

10-11/25

dotnetpro

by DWX

.NET 10



.NET, C#, ASP.NET Core, Blazor, EF Core, SQL Server und Co.: Welche Neuerungen bringt Version 10? Und für wen lohnt sich der Umstieg besonders? Ab Seite 6

PROGRAMM AB SEITE 65

.NET DEVELOPER CONFERENCE '25

Die Konferenz für .NET-Entwickler:innen

DWX
EVENT

24. BIS 27. NOVEMBER '25
Pullman Cologne Hotel in Köln



Du willst ein Thema durchdringen – ohne Zeitverlust, ohne heiße Luft?

Ein digitales Kompaktformat, das auf den
Punkt bringt, was du wirklich brauchst:

- **READ:** Finde mit unseren Online-Beiträgen deinen Einstieg in das Spotlight-Fokusthema.
- **LISTEN:** Im Interview ordnen Redaktion und Expert:innen relevante Aspekte für dich ein.
- **LEARN:** Unsere Lernvideos vertiefen das Fokusthema – kompakt und übersichtlich, professionell und praxisnah.



NEW SPOTLIGHT

„KI-gestützte Workflows
für Developer“

Ab sofort online verfügbar





Wissen schlägt KI

Macht Künstliche Intelligenz Entwickler:innen überflüssig? Nein, ganz im Gegenteil: Mit der richtigen Expertise sind sie wichtiger denn je.

Künstliche Intelligenz kann jetzt auch coden! Dieser Satz ist in unserer Branche seit Langem ein Allgemeinplatz. Mehr noch: KI produziert nicht nur Code, sondern generiert inzwischen auch Softwaretests, erstellt Architekturkonzepte oder erledigt komplette Refactorings. Doch fest steht auch: Ohne den Menschen bleibt die KI ein Werkzeug mit eingebauter Unschärfe.

Denn seien wir mal ehrlich: So leistungsfähig KI auch sein mag, sie ist nur so gut wie die Prompts, die man ihr gibt, und die Bewertung dessen, was sie ausspuckt. Wer nicht weiß, wie saubere Architektur aussieht, kann nicht erkennen, ob der generierte Code wartbar ist. Wer die Prinzipien von Clean Code nicht verinnerlicht hat, wird kaum bemerken, wenn ein vermeintlich funktionierendes Snippet langfristig technische Schulden anhäuft. Und wer nicht versteht, was ein wirklich gutes Interface ausmacht, vertraut blind auf Aussagen der KI zur Nutzerfreundlichkeit, mit allen Risiken und Nebenwirkungen.

Deshalb gilt: Nur diejenigen Entwickler:innen, die über profunde, solide und breit gefächerte Kenntnisse verfügen, können KI als das nutzen, was sie ist – ein fähiger Co-Worker, der einen selbst tatkräftig unterstützt und hilfreiche Anregungen gibt, aber keinen Freifahrtschein hat.

„Im Zeitalter der Künstlichen Intelligenz ist das Wissen um ihren richtigen Einsatz essenziell.“

Regelmäßige Fort- und Weiterbildung ist unabdingbare Voraussetzung für solch einen Wissensschatz. Nur wer sich fortbildet, lernt neue Technologien kennen, schärft das eigene Urteilsvermögen, entwickelt ein besseres Gespür für die Qualität von Software, weiß die Arbeit von KI kritisch zu reflektieren und sie so optimal einzusetzen.

Die Lektüre einer Fachzeitschrift wie der dotnetpro ist eine Möglichkeit, sich das dafür nötige Wissen anzueignen. Ebenso wichtig ist es aber, dieses Wissen zu hinterfragen und auf den Prüfstand zu stellen. Mein Rat: Diskutieren Sie mit Kolleg:innen über das richtige Werkzeug, das beste Vorgehen, den optimalen Lösungsansatz. Oder tauschen Sie sich mit Gleichgesinnten aus, wo und wann immer es Ihnen möglich ist. Haben Sie Mut und trauen Sie sich – es lohnt!

Als .NET-Entwickler:innen haben Sie übrigens bald wieder die Gelegenheit dazu – auf der .NET Developer Conference Ende November in Köln.

Bis dahin gilt: Ich wünsche Ihnen eine schöne Zeit mit der neuen dotnetpro!

Herzlichst

Fernando Schneider
Chefredakteur dotnetpro



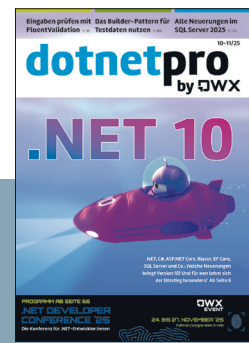
Marc Müller

zeigt auf, wie .NET 10 moderne Cloud-Native-Architekturen effizienter, sicherer und performanter macht (S. 6)



Stefan Lieser

verrät, warum Sie Ressourcen und Dialoge klar vom Kern Ihrer Anwendung trennen sollten (S. 112)



INHALT

Focus

Verteilte Cloud– Applikationen mit .NET 10

Was ist neu mit .NET 10?

6

Web–Framework, frisch poliert

Neues in ASP.NET Core 10

14

Weiterhin mit ordentlich Tempo

Neuerungen in Blazor 10.0, Teil 1

19

.NET Stack

NETirol: FluentValidation

Komplexe Prüfregeln für Benutzereingaben anlegen

30

Aloha für MAUI

.NET MAUI und iOS

32

API Contracts perfekt im Griff

OpenAPI, Swagger und Scalar in ASP.NET Core meistern

38

SignalRC – in Echtzeit ans Steuer

Auf in die Welt mit SignalR, Raspberry PI und sechs Rädern

46

Data Access: Wenig Neues

Entity Framework Core und LINQ in .NET 10.0 Preview 1 bis 6

54

Artificial Intelligence

Pizza bestellen mit Struktur

Das Framework Semantic Flow

70

Vom Co–Pilot zum Co–Worker

KI-gestützte Development–Workflows

79

KI im Mittelstand braucht mehr als Technik

Zwischen Hype und Realität

88

In gemeinsamer Mission

KI-Agenten mit OpenAI und Semantic Kernel entwickeln

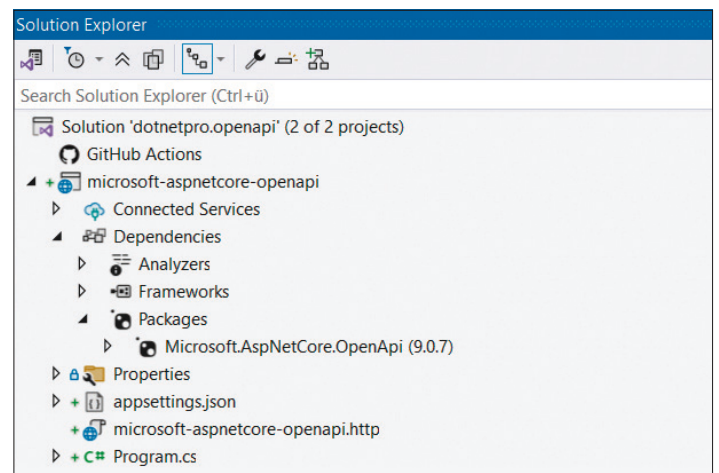
95



Bild: Microsoft

Alle Jahre wieder: Mit welchen Neuerungen .NET 10 aufwartet und was Sie sich unbedingt anschauen sollten.

6



Alles im Griff: Mit OpenAPI, Swagger, Scalar und ASP.NET Core Schnittstellen beschreiben und dokumentieren.

38

Experten in dieser Ausgabe



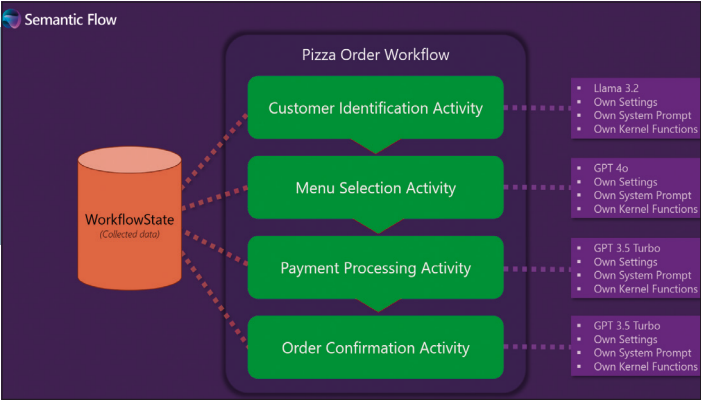
Dotnet–Doktor
Holger Schwichtenberg nimmt Version 10 des Entity Framework Core unter die Lupe.

54



David Tielke
ist davon überzeugt, dass der Product Owner in Softwareprojekten unersetzlich ist.

102



Alles im Fluss: Wie Sie mit Semantic Kernel und Semantic Flow unter .NET komplexe KI-Prozesse im Griff haben. 70

KI-gestützte Workflows: Welche Vorteile sich aus dem Einsatz von GitHub Copilot, Copilot Spaces, Copilot Spark und Coding Agents für Ihre Arbeit als Entwickler:in ergeben.

79

PostgreSQL-Serie, Teil 4: Installationen absichern sowie Backup- und Restore-Operationen durchführen. 132

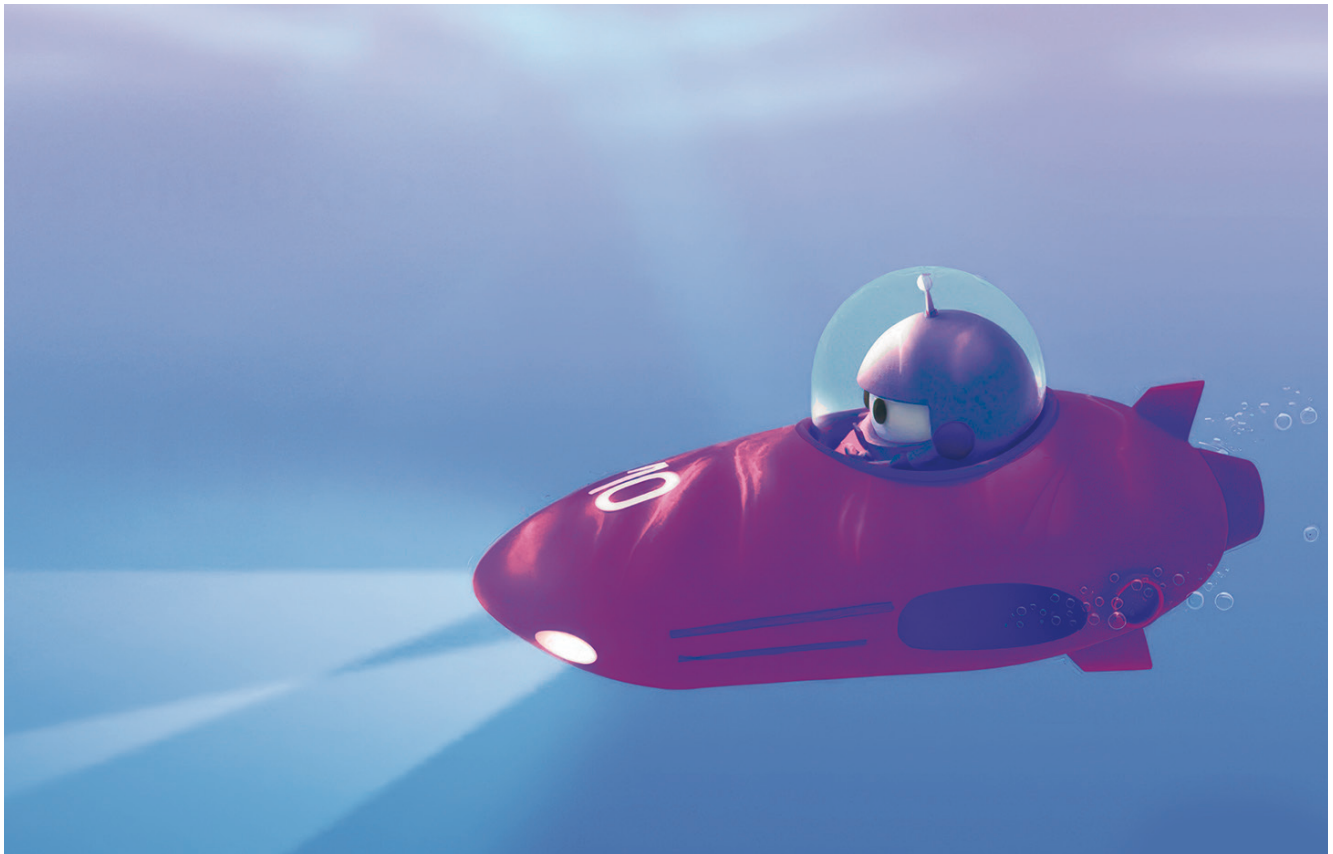


Cross Concerns

Dauids Deep Dive: Der Process Owner	
Rollen in der Softwareentwicklung, Teil 2	102
Bob, der Testdaten-Baumeister	
Skalierbare Testarchitekturen mit KI-Tools und Testdata-Buildern entwickeln	104
Ressourcen und Dialoge	
Clean Code und Architektur, Teil 3	112
What else?	
Viel Licht, aber auch Schatten	
SQL Server 2025 für Entwickler	120
Backup und Restore	
PostgreSQL-Installationen verwalten	132

People & Soft Skills

Time-out: Wann ist die KI ein Mensch?	
Über die Frage, ob Identität eine Nebenwirkung unseres eigenen neuronalen Netzes ist	140
Editorial	3
Inhalt	4
Arbeitsmarkt	142
Lernstoff	144
Impressum/Vorschau/Inserenten	146



Bildquelle: Microsoft

WAS IST NEU MIT .NET 10?

Verteilte Cloud-Applikationen mit .NET 10

Wie der kommende LTS-Release von .NET 10 moderne Cloud-Native-Architekturen mit ASP.NET Core 10 effizienter, sicherer und performanter macht.

Der Release von .NET 10 (Long-Term Support) rückt näher – der finale Launch ist für November 2025 geplant. Besonders spannend: ASP.NET Core 10 Preview bietet gezielte Verbesserungen für moderne, verteilte Cloud-Native-Anwendungen, bei denen Skalierbarkeit, Sicherheit und Betriebstransparenz entscheidend sind.

In diesem Artikel beleuchten wir, wie .NET 10 diese Anforderungen adressiert – mit Fokus auf Sicherheit, Observability, Performance und Effizienz. Auch klassische modulare Monolithen profitieren von den SDK-Optimierungen, der verbesserten Diagnostik und modernen API-Features. Im Folgenden sehen wir uns im Detail die besonders relevanten Neuerungen in ASP.NET Core 10 für Cloud-Architekturen an:

- Sichere Cloud-Native-Authentifizierung, inklusive Passkey-Support und HTTPS-Entwicklung mit `*.localhost`

- Server-Sent Events (SSE) für effizienteres Event-Streaming und reaktive Anwendungen
- Verbesserungen bei Observability und Diagnostics, etwa durch konfigurierbares Exception-Logging und integrierte Auth-Metriken
- Performance-Optimierungen und Produktivitäts-Features wie native Container-Images und neue CLI-Erweiterungen
- Moderne APIs und Minimal APIs, mit neuen Templates, Validierungsmöglichkeiten und OpenAPI 3.1

Sichere Applikationen zu bauen soll einfach sein

Die Zahl der Cyberangriffe ist auf einem Allzeithoch, weshalb es die Pflicht jedes Softwareentwicklungsteams ist, Applikationen entsprechend zu härten und auf die neuesten Standards zu aktualisieren. Grundsätzlich war es schon immer möglich,

sichere Applikationen mit ASP.NET Core zu implementieren. Ein häufiger Kritikpunkt der Entwickler-Community war aber, dass Informationen schwer zu finden waren und oft Drittanbieter- oder Open-Source-Libraries verwendet werden mussten. Wichtig für die Sicherheit ist daher, dass es einfach und verständlich sein sollte, Sicherheitsfunktionen zu integrieren.

Mit ASP.NET Core 10 stehen zwei Bereiche im Fokus:

- Microsoft verbessert die Dokumentation zu Identity und Authentication, um eine breitere und zugleich korrekte Nutzung der Sicherheitsfunktionen zu fördern.
- Neu unterstützt ASP.NET Core Identity WebAuthn und Passkeys.

In einer verteilten Anwendung bearbeiten mehrere Services eine Aufgabe, die oft von einem Benutzeraufruf initiiert wird. Auch wenn die Services untereinander so sicher wie möglich kommunizieren, kann ein Angreifer bei kompromittierten Benutzer-Credentials dennoch problemlos auf das System zugreifen. Dies verdeutlicht erneut, dass der Mensch und insbesondere Benutzerpasswörter das schwächste Glied der Sicherheitskette darstellen. Sofern die Authentisierung nicht an einen externen Identity-Provider ausgelagert wird, sollten sich Teams intensiv mit Passkeys beschäftigen.

Passkeys sind eine neue Form kryptografischer Credentials, die als Haupt-Authentisierungsfaktor oder als Ergänzung zum Passwort verwendet werden können. Technisch handelt es sich um ein Public-/Private-Key-Paar, das auf einem physischen Gerät sicher gespeichert oder über Apps (mit und ohne Hardware-Verschlüsselung) in der Cloud verwaltet wird. Bekannte Vertreter sind YubiKey, Microsoft Authenticator oder iCloud Keychain. Der Private Key wird für Signaturen verwendet und verlässt den Authenticator niemals, während der Public Key serverseitig zur Verifikation dient. Zusätzlich sind Schlüsselpaare sowohl an eine spezifische Anwendung (Relying Party) als auch an einen Benutzer gebunden, wodurch sie nicht für andere Applikationen genutzt werden können. **Bild 1** und **Bild 2** illustrieren die Funktionsweise von Passkeys.

Vor ASP.NET Core 10 konnten Passkeys bereits mithilfe der Open-Source-Bibliothek `fido2-net-lib` integriert werden. Nun hat Microsoft Teile davon direkt in ASP.NET Core Identity übernommen. Ab ASP.NET Core 10 können Passkeys daher nativ in Identity verwaltet werden; Browser-Integrationen stellen die Kommunikation mit Authenticators sicher. Micro-

soft hat bewusst nur stabile Funktionen integriert, während Features, die sich noch in dynamischer Entwicklung befinden, vorerst fehlen.

Wer die neuen Passkeys-Funktionen ausprobieren möchte, kann dies mit den neuen Blazor-Scaffolding-Templates tun. Hierzu erstellt man eine neue Blazor-Applikation mit individueller Authentisierung mit dem Kommando:

```
dotnet new blazor -au Individual
```

Darüber hinaus hat Microsoft die Authentication-Scaffolders erweitert. Dabei handelt es sich um interaktive CLI-Befehle, die ASP.NET-Core-Anwendungen automatisch um Authentifizierungsendpunkte für Entra ID und Identity ergänzen.

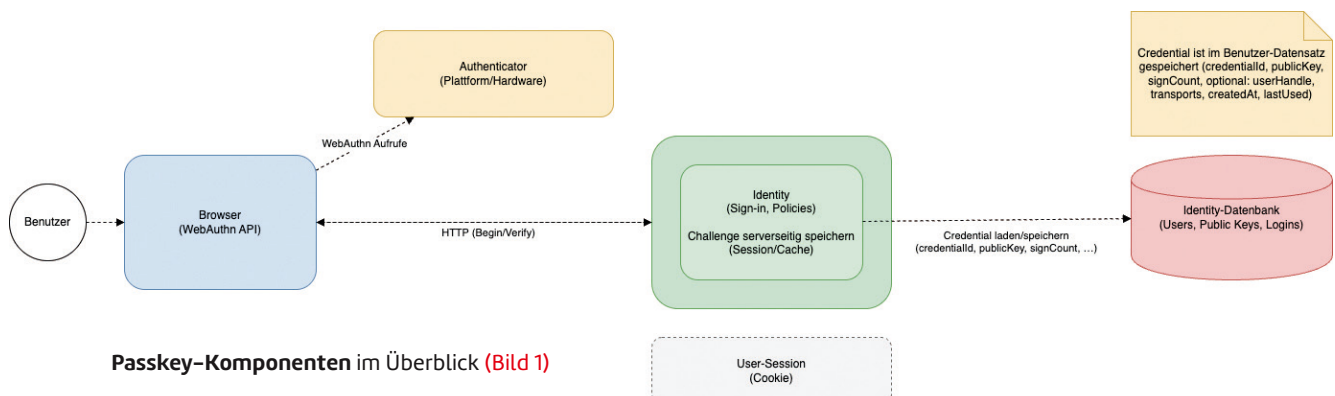
Ein weiteres Detail betrifft die Cookie-Authentifizierung in APIs. Bisher leitete ASP.NET Core bei fehlender Authentifizierung standardmäßig auf eine Login-Seite um – ein Verhalten, das für Browser-Szenarien sinnvoll ist, bei APIs jedoch problematisch sein kann. Ab ASP.NET Core 10 wird stattdessen standardmäßig ein `401 Unauthorized`-HTTP-Statuscode zurückgegeben, wenn API-Endpunkte ohne gültige Authentifizierung aufgerufen werden. Dieses Verhalten ist konsistent mit RESTful-API-Design und erleichtert die Integration mit API-Clients oder automatisierten Tests.

Validierung und OpenAPI

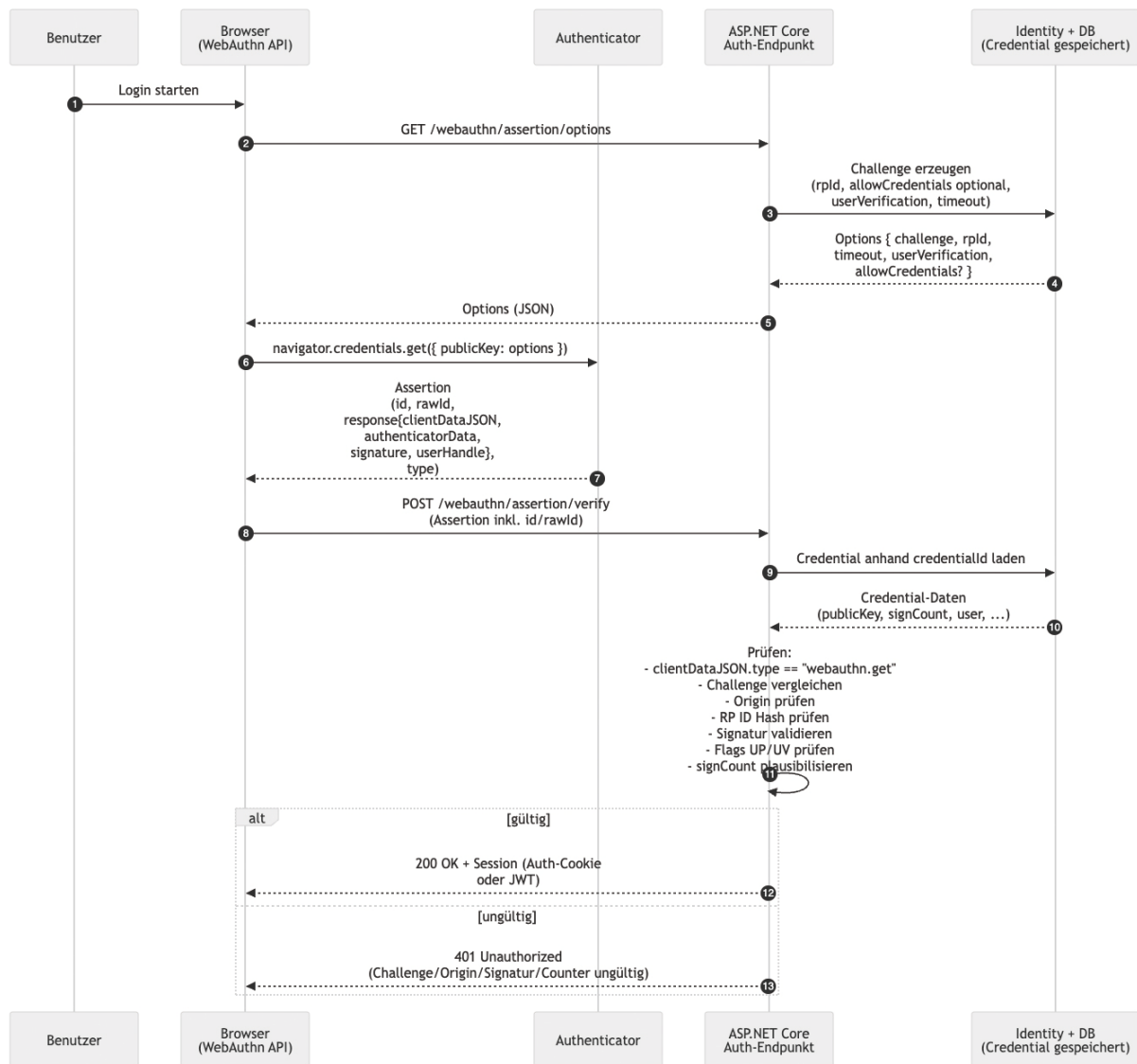
Herzstück jeder verteilten Anwendung ist das API – implementiert in ASP.NET Core wahlweise über klassische API-Controller oder die leichtgewichtigen Minimal APIs. Bisher war die Anpassung von Validierungsfehlermeldungen in Minimal APIs eher aufwendig. .NET 10 verbessert dies grundlegend:

- Minimal APIs integrieren nun die *IProblemDetailsService*-Infrastruktur, die bislang MVC vorbehalten war.
- Der neue eingebaute Validierungssupport wird mit *builder.Services.AddValidation()* aktiviert.
- Ein Endpoint-Filter validiert automatisch alle Parameter- und Body-Typen der Minimal-API-Handler.
- Entwickler können bei Bedarf eine eigene *ProblemDetailsService*-Implementierung registrieren, um Fehlerantworten individuell anzupassen.

Damit stehen Minimal APIs in puncto Validierung und einheitliche Fehlerausgaben erstmals auf Augenhöhe mit MVC. ►



Passkey-Komponenten im Überblick (Bild 1)



Flow eines Passkey-Logins (Bild 2)

Ein weiterer wichtiger Fortschritt: Validierungsattribute können nun konsistent auf Klassen, Records und deren Primärkonstruktoren angewendet werden. So lassen sich Modelle flexibel gestalten, ohne auf Validierung zu verzichten.

Listing 1 zeigt eine Minimal-API-Anwendung mit aktueller .NET-10-OpenAPI-Integration und Validierung von Records und Klassen. Was dieses Beispiel verdeutlicht:

- **Automatische Validierung:** Durch `builder.Services.AddValidation()` prüft ASP.NET Core eingehende Requests automatisch. Ungültige Anfragen werden vor Ausführung des Handlers abgefangen, und es wird eine standardisierte 400 *Bad Request*-Antwort mit *ProblemDetails* zurückgegeben.
- **Konsistente Attributverwendung:**
 - *CreateOrder*: Record mit Validierungsattributen am Primärkonstruktor.
 - *OrderDto*: Klassisches Modell mit Attributen auf Properties, inklusive verschachtelter Klasse *Address*.

Beide Varianten funktionieren nahtlos und zeigen die Flexibilität der neuen Validierungslogik.

- **Native OpenAPI-Integration:** Dank `AddOpenApi()` und `MapOpenApi()` in .NET 10 werden Minimal APIs nativ per OpenAPI-Schema dokumentiert. Mit `Scalar.AspNetCore` lässt sich zusätzlich eine moderne, interaktive Referenz (`/scalar/v1`) bereitstellen. Validierungsattribute erscheinen automatisch im Schema (Constraints wie `minLength`, `maxLength` oder `pattern` et cetera).
- **ProblemDetails-Standardisierung:** Fehlerantworten folgen RFC 7807. Ein Beispielauszug bei ungültigen Eingaben ist in **Listing 2** ersichtlich.
- **Feature-Parität MVC <=> Minimal APIs:** Entwickler erhalten denselben Komfort wie in MVC: automatische Validierung, standardisierte Fehlerausgaben und flexible Anpassbarkeit über eigene *ProblemDetailsService*-Implementierungen.

Mit .NET 10 erhält ASP.NET Core erstmals vollständige Unterstützung für OpenAPI 3.1. Das bedeutet, dass Minimal APIs und Controller-basierte Endpoints nun OpenAPI-Schemas erzeugen, die aktuelle Standards wie JSON Schema 2020-12 und präzisere Typdefinitionen berücksichtigen. Auch `nullable`-Felder,

oneOf/anyOf-Kombinationen und weitere Features werden korrekt beschrieben. Entwicklerinnen und Entwickler profitieren von einer verbesserten Werkzeugkette, da moderne API-Clients und Drittsysteme OpenAPI 3.1 zunehmend voraussetzen. Die neue native OpenAPI-Integration in ASP.NET Core macht es einfacher, aktuelle Spezifikationen ohne zusätzliche Pakete zu nutzen.

Server-Sent Events (SSE)

Ein weiteres neues Feature in ASP.NET Core 10 ist die native Unterstützung für Server-Sent Events (SSE). Dabei handelt es sich um einen Mechanismus, bei dem ein Server Nachrichten unidirektional an einen Client senden kann – ideal für Echtzeit-Updates ohne komplizierte Infrastruktur.

Auf den ersten Blick erinnert SSE an SignalR oder WebSockets – allerdings ist es deutlich leichtgewichtiger. Im Gegensatz zu WebSockets, die eine bidirektionale Verbindung er-

möglichen, basiert SSE auf einem einfachen HTTP-GET-Request. Die Verbindung bleibt geöffnet, und der Server sendet Nachrichten fortlaufend über den Response-Stream. Die Übertragung erfolgt chunked, was ein einfaches Streaming vom Server zum Client erlaubt.

Technisch ist SSE nicht neu, aber die Implementierung in ASP.NET Core war bislang umständlich und erforderte individuelle Lösungen. Mit .NET 10 gibt es dafür direkten Framework-Support: *TypedResults.ServerSentEvents* übernimmt die korrekte HTTP-Konfiguration, sodass sich SSE-Endpunkte mit wenigen Zeilen Code umsetzen lassen.

Listing 3 zeigt einen minimalistischen SSE-Endpunkt mit Minimal API. Die Nachrichten, hier automatisch erzeugte Zufallszahlen, werden in diesem Beispiel über ein *IAsyncEnumerable* generiert. Jeder HTTP-Client kann die Events konsumieren, und im Browser lassen sie sich einfach mit *EventSource* abonnieren. **Listing 4** zeigt den dazugehörigen ►

● Listing 1: Validierung mit Minimal API (Teil 1)

```
// Program.cs (.NET 10 Preview)
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Http.HttpResults;
using Microsoft.AspNetCore.Http;
using Scalar.AspNetCore;

namespace Validation;

// --- Models ---
public record CreateOrder(
    [Required, StringLength(100)] string Product,
    [Range(1, 1000)] int Quantity,
    [Required, EmailAddress] string CustomerEmail
);

public class Address
{
    [Required, StringLength(120)]
    public string Street { get; init; } = default!;

    [Required, StringLength(80)]
    public string City { get; init; } = default!;
}

public class OrderDto
{
    [Required, StringLength(100)]
    public string Product { get; init; } = default!;

    [Range(1, 1000)]
    public int Quantity { get; init; }

    [Required, EmailAddress]
    public string CustomerEmail { get; init; } =
        default!;

    [Required]
    public Address Address { get; init; } = default!;
}

// Shape returned by the API (includes
// server-generated fields)
public record OrderResponse(Guid Id, string Product,
    int Quantity, string CustomerEmail, Address Address);

// --- Program ---
public static class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // .NET 10: native OpenAPI-Services registrieren
        builder.Services.AddOpenApi();

        // ASP.NET Core 10 minimal validation
        builder.Services.AddValidation();

        var app = builder.Build();

        if (app.Environment.IsDevelopment())
        {
            app.MapOpenApi(); // /openapi/v1.json (and
                            // .yaml if mapped)
            app.MapScalarApiReference(); // /scalar/v1
        }

        // Group + tag endpoints
```

● Listing 1: Validierung mit Minimal API (Teil 2)

```

var orders = app.MapGroup("/orders")
    .WithTags("Orders");

// Create (rich DTO) – union results let OpenAPI
// infer 201/400
orders.MapPost("/", (OrderDto dto) =>
{
    var id = Guid.NewGuid();
    var response = new OrderResponse(
        id, dto.Product, dto.Quantity,
        dto.CustomerEmail, dto.Address);
    return TypedResults.Created($" /orders/{id}",
        response);
})
.WithName("CreateOrder")
.WithSummary("Create a new order")
.WithDescription("Creates an order and returns

it with a Location header.");

// Create (simple record)
orders.MapPost("/simple", (CreateOrder order) =>
{
    var id = Guid.NewGuid();
    var response = new { Id = id, order.Product,
        order.Quantity, order.CustomerEmail };
    return TypedResults.Created(
        $" /orders/simple/{id}", response);
})
.WithName("CreateOrderSimple")
.WithSummary("Create a new simple order");

app.Run();
}
}

```

JavaScript-Code, der auf Basis der empfangenen Events die Webseite aktualisiert.

Das bisherige Beispiel ist bewusst einfach gehalten und soll vor allem den Einstieg in die Arbeit mit SSE verdeutlichen. In realen Anwendungen stellen sich jedoch schnell komplexere Anforderungen: Oft soll der Datenstrom benutzerspezifisch sein, Verbindungen sollen nach Unterbrechungen automatisch fortgesetzt werden, und es ist häufig sinnvoll, Ereignistypen zu unterscheiden. Listing 5 zeigt eine weiterentwickelte Implementierung, die genau diese Herausforderungen adressiert. Hier werden Fortschrittsmeldungen zu laufenden Aufgaben eines bestimmten Benutzers in Echtzeit an den Client übertragen. Die Identifikation des Benutzers erfolgt über ein URL-Segment, könnte aber ebenso über eine Authentisierungs-lösung oder eine Session-Abfrage erfolgen. Wird die Verbindung unterbrochen, sendet der Browser bei einem erneuten Verbindungsaufbau automatisch die zuletzt empfangene Event-ID im HTTP-Header mit. Der Server kann diese Information nutzen, um den Stream nahtlos an der richtigen Stelle fortzuführen. Darüber hinaus werden die Ereignisse mit einem Typ versehen, sodass der Client sie gezielt verarbeiten und beispielsweise unterschiedliche UI-Elemente aktualisieren kann. Aus Platzgründen zeigen wir hier nur Codeauszüge; der vollständige Quellcode ist unter [1] auf GitHub verfügbar.

Während Server-Sent Events (SSE) und WebSockets beziehungsweise SignalR alle für Echtzeit-Kommunikation zwischen Server und Client eingesetzt werden, unterscheiden sie sich in Konzept, Leistungsfähigkeit und Einsatzszenarien deutlich, wie Tabelle 1 zeigt. Fazit für die Praxis:

- SSE überzeugt durch geringen Implementierungsaufwand und breite Browser-Unterstützung. Es eignet sich besonders für Dashboards, Live-Metriken, IoT-Meldungen und Benachrichtigungen, bei denen nur der Server Nachrichten an den Client sendet.

● Listing 2: Validierungsserror gemäß RFC 7807

```

HTTP/1.1 400 Bad Request
Connection: close
Content-Type: application/json; charset=utf-8
Date: Tue, 02 Sep 2025 12:54:22 GMT
Server: Kestrel
Transfer-Encoding: chunked

{
  "title":
    "One or more validation errors occurred.",
  "errors": {
    "Product": [
      "The Product field is required."
    ],
    "Quantity": [
      "The field Quantity must be between 1 and
        1000."
    ],
    "CustomerEmail": [
      "The CustomerEmail field is not a valid
        e-mail address."
    ],
    "Address.Street": [
      "The Street field is required."
    ],
    "Address.City": [
      "The City field is required."
    ]
  ]
}

```


- WebSockets/SignalR sind die richtige Wahl, wenn bidirektionale Kommunikation oder komplexe Interaktionen benötigt werden, etwa in Chat-Anwendungen, Echtzeitspielen oder kollaborativen Tools.

Mit ASP.NET Core 10 lassen sich beide Ansätze komfortabel nutzen: SignalR und WebSockets sind seit Jahren etabliert, und SSE ist nun erstmals direkt im Framework integriert und ähnlich einfach zu implementieren wie ein Minimal API.

Entwicklung verteilter Anwendungen

Die Entwicklung verteilter Anwendungen bringt zahlreiche Herausforderungen mit sich. Kleine, individuell deploybare und dynamisch skalierbare Services sind zwar ein Segen für kurze Releasezyklen und eine optimale Auslastung von Ressourcen, doch die Komplexität steigt erheblich. Beim Debugging reicht es selten aus, einen einzelnen Service zu starten; meist müssen weitere Abhängigkeiten oder entsprechende Mocks bereitgestellt werden. Auch die Fehlersuche gestaltet sich schwieriger: Nachrichten und Events zwischen vielen Instanzen sind ohne Distributed Tracing kaum nachvollziehbar.

An dieser Stelle lohnt es sich, .NET Aspire zu betrachten – auch wenn es als eigenständiges Tool entwickelt wird und nicht direkt an den .NET-10-Release gekoppelt ist. Aspire ►

Listing 3: Implementierung eines SSE-Endpunkts

```
app.MapGet("/api/randomnumbers",
    (Cancellation token ct) =>
    {
        return TypedResults.ServerSentEvents(
            GenerateRandomNumbers(ct));
    });

static async IAsyncEnumerable<int>
    GenerateRandomNumbers([EnumeratorCancellation]
        CancellationToken cancellationToken)
    {
        var random = new Random();

        while (!cancellationToken.IsCancellationRequested)
        {
            yield return random.Next(1, 1000);
            await Task.Delay(1000, cancellationToken);
            // Wait 1 second
        }
    }
```

Listing 4: Konsumieren eines SSE-Endpunkts im Browser

```
<script>
    let count = 0;
    const numberDisplay =
        document.getElementById('numberDisplay');
    const statusDiv = document.getElementById('status');
    const countSpan = document.getElementById('count');

    // Connect to the SSE endpoint
    const eventSource =
        new EventSource('/api/randomnumbers');

    // Handle successful connection
    eventSource.onopen = function() {
        statusDiv.textContent = 'Connected to SSE stream';
        statusDiv.className = 'status connected';
        console.log('SSE connection opened');
    };

    // Handle incoming messages
    eventSource.onmessage = function(event) {
        const randomNumber = event.data;

        // Update the display
        numberDisplay.textContent = randomNumber;
        numberDisplay.classList.add('updated');

        // Remove the animation class after a short delay
        setTimeout(() => {
            numberDisplay.classList.remove('updated');
        }, 300);

        // Update counter
        count++;
        countSpan.textContent = count;

        console.log('Received random number:',
            randomNumber);
    };

    // Handle connection errors
    eventSource.onerror = function(error) {
        statusDiv.textContent =
            'Connection error - will automatically retry';
        statusDiv.className = 'status disconnected';
        console.error('SSE connection error:', error);
    };

    // Clean up on page unload
    window.addEventListener('beforeunload', function() {
        if (eventSource) {
            eventSource.close();
        }
    });
</script>
```

● **Tabelle 1: Übersicht über Echtzeit-Kommunikationsmöglichkeiten zwischen Server und Client**

Merkmal	Server-Sent Events (SSE)	WebSockets / SignalR
Kommunikationsrichtung	Unidirektional: Server → Client	Bidirektional: Server ↔ Client
Transport	Reines HTTP/1.1, GET-Request, <i>text/event-stream</i>	Spezielles WebSocket-Protokoll; HTTP/1.1-Upgrade oder HTTP/2
Implementierungskomplexität	Sehr einfach, nativer Browser-Support über EventSource	Aufwendiger, Client- und serverseitige API-Layer notwendig
Verbindungs-Handling	Automatisches Reconnect, Last-Event-ID für Fortsetzung	Reconnect und State-Handling müssen explizit implementiert werden
Datenfluss	Nur Push vom Server zum Client	Beidseitige Kommunikation, etwa für Chats, Multiplayer-Spiele
Performance und Skalierung	Leichtgewichtig, ideal für viele Clients mit Broadcasts	Höhere Flexibilität, kann für viele gleichzeitige Clients teurer sein
Protokoll-Features	Nur Text-Events; keine Binärdaten	Text- und Binärdaten, Streams
Browser-Support	Nativ seit vielen Jahren	Browser-API vorhanden, aber Implementierung komplexer
ASP.NET Core 10 Support	Neu: TypedResults.ServerSentEvents	SignalR/WebSockets sind etabliert und „feature-complete“
Typische Szenarien	Live-Updates, Dashboards, Benachrichtigungen, IoT-Events	Chats, Games, Kollaborationstools, komplexe Interaktionen

ist ein Opinionated Stack für die Entwicklung verteilter Cloud-Anwendungen mit .NET. Es umfasst Tools, Bibliotheken, Templates und ein Dashboard, das komplexe Aufgaben wie Service-Orchestrierung, Service-Discovery, Telemetrie, Health Checks und lokale Entwicklung erheblich vereinfacht. Was zuvor meist nur mit Docker Compose möglich war, gelingt mit Aspire direkt im .NET-Ökosystem.

Teams, die in ihrer Systemlandschaft konsequent auf .NET setzen, können die Orchestrierung sogar direkt in C# schreiben und verwalten. Die integrierte Observability mit Dashboard erleichtert Analyse und Tracing, ohne dass zusätzliche Tools benötigt werden. Doch auch für Teams, die weiterhin mit Docker Compose oder einem technologieübergreifenden Observability-Stack arbeiten möchten, bietet Aspire Vorteile: Es ist optional und wird über NuGet-Pakete und sogenannte Service Defaults eingebunden, wodurch ASP.NET-Core-Anwendungen automatisch für Tracing, Metriken und Logging konfiguriert werden. Dieselben Features können natürlich auch ohne Aspire manuell aktiviert werden – Grund genug also, einen Blick auf die Neuerungen in ASP.NET Core 10 selbst zu werfen.

ASP.NET Core setzt konsequent auf den OpenTelemetry-Standard. Mit .NET 10 wurden Metriken und Telemetriedaten deutlich erweitert. Zu den neuen, direkt im Framework integrierten Kennzahlen gehören unter anderem Requests per Second, Request Duration, Authentication Duration und Authorization Duration. Wer Aspire verwendet, aktiviert diese Metriken automatisch über die Service Defaults. Ohne Aspire können sie manuell über das Paket Microsoft.AspNetCore.OpenTelemetry im Startup konfiguriert werden.

Eine weitere Hürde in der lokalen Entwicklung sind host-basierte Szenarien mit Authentifizierung. Komplexe Tests funktionieren häufig nicht zuverlässig, wenn Dienste nur über IP-Adressen und Ports erreichbar sind. Mit .NET 10 unterstützt Kestrel nun Wildcard-Hostnames wie **.localhost*, die

automatisch auf 127.0.0.1 gebunden werden. So lassen sich Services bequem unter Hostnamen wie *orders.api.localhost* oder *inventory.api.localhost* erreichen. Auch das Entwickler-zertifikat-Tooling wurde erweitert: Zertifikate für Subdomains wie **.dev.localhost* können nun einfach erstellt werden.

Neben der Orchestrierung hat Microsoft auch den Container-Build-Prozess vereinfacht. Projekte lassen sich jetzt direkt über *dotnet publish* zu Container-Images bauen; alle Build-Parameter können in den *Containerize*-Settings der *.csproj*-Datei konfiguriert werden. Parallel dazu wurden die offiziellen .NET-Container-Images für Version 10 aktualisiert. Standardmäßig kommt nun Ubuntu 24.04 LTS für Basis-Images wie 10.0-sdk und 10.0-runtime zum Einsatz; alternativ stehen auch Debian-13-Images (Trixie) zur Verfügung. Für produktive Szenarien empfiehlt Microsoft zunehmend sogenannte Chiseled Images. Diese distributionslosen, gehärteten Images enthalten nur das Nötigste und sind besonders sicher. Mit den neuen Chiseled Manifests ist die Verwaltung der Komponenten deutlich einfacher geworden – ein Fortschritt, der nun auch in die offiziellen .NET-10-Images eingeflossen ist.

Performance-Optimierungen

Wie bei jedem neuen Release hat Microsoft auch in .NET 10 erheblich die Performance optimiert. Die Liste der Verbesserungen ist lang, weshalb wir uns hier auf die wichtigsten Neuerungen konzentrieren.

Eine besonders relevante Optimierung für verteilte Anwendungen und Hyperscaling betrifft das Memory-Pool-Management von Kestrel. Der Webserver verwendet intern Speicher-pools, um Buffers wiederzuverwenden und so unnötige Allokationen zu vermeiden. Das Problem in früheren Versionen: Unter Last wurden die Pools zwar erweitert, dann aber nicht wieder verkleinert. Dadurch konnte bei Lastspitzen sehr viel Speicher dauerhaft gebunden bleiben. Mit .NET 10 verfügt

● Listing 5: Implementierung eines SSE-Endpunkts

```
app.MapGet("/api/task-progress/{userId}", (
    string userId,
    TaskProgressService taskService,
    UserSessionService userService,
    HttpRequest httpRequest,
    HttpResponse httpResponse,
    CancellationToken ct) =>
{
    try
    {
        // Handle reconnection with Last-Event-ID
        var lastEventId = httpRequest.Headers.TryGetValue(
            "Last-Event-ID", out var id)
            ? id.ToString()
            : null;

        if (!string.IsNullOrEmpty(lastEventId))
        {
            app.Logger.LogInformation("User {UserId}
                reconnected, last event ID: {LastEventId}",
                userId, lastEventId);
        }
        else
        {
            app.Logger.LogInformation("User {UserId}
                connected to SSE stream", userId);
        }

        // Register user session
        userService.RegisterUser(userId);

        // Create the SSE stream with user-specific
        // filtering
        var stream = taskService.GetTaskProgressUpdates(
            userId, lastEventId, ct);

        return TypedResults.ServerSentEvents(
            stream, eventType: "taskUpdate");
    }
    catch (Exception ex)
    {
        app.Logger.LogError(ex, "Error in SSE endpoint
            for user {UserId}", userId);
        return Results.StatusCode(500);
    }
});
```

Kestrel nun über eine automatische Eviction-Strategie, die ungenutzte Puffer zurückgibt und den Speicherverbrauch deutlich stabilisiert – ein großer Vorteil bei variablen Workloads.

Auch beim JSON-Handling hat sich viel getan. System.Text.Json unterstützt jetzt PipeReader direkt. ASP.NET Core nutzt diese Erweiterung für eingehende Requests, wodurch sich zusätzliches Buffering und unnötige Byte-Array-Allokationen vermeiden lassen. Gleichzeitig hat es das lange erwartete Update für JSON Patch in den Release geschafft: Die Implementierung basiert nun vollständig auf System.Text.Json und nicht mehr auf Newtonsoft.Json. Die Performance-Gewinne sind beeindruckend – ein JSON-Patch-Dokument wird bis zu 100-mal schneller verarbeitet, bei gleichzeitig deutlich reduziertem Speicherbedarf.

Über diese beiden Highlights hinaus profitiert .NET 10 von vielen weiteren Optimierungen: Verbesserungen am JIT-Compiler (RyuJIT), effizientere Loop-Invariant-Code-Motion und erweitertes On-Stack Replacement (OSR) sorgen für schnelleren Code zur Laufzeit. Hinzu kommen gezielte Anpassungen in der Thread-Pool-Steuerung, die insbesondere bei hoher Parallelität für konstantere Latenzen sorgen.

In der Summe ergibt sich ein Release, der nicht nur punktuelle Hotspots adressiert, sondern in allen Schichten spürbar an Performance gewinnt – vom Webserver über die JSON-Verarbeitung bis hin zum Compiler und dem Runtime-Scheduling. Für Entwickler verteilter Anwendungen heißt das: Mehr Durchsatz, weniger Speicherverbrauch und eine insgesamt stabilere Performance, selbst unter Last.

Fazit

.NET 10 und ASP.NET Core 10 liefern keine radikale Neuausrichtung, sondern eine konsequente Weiterentwicklung des Frameworks. Neben besserer Observability und nativer OpenAPI-Unterstützung sorgen neue Funktionen wie SSE, JSON-Patch mit System.Text.Json und zahlreiche Performance-Optimierungen für spürbare Produktivitäts- und Effizienzgewinne.

In Kombination mit .NET Aspire und den erweiterten Container-Features steht ein moderner Werkzeugkasten bereit, um skalierbare, verteilte Anwendungen noch einfacher zu entwickeln, zu betreiben und zu beobachten. Damit bestätigt Microsoft, dass .NET nicht nur für klassische Web-Apps, sondern auch für komplexe Cloud-native-Szenarien hervorragend gerüstet ist. ■

[1] Quellcode zum Artikel auf GitHub,
<https://github.com/marc-mueller/sse-demo>



Marc Müller

arbeitet als Principal Consultant für DevOps und .NET-/Azure-Lösungen bei 4tecture und ist Microsoft MVP für DevOps. Zu seinen Schwerpunkten zählen Ausbildung und Coaching von DevOps- und .NET-Projektteams.
publications@4tecture.ch

NEUES IN ASP.NET CORE 10

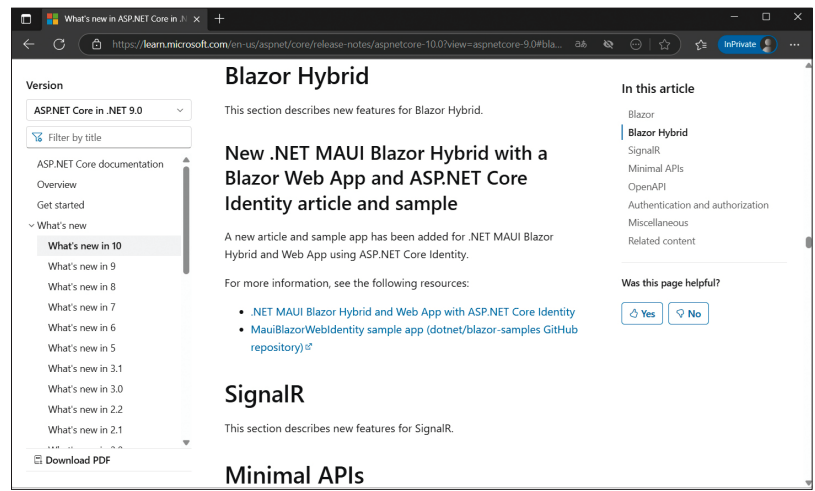
Web-Framework, frisch poliert

Was ändert sich bei der Webentwicklung mit .NET 10?

Herzlichen Glückwunsch zum 10. Geburtstag, liebes .NET. Oder .NET Core? Nun ja, selbst das mit der „10“ ist nicht ganz präzise. .NET Core 1.0 erschien im Juni 2016, das „neue“ .NET ist also zum Release von .NET 10 etwa neunzehn Jahre alt. Und genau zehn Versionen waren es auch nicht: .NET Core 4 wurde übersprungen, und bei den Versionen 1 bis 3 gab es jeweils relevante Unter-Releases. Dennoch ist die runde Zahl ein Grund zum Feiern, und es ist ein Release, auf den mit Hochdruck hingearbeitet wurde.

Was also gibt es an aufregenden Neuerungen für die Webentwicklung in dieser Version? Ein Blick auf das Dokument mit allen Änderungen und hinzugefügten Features [1] verrät: Gar nicht so viel. Das klingt ein wenig zynisch und ignoriert geflissentlich den Bereich Blazor, denn Microsofts SPA-Framework wird in einem direkt anschließenden Artikel in dieser dotnetpro-Ausgabe gesondert behandelt [2]. Doch streicht man das Thema Blazor aus dem Neuerungen-Dokument von Microsoft, so eliminiert man fast die Hälfte des Inhalts. Direkt danach folgen die Abschnitte zu Blazor Hybrid (ein neuer Artikel zu diesem Feature samt Beispielanwendung) und SignalR (gar keine Neuerungen, zumindest zum Entstehungszeitpunkt dieses Artikels eine Woche nach Veröffentlichung von Preview 7 von .NET 10, siehe auch Bild 1). Zum Thema „ASP.NET Core“ gibt es gar keinen eigenen Bereich, bei MVC und Razor Pages ändert sich also überhaupt nichts. Sehen wir es positiv: Die Technologien sind „feature-complete“, und die laut Stack-Overflow-Mitgründer Joel Spolsky häufig bei Microsoft vorhandenen „Architect Astronauts“ [3] streben auch kein aufwendiges Refactoring an. Das sind nach meinem Dafürhalten gute Nachrichten, die auch für die Zukunftssicherheit des Stacks sprechen. Bei Blazor tut sich in .NET 10 in der Tat mehr, die Technologie ist ja auch deutlich jünger.

Für diesen Artikel ist die Situation allerdings eine Herausforderung, denn die neuen Features sind teilweise sehr auf bestimmte Szenarien oder Randfälle hin optimiert – statt umfassender neuer Möglichkeiten für alle gibt es viele kleinere, aber eben auch speziellere Verbesserungen. Daher wurde für diesen Beitrag eine subjektive Auswahl für jeden Einzelbestandteil von ASP.NET Core 10 getroffen: Dinge, die für viele



Wo ist Waldo? In Teilen nur homöopathische Neuerungen (Bild 1)

Projekte spürbar sind, oder solche, die man sich eigentlich gar nicht gewünscht hat, auf die man jedoch nicht mehr verzichten möchte. Außerdem skizzieren wir – analog zum Beitrag vom vergangenen Jahr zu .NET 9 [4] – kurz den Weg eines Community-Beitrags, der ebenfalls den Weg in den neuen Release geschafft hat. Schnallen Sie sich an, die Fahrt geht los!

Neues für APIs

Für den Einsatz von ASP.NET Core Web API gibt es in der Tat eine ganze Reihe von Neuerungen. Viele davon hängen mit der Unterstützung von OpenAPI zusammen, die insbesondere in ASP.NET Core 9 stark erweitert wurde. Zur Erinnerung: OpenAPI ist ein Standard zur Beschreibung von APIs, die sich am REST-Prinzip orientieren. Das Ganze wird von der OpenAPI Initiative [5] verwaltet. Die zum Redaktionsschluss aktuellste Version der OpenAPI-Spezifikation war 3.1.1 [6] aus dem Oktober 2024, also ganz kurz vor dem Release von .NET 9 – offensichtlich zu kurz. Der Vorgänger, OpenAPI 3.1.0, ist schon älter und wurde Mitte Februar 2021 veröffentlicht. Trotzdem verwendete .NET bisher „nur“ OpenAPI 3. Das ändert sich mit .NET 10. Jetzt steht auch Version 3.1 zur Verfügung. Der Pull Request für diese Erweiterung wurde bereits einen Monat nach dem Release von .NET 9 erstellt [7] und einen Monat später gemergt. Sieht nach einem typischen Fall von „nicht rechtzeitig fertig geworden“ aus, aber fairerweise müssen wir darauf hinweisen, dass bei den jährlichen .NET-

Releases nach Kalender, nicht nach Feature-Fertigstellung, so etwas nur schwer vermieden werden kann.

OpenAPI 3.1 ist die neue Standardversion, aber dieses Verhalten lässt sich anpassen. Ausgangspunkt ist eine Web-API-Anwendung mit aktiviertem OpenAPI-Support. Das geht am einfachsten bei der Projektanlage in Visual Studio und der dazugehörigen Checkbox bei dem Web-API-Template (Bild 2, auch wenn der Hinweis auf Swagger irreführend ist, denn dafür bedarf es noch eines weiteren NuGet-Pakets); beim Einsatz des .NET CLI muss auf den Schalter `--no-openapi` verzichtet werden. Apropos: Dass die OpenAPI-Generierung standardmäßig aktiv ist, gilt in .NET 10 sogar für das AOT-Template für Web API!

Der generierte Code in der Datei *Program.cs* enthält jetzt einen Aufruf der Methode *AddOpenApi()*, der nach folgendem Muster konfiguriert werden kann:

```
builder.Services.AddOpenApi(options =>
{
    options.OpenApiVersion =
        Microsoft.OpenApi.OpenApiSpecVersion.OpenApi3_0;
});
```

Es stehen die Versionen 2.0, 3.0 und 3.1 zur Verfügung. Wir wollen aber 3.1 einsetzen, um Unterschiede zu den Vorgängern nachvollziehen zu können.

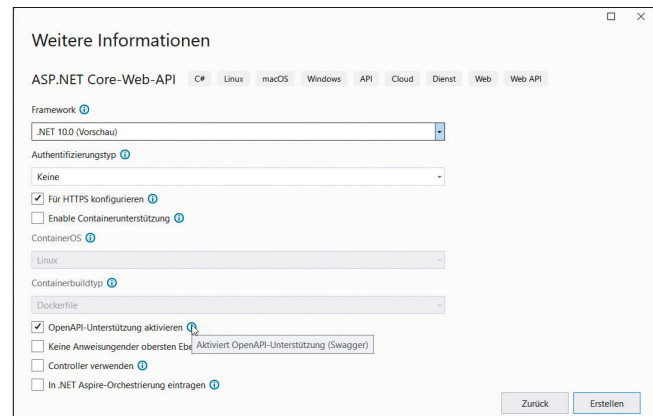
Der wohl akuteste Änderungsbedarf steht an, wenn Schema-Transformationen eingesetzt werden – es handelt sich dabei um Code, der in die Generierung eines OpenAPI-Dokuments eingreift. Wir haben das in [4] bereits gemacht, die neuralgische Codestelle sah damals so aus:

```
options.AddSchemaTransformer((schema, context,
    cancellationToken) =>
{
    schema.Example = new OpenApiObject()
    {
        ["id"] = new OpenApiInteger(1),
        ["title"] = new OpenApiString(
            "Bezeichnung des Todos"),
        ["dueBy"] = new OpenApiDate(DateTime.UtcNow),
        ["isComplete"] = new OpenApiBoolean(false)
    };

    return Task.CompletedTask;
});
```

Und das ist der neue Code:

```
options.AddSchemaTransformer((schema, context,
    cancellationToken) =>
{
    if (context.JsonTypeInfo.Type == typeof(Todo))
    {
        schema.Example = new JsonObject()
        {
            ["id"] = 1,
```



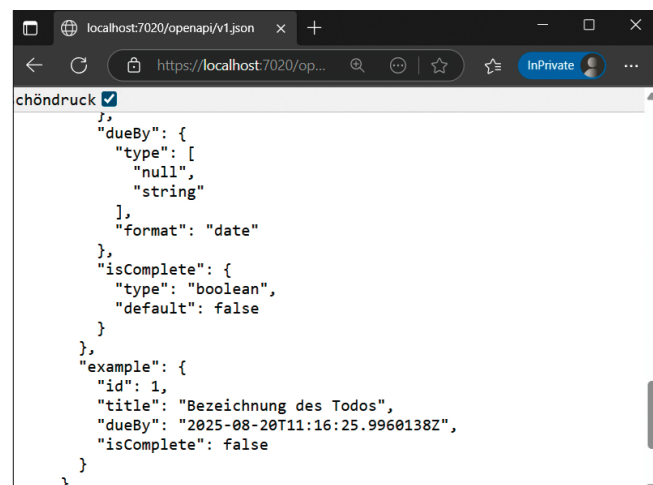
Auf Knopfdruck: Das Web-API-Template ist bereits für OpenAPI vorbereitet (Bild 2)

```
["title"] = "Bezeichnung des Todos",
["dueBy"] = DateTime.UtcNow,
["isComplete"] = false
};
};

return Task.CompletedTask;
});
```

Zum einen kann also wieder überprüft werden, ob überhaupt ein Objekt vom Typ *Todo* an die Schema-Transformation übergeben wird – zum Entstehungszeitpunkt des Artikels zu .NET 9 war diese Möglichkeit nicht gegeben, wurde dann aber später nachgereicht. Und, viel wichtiger: *OpenApiObject()* kann durch das generischere *JsonObject()* ersetzt werden, die übergebenen Werte können dann ohne Umwandlung in eine spezielle *OpenApiXXX*-Klasse zum Einsatz kommen.

Über den URL `https://<Servername>/openapi/v1.json` erhalten wir nun Zugriff auf das generierte OpenAPI-Dokument. Trotz des „v1“ im Namen wird die Versionsnummer als 3.1.1 ausgegeben. Die in der Transformation angegebenen Werte für „Example“ erscheinen wie gewünscht (Bild 3). ▶



Optimus Prime: Das Ergebnis der Transformation im Browser (Bild 3)

Ein weiteres praktisches Feature besteht darin, dass XML-Dokumentationskommentare mit in die OpenAPI-Beschreibung übernommen werden können. Hier ein einfaches Beispiel. Bisher erfolgt die Implementierung der Endpunkte für das API-Beispiel inline, etwa so:

```
todosApi.MapGet("/", () => sampleTodos);
```

An dessen Stelle müssen wir die Implementierung (also im Wesentlichen: *return sampleTodos*) in eine Methode auslagern. Am einfachsten ist es, wir erzeugen diese direkt in der *Program.cs*. Das geht so (Darstellung etwas gekürzt):

```
public partial class Program
{
    private static Todo[] sampleTodos = new Todo[] {
        // ...
    };

    /// <summary>
    /// Alle Todos auf einmal
    /// </summary>
    /// <returns>Eine Liste von Todos</returns>
    public static Todo[] AllTodos()
    {
        return sampleTodos;
    }
}
```

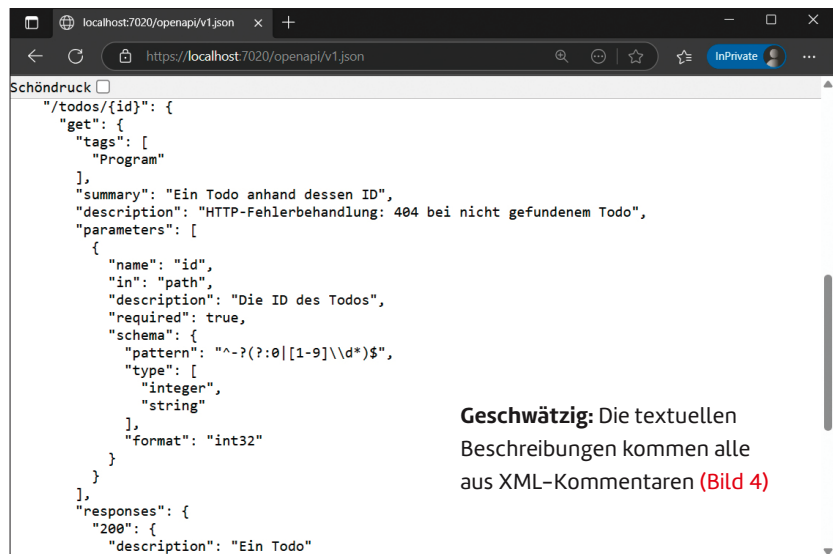
Beachten Sie, wie diese Methode XML-Dokumentationskommentare erhalten hat. Dasselbe können wir auch für den zweiten Endpunkt umsetzen, der ein To-do anhand einer ID zurückliefert.

Jetzt fehlt nur noch eine Einstellung: In der Projektdaten (*.csproj*) ist das folgende XML-Element notwendig:

```
<GenerateDocumentationFile>true
</GenerateDocumentationFile>
```

Jetzt erscheinen in dem OpenAPI-Dokument alle Angaben aus den XML-Kommentaren (sowie eine Reihe von Warnungen, dass wir die verwendeten Typen ja auch ordentlich beschreiben könnten), wie in **Bild 4** gezeigt.

Von OpenAPI abgesehen gibt es in .NET 10 auch noch einige Neuerungen für Minimal APIs, um die Lücke zu Controller-basierten APIs weiter zu schließen. Besonders interessant: Validierung über Data Annotations ist jetzt endlich möglich. Ein Beispiel: Die folgende Klas-



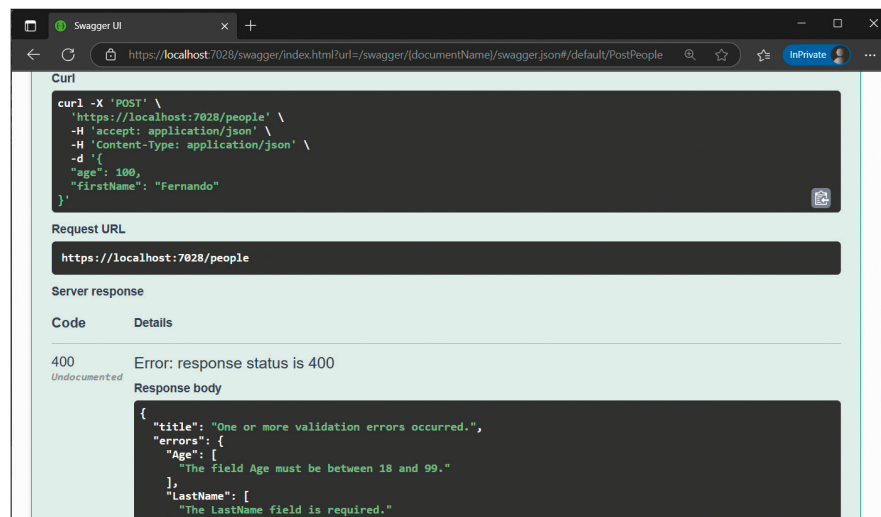
se (streng genommen ist es ein Record, aber auch dafür funktioniert das neue Feature) setzt auf Data Annotations, um bestimmte Beschränkungen für die Eigenschaftswerte festzulegen, etwa eine Altersdiskriminierung (ab 100 Jahren wird man vom System nicht mehr akzeptiert) sowie diverse Pflichtfelder:

```
using System.ComponentModel.DataAnnotations;

internal record Person([Range(18, 99)] int Age, [Required]
    string FirstName, [Required] string LastName);
```

Ein entsprechender Endpunkt nimmt eine (serialisierte) Person entgegen und verarbeitet diese. Normalerweise käme eine Speicherung in der Datenbank in Betracht, aber da wir auf die Validierung schauen, ist dieser Schritt entbehrlich und wir können mit einer reduzierten Implementierung arbeiten:

```
app.MapPost("/people", (Person person) =>
{
```



Shaming mit .NET 10: Ohne Nachname und zu alt (**Bild 5**)

```
return TypedResults.Ok(person);
});
```

Damit die Validierung auch durchgeführt wird, muss noch eine zusätzliche Middleware aktiviert werden:

```
builder.Services.AddValidation();
```

Bild 5 zeigt das Ergebnis: Die Eingabe verletzt zwei der Bedingungen durch die Data Annotations, und das Web-API gibt dazu detaillierte Fehlermeldungen in der HTTP-Antwort zurück.

Abschließend soll noch eine wichtige Änderung für Controller-basierte Web-APIs kurz angerissen werden, also APIs, die auf Controller-Klassen inklusive `[ApiController]`-Attribut setzen und Cookie-Authentifizierung verwenden. Bei unautorisierten Zugriffen zu API-Endpunkten sorgt das zugrunde liegende MVC-Framework dafür, dass ein HTTP-Redirect auf die Login-Seite vorgenommen wird – so funktioniert das eben bei traditionellen Webanwendungen. APIs jedoch haben ein anderes Mittel, auf fehlende Autorisierung hinzuweisen: die HTTP-Statuscodes 401 (*Unauthorized*) und 403 (*Forbidden*). Das ist jetzt anders – beim Einsatz von `[ApiController]` gibt es keine Weiterleitung mehr, sondern ordentliche HTTP-Fehlercodes. Dies gilt übrigens auch für Minimal APIs, sofern sie mit JSON arbeiten (in der Payload der HTTP-Anfrage und/oder in der HTTP-Antwort). Ursprünglich für .NET 11 geplant, hat es dieses Feature noch kurz vor Schluss in .NET 10 geschafft.

Auf und mit localhost entwickeln

Bereits im Jahr 1999 hat das World Wide Web Consortium ein paar Top-Level-Domains (TLDs) „reserviert“, sprich für anderweitige Nutzung gesperrt (der zugehörige RFC hat die Nummer 2606 und definiert auch die Domain `example.com/.net/.org` [8]):

- `.example`
- `.invalid`
- `.localhost`
- `.test`

Diese Spezial-TLDs tauchten knapp 14 Jahre später im RFC 6761 [9] wieder auf, der das genaue erwartete Verhalten bei deren Einsatz beschrieb. Kurzum: Alles, was auf `.localhost` endet, sollte auf das lokale Gerät (genauer: das Loopback-Device) aufgelöst werden. Das machen beispielsweise moderne Browser automatisch so.

Auch ASP.NET Core reiht sich in die Unterstützer dieser RFCs ein und spendiert den .NET-Templates `web` und `blazor` eine neue Option: Der Schalter `--localhost-tld` sorgt dafür, dass die Anwendung (bei Verwendung von Kestrel) nicht nur unter `localhost`, sondern auch unter `<Projektname>.dev.localhost` erreichbar ist. **Bild 6** zeigt sowohl die Erzeugung des Projekts als auch den Domain-Namen beim Starten der Webanwendung. In Visual Studio gab es hier zum Redaktionsschluss noch keine Option, aber das wird möglicherweise nachgeholt. Da die Verwendung von `.localhost` in einigen anderen Web-Frameworks Usus ist, freue ich mich persönlich sehr über diese kleine, aber feine Ergänzung.

Community rocks!

Zum Abschluss noch ein kurzer Blick auf ein Feature, das von einem Community-Mitglied implementiert wurde, und den Weg dorthin. Den Anstoß gab der damalige JetBrains-Mitarbeiter Khalid Abuhakmeh, der sich im Sommer 2020 in einem GitHub-Issue [10] einen einfachen Weg gewünscht hat, um festzustellen, ob ein URL als lokal angesehen werden kann oder nicht. Dies ist insbesondere wichtig, um Phishing durch Weiterleitung an eine Third-Party-Site zu verhindern. Zwar gab es diese Funktionalität bereits in der Klasse `UrlHelperBase`, die aber in einem Minimal API mühsam instanziiert werden musste (Haupteinsatzzweck waren Controller-basierte APIs).

Passiert ist zunächst nichts, außer dass das Ticket ins Backlog gelegt wurde. Ziemlich genau vier Jahre später jedoch hatte Martin Costello von Grafana einen ähnlichen Wunsch und legte ebenfalls ein Ticket an [11]. Allerdings ging er geschickter vor und lieferte gleich einen API-Vorschlag mit (statische Methode innerhalb von `RedirectHttpResult`). Dieser wurde einen Monat später angenommen, und Costello ►

Mein Localhost,
dein Localhost:
ASP.NET Core 10
unterstützt per
Schalter eine
spezielle TLD
(Bild 6)

```
C:\Windows\System32\cmd.exe x + v

> dotnet new web --localhost-tld --name dnp.AspNetCore10Demos.LocalhostTLD
The template "ASP.NET Core Empty" was created successfully.

Processing post-creation actions...
Restoring C:\Users\cwenz\source\repos\dnp.AspNetCore10Demos\dnp.AspNetCore10Demos.LocalhostTLD\dnp.AspNetCore10Demos.LocalhostTLD.csproj:
Restore succeeded.

> dotnet sln add dnp.AspNetCore10Demos.LocalhostTLD
Project `dnp.AspNetCore10Demos.LocalhostTLD\dnp.AspNetCore10Demos.LocalhostTLD.csproj` added to the solution.

> cd dnp.AspNetCore10Demos.LocalhostTLD

> dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://dnp.aspnetcore10demos_localhosttld.dev.localhost:5141
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5141/
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\cwenz\source\repos\dnp.AspNetCore10Demos\dnp.AspNetCore10Demos.LocalhostTLD
```

● Neues in .NET Aspire

Zwar wird .NET Aspire auch als Teil von ASP.NET Core 10 angesehen, aber die Release-Zyklen sind unterschiedlich. Beispielweise wurde .NET Aspire 9.4 zeitnah mit .NET 10 Preview 6 veröffentlicht (und als damit kompatibel ausgewiesen). Die Folgeversion wird auch nicht 10 heißen, sondern 9.5.

Wie immer gilt: Um auf die neuesten Aspire-Features setzen zu können, müssen alle Projekte auf die neue SDK-Version verweisen und mit `dotnet new install Aspire.ProjectTemplates` die neuesten Projektvorlagen eingespielt werden.

Die Liste der Neuerungen fällt lang und detailliert aus und ist mit zahlreichen Codebeispielen versehen [12]. Hauptneuerung von Version 9.4 ist die

Verfügbarkeit eines eigenen Kommandozeileninterfaces, des Aspire CLI. Dieses kann mit `dotnet tool install -g Aspire.Cli` installiert werden und bietet dann diverse Möglichkeiten analog zum .NET CLI an, etwa die Anlage (`aspire new`) oder Ausführung (`aspire run`, siehe Bild 7) von Aspire-Projekten. Wie üblich wurde das Dashboard um weitere Features ergänzt, beispielsweise einen Hinweis, wenn es eine neuere Aspire-Version gibt. Hinzugekommene Integrationen, etwa von KI-Modellen auf GitHub („GitHub Models“), vereinfachen den Einbau von zusätzlichen Systemen in die Aspire-Orchestrierung.

```

> dotnet tool install -g Aspire.Cli
You can invoke the tool using the following command: aspire
Tool 'aspire.cli' (version '9.4.1') was successfully installed.

> aspire new --help
Description:
  Create a new Aspire project.

Usage:
  aspire new [command] [options]

Options:
  -n, --name           The name of the project to create.
  -o, --output          The output path for the project.
  -s, --source          The NuGet source to use for the project templates.
  -v, --version         The version of the project templates to use.
  -?, -h, --help       Show help and usage information
  -d, --debug           Enable debug logging to the console.
  --wait-for-debugger   Wait for a debugger to attach before executing the command. [default: False]

Commands:
  aspire-starter        Starter template
  aspire                AppHost and service defaults
  aspire-apphost        AppHost
  aspire-servicedefaults Service defaults
  aspire-test           Integration tests
  
```

1980er-Revival: Aspire per CLI steuern (Bild 7)

legte sofort mit einem Pull Request nach. Ende der Geschichte: Costellos Code wurde akzeptiert und in .NET integriert, allerdings erst in .NET 10, für .NET 9 war es zu spät oder nicht relevant genug.

Und so sieht ein typisches Einsatzszenario der Methode jetzt aus – erst dann per `LocalRedirect()` weiterleiten, wenn es ein lokaler URL ist:

```

if (RedirectHttpResult
    .IsLocalUrl(url))
{
    return Results
        .LocalRedirect(url);
}
  
```

Die Moral von der Geschichte: Mache dem .NET-Team die Arbeit möglichst einfach, sonst klappt es nicht. Aber es gilt eben auch: Die Community kann bei .NET und insbesondere auch bei ASP.NET Core definitiv etwas bewirken! ■

[1] Neues in ASP.NET Core 10, <https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-10.0>

[2] Holger Schwichtenberg, Weiterhin mit ordentlich Tempo, *dotnetpro* 10-11/2025, Seite 19 ff.

[3] Joel Spolsky über „Architect Astronauts“, www.joelonsoftware.com/2008/05/01/architecture-astronauts-take-over/

[4] Christian Wenz, *Ausbesserungsarbeiten im Web*, *dotnetpro* 11/2024, Seite 10 ff.

[5] OpenAPI Initiative, www.openapis.org

[6] OpenAPI-Spezifikation 3.1.1, <https://spec.openapis.org/oas/v3.1.1.html>

[7] Pull Request für OpenAPI-3.1-Support, <https://github.com/dotnet/aspnetcore/pull/59480>

[8] Durch das W3C reservierte Top-Level-Domains, www.rfc-editor.org/rfc/rfc2606

[9] Spezielle Domains, inklusive „.localhost“, www.rfc-editor.org/rfc/rfc6761

[10] Wunsch nach `IsLocalUrl`, <https://github.com/dotnet/aspnetcore/issues/24486>

[11] Zweiter Wunsch nach `IsLocalUrl`, <https://github.com/dotnet/aspnetcore/issues/56770>

[12] Neues in .NET Aspire 9.4, <https://learn.microsoft.com/en-us/dotnet/aspire/whats-new/dotnet-aspire-9.4>



Christian Wenz

ist Softwarearchitekt, Berater und Spezialist für die Themen Performance und Web Security. Seit 2004 ist er Microsoft MVP. Seine neuesten Bücher sind „ASP.NET Core Security“ (erschienen bei Manning) und „C# und .NET 8“ (Hanser). www.christianwenz.de

NEUERUNGEN IN BLAZOR 10.0, TEIL 1

Weiterhin mit ordentlich Tempo

Microsoft gibt bei Blazor weiterhin Gas und bringt spannende Neuerungen insbesondere in den Bereichen Zustandsverwaltung, Navigation, Validierung und Sicherheit.

Blazor 10.0 ist Teil von ASP.NET Core 10.0, das wiederum ein Teil von .NET 10.0 ist. Zum Redaktionsschluss für diesen Artikel am 5. September 2025 sind von .NET 10.0 bereits sieben Preview-Versionen erschienen.

Dieser Beitrag stellt die darin enthaltenen Neuerungen für Blazor gegenüber Blazor 9.0 vor. Die fertige Version soll im November 2025 mit Long-Term-Support von drei Jahren erscheinen. Vorher wird es noch zwei Release-Candidate-Versionen im September und Oktober geben.

Bezugsquelle und Voraussetzungen

Blazor 10.0 installiert man als Teil des .NET 10.0 SDK [1]. Zum Entwickeln braucht man Visual Studio 2022 in der Version 17.14 oder höher. Alternativ kann man mit Visual Studio Code und dem aktuellen C# Dev Kit [2] arbeiten.

Persistent Component State per Annotation

Blazor Server und Blazor WebAssembly erlauben schon vor Blazor 10.0 die Übergabe von Daten zwischen dem Prerendering und dem Hauptrendering einer Razor Component via JSON-Anhang in den zum Client gesendeten HTML-Dokumenten. Dieser Persistent Component State war für die Entwicklerinnen und Entwickler in der Realisierung bisher allerdings recht aufwendig, denn man musste zunächst ein Objekt per Dependency Injection bekommen:

```
@inject PersistentComponentState ApplicationState
```

Anschließend musste man den Zustand als Objekt dort ablegen:

```
private PersistingComponentStateSubscription?
    persistingSubscription;

persistingSubscription =
    ApplicationState.RegisterOnPersisting(() =>
```

```
{
    ApplicationState.PersistAsJson("name", daten);
    return Task.CompletedTask;
});
```

Und jeweils vor der Nutzung galt es den Zustand dann wieder explizit aus dem Persistent Component State herauszuholen:

```
if (ApplicationState.TryTakeFromJson<Typ>("name", out
    var daten))
{
    Daten = daten;
}
```

Dies hat Microsoft mit .NET 10.0 Preview 3 mit dem deklarativen Persistent Component State radikal vereinfacht. Entwicklerinnen und Entwickler müssen jetzt nur noch eine oder mehrere Properties in der Razor Component mit der Annotation `[PersistentState]` versehen und sich sonst um gar nichts kümmern:

```
[PersistentState]
public Daten? daten { get; set; }
```

In Listing 1 sieht man auskommentiert die alte, codebasierte Variante im Vergleich zur neuen Variante mit der Annotation.

Aber Achtung, es gibt noch einen Bug Stand .NET 10.0 Preview 7: Es kommt zu einem Absturz von Blazor Server bei etwas mehr Daten, im konkreten Fall ab 59 Flügen, die in den Component State geladen werden (Bild 1). Das passiert bei der alten und der neuen Lösung!

Circuit-Zustände bei Blazor Server persistieren

Blazor Server besitzt für jeden per WebSocket angeschlossenen Browser einen sogenannten Circuit, der für alle aktiven ►

✖ ► [2025-06-18T07:31:33.899Z] Error: Connection disconnected with error 'Error: [blazor.web.6q4jeuced1.js:1](#) Server returned an error on close: Connection closed with an error.'

Der Blazor-Server-Circuit stürzt sang- und klanglos ab, wenn man eine etwas größere Datenmenge in den Component State legt (Bild 1)

● Listing 1: Der Einsatz von [PersistentState] macht den Code deutlich schlanker

```

@page "/State"
@using BO.WWWings

@* ----- auskommentiert die alte
Variante ohne [PersistentState] (vor .NET 10.0)*@
@* @inject PersistentComponentState ApplicationState *@

<PageTitle>Counter</PageTitle>

<h1>Flight-Counter</h1>

<p role="status" title="@pageState.LastChange">Current
count: @pageState.CurrentCount</p>

<button class="btn btn-primary"
    @onclick="IncrementCount">+</button>
<button class="btn btn-primary"
    @onclick="DecrementCount">-</button>

@if (this.pageState?.FlightSet != null)
{
    <hr noshade>
    <ol>
        @foreach (Flight f in this.pageState.FlightSet.Take(
            this.pageState.CurrentCount))
        {
            <li>Flug #@f.FlightNo (
                @f.FlightDate.ToShortDateString())
                @f.Departure -> @f.Destination </li>
        }
    </ol>
}

@code {
    [PersistentState]
    public PageState pageState { get; set; } = null;

    private void IncrementCount()
    {
        if (pageState.CurrentCount < maxCount)
            pageState.CurrentCount++;
    }

    private void DecrementCount()
    {
        if (pageState.CurrentCount > 0)
            pageState.CurrentCount--;
    }

    // private PersistingComponentStateSubscription?
    // persistingSubscription;

    int maxCount = 59; // > 58 Führt zum Absturz von
                        // Blazor Server (Stand Preview 7)

```

The screenshot shows a web browser displaying a Blazor Server application. The application's title is "Flug-Daten (Component State Demo)". The current time is 20:58:42. The left sidebar contains a navigation menu with items: Home, Metriken, Trace, State (highlighted), Interop, Validation, QuickGrid, Navigation, Not Found, Counter, and Weather. The main content area displays a list of 20 flights, each with a number, time, and route. The right sidebar shows the console output, which includes messages from the Blazor Circuit and the application's state changes.

Persistierung eines Blazor-Server-Circuits (Bild 2)

```

protected override void OnInitialized()
{
    // persistingSubscription =
    //     ApplicationState.RegisterOnPersisting(() =>
    // {
    //     ApplicationState.PersistAsJson(nameof(
    //         PageState), pageState);
    //     return Task.CompletedTask;
    // });

    if (pageState == null)
    {
        DA.WWWings.WwwingsV1EnContext ctx = new();
        var data = ctx.Flights.Take(maxCount).ToList();
        pageState = new()
        {
            FlightSet = data,

            CurrentCount = data.Count,
            Created = DateTime.Now
        };
    }

    // if (ApplicationState.TryTakeFromJson<PageState>(
    //     nameof(PageState), out PageState daten))
    // {
        //     pageState = daten;
        // }

    /// <summary>
    /// Beliebige eigene Klasse mit dem Zustand der
    /// Seite.
    /// </summary>
    public class PageState
    {
        public DateTime Created { get; set; }
        public DateTime LastChange { get; set; }
        public int _CurrentCount;
        public int CurrentCount
        {
            get
            {
                return _CurrentCount;
            }
            set
            {
                _CurrentCount = value;
                LastChange = DateTime.Now;
            }
        }
        public List<Flight> FlightSet { get; set; }
    }

```

Razor Components sowohl das Document Object Model (DOM) als auch die Werte aller Variablen auf dem Webserver speichert. Bisher ging dieser Circuit komplett verloren, wenn mehrere Sekunden lang die Netzwerkverbindung abreißt oder der Browser die Anwendung schlafen legt (zum Beispiel bei mobilen Geräten), denn der Webserver erwartet zur Beibehaltung des Circuits eine regelmäßige kurze Keep-alive-Nachricht über die WebSocket-Verbindung.

Seit .NET 10.0 Preview 6 können Entwicklerinnen und Entwickler mit Blazor Server nun erstmalig tun, was sie mit dem Konkurrenzprodukt Wisej.NET schon länger konnten: Zustände über solche Abbrüche hinweg persistieren. Während dies jedoch in Wisej.NET automatisch erfolgt, müssen Entwicklerinnen und Entwickler bei Blazor Server an dieser Stelle selbst etwas implementieren.

Die Persistenz von Circuit-Zuständen bei Blazor Server erfolgt im Standard nur im RAM des Webserver. Sie kann jedoch optional auch in getrennten Cache-Prozessen (zum Beispiel Redis) oder einem Datenbankmanagementsystem (zum Beispiel Microsoft SQL Server) erfolgen.

Blazor Server persistiert dabei nicht den kompletten Circuit und auch nicht alle Variablen einer Razor Component, sondern nur diejenigen Werte, die explizit in den Persistent Component State gelegt werden – wahlweise codebasiert oder mit der in

.NET 10.0 neu eingeführten Annotation *[PersistentState]*. Dafür zeigt **Listing 2** ein aussagekräftiges Beispiel: Die Razor Component *State.razor* persistiert via *[PersistentState]* eine Instanz der selbst geschriebenen Klasse *PageState*. Die Klasse *PageState* umfasst eine aus einem Datenbankmanagementsystem via Entity Framework Core in *OnInitialized()* geladene Menge von Flugdaten sowie die Anzahl der Flüge, die in der Benutzeroberfläche angezeigt werden sollen. Zudem gibt es auf der Benutzeroberfläche eine laufend aktualisierte Zeitanzeige. Zu Demonstrationszwecken schreibt der Programmcode jeweils beim Aktualisieren der Zeit die aktuelle Zeit zusätzlich auch in die Browserkonsole.

In der zentralen Razor Component *App.razor* wird zudem der JavaScript-Code aus **Listing 3** hinterlegt, der dafür sorgt, dass Circuits und die zugehörige WebSocket-Verbindung beendet werden, wann immer die Webanwendung nicht mehr sichtbar ist, und dass eine Wiederherstellung des Circuits und der WebSocket-Verbindung erfolgen, sobald die Webanwendung wieder sichtbar wird. Auf diese Weise kann man bei Blazor Server einige Ressourcen (RAM und Rechenzeit) auf dem Webserver einsparen und damit die Skalierbarkeit der Webanwendung verbessern.

Bild 2 zeigt das Verhalten der Razor Component, die für kurze Zeit unsichtbar gemacht wurde (zum Beispiel durch das ►

● Listing 2: State.razor

```

@page "/State"
@using B0.WWWings
@using ITVisions
@inject ITVisions.Blazor.BlazorUtil util

<PageTitle>Counter</PageTitle>

<h1>Flug-Daten (Component State Demo)</h1>
<p><b>Aktuelle Uhrzeit:</b>
    @currentTime.ToString("HH:mm:ss")</p>

<hr />
<button class="btn btn-primary"
    onclick="Blazor.pauseCircuit()">Pause</button>
<button class="btn btn-primary"
    onclick="Blazor.resumeCircuit()">Resume</button>
<hr />

<p role="status" title="@pageState.LastChange">
    Anzahl Flüge: @pageState.CurrentCount

<button class="btn btn-primary btn-sm"
    @onclick="IncrementCount">+</button>
<button class="btn btn-primary btn-sm"
    @onclick="DecrementCount">-</button>
</p>

@if (this.pageState?.FlightSet != null)
{
    <hr noshade>

```

```

<ol>
    @foreach (Flight f in this.pageState.FlightSet.Take(
        this.pageState.CurrentCount))
    {
        <li>Flug #@f.FlightNo <span class="badge
            bg-primary"> @f.FlightDate.ToShortTimeString()
        </span> @f.Departure <span style="font-size:
            1.2em;">&#10230;</span> @f.Destination </li>
    }
</ol>
}

@code {
    [PersistentState] // NEU ab .NET 10.0
    public PageState pageState { get; set; } = null;

    private DateTime currentTime = DateTime.Now;
    private System.Timers.Timer? timer;

    private void IncrementCount()
    {
        if (pageState.CurrentCount < maxCount)
            pageState.CurrentCount++;
    }

    private void DecrementCount()
    {
        if (pageState.CurrentCount > 0)
            pageState.CurrentCount--;
    }
}

```

Öffnen eines anderen Browser-Tabs oder durch Minimieren des Browsers). Zunächst gibt es im Sekundentakt eine Ausgabe in die Browserkonsole jeweils beim Aktualisieren der Uhrzeit auf dem Bildschirm. Sobald die Razor Component unsichtbar wird, wird der Circuit-Zustand durch den JavaScript-Aufruf *blazor.pauseCircuit()* persistiert und die WebSocket-Verbindung beendet.

23 Sekunden später wird die Webseite wieder sichtbar. Durch den Aufruf *blazor.resumeCircuit()* in JavaScript wird ein neuer Circuit erstellt, der persistente Zustand des Circuits geladen und die WebSocket-Verbindung wieder aufgebaut. Die Razor Component wird neu gerendert und macht aus Benutzersicht dort weiter, wo sie aufgehört hat: Sie besitzt noch die gleichen Flugdaten wie zuvor (ohne diese neu aus dem Daten-

● Listing 3: Pausieren und Wiederaufnahme von Circuits und zugehöriger WebSocket-Verbindung

```

<script>
    // Für Persistenz von Circuits
    window.addEventListener('visibilitychange', () => {
        if (document.visibilityState === 'hidden') {
            Blazor.pauseCircuit();
            console.warn(new Date() + ': Blazor Circuit
                paused due to visibility change');
        } else if (document.visibilityState === 'visible')

```

```

    {
        Blazor.resumeCircuit();
        console.warn(new Date() + ': Blazor Circuit
            resumed due to visibility change');
    }
});
</script>

```



```

int maxCount = 50; // > 58 Führt zum Absturz von
                // Blazor Server (Stand Preview 7)
protected override void OnInitialized()
{
    // Uhrzeit-Timer starten
    timer = new System.Timers.Timer(1000);
    timer.Elapsed += (s, e) => {
        currentTime = DateTime.Now;
        InvokeAsync(StateHasChanged);
        util.Log(currentTime);
    };
    timer.Start();

    if (pageState == null)
    {
        // Lade Daten
        var renderMode = this.RendererInfo.Name;
        var logMessage = "Lade bis zu " + maxCount +
            " Flugdatensätze im Blazor-Rendermode=" +
            renderMode;
        DA.WWWings.WwwingsV1EnContext ctx = new();
        var data = ctx.Flights.Take(maxCount).ToList();
        // LINQ --> SQL
        pageState = new()
        {
            FlightSet = data,
            CurrentCount = 20,
            Created = DateTime.Now
        };
    }
}

}

public void Dispose()
{
    timer?.Dispose();
}

/// <summary>
/// Beliebige eigene Klasse mit dem zu persistierenden
/// Zustand der Seite.
/// </summary>
public class PageState
{
    public DateTime Created { get; set; }
    public DateTime LastChange { get; set; }
    public int _CurrentCount;
    public int CurrentCount
    {
        get
        {
            return _CurrentCount;
        }
        set
        {
            _CurrentCount = value;
            LastChange = DateTime.Now;
        }
    }
    public List<Flight> FlightSet { get; set; }
}

```

bankmanagementsystem laden zu müssen) und zeigt die gleiche Anzahl von Flügen. Die Uhrzeitausgabe in der Webseite und der Browserkonsole läuft weiter. Dass die Uhrzeitausgabe in der Browserkonsole unterbrochen ist, beweist, dass die Webanwendung zwischenzeitlich tatsächlich inaktiv war.

Im Startcode der Blazor-Server-Anwendung können Entwicklerinnen und Entwickler die Persistierung via *Configure<CircuitOptions>()* konfigurieren. Listing 4 zeigt die Festlegung der maximalen Anzahl der persistierten Circuits-Zustände auf 500 (Standard ist hier 1000), die Dauer der Persistierung im RAM auf 10 Sekunden (der Standard ist hier zwei Stunden) und die Dauer der Persistierung im Second-Level-Cache auf eine Stunde (der Standard ist hier ebenfalls zwei Stunden) sowie die Festlegung eines Microsoft SQL Servers als Second-Level-Cache zusätzlich zum RAM-Cache (Bild 3). Einen Hybrid Cache muss man dabei auch hinzufügen, denn Microsoft verwendet die in .NET 9.0 eingeführte Hybrid-Cache-Bibliothek [3] als Abstraktion.

Der Second-Level-Cache soll eine serverübergreifende Nutzung der persistierten Circuit-Daten erlauben, wenn man dem Tooltip der neuen Eigenschaft *HybridPersistenceCache*

in der Klasse *CircuitOptions* glauben will: „Gets or sets the HybridCache instance to use for persisting circuit state across servers.“ Eine Dokumentation dazu gibt es aber bislang leider noch nicht.

Die hier beschriebene Persistenz funktioniert nur bei Blazor Server; bei Blazor WebAssembly oder Blazor Hybrid ist sie gar nicht notwendig, denn dort gibt es keine Circuits. Bei einem manuellen Browser-Refresh (Wisej.NET behält dabei den Zustand) oder manuellem Schließen des Browser-Fensters funktioniert die Persistenz nicht. Diese und weitere Einschränkungen dokumentiert Microsoft unter [4]. Ebenso findet man dort Hinweise, wie man Daten in per Dependency Injection injizierten Diensten (sofern diese „Scoped Services“ sind) persistieren kann.

Verbesserungen in der Klasse *NavigationManager*

Die in Blazor eingebauten Implementierungen der abstrakten Basisklasse *Microsoft.AspNetCore.Components.NavigationManager* haben in Blazor 10.0 Preview 4 zwei neue Mitglieder erhalten: die Methode *NotFound()* und das Ereignis *OnNotFound()*. ►

Listing 4: Circuit-Zustandspersistenz konfigurieren

```
builder.Services.Configure<CircuitOptions>(options =>
{
    options.PersistedCircuitInMemoryMaxRetained = 500;
    // The maximum number of circuits to retain.
    // The default is 1,000 circuits.
    options.PersistedCircuitInMemoryRetentionPeriod =
        TimeSpan.FromSeconds(1);
    // The maximum retention period as a TimeSpan.
    // The default is two hours.
    options.PersistedCircuitDistributedRetentionPeriod =
        TimeSpan.FromSeconds(60);
    // The maximum retention period for distributed
    // circuits. The default is two hours.
    options.DetailedErrors = true;
});

// ----- Hybrid Cache konfigurieren
var hyb = builder.Services.AddHybridCache(options =>
// optionale Einstellungen
{
    options.DefaultEntryOptions =
        new HybridCacheEntryOptions
        {
            Flags = HybridCacheEntryFlags.DisableCompression
            // nur als Beispiel für Einstellungen
        };
    // ----- Second-Level-Cache konfigurieren
    builder.Services.AddDistributedSqlServerCache(options =>
    {
        options.ConnectionString = "Data Source=" +
            DB_SERVERNAME + ";Initial Catalog=NET_Cache;" +
            "Integrated Security=True;Connect Timeout=30;" +
            "Encrypt=False;Trust Server Certificate=True;" +
            "Application Intent=ReadWrite;" +
            "Multi Subnet Failover=False";
        options.SchemaName = "dbo";
        options.TableName = "Cache2";
    });
});
```

Die neue Methode *NotFound()* signalisiert Blazor, dass eine angeforderte Ressource nicht vorhanden ist. Beim statischen Server-Side Rendering bekommt der Webbrowser dann eine HTTP-Antwort mit Statuscode 404. Beim interaktiven Rendering (Blazor Server, Blazor WebAssembly, Blazor Hybrid) wird gerendert, was im Router bei *<NotFound>* definiert ist.

OnNotFound() ist ein Ereignis, das von Blazor ausgelöst wird, wenn die Methode *NotFound()* aufgerufen wird.

NavigationManager.NavigateTo() funktioniert nun auch beim statischen Server-Side Rendering. Bisher lieferte dies immer den Laufzeitfehler *NavigationException*. Für Anwendungen, die diesen Fehler erwarten, hat Microsoft einen Schalter eingebaut, der das bisherige Verhalten zurückholt:

```
AppContext.SetSwitch("Microsoft.AspNetCore.Components
.Endpoints.NavigationManager
.EnableThrowNavigationException", isEnabled: true);
```

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' shows the database structure, including the 'dbo.Cache2' table. The table's columns are listed: 'Id' (PK, nvarchar(449), not null), 'Value' (nvarchar(max), not null), 'ExpiresAtTime' (datetimeoffset(7)), 'SlidingExpirationInSeconds' (bigint), and 'AbsoluteExpiration' (datetimeoffset). The main pane shows a SQL query: 'SELECT TOP (1000) [Id], [Value], [ExpiresAtTime], [SlidingExpirationInSeconds], [AbsoluteExpiration] FROM [NET_Cache].[dbo].[Cache2]'. Below the query, the 'Results' pane shows a table with 17 rows of data. The columns are 'Id', 'Value', 'ExpiresAtTime', 'SlidingExpirationInSeconds', and 'AbsoluteExpiration'. The data includes various IDs, values, and expiration times.

Persistierte Circuit-Zustände in einer Microsoft-SQL-Server-Tabelle, deren Aufbau durch den Distributed Cache Provider (SqlServer-Cache) vorgegeben ist (Bild 3)

Listing 5: <NotFound>-Tag im Blazor-Router

```
<Router AppAssembly="@typeof(App).Assembly">
  <Found Context="routeData">
    <RouteView RouteData="@routeData"
      DefaultLayout="@typeof(MainLayout)" />
    <FocusOnNavigate RouteData="@routeData"
      Selector="h1" />
  </Found>
  <NotFound>
    <PageTitle>Nicht gefunden</PageTitle>
    <LayoutView Layout="@typeof(MainLayout)">
      <p role="alert">Sorry, diese Seite gibt es
        nicht!</p>
    </LayoutView>
  </NotFound>
</Router>
```

Die Methode `NavigateTo()` in der Blazor-Klasse `NavigationManager` führt den Nutzer beziehungsweise die Nutzerin zudem nicht mehr zum Beginn der Seite zurück, falls der Entwickler beziehungsweise die Entwicklerin nur URL-Parameter an die Seite angehängt hat:

```
nav.NavigateTo(nav.GetUriWithQueryParameter(
  "count", currentCount));
```

Ein Rücksprung an den Seitenanfang findet allerdings weiterhin statt bei Angabe von `true` für `forceLoad`

```
nav.NavigateTo(nav.GetUriWithQueryParameter(
  "count", currentCount), forceLoad: true);
```

oder bei Änderungen eines Teils des relativen Pfades:

```
nav.NavigateTo("/counter/" + currentCount);
```

Eigene Seite bei 404

Blazor konnte bisher schon den Fall nicht gefundener Seiten behandeln. Tritt dieses Ereignis auf, so wird bisher dargestellt, was im Tag `<Router>` im Unterelement `<NotFound>` steht (Listing 5).

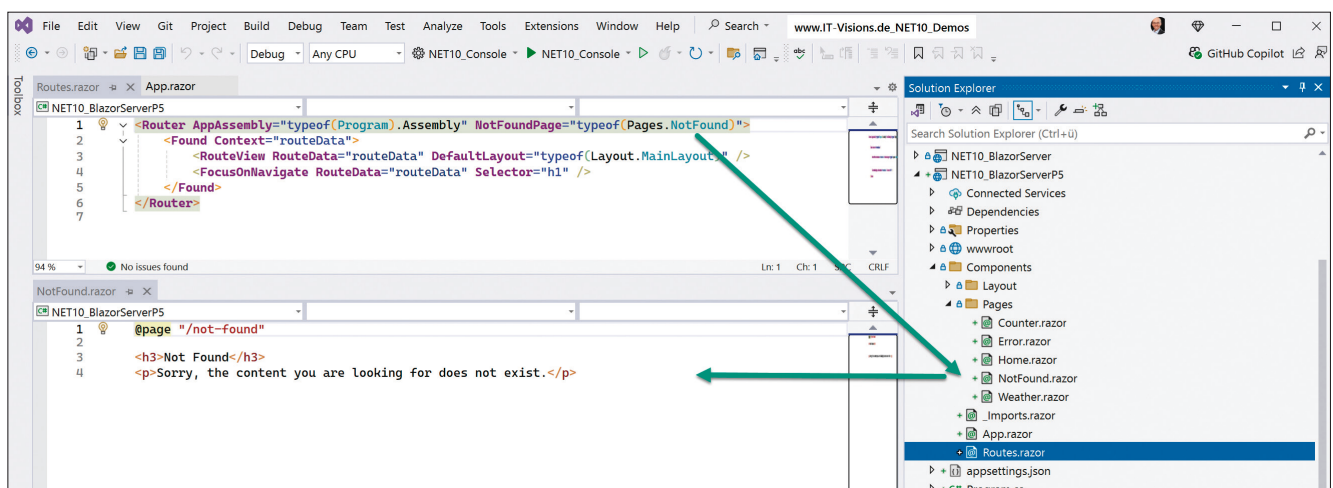
Seit .NET 10.0 Preview 5 bietet Microsoft als weitere Möglichkeit an, eine eigene Razor Component für den Fall nicht gefundener Seiten zu definieren. Den Namen dieser Razor Component erfassen Entwicklerinnen und Entwickler in der neuen Eigenschaft `NotFoundPage` direkt im `<Router>`-Tag (Listing 6). Falls es zusätzlich ein Unterelement `<NotFound>` gibt, wird dieses dann ignoriert. Microsoft reagiert mit dieser Erweiterung auf einen Benutzerwunsch aus dem Dezember 2022 unter [5], der Herausforderungen bei dem bisherigen Ansatz meldete.

In den Projektvorlagen *Blazor Web App* und *Blazor Web-Assembly Standalone* existiert nun eine eigene Razor Component `NotFound.razor`, die im Tag `<Router>` in `Routes.razor` referenziert wird (siehe Bild 4).

Laut der Dokumentation unter [6] wirkt die neue Fehlerseite zwar beim Aufruf von `NavigationManager.NotFound()` im Programmcode der Blazor-Anwendung, aber nicht, wenn der Benutzer einen falschen URL direkt ansteuert. Im Test ►

Listing 6: NotFoundPage-Eigenschaft im Blazor-Router

```
<Router AppAssembly="@typeof(Program).Assembly"
  NotFoundPage="typeof(Pages.NotFound)">
  <Found Context="routeData">
    <RouteView RouteData="@routeData" />
    <FocusOnNavigate RouteData="@routeData"
      Selector="h1" />
  </Found>
  <NotFound>This content will be ignored because we
    have NotFoundPage defined.</NotFound>
</Router>
```



NotFound.razor in der Projektvorlage „Blazor Web App“ ab .NET 10.0 Preview 5 (Bild 4)

● Listing 7: Beispiel für eine Razor Component NotFound.razor

```
@page "/not-found"
<div class="text-center mt-5">
    
    <h2 class="mb-3" style="color:#6c757d;">
        Seite nicht gefunden</h2>
    <p class="mb-4">Die angeforderte Adresse existiert
        leider nicht.<br />Bitte überprüfe den URL oder
        kehre zur <a href="/">Startseite</a> zurück.</p>
    <br />
    <a class="btn btn-primary" href="/">Zurück zur
        Startseite</a>
</div>
```

zeigte sich, dass dies leider korrekt ist. Für den Direktansprung falscher URLs müssen Entwicklerinnen und Entwickler wie bisher im Startcode der Blazor-Anwendung (in *Program.cs*) ein Ziel für den Statuscode 404 explizit definieren. Das Ziel kann dann aber die Seite sein, die im Blazor-Router festgelegt ist.

```
app.UseStatusCodePages(async statusCodeContext =>
{
    if (statusCodeContext.HttpContext.Response.StatusCode
        == 404)
    {
```

```
        statusCodeContext.HttpContext.Response.Redirect(
            "/not-found");
    }
});
```

Die NotFound-Seite muss dafür eine *@page*-Direktive mit dem relativen URL besitzen, den man bei *Redirect()* angibt (Listing 7).

Validierung für Objekthierarchien

Die in Blazor eingebauten Eingabesteuerelemente (*<InputText>*, *<InputDate>*, *<InputNumber>*, *<InputSelect>*, *<Input*

● Listing 8: Datenmodell Company mit Unterobjekt Contact

```
using System.ComponentModel.DataAnnotations;
using Microsoft.Extensions.Validation;

namespace NET10_BlazorServer.Model;

#pragma warning disable ASP0029 // Type is for
// evaluation purposes only and is subject to change
// or removal in future updates. Suppress this
// diagnostic to proceed.
[ValidatableType]
#pragma warning restore ASP0029 // Type is for
// evaluation purposes only and is subject to change
// or removal in future updates. Suppress this
// diagnostic to proceed.
public class Company
{
    public int CompanyID { get; set; }
    [Required(ErrorMessage =
        "Firmenname ist Pflichtfeld")]
    public string CompanyName { get; set; }
    [Required(ErrorMessage =
        "Gründungsdatum ist Pflichtfeld")]
    [Range(typeof(DateTime), "01/01/1900",
        "31/12/2024", ErrorMessage =
        "Datum muss zwischen 1900 und 2024 liegen.")]
    public DateTime Foundation { get; set; }
    public Contact Contact { get; set; } =
        new Contact();
}

public class Contact
{
    [Required(ErrorMessage = "Website ist Pflichtfeld")]
    public string Website { get; set; }

    [Required(ErrorMessage = "E-Mail ist Pflichtfeld")]
    [EmailAddress(ErrorMessage =
        "Ungültige E-Mail-Adresse")]
    public string Email { get; set; }

    public Address Address { get; set; } =
        new Address();
}

public class Address
{
    [Required(ErrorMessage = "Adresse ist Pflichtfeld")]
    public string AddressText { get; set; }
```

`TextArea`> und so weiter) beherrschen seit der ersten Blazor-Version Eingabevalidierung gegen mit Datenvalidierungsannotationen versehene .NET-Objekte, wenn das Tag `<DataAnnotationsValidator>` in der Razor Component steht.

Allerdings funktionierte die Validierung bisher nur für die Properties der Objekte auf der obersten Ebene (Bild 5). Im Beispiel aus Listing 8, bei dem ein *Company*-Objekt ein *Contact*-Objekt und dieses wieder ein *Address*-Objekt enthält, wurden leider nur die Datenvalidierungsannotationen in der

Klasse *Company* ausgewertet. In dem in Listing 9 gezeigten Formular wurde daher bisher lediglich geprüft, ob die Eingaben bei Firma und Gründungsdatum stimmen, nicht aber bei Website und E-Mail sowie Adresse.

Seit .NET 10.0 Preview 6 kann Blazor nun auch Unterobjekte (komplexe Objekte beziehungsweise ganze Objekthierarchien) validieren (Bild 6). Blazor verwendet dabei die gleiche Implementierung wie ASP.NET Core Minimal WebAPI seit .NET 10.0 Preview 3. Die Implementierung liegt seit ►

● Listing 9: Eingabeformular für das Datenmodell aus Listing 8

```
@page "/Validation"
@using BO.WWWings
@using ITVisions
@using System.ComponentModel.DataAnnotations
@using NET10_BlazorServer.Model
@inject ITVisions.Blazor.BlazorUtil util

<h3>Eingabevalidierung für komplexe Objekte in Blazor
10.0</h3>

<EditForm EditContext="@editContext">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <div>
            <label for="companyName">Firma</label>
            <InputText class="form-control"
                id="customerFullName"
                @bind-Value="company.CompanyName" />
            <ValidationMessage For="@(() =>
                company.CompanyName)" />
        </div>
        <div>
            <label for="companyName">Gründungsdatum</label>
            <InputDate class="form-control"
                id="customerFullName"
                @bind-Value="company.Foundation" />
            <ValidationMessage For="@(() =>
                company.Foundation)" />
        </div>
        <div>
            <label for="website">Website</label>
            <InputText class="form-control"
                id="customerFullName"
                @bind-Value="company.Contact.Website" />
            <ValidationMessage For="@(() =>
                company.Contact.Website)" />
        </div>
        <div>
            <label for="email">E-Mail</label>
            <InputText class="form-control"
                id="customerEmail"
                @bind-Value="company.Contact.Email" />
            <ValidationMessage For="@(() =>
                company.Contact.Email)" />
        </div>
        <div>
            <label for="email">Adresse</label>
            <InputTextArea class="form-control"
                id="customerAddress" @bind-Value=
                "company.Contact.Address.AddressText" />
            <ValidationMessage For="@(() =>
                company.Contact.Address.AddressText)" />
        </div>
        <div>
            <button class="mt-2 btn btn-primary"
                @onclick="Submit">Speichern</button>
        </div>
    </div>
</EditForm>

@code {
    private Company company = new Company();
    private EditContext editContext;

    protected override void OnInitialized()
    {
        editContext = new EditContext(company);
    }

    void Submit()
    {
        if (editContext.Validate())
        {
            // Formular ist gültig
            util.Log("Formular ist gültig!");
        }
        else
        {
            // Formular ist NICHT gültig
            util.Log("Formular ist NICHT gültig!");
        }
    }
}
```


Preview 6 im neuen NuGet-Paket `Microsoft.Extensions.Validation` [7].

Dazu müssen Entwicklerinnen und Entwickler zwei Dinge hinzufügen:

- Einen Aufruf von `builder.Services.AddValidation()` im Startcode der Anwendung in `Program.cs`.
- Man muss die folgenden Zeilen in der Datei ergänzen, in der die Modellklassen stehen, wobei dies eine `.cs`-Datei sein muss; es funktioniert nicht, wenn die Modellklassen in `.razor`-Dateien liegen:

```
#pragma warning disable ASP0029
// Type is for evaluation purposes
// only and is subject to change
// or removal in future updates.
```

Eingabevalidierung für komplexe Objekte in Blazor 9.0

Die Validierung findet bei Blazor 9.0 nur auf der obersten Ebene in der Objekthierarchie statt (Bild 5)

Eingabevalidierung für komplexe Objekte in Blazor 10.0

In Blazor 10.0 können auch die Benutzereingaben für das Unterobjekt „Contact“ validiert werden (Bild 6)

Anlegen eines neuen Blazor-Projekts mit dem Authentifizierungstyp „Individual Accounts“ (Bild 7)

```
// Suppress this diagnostic to proceed.
[ValidatableType]
#pragma warning restore ASP0029
// Type is for evaluation purposes only and is
// subject to change or removal in future updates.
// Suppress this diagnostic to proceed.
```

Da ein Teil der Funktionen noch als „experimentell“ gilt, braucht man die Deaktivierung der Warnung.

Dies führt dazu, dass ein Source Code Generator den Validierungscode zur Kompilierungszeit erzeugt. Einsehen können Entwicklerinnen und Entwickler den generierten Programmcode im Projekt im Ast `Dependencies/Analyzers/Microsoft.Extensions.Validation.ValidationsGenerator`.

Passkeys-Anmeldung in ASP.NET Core Identity

ASP.NET Core Identity ist eine von Microsoft vordefinierte Benutzerverwaltung mit Weboberfläche und REST-Diensten,

Anlegen eines Passkeys innerhalb der Benutzerverwaltung (Bild 8)

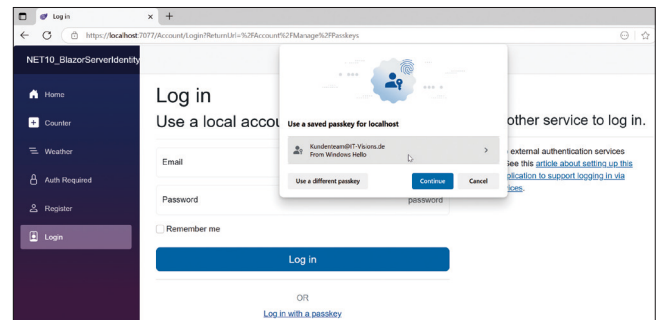
Man muss dem Passkey einen Namen geben (Bild 9)

Der Passkey wurde gespeichert (Bild 10)

die es schon lange gibt. Seit .NET 10.0 Preview 6 unterstützt ASP.NET Core Identity nun auch das Web Authentication API (WebAuthn) [8] alias Passkeys. Die Passkey-Implementie-

nung in .NET 10.0 unterstützt Resident Keys (Discoverable Credentials) und Non-Resident Keys, allerdings keine Attestation.

Der einfachste Weg zur Passkey-Unterstützung in einer Blazor-Anwendung führt über das Anlegen eines neuen Blazor-Projekts mit dem Authentifizierungstyp *Individual Accounts* (Bild 7) per Kommandozeile: `dotnet new blazor -au Individual`. Man findet dann zusätzliche Dateien wie *PasskeyOperation.cs*, *PasskeyInputModel.cs*, *Passkeys.razor* und *PasskeySubmit.razor* im Projekt in den Ordnern */Component/Account*, */Component/Account/Shared* und */Component/Account/Pages/Manage*. Diese Dateien kann man (wie bei ASP.NET Core üblich)



Mit dem gespeicherten Passkey kann man sich wieder anmelden (Bild 11)

CredentialId	UserId	PublicKey	Name	CreatedAt	SignCount	Transports	IsUserVerified	IsBackupEligible	IsBackedUp	AttestationObject	ClientData/son
390783C6B41630 027405d-e9f...	0uA40103033...	0uA40103033...	Passkey	22.07.2025 12:23...	1	["internal"]	True	False	False	0uA363666D...	0x782274797...

Tabelle „AspNetUserPasskeys“ im Microsoft SQL Server (Bild 12)

an eigene Bedürfnisse anpassen (Bild 8, Bild 9, Bild 10 und Bild 11). Die Passkey-Daten speichert ASP.NET Core Identity im Standard in seiner Microsoft SQL-Server-Datenbank in der neuen Tabelle *AspNetUserPasskeys* (Bild 12). ■

- [1] .NET 10.0 SDK, <https://dotnet.microsoft.com/en-us/download/dotnet/10.0>
- [2] C# Dev Kit, <https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csdevkit>
- [3] Hybrid-Cache-Bibliothek, <https://learn.microsoft.com/de-de/aspnet/core/performance/caching/hybrid?view=aspnetcore-9.0>

- [4] Circuit state persistence, <https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-10.0?#circuit-state-persistence>
- [5] Create simpler way to show Not Found 404 UI with Blazor, Issue #45654, <https://github.com/dotnet/aspnetcore/issues/45654>
- [6] NotFoundPage parameter, <https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-10.0?#blazor-router-has-a-notfoundpage-parameter>
- [7] NuGet-Paket Microsoft.Extensions.Validation, <https://www.nuget.org/packages/Microsoft.Extensions.Validation>
- [8] Web Authentication API, https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API

● Mehr Blazor-Neuerungen in .NET 10

Weitere Verbesserungen wird Blazor 10.0 in den Bereichen Monitoring, QuickGrid-Steuerelement und C#-JavaScript-Interoperabilität bieten. Dazu lesen Sie mehr im zweiten Teil dieses Artikels, der in der *dotnetpro* 12/2025 sowie Ende Oktober online unter developer-world.de/insights veröffentlicht wird.



Dr. Holger Schwichtenberg

zählt zu den bekanntesten .NET- und Webexperten in Deutschland. Microsoft zeichnet ihn seit 22 Jahren als MVP aus. Mit dem 54-köpfigen Expertenteam bei www.IT-Visions.de bietet er Schulungen und Softwareentwicklung an. www.dotnetdoktor.de



BENUTZEREINGABEN CHECKEN

FluentValidation

Komplexe Prüfreden für die Dateneingabe anlegen.

Das Thema Datenvalidierung verfolgt uns Anwendungsentwickler schon seit Jahren. Die zentrale Frage lautet: „Ist das die Goldene Lösung?“ Sowohl in Desktop- als auch in Webanwendungen sind alle Daten, die der Benutzer eingeben kann oder die „gegen das API geworfen werden“, auf Gültigkeit zu prüfen. Microsoft hat versucht, mehrere Konzepte in unterschiedlichen Frameworks zu etablieren. Stand heute sind

DataAnnotations das Mittel der Wahl. Ein etwas anderer Ansatz ist die Validierung mit dem Open-Source-Framework FluentValidation, das hier vorgestellt wird.

Microsofts Ansatz

Microsoft hat DataAnnotations schon vor vielen Jahren für .NET und ASP.NET MVC eingeführt. Hier ein Beispiel: Angenommen, es gibt eine Datenklasse *Person* mit Vorname und Geburtstag. Bild 1 zeigt den minimalen Code mit Attributen aus dem Namensraum *System.ComponentModel.DataAnnotations*. Die Attribute (hier *[Required]*) bestimmen, welche Regeln gelten. Die Standardattribute sind:

- *Required*,
- *NotEmpty*,
- *Length*,
- *Range*,
- *CreditCard*,
- *Phone*,
- *RegularExpression*,
- *Url*.

Diese Attribute werden einfach an die Properties geheftet und damit dann an einer beliebigen Stelle eine Validierung angefordert. Der Code für die Validierung selbst ist dabei ebenfalls überschaubar – siehe Bild 2.

Damit lässt sich eine beliebige Instanz einer *Person* validieren, und man erhält in der Collection *results* die Erfolgs- beziehungsweise die Fehlermeldung.

Die Grenzen von DataAnnotations

So weit, so einfach, doch wie jeder Ansatz hat auch dieser seine Tücken. Einerseits muss man Zugriff auf die Datenklasse haben, andererseits gibt es nur sehr einfache Regeln. Sobald komplexe Re-

```
public class Person
{
    [Required]
    public string FirstName { get; set; }

    [Required]
    public DateTime? Birthdate { get; set; }
}
```

Beispielklasse Person (Bild 1)

```
var results = new List<ValidationResult>();
var instance = new Person();
Validator.TryValidateObject(instance,
    new ValidationContext(instance),
    results,
    validateAllProperties: true);
foreach (var result in results)
{
    Console.WriteLine(result.ErrorMessage);
}
```

DataAnnotations zur Validierung (Bild 2)

geln erforderlich sind, zum Beispiel „Wenn die Person mindestens 18 ist, darf der Vorname auch leer sein“, wird dieser Ansatz sehr komplex. Microsofts Lösung dafür: Das Regelwerk ist entweder mittels eines eigenen Attributs zu implementieren, oder das Objekt selbst soll das Interface *IValidatableObject* implementieren und die Datenklasse wird um die neuen Regeln erweitert. Das bläht die Datenklasse immer weiter auf und kann zu schwerfälligen Datenklassen führen.

FluentValidation

Die Open-Source-Bibliothek FluentValidation [1] verfolgt einen anderen Ansatz, den ich nur empfehlen kann. Statt mit Attributen zu arbeiten, beschreibt eine eigene *Validator*-Klasse die erforderlichen Regeln. Knapp 700 Millionen Downloads (Bild 3) auf NuGet belegen den Verbreitungsgrad von FluentValidation, dazu kommen noch weitere NuGet-Pakete, welche die Integration in Anwendungsframeworks wie ASP.NET Core oder WPF erleichtern. Nach erfolgreicher Installation (Bild 4) können Sie mit FluentValidation sofort loslegen und die Beispielklasse prüfen. Dafür wird eine neue, eigenständige Klasse angelegt, die von *AbstractValidator<T>*



FluentValidation ist sehr beliebt (Bild 3)

```
<ItemGroup>
  <PackageReference Include="FluentValidation" Version="12.0.0" />
</ItemGroup>
```

Installation von FluentValidation mittels NuGet (Bild 4)

```
public class PersonValidator : AbstractValidator<Person>
{
    [1 usage]
    public PersonValidator()
    {
        RuleFor(x:Person => x.FirstName).NotEmpty();
        RuleFor(x:Person => x.Birthdate).NotEmpty();
    }
}
```

Die Klasse `PersonValidator` enthält das Regelwerk im Code (Bild 5)

```
var validator = new PersonValidator();
var validationResult = validator.Validate(new Person());
var isValid = validationResult.IsValid;
Console.WriteLine(isValid);
foreach (var error:ValidationFailure in validationResult.Errors)
{
    Console.WriteLine(error.ErrorMessage);
}
```

Die Validierung erfolgt ähnlich wie bei `DataAnnotations` (Bild 6)

ableitet, wobei das Argument *T* der Datentyp ist, der die zu prüfenden Daten enthält (Bild 5). Diese Klasse *Validator* kann anschließend im Konstruktor beliebige Regeln aufnehmen. Diese werden fließend geschrieben: *RuleFor(x => ...)* gibt an, welche Property gemeint ist, und anschließend können beliebige Methoden dazu definiert werden – wie etwa *NotEmpty()* für Werte, die nicht leer oder null sein dürfen.

Validierung starten

Um nun die Datenklasse mittels *FluentValidation* zu validieren, genügt es, die neue Klasse zu instanzieren, um sie anschließend zu nutzen (Bild 6). Die Klasse *PersonValidator* bietet eine synchrone und eine asynchrone Variante, die aufgerufen werden können, um die Validierungsergebnisse abzuholen – so weit, so easy. Bereits jetzt ist die Trennung komplett vollzogen. In der Datenklasse gibt es keine Regeln für die Validierung – eventuell vorhandene *DataAnnotations* werden dabei vollständig ignoriert.

Komplexe Regeln

Will man nun zum Beispiel einen Zusammenhang der Daten herstellen, bietet *FluentValidation* ebenfalls komfortable Wege an, wie folgendes Beispiel einer bedingten Validierung zeigt. Nur wenn das Geburtsdatum (*BirthDate*) mindestens 18 Jahre zurückliegt, muss auch der Vorname (*FirstName*) gesetzt werden (Bild 7). Durch den Code, den wir hier schreiben können, lassen sich beliebige Verkettungen und Bedingungen erzeugen.

Will man komplexere Regeln aufstellen, so kann man jederzeit auf die Methode *Must* zurückgreifen (Bild 8). *Must* kann dabei nicht nur den neuen Wert entgegennehmen, sondern auch die gesamte *Person*-Instanz. Damit kann man kunterbunte Varianten von Regelwerken niederschreiben.

Durch die Trennung der Regeln von den Daten kann die Validierungsklasse einerseits im Anwendungsteil (UI) verwendet werden, aber – falls gewünscht – gleichzeitig auch für die API-Validierung. Wer dann noch Unit-Tests schreiben will, dem wird es mit diesem Ansatz wirklich leicht gemacht.

Fazit

Man wird kaum jemanden finden, der sagt, Validierung sei sein Lieblingsthema, aber mit *FluentValidation* hat sich eine

```
public PersonValidator()
{
    RuleFor(x:Person => x.FirstName) // IRuleBuilderInitial<Person,string>
        .NotEmpty()
        .When(x:Person => x.Birthdate < DateTime.Now.AddDays(365 * 18));

    RuleFor(x:Person => x.Birthdate) // IRuleBuilderInitial<Person,DateTime?>
        .NotEmpty();
}
```

Komplexe Regeln? Kein Problem! (Bild 7)

```
public PersonValidator()
{
    RuleFor(x:Person => x.FirstName) // IRuleBuilderInitial<Person,string>
        .NotEmpty()
        .When(x:Person => x.Birthdate < DateTime.Now.AddDays(365 * 18));

    RuleFor(x:Person => x.Birthdate) // IRuleBuilderInitial<Person,DateTime?>
        .NotEmpty()
        .Must(BeGreater18);
}

[1 usage]
private bool BeGreater18(Person person, DateTime? givenDate)
{
    return true;
}
```

Eigene Regeln? Kein Thema! (Bild 8)

Lösung etabliert, mit dem man das eigene Regelwerk wirklich sehr, sehr einfach hinterlegen kann. Selbst die komplexesten Regeln lassen sich damit sehr elegant in der Anwendung hinterlegen. ■

[1] *FluentValidation*,
<https://github.com/FluentValidation/FluentValidation>



Christian Giesswein

studierte Wirtschaftsinformatik in Wien und entwickelt von klein auf Software mit .NET und C#. In Tirol hat er das Unternehmen Giesswein Software-Solutions gegründet, das sich auf Individualsoftware und Consulting spezialisiert hat.

christian@software.tirol

.NET MAUI UND IOS

Aloha für MAUI

In der Theorie funktionieren MAUI-Anwendungen mit geringem Aufwand auch unter iOS. Was in der Praxis bei der Bereitstellung einer Applikation zu beachten ist, zeigt der Artikel.

Der Autor dieser Zeilen durfte als Nebenprodukt eines Consultingprojekts eine iOS-Applikation entwickeln und vertreiben, die mit einem externen Hardware-Peripheriegerät kommunizierte. Der hier vorliegende Artikel basiert auf den dabei gewonnenen Erfahrungen.

Zu beachten ist dreierlei: Erstens ist der Artikel kurz gehalten und verzichtet, wo möglich, auf Bilder und Klickfolgen. Unter [1] finden sich aber Videos, die die diversen Klickfolgen en détail illustrieren.

Zweitens gilt, dass sich die iOS-Welt permanent verändert; es ist gut möglich (und von Apple gelebte Praxis), dass ein nächstens ausgeliefertes Update Probleme verursacht.

Zu guter Letzt sei darauf hingewiesen, dass der Artikel Vorkenntnisse im Bereich MAUI voraussetzt.

Arbeitsumgebung – oder: Was wird benötigt

MAUI- und Xamarin-Applikationen lassen sich per definitionem nur auf einem unter macOS laufenden Rechner „finalisieren“. Erste Aktion des Entwicklungswilligen ist daher der Kauf eines Mac-Rechners – Visual Studio für macOS ist abgekündigt und vom Komfort her in keiner Weise mit der unter Windows laufenden Vollversion vergleichbar. Stattdessen kommt die in **Bild 1** gezeigte Architektur zum Einsatz.

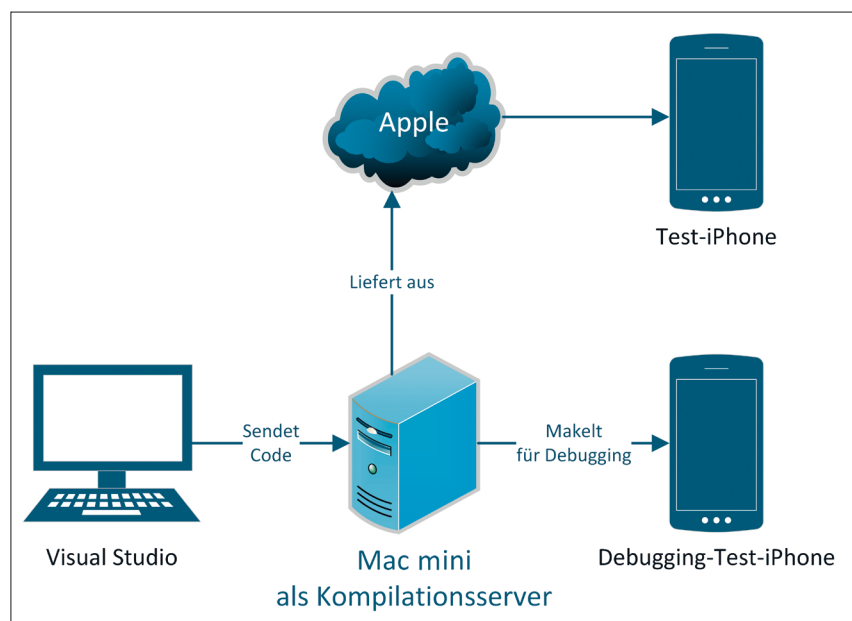
Eine vernünftige Mindestausstattung ist ein Mac mini, der im Gebrauchtmrkt zum Preis von etwa 450 Euro erhältlich ist. Der Autor verwendet derzeit einen Mac mini des Modelljahrs 2018 mit einer Core-i3-CPU – zu beachten ist hier allerdings, dass die Abkündigung von macOS-Updates über kurz oder lang einen Neukauf erforderlich machen wird.

Diese Maschine muss danach unbedingt per Ethernet mit dem Firmennetzwerk verbunden werden, um Latenzen im Bereich der Kommunikation zu eliminieren und die Arbeitseffizienz zu steigern.

128 GB an Remanentspeicher reichen dabei im Allgemeinen aus. Wichtig ist, regelmäßig den Mac aufzuräumen und in Xcode nicht mehr benötigte Versionen des iOS-SDK zu löschen. Apple hat nämlich die unangenehme (und durchaus umsatzfördernde) Eigenheit, alte Versionen anzusammeln

und den Entwicklungs-Mac auf diese Weise so gut wie möglich zuzumüllen.

Eine weitere Falle betrifft die Steuerung des Macs. Im Apple Store versucht man, ein als Apple Remote Desktop bezeichnetes und rund 90 Euro kostendes Softwarepaket mit dem VNC-Server zu verkaufen. In der Praxis ist dies allerdings nicht notwendig, die relevanten Einstellungen lassen sich wie unter [2] gezeigt auch lokal aktivieren. Da der Mac im Allgemeinen nur als Kompilationsrechner dient, sind Monitor und Maus beziehungsweise Tastatur nicht erforderlich. Empfehlenswert sind allerdings einige (lange) USB-Kabel, um die Test-iPhones während der Auslieferung der Kompilate mit dem Mac zu verbinden.



Schematische Darstellung der erforderlichen Architektur (Bild 1)

Pro aktiv mit iOS arbeitendem Entwickler ist dabei ein Mac empfehlenswert: Zum Testen der Applikation sollte das zwangsweise mit dem Mac verbundene iPhone in Griffweite liegen. Im Unternehmen des Autors wurde dies durch einen dedizierten „Macplatz“ gelöst, an dem der mit iOS testende Entwickler Platz nimmt – der Gutteil der Entwicklung erfolgt dagegen unter Android, weil Kompilation und Debugging dort bequemer sind.

iPhone kaufen ist Pflicht!

Wer mit realen Bluetooth-Geräten experimentiert, benötigt unbedingt ein reales iPhone. Im Simulator finden sich zwar einige Funktionen, eine Durchreichung des virtuellen iPhone-Bluetooth-Stacks an den Transceiver des Macs ist allerdings nicht implementiert.

Da der MAUI-GUI-Stack unter iOS mitunter seltsame und in Android nicht sichtbare Fehler aufweist, ist der Erwerb eines iPhones auch sonst dringend zu empfehlen. Ein iPad ist dagegen nur dann erforderlich, wenn diese Plattform den Nutzern der Applikation auch aktiv zur Verfügung stehen soll.

iPhones haben die unangenehme Eigenschaft, auch im gebrauchten Zustand noch richtig Geld zu kosten. Da das Unternehmen des Autors keinerlei iOS-Nutzer hat, erfolgte der Ankauf über diverse Gebrauchthändler. Empfehlenswert ist es dabei, ein Telefon mit hochauflösendem Bildschirm zu erwerben, weil dies die weiter unten besprochene Generierung der Screenshots für das iTunes-Listing erleichtert. Unbedingt notwendig ist dies indes nicht.

Insbesondere für Hardwareentwickler empfehlenswert ist es, ein Telefon mit brandaktuellem iOS und ein altes Telefon (lies: Telefon ohne weitere Major Updates) mit iOS 15 zu haben. Grund dafür ist, dass Apple zwischen iOS 15 und iOS 16 ein nicht wirklich dokumentiertes Update des Bluetooth-Stacks durchgeführt hat. Dies ist insofern heimtückisch, als dieses Update die Sichtbarkeit von Endstellen beschneidet. **Bild 2** zeigt zwei Geräte – das iOS-15-Gerät kann die Gegenstelle sehen, während sie für iOS 18 unsichtbar ist.

Die iOS-Flotte des Autors besteht aus einem iPhone SE der ersten Generation und einem aus dem Jahre 2020. Ein Ladegerät ist ebenfalls empfehlenswert und beim „Billigladen“ nebenan preiswert erhältlich – das Zubehör zertifiziert bei Apple zu kaufen ist natürlich ebenfalls möglich, aber nicht notwendig.

VS-Updates können schädlich sein

Die in **Bild 1** schematisch gezeigte Architektur setzt das Zusammenspiel diverser Komponenten sowohl aufseiten des Macs als auch aufseiten der Windows-Workstation voraus. Zu beachten ist, dass der auf dem Mac laufende und als IDB bezeichnete Agent nicht oder nur teilweise abwärtskompatibel ist.

Daraus ergeben sich mehrererlei besondere Handlungsanweisungen. Erstens gilt, dass die Aktualisierung einer einzelnen Maschine zwangsläufig auch zur Aktualisierung der am Mac befindlichen IDB-Version führt – in vielen Fällen hat dies zur Konsequenz, dass alle anderen Entwicklermaschinen fortan vom Pairing gegen ihren jeweiligen Kompilationsserver ausgeschlossen sind.

Außerdem gilt, dass die Codequalität im Bereich Visual Studio beziehungsweise Mac-Pairing nicht unbedingt ausgeprägt ist. So erlebte der Autor mehrfach, dass nach einem Visual-Studio-Update die vorher problemlos funktionierende Verbindung zwischen Windows-Workstation und Kompilationsserver plötzlich das Funktionieren einstellte.

Umgebungsmethode: Not-Mac

Ab einer gewissen Flottengröße beziehungsweise Empfindlichkeit der bearbeiteten Projekte kann es empfehlenswert sein, ein gebrauchtes ThinkPad und einen preiswerten Mac als „Not-Verbund“ zu definieren. Die beiden Geräte erhalten Updates prinzipiell erst dann, wenn die Hauptflotte aktualisiert ist – im Notfall lässt sich unter Nutzung dieser beiden Restanten noch ein Update der im Test befindlichen Elemente durchführen.

Aus diesen beiden Richtlinien ergibt sich eine Konsequenz: Insbesondere wer Applikationen im Test Flight hat, ist gut beraten, vor jeder Aktivierung des Visual Studio Installer eine Prüfung der noch verbleibenden Build-Lebenszeiten durchzuführen. In der Praxis braucht Microsoft zur Behebung der Probleme mitunter bis zu zwei Wochen – laufen die Builds in dieser Zeit ab, so gibt es wie weiter unten besprochen nur wenig Aussicht auf Errettung.

Außerdem gilt, dass Aktualisierungen von IDB mitunter auch Anpassungen der auf den Mac vorausgesetzten Mindest-Versionen mit sich bringen. Sofern eine SDK-Aktualisierung erforderlich wird, gilt, dass die Pflege des Macs durchgeführt werden muss, bevor die Applikation wieder kompiliert und ausgeliefert werden kann – auch hier also ein Fall, der eine eher sorgfältige Planung notwendig macht.

Zu guter Letzt zeigt die Erfahrung, dass Visual Studio in der englischen Sprachversion am besten funktioniert. In der ►



Foto: tam.hanna auf Instagram

Eines der Geräte ist nur am alten iPhone sichtbar (**Bild 2**)

Vergangenheit gab es immer wieder Fehler, in denen die deutsche Version von Visual Studio die Antworten des Macs aufgrund anders hart codierter Strings nicht richtig erkennen konnte – man ist also gut beraten, sowohl bei Visual Studio als auch beim Mac auf die englische Sprache zu setzen.

Verwaltung des iTunes-Entwicklerkontos

In Zeiten der Entwicklung für Symbian und Co. galt, dass aufgrund der OMA-DRM-Richtlinien nur Kapitalgesellschaften Applikationen zur Zertifizierung freigeben durften; eine Situation, an der Anbieter englischer Limiteds gutes Geld verdienten. Mittlerweile erlaubt Apple auch natürlichen Personen den Zugang – die Ausführungen in den folgenden Schritten beziehen sich allerdings auf Konten, die zu einer Kapitalgesellschaft (hier einer slowakischen k.s./s.r.o.) gehören.

Als Dreh- und Angelpunkt für die Kommunikation zwischen Apple und der Entwicklerschaft eines Unternehmens dient dabei ein mehr oder weniger gewöhnliches iTunes-Konto. Da dieses beziehungsweise seine Zugangsdaten bei der Arbeit immer wieder erforderlich sind, ist es in höchstem Maße empfehlenswert, für die Entwicklerschaft ein dediziertes iTunes-Konto zu verwenden und dieses sowohl in Entwicklungs-iPhones als auch im Kompilationsserver als Benutzer-Eigentümer-Account anzulegen.

Schon hier sei angemerkt, dass dies nicht bedeutet, dass entwicklereigene iPhones nicht ebenfalls zum Testen herangezogen werden können – weitere Informationen hierzu finden sich weiter unten oder in der iTunes-Entwicklerdokumentation unter dem Stichwort „Teamverwaltung“.

Die erste Aktion ist dann das Anlegen dieses neuen iTunes-Kontos. Hierfür ist logischerweise eine E-Mail-Adresse erforderlich – alternativ dazu kann man auch einen iCloud-Account anlegen. In diesem Fall muss allerdings dringendst darauf geachtet werden, dass dieser iCloud-Account regelmäßig überprüft wird – viele für den Geschäftsbetrieb höchst relevante Nachrichten stellt Apple nämlich nur, und zwar explizit nur, an dieses Konto zu.

Als Nächstes ist – die folgenden Schritte beziehen sich, wie weiter oben erwähnt, ja auf ein Unternehmen – eine Dun-&Bradstreet-ID erforderlich. Dun & Bradstreet ist eine Art „globales Firmenregister“, das so gut wie alle online einsehbaren Firmenregister durchsucht und den Firmen D-U-N-S-Nummern zuweist. Apple bietet unter [3] weitergehende Informationen dazu an, wie man die – wahrscheinlich bereits dem Unternehmen zugewiesene – ID ermittelt.

Als nächster Schritt folgt der Beginn der Freischaltung des Apple-Entwicklerkontos: Dabei handelt es sich um einen

Abonnementdienst, den Apple mit rund 100 US-Dollar pro Jahr in Rechnung stellt.

Die Mitgliedschaftsgebühr kann dabei im Rahmen der Zustimmung zu den Bedingungen des Formulars bezahlt werden. Wichtig ist, dass Apple im Rahmen dessen auch eine Handynummer abfragt, unter der ein Verifikations-Anruf eingeht. Wie Tests des Autors zeigten, kann dieser auch zu höchst ungewöhnlichen Zeiten und von seltsamen Nummern aus erfolgen. Empfehlenswert ist, bis zum Eintreffen dieses Anrufs – der Prozess kann bis zu einer Woche dauern – auch Anrufe von unbekannten Nummern entgegenzunehmen.

Im Fall des Autors erfolgte die eigentliche Verifikation dann eher benevolent. Die Anruferin wollte die D-U-N-S-Nummer des Unternehmens, die E-Mail-Adresse und die von Apple im Rahmen des Anmeldevorgangs ausgegebene ID wissen – es ist naturgemäß auch empfehlenswert, Informationen zu Firmenadresse und Co. bereitzuhalten. Nach der erfolgreichen Verifikation vergehen dann nochmals zwischen 24 und 48 Stunden, bevor das Entwicklerkonto zur Gänze einsatzbereit geworden ist.

An dieser Stelle findet sich eine weitere, potenziell existenzbedrohende Falle. Apple verlangt vom Entwickler, das Konto einmal jährlich zu verlängern. Diese Verlängerung wird allerdings – selbst bei gültiger Kreditkarte – nicht automatisch durchgeführt, sondern setzt einen manuellen Eingriff voraus. Unterbleibt dieser, so verschwinden alle Applikationen für einige Zeit auf dem Store.

User-Facing-Demovideo

Unter [6] findet sich ein Video, das auch den am User-iPhone durchzuführenden Flow demonstriert.

Zu allem Überfluss erlaubte Apple bei Abfassung dieses Artikels die Verlängerung erst dann, wenn nur noch 30 Tage „Kredit“ auf dem Konto sind, und sendet oft keine Warnmeldung. Es ist also in höchstem Maße empfehlenswert, diese Termine in den Terminkalender zu speichern und sich zeitgerecht dieses Problems anzunehmen.

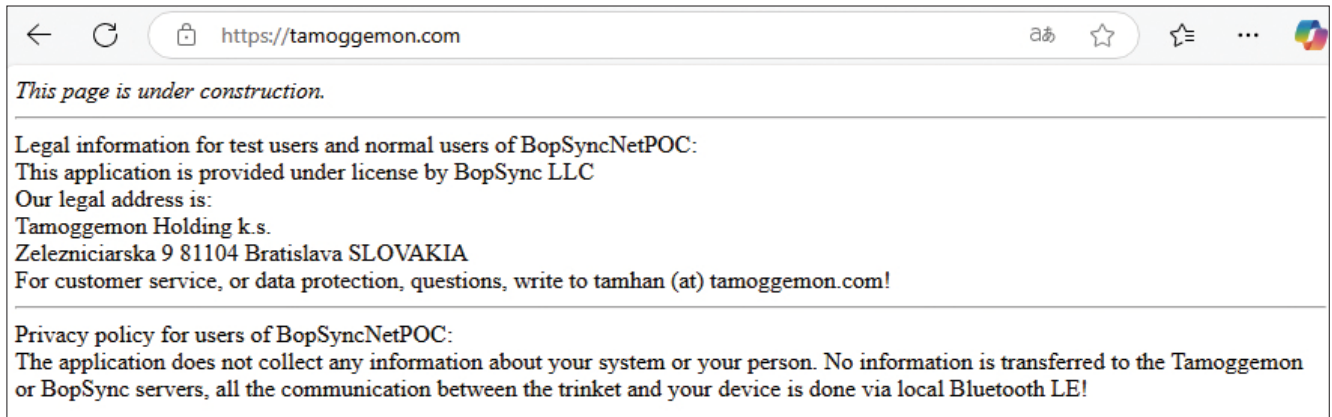
Applikationstest mit Test Flight

Die Auslieferung von Kompilaten an per Kabel mit dem Mac verbundene iPhones mag ein technisch gültiger Weg zur Auslieferung von Code sein, ist in der Praxis aber wenig befriedigend. Insbesondere wenn Kunden neu angeworben werden sollen oder müssen, ist das Test-Flight-Programm der bessere Weg. Auch für die Auslieferung per Test Flight verlangt Apple verschiedenste rechtliche Informationen auf der Webseite des Unternehmens. Eine Barebone-Variante, die von Apple allerdings problemlos akzeptiert wurde, präsentiert sich wie in Bild 3 gezeigt.

Im nächsten Schritt ist das „Hochladen“ eines Builds aus Visual Studio heraus erforderlich. Danach folgt der Wechsel

Videos zur Inbetriebnahme und Verbindung

Die unter [2] und [5] bereitstehenden Videos des Autors enthalten Klickfolgen und Hintergrundinformationen, die die Inbetriebnahme und Verbindung des Kompilationsrechners und auch die Konfiguration des VNC-Servers en détail erklären.



Mit dieser Webseite zum Erfolg (Bild 3)

in die Developer Console, wo eine neue Applikationsstruktur anzulegen ist.

Je nach Tageslaune verlangt Apple hier mitunter das Hochladen von hochauflösenden Screenshots. In Tests des Autors war es allerdings kein Problem, die auf dem iPhone SE 2020 aufgenommenen Screenshots unter Nutzung der Resize-Funktion von Gimp in das gewünschte Format zu bringen und diese danach in den Store hochzuladen. Zu beachten ist, dass die diversen gemachten Angaben auch für Betatester sichtbar sind – es ist also empfehlenswert, auf dümmliche oder falsche Angaben zu verzichten.

Nach dem Einrichten des Applikationsprofils folgt ein Wechsel in die Rubrik „Test Flight“, wo der eigentlich auszuliefernde Build ausgesucht werden kann.

Bitte beachten Sie, dass mit „Internal“ und „External“ zwei Benutzergruppen im Backend angelegt werden. Internal ist

eine Gruppe, in die man Benutzer nur von Hand hinzufügen kann. Die Gruppe External bekommt stattdessen einen nach dem Schema <https://testflight.apple.com/join/LFS1H99Q> aufgebauten URL eingeschrieben, über den sich mehr oder weniger beliebige User anmelden können, siehe Bild 4. Zu beachten ist dabei, dass das Backend in vielen Fällen das Vorhandensein einer Internal-Testing-Gruppe voraussetzt, bevor die External-Testing-Funktion freigeschaltet wird.

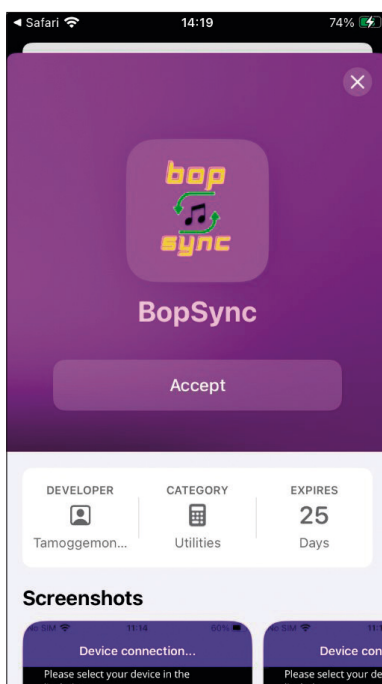
In Bezug auf Klickfolgen und sonstige Listings sei – wie weiter oben erwähnt – auf den YouTube-Kanal des Autors hingewiesen. Wichtig zu wissen ist indes, dass die Nutzung von Test Flight mit verschiedenen Besonderheiten einhergeht. Erstens gilt, dass eine über den Test-Flight-Kanal verteilte Applikation – wie in Bild 5 gezeigt – von Apple prinzipiell und immer mit einem Ablaufdatum ausgestattet wird. Dieses Ablaufdatum wird von Apple extrem streng gehandhabt und wirkt sich auch auf Betatester aus. Das bedeutet, dass das Nicht-Hochladen neuer Releases dazu führt, dass die Applikationen der Endkunden irgendwann schlicht nicht mehr funktionieren.

Erwähnenswert ist außerdem die Dauer der QA-Prüfungen. Bis zur Mitte 2024 erfolgte die Freischaltung in den Betaversionen so gut wie immer „instantan“, während Apple seither eine ein bis zwei Tage lange Schaffenspause eingelegt.

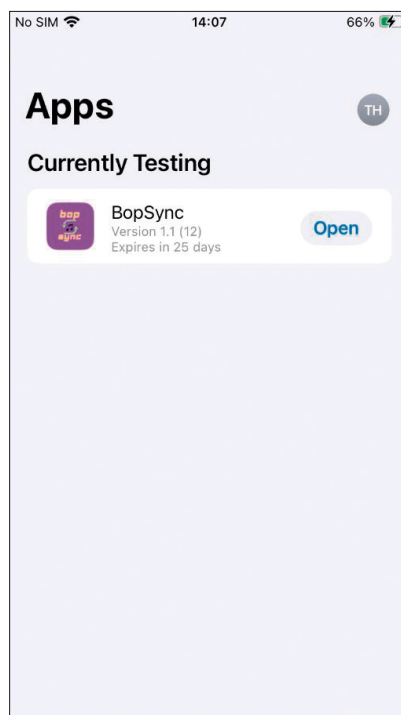
Zudem scheint es so zu sein, dass die interne Gruppe schneller aktualisiert wird als ihre externen Kollegen. Eng mit dem Unternehmen verbundene Tester sind deshalb dort besser aufgehoben. Zu beachten ist allerdings, dass insbesondere in den USA das zum Einpflegen notwendige Abfragen der Apple-ID des jeweiligen Benutzers in vielen Fällen als „feindselige“ Handlung verstanden wird.

Auf dem Weg zur Auslieferung

Aufgrund der diversen mit Test Flight einhergehenden Probleme ist es empfehlenswert, irgendwann eine „finale“ Auslieferung ►



Dieser User möchte dem System beitreten (Bild 4)



App, willst du ewig leben? (Bild 5)

anzustreben. Insbesondere bei der Arbeit mit US-amerikanischen Kunden ist außerdem dringend anzuraten, immer eine stabile Version des Programms in normaler Distribution zu haben – aus dem Autor unverständlichen Gründen wird eine Test-Flight-Bereitstellung dort vielfach als massiver Image-nachteil angesehen.

Wichtig ist im ersten Schritt, so viele Verträge wie möglich mit Apple zu unterschreiben: Selbst bei kostenloser Distribution ist es notwendig, wie unter [4] beschrieben, auch einen Kontrakt zum Vertreiben bezahlter Apps zu erstellen.

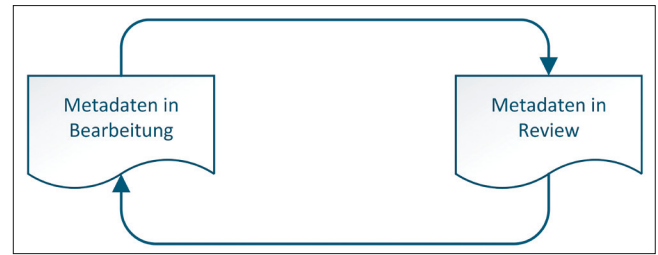
Als besonderes Hindernis erweist sich dabei häufig der Dialog für die Eingabe der Konten-Informationen. Statt freier Eingabe der Adresse der Bank ist es unbedingt erforderlich, den von Apple angebotenen Assistenten Schritt für Schritt abzuarbeiten.

Eine zweite Besonderheit betrifft das Backend, das bei der Release-Verwaltung wie in **Bild 6** schematisch gezeigt arbeitet. Eine Version der Metadaten ist dabei zur Bearbeitung freigegeben, während eine andere in Richtung von Apple zur Überprüfung geschickt wird.

Zu beachten ist, dass das erfolgreiche Bestehen der Prüfung nicht bedeutet, dass die Applikation sofort im Store verfügbar wird. Apple schickt stattdessen (siehe **Bild 7**) eine E-Mail, die zum Besuch des Backends und dort zur manuellen Freischaltung des Builds auffordert. Erst einige Stunden nach dieser manuellen Freigabe ist der Build in der Praxis verfügbar – wichtig ist das, weil das Unterlassen der manuellen Freigabe zu einer Applikation führt, die nie zum Download zur Verfügung stehen wird.

Fallstricke, insbesondere mit Bluetooth

Das vom Autor für diesen Artikel verwendete Projekt war der in **Bild 8** gezeigte intelligente Aschenbecher, dessen Einstellungen durch eine iOS-Applikation veränderbar waren.



Release-Verwaltung im Backend, schematische Darstellung (Bild 6)

Zuallererst sei angemerkt, dass Apple die Zusendung von realer Hardware nicht verlangte. Stattdessen waren einige Besonderheiten zu beachten. Erstens lehnte Apple die Auslieferung des Programms ab, weil der verwendete Beispielcode in der PLIST-Datei keine Gründe für die Verwendung der diversen Permissions lieferte.

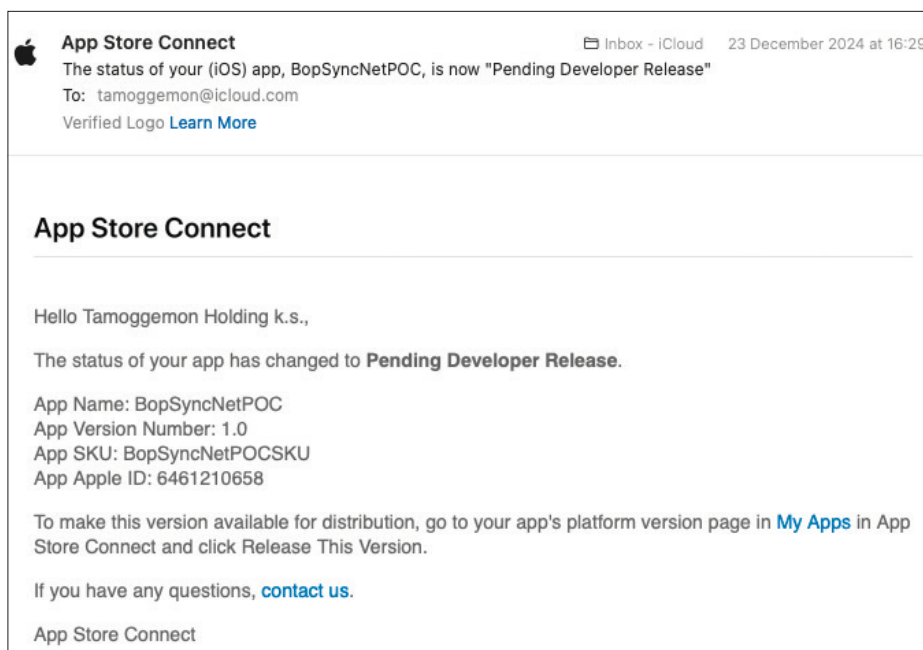
Zur Behebung reichte es aus, in der Datei nach folgendem Schema textuelle Begründungen bereitzustellen:

```

<key>NSBluetoothPeripheralUsageDescription</key>
<string>This app needs Bluetooth permission to connect
to the BopSync product, so that we can modify
the displayed colours and interact with its
settings!</string>
<key>NSBluetoothAlwaysUsageDescription</key>
<string>This app needs Bluetooth permission to connect
to the BopSync product, so that we can modify
the displayed colours and interact with its
settings!</string>
  
```

Problem Nummer zwei bildete die Entwicklung in der Rolle als Subunternehmer: Der Vertrieb der Applikation im iTunes App Store erfolgte durch die Tamoggemon Holding k.s., während die diversen Patente und Lizenzen auf ein anderes Unternehmen lauteten. Zur Behebung dieses Hindernisses reichte der im Kasten **Vollmacht** abgedruckte Text aus, der – von der Geschäftsführerin ausgedruckt, unterschrieben und als PDF eingescannt – im iTunes-Portal hochgeladen wurde.

Problem Nummer drei war dann ein Demovideo, das die Interaktion zwischen Hardware und iOS-Applikation illustrierte. Da Videoinhalte im Allgemeinen textuell schwer zu beschreiben sind, sei hier einfach auf das unter [4] bereitstehende Beispieldemo verwiesen – es enthält die gesamte von Apple akzeptierte Filmaufnahme. Die Bereitstellung der Aufnahme erfolgte übrigens durch ein YouTube-Video, das einfach als „unlisted“ hochgeladen wurde.



Wer diese E-Mail verpasst, muss sich nicht wundern (Bild 7)

● Vollmacht

Power of Attorney

On the *DATUM*, I, *NAME*, citizen of *LAND* furthermore trustor, acting both personally and as the owner and managing director of *FIRMA*, domiciled in *ADRESSE*, Company number being *ZAHL*, registered in the company court of *LAND* in accordance to *ENTSCHEIDUNG* (nur bei US-Firma), decide to, out of my free will, erect a power of attorney empowering

Ing. Tam HANNA / Bsc, born 10th September 1988, citizen of Austria, furthermore trustee, acting both personally and as the owner and managing director of Tamoggemon Holding k.s., company number ICO 46715983, registered in the company court of Bratislava III.

The power of attorney is limited to representing the interests of the trustor in front of all companies directly or indirectly owned, or beneficially owned, or directly or independently controlled, or affiliated to *Apple Inc.*, located at *One Apple Park Way, Cupertino, California*. This agglomeration is furthermore referred to as *Apple* in this document.

The power of attorney is limited in scope to handle trustors interests in front of the Apple iTunes App store personnel, its cadres, legal representatives, assessors and workers. The trustee is to upload, manage, promote and maintain applications developed by him or his associates, intended to enable the use of *FIRMENNAME* products with Apple hardware.

Should the trustee need to use *FIRMENNAME* trademarks (such as *NAME*, number *NUMMER*) or patents (such as number *NUMMER*) to achieve his goals as set out in this power of attorney, trustor agrees to this and provides a license and permission as required by law. This license also extends to Apple.

In principle, this power of attorney is erected in perpetuity, until canceled in writing by trustor or trustee. In the case of a significant material change in economic status at trustor or trustee, this power of attorney remains valid until withdrawn in writing by trustor or trustee.

As the trustee, I accept the erection of the power of attorney and promise under penalty of perjury to only act in the interests of the trustor as to the best of my knowledge, and to keep away the damages and harms from the property and reputation of the trustor.

Signature for trustor
Date, place

Signature for trustee
Date, place

Interessant ist, dass Apple hier sogar über Feiertage hinweg Aktivität demonstrierte – in Tests erhielt der Autor teilweise eine Reaktion binnen drei Kalendertagen. Oft gelang der QA-Durchlauf auch in nur zwölf Stunden. Es ist also empfehlenswert, eine gewisse Latenz einzuplanen, aber dennoch regelmäßig die erwähnte iTunes-Mailbox zu kontrollieren.



Das Corpus Delicti (Bild 8)

Fazit

Böse Zungen bezeichnen Apple als einzige Religion im Computerbereich. Nach einigen Jahren des Experimentierens und praktischer Herausforderungen kann der Autor dieser Einschätzung nur zustimmen – wer MAUI-Applikationen unter iOS vertreibt, lebt mit Sicherheit interessanter ...

- [1] Tam Hanna, Deploy and beta test MAUI apps to iOS beta testers, www.youtube.com/watch?v=etzE9wStb2M&list=PL1rWqPR22pnS9AFWZRD21hHRL8Y-1rPTG
- [2] Tam Hanna, Set up Visual Studio 2022 for iOS development with MAUI and a Mac, www.youtube.com/watch?v=Lb3WHoQ71io
- [3] Apple Developer, D-U-N-S Number, <https://developer.apple.com/support/D-U-N-S/>
- [4] Tam Hanna, Release MAUI apps to the iTunes App Store, www.youtube.com/watch?v=_Zuy9Rdq520
- [5] Tam Hanna, BackMarket Mac Mini unbox and Xamarin/MAUI developer setup, www.youtube.com/watch?v=v0zUEgIS9y8
- [6] Tam Hanna, Beta test MAUI apps on iOS using Test Flight, www.youtube.com/watch?v=dNvyxk8qrws



Tam Hanna

entwickelt Programme für verschiedene Plattformen, betreibt Online-Newsdienste zum Thema und steht für Fragen, Trainings und Vorträge gern zur Verfügung. Sie erreichen ihn unter der E-Mail-Adresse tamhan@tamoggemon.com.

OPENAPI, SWAGGER UND SCALAR IN ASP.NET CORE MEISTERN

API Contracts perfekt im Griff

Schnittstellen zu dokumentieren ist einfach – aber stabile und wartbare APIs entstehen erst, wenn aus bloßer Beschreibung ein verlässlicher Vertrag wird, der Standards folgt.

Verträge sind das unsichtbare Fundament unseres gesellschaftlichen Zusammenlebens. Sie begegnen uns in nahezu allen Lebensbereichen: Beim Abschluss eines Mietvertrags, der genau regelt, wer welche Rechte und Pflichten rund um eine Wohnung hat. Im Arbeitsvertrag, der festlegt, welche Aufgaben zu erfüllen sind und welche Gegenleistungen erbracht werden – und wie viel Urlaub man hat. Oder auch in den kleinen, alltäglichen Absprachen zu Hause – etwa, wer den Müll rausbringt.

All diese Vereinbarungen sorgen dafür, dass Erwartungen klar sind und jeder weiß, woran man ist. Vereinbarungen schaffen Vertrauen, minimieren Missverständnisse und bilden die Basis für reibungslose Zusammenarbeit. Ohne diese Grundpfeiler würde unser Alltag schnell im Chaos versinken; denn erst durch klare Abmachungen werden Rechte, Pflichten und Verantwortlichkeiten für alle Beteiligten transparent.

sich stabile, wartbare und skalierbare Anwendungen entwickeln. API Contracts sind somit das Rückgrat moderner Softwarearchitekturen und sorgen dafür, dass alle Beteiligten – ob nun Mensch oder Maschine – die gleichen Spielregeln kennen und einhalten. Und sie werden noch viel wichtiger, sobald Anwendungen mit künstlicher Intelligenz ins Spiel kommen.

In meinen rund 25 Jahren Softwareentwicklung habe ich vieles gesehen – und auch Softwareprodukte, die sich nicht an Standards halten; weder projektinterne Standards noch allgemeine Standards. Da ich mich vor allem im Cloud- und Web-Umfeld bewege, habe ich besonders viel mit HTTP-Schnittstellen und deren Standards zu tun.

Aus diesem Grund habe ich vor mehreren Jahren das Projekt `dotnet.rest` [1] gestartet, das eine Fülle an Standards beschreibt – mittlerweile nicht mehr nur HTTP und REST. Es deckt eine breite Anzahl von API-Contract-Standards für .NET-Entwickler ab.

● **Tabelle 1: Beispiele für den Einsatz der HTTP-Verben und URI-Strukturen**

HTTP-Verb	Beispiel-URI	Bedeutung / Verwendung
GET	/customers	Liste aller Kunden abrufen
GET	/customers/31	Einzelnen Kunden mit der ID 31 abrufen
POST	/customers	Neuen Kunden anlegen
PUT	/customers/31	Bestehenden Kunden #31 vollständig ersetzen
PATCH	/customers/31	Bestehenden Kunden #31 teilweise aktualisieren
DELETE	/customers/31	Kunden löschen

Was im täglichen Leben gilt, ist in der Welt der Schnittstellenentwicklung von Softwareanwendungen umso wichtiger; auch hier treffen unterschiedliche Parteien aufeinander: Entwicklerteams, externe Dienstleister, Frontend und Backend, Mensch und Maschine.

Die Komplexität moderner Anwendungen wächst stetig und mit ihr die Notwendigkeit, klare Schnittstellen und Absprachen zu definieren. Damit diese Zusammenarbeit funktioniert, braucht es klare Verträge – sogenannte API Contracts. Sie legen fest, wie Systeme miteinander kommunizieren, welche Daten und Formate erwartet und geliefert werden und welche Regeln dabei einzuhalten sind. Nur so lassen

von der hinterherhängenden Dokumentation unterschieden; der Frust war hoch. Swagger brachte einen maschinenlesbaren Standard ins Spiel, mit dem sich APIs nicht nur dokumentieren, sondern auch automatisch testen und generieren ließen. Besonders die Funktionalität der Code-beziehungsweise Contract-Generierung durch Code fand schnell Anklang in der Entwickler-Community und wurde zum De-facto-Standard für HTTP-API-Beschreibungen.

2016 wurde das Projekt an die OpenAPI Initiative [3] der Linux Foundation „gespendet“ und in „OpenAPI“ umbenannt. Wenn wir also heute von OpenAPI reden, reden wir im Endeffekt vom Nachfolger von Swagger.

Swagger versus OpenAPI

Im HTTP-Umfeld hat sich Swagger [2] zu einem echten Synonym für API Contracts entwickelt, obwohl Swagger ursprünglich um die 2010er-Jahre als Dokumentationstool für RESTful APIs ins Leben gerufen wurde. Damals war es üblich, Schnittstellen in Wikis oder Textdateien zu beschreiben – ein fehleranfälliger und oft inkonsistenter Prozess. Die reale Welt hat sich meist stark

Eine sehr große Bedeutung bei OpenAPI als Standard hat weiterhin, dass Best Practices wie das Richardson Maturity Model [4] – insbesondere Level 2 – angewandt werden. Das bedeutet, dass die URIs der Ressourcen sinnvoll und konsistent gestaltet werden und die HTTP-Verben (*GET*, *POST*, *PUT*, *DELETE* et cetera) korrekt und gemäß ihrer Bedeutung verwendet werden. Nur so lassen sich wirklich saubere, wartbare und interoperable APIs schaffen. Typische Beispiele für den Einsatz der HTTP-Verben und URI-Strukturen nach Level 2 des Richardson Maturity Models zeigt [Tabelle 1](#). Ressourcen werden über sprechende URIs adressiert, und die HTTP-Methoden werden semantisch korrekt verwendet.

Im Nachfolgenden wird OpenAPI als modernere Begrifflichkeit verwendet; die Aussagen sind aber auf Swagger eins zu eins übertragbar.

Contract Development

Bei der Konzeption von OpenAPI Contracts gibt es maßgeblich zwei weit verbreitete Herangehensweisen: den Top-down- und den Bottom-up-Ansatz. Beim Top-down-Ansatz wird der Contract – also die Beschreibung der Schnittstelle – zuerst entwickelt. Dies geschieht in der Regel durch den Dev-Lead oder das gesamte Team, oft gemeinsam mit den Frontend- und Backend-Entwicklern. Der Vorteil: Alle Beteiligten einigen sich frühzeitig auf die Schnittstelle, was Missverständnisse minimiert und eine klare Grundlage für die weitere Entwicklung schafft. Dadurch lässt sich das Vorgehen sehr gut in einen agilen Prozess integrieren, da die Implementierung nach dem Refinement sofort angegangen werden kann und zum Beispiel das Frontend-Team nicht auf das Backend-Team warten muss.

Der Contract dient dabei als Ziel der Entwicklung für Backend und Frontend beziehungsweise es kann anschließend automatisch Server- und Client-Code generiert werden. Dabei dient der Contract direkt auch als Basis für Unit-Tests. Für die Entwicklung des OpenAPI Contracts auf YAML- oder JSON-Basis stehen verschiedene Tools zur Verfügung, aber auch Visual-Studio-Code-Extensions wie der OpenAPI Editor von 42Crunch [5] oder Redocly OpenAPI [6].

Beim Bottom-up-Ansatz hingegen wird zunächst der Code – meistens das Backend – geschrieben. Aus dem bestehenden Code wird dann mithilfe von Tools wie Swagger automatisch ein OpenAPI Contract generiert. Dieser Ansatz ist bedeutend schneller in der Umsetzung, da sich dadurch der API Contract automatisch durch die Endpunkte-Definition – zum Beispiel ASP.NET Core Controller implizit beziehungsweise durch Minimal API explizit – generieren lassen. In der Realität ist jedoch oft zu sehen, dass es vielen generierten OpenAPI-Definitionen an Qualität mangelt: Beschreibungen fehlen oder Standards von RESTful beziehungsweise OpenAPI werden – meist durch Leichtsinnsfehler – missachtet. Des Weiteren wird oft auf Tests der generierten OpenAPI-Definitionen verzichtet: „Es wird ja alles generiert.“

Die allgemeine Empfehlung lautet deshalb: Top-down für Entwicklungsteams für eine potenziell bessere Contract-Qualität, Bottom-up für Prototypen, kleinere Tools oder rein teaminterne APIs. Besonders APIs, die zur externen Verwen-

dung gedacht sind, profitieren von Top-down durch eine in der Regel höhere Qualität der Dokumentation und dadurch geringeren Frust bei externen Nutzern.

OpenAPI mit .NET

Die Adaption von OpenAPI in der .NET-Welt kam maßgeblich durch Open-Source-Projekte von Dritten zustande. Besonders hervorzuheben sind die beiden Projekte NSwag [7] (vorrangig durch Rico Suter entwickelt, mit rund 90 Millionen

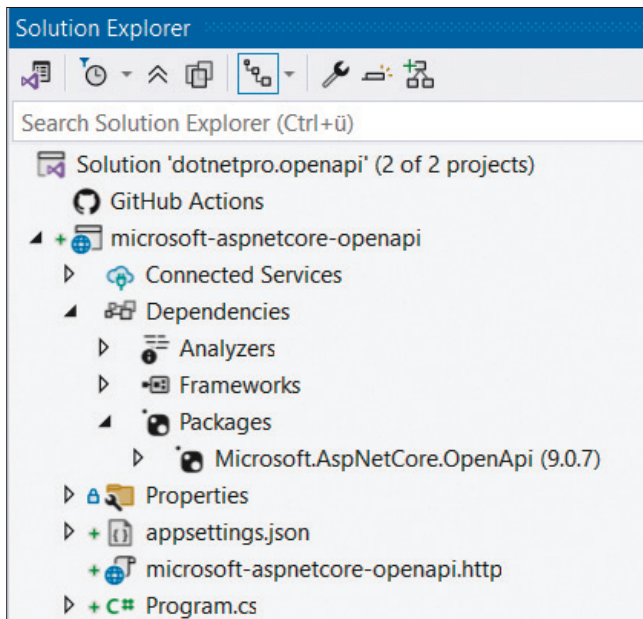
The screenshot shows the 'Additional information' configuration window in Visual Studio. It has tabs for ASP.NET Core Web API, C#, Linux, macOS, Windows, API, Cloud, Service, Web, and Web API. The 'Framework' dropdown is set to '.NET 9.0 (Standard Term Support)'. The 'Authentication type' is set to 'None'. There are checkboxes for 'Configure for HTTPS' (checked), 'Enable container support' (unchecked), 'Container OS' (set to 'Linux'), 'Container build type' (set to 'Dockerfile'), 'Enable OpenAPI support' (checked), 'Do not use top-level statements' (unchecked), 'Use controllers' (unchecked), and 'Enlist in .NET Aspire orchestration' (unchecked). The 'Aspire version' dropdown is set to '9.2'.

Erstellen einer ASP.NET-Core-Anwendung mit OpenAPI-Support in Visual Studio (Bild 1)

Downloads des Pakets NSwag.AspNetCore [8]) und Swashbuckle [9] (beeindruckende 780 Millionen Downloads des Pakets Swashbuckle.AspNetCore [10]). Diese beiden Bibliotheken haben entscheidend dazu beigetragen, dass OpenAPI/Swagger in .NET-Anwendungen weit verbreitet wurde und sich als Standard etablierte. Diese Community-getriebenen Lösungen füllten über Jahre hinweg eine wichtige Lücke im .NET-Ökosystem und etablierten sich als De-facto-Standards für OpenAPI-Integration in .NET. Ihre breite Akzeptanz und kontinuierliche Weiterentwicklung machten sie zu unverzichtbaren Bibliotheken für API-Entwickler, lange bevor Microsoft im Jahr 2022, beginnend für .NET 7, eine eigene offizielle Unterstützung bereitstellte.

ASP.NET Core mit Microsoft.AspNetCore.OpenApi

Mit der Veröffentlichung von .NET 7 hat Microsoft zusammen mit dem ASP.NET-Core-Team einen wichtigen Meilenstein erreicht: die Einführung einer eigenen, hochqualitativen Lösung für das weit verbreitete „Bottom-up“-Vorgehen namens Microsoft.AspNetCore.OpenApi [11]. Dieses offizielle Paket wurde speziell für die automatische Generierung von OpenAPI-Dokumentation aus ASP.NET-Core-API-Endpunkten entwickelt und zeichnet sich durch seine nahtlose Integration in das .NET-Ökosystem aus. Ein wichtiger Unterschied zu etablierten Lösungen wie NSwag liegt darin, dass Microsoft.AspNetCore.OpenApi bewusst auf Code-Generatoren verzichtet und sich ausschließlich auf die Dokumentationsgenerierung ►



Ansicht der Paket-Abhängigkeiten in Visual Studio (Bild 2)

konzentriert. Diese Fokussierung ermöglicht eine optimierte Performance und eine schlanke, wartbare Codebasis.

Die Integration von Microsoft.AspNetCore.OpenApi in den Entwicklungsworkflow wurde seit .NET 9 erheblich vereinfacht. Bei der Erstellung eines neuen ASP.NET-Core-Projekts in Visual Studio wird Entwicklern im Projekterstellungsdialog automatisch die Option *OpenAPI Support* angeboten, was die Einrichtung zu einem Ein-Klick-Vorgang macht (Bild 1).

Bei der Projekterstellung wird automatisch die jeweils aktuellste Version des Pakets referenziert – im Beispiel v9.0.7 –, und alle notwendigen Abhängigkeiten werden konfiguriert.

Das generierte Projekt enthält nicht nur den standardmäßigen ASP.NET-Core-WebAPI-Beispielcode für Wettervorhersagen, sondern auch die beiden essenziellen Konfigurationszeilen: *AddOpenApi()* für die Registrierung der OpenAPI-Dienste in der Dependency Injection sowie *MapOpenApi()* für die Registrierung der OpenAPI-Endpunkte. Diese Konfiguration stellt automatisch den Endpunkt */openapi/v1.json* zur Verfügung, der die vollständige OpenAPI-JSON-Definition basierend auf den definierten Minimal-API-Endpunkten generiert und ausliefert (Bild 2).

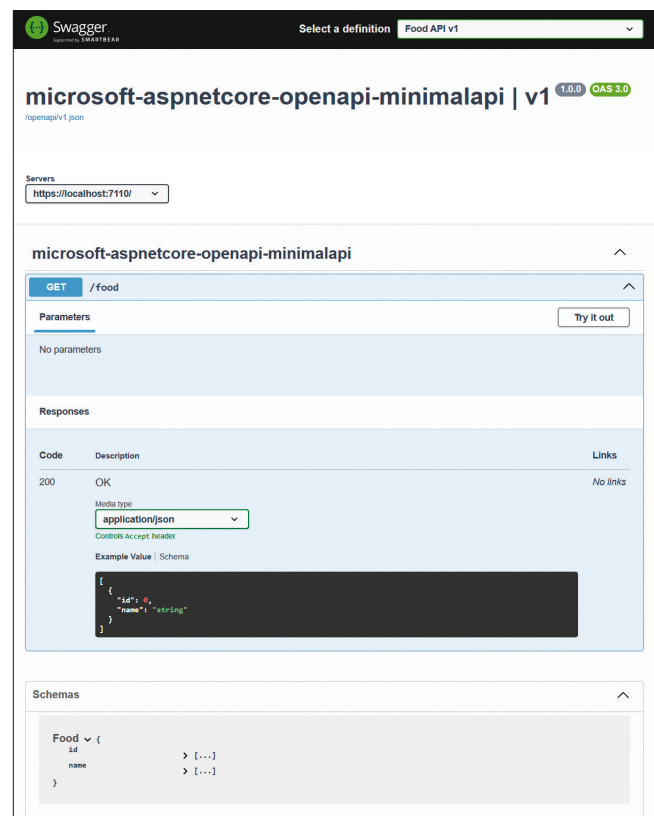
Ein besonderer Vorteil von Microsoft.AspNetCore.OpenApi ist die vollständige Unterstützung für Native AOT (Ahead-of-Time Compilation): Native AOT [12] kompiliert C#-Code bereits zur Build-Zeit in nativen Maschinencode, wodurch sich drastisch verkürzte Startzeiten und ein reduzierter Speicherverbrauch ergeben. Diese Compiler-Option ist besonders in Container-Umgebungen von enormer Bedeutung, da sie zu schnelleren Container-Starts, effizienterer Ressourcennutzung und damit zu kostengünstigerem Betrieb und schnelleren Deployments führt. Wichtig: Um den enormen Vorteil von Native AOT nutzen zu können, muss das API via Minimal API umgesetzt sein. ASP.NET Core Controller unterstützen Native AOT im Gegensatz zum Minimal API nicht – und sie werden auch keine Unterstützung für Native AOT erhalten.

Ein Aspekt, der bei der Verwendung von Microsoft.AspNetCore.OpenApi zu beachten ist: Das Paket bringt im Gegensatz zu NSwag oder Swashbuckle keine integrierte Benutzeroberfläche mit; der Endpunkt */openapi/v1.json* liefert ausschließlich den OpenAPI Contract. Für eine interaktive API-Dokumentation und API-Tests kann jedoch problemlos auf bewährte Lösungen wie SwaggerUI [2] zurückgegriffen werden, die sich nahtlos mit der generierten OpenAPI-Definition integrieren lassen:

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
    app.UseSwaggerUI(options =>
    {
        options.SwaggerEndpoint("/openapi/v1.json",
            "Food API v1");
    });
}
```

Minimal APIs: Dokumentation am Beispiel eines Food-API

Gerade bei Minimal APIs in ASP.NET Core zeigt sich, wie stark die Qualität des OpenAPI Contracts von der Sorgfalt der Entwicklung abhängt. Im Folgenden werden zwei Beispiele anhand eines Food-API-Endpunkts vorgestellt, die die Auswirkungen auf die generierte OpenAPI-Beschreibung verdeutlichen.



Swagger-Frontend ohne Endpunktdokumentation (Bild 3)

Beispiel 1:

Minimal API Endpoint mit unzureichender Dokumentation

Im ersten Beispiel wird ein Endpoint definiert, der eine Liste von Gerichten zurückgibt:

```
// fehlende Dokumentation
app.MapGet("/food", Ok<List<Food>>() =>
    TypedResults.Ok(foods)
);
```

Auf zusätzliche Metadaten oder Dokumentation wird ganz im Sinne des Minimalisten vollständig verzichtet.

Die Implementierung ist zwar funktional, doch für die Open-API-Generierung fehlen sämtliche Hinweise zu Zweck, Rückgabewerten, möglichen Fehlerfällen oder semantischen Details. Die resultierende äußerst rudimentäre OpenAPI-Beschreibung bietet für Konsumenten des API kaum Mehrwert (Bild 3). Eine klassische Endpunktbeschreibung eines absoluten Minimalisten: Wichtige Informationen wie Beschreibung, Tags, Response-Typen oder Fehlercodes fehlen. Gerade bei der automatisierten Client-Generierung oder bei der Nutzung von Tools wie SwaggerUI wird das deutlich: Das API bleibt für Außenstehende schwer verständlich und wenig wartbar.

Beispiel 2:

Minimal API Endpoint mit umfassender Dokumentation

Das zweite Beispiel zeigt, wie mit wenig zusätzlichem Code die Qualität in die Höhe springt und eine deutlich bessere Dokumentation und damit ein qualitativ hochwertiger OpenAPI Contract erzeugt werden kann:

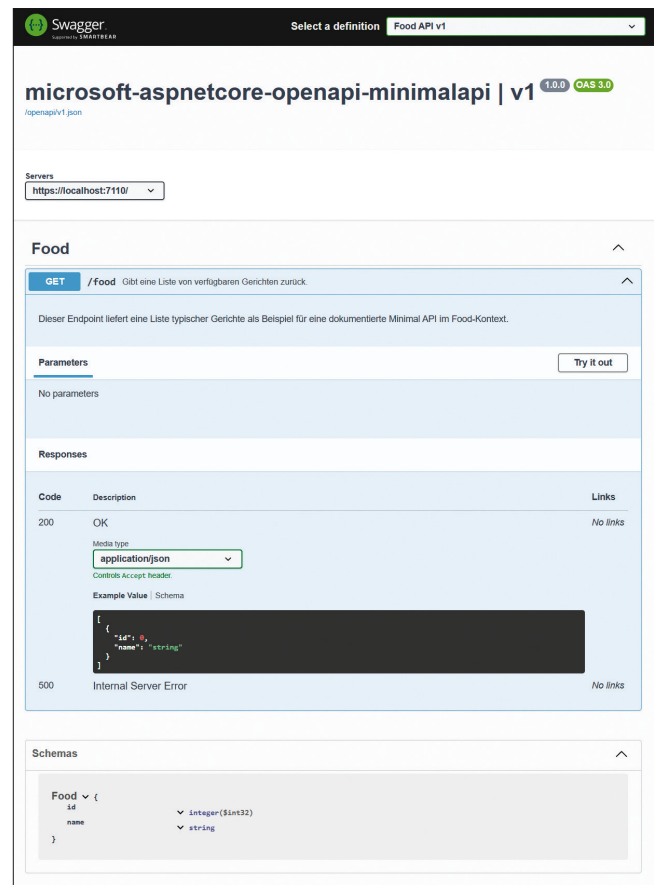
```
// Mit Dokumentation
app.MapGet("/food", Ok<List<Food>>() =>
    TypedResults.Ok(foods)
)
.WithName("GetFoodList")
.WithTags("Food")
.WithSummary(
    "Gibt eine Liste von verfügbaren Gerichten zurück.")
.WithDescription("Dieser Endpoint liefert eine Liste typischer Gerichte als Beispiel für ein dokumentiertes Minimal API im Food-Kontext.")
.Produces<List<Food>>(StatusCodes.Status200OK, "application/json")
.Produces(StatusCodes.Status500InternalServerError);
```

Das Resultat ist eine OpenAPI-Beschreibung, die nicht nur maschinenlesbar, sondern auch für Menschen verständlich und wartbar ist (Bild 4).

Gerade in größeren Teams oder bei der Veröffentlichung von APIs für Dritte ist diese Sorgfalt entscheidend, um Missverständnisse zu vermeiden und die Integration zu erleichtern.

Verbesserungen mit Microsoft.AspNetCore.OpenApi v10.x

Wie leider oft bei offiziellen .NET-Paketen üblich, mangelt es bei den ersten Releases einer Standardfunktionalität an De-



Swagger-Frontend mit Endpunktdokumentation (Bild 4)

tails: So steht bis einschließlich .NET 9 kein direkter Support der Integration von XML-Code-Kommentaren in die UI-Ansicht durch `UseSwaggerUI()` zur Verfügung. Wie in Bild 4 zu sehen ist, fehlen die XML-Kommentare des Food-Modells. Diese bequem zu integrieren ist derzeit nur mit weiteren Bibliotheken möglich.

Mit dem Release der Version 10 von Microsoft.AspNetCore.OpenApi (aktuell als Preview Version 10.0.0-preview.6.25358.103) werden durch Source Generators ohne weiteren Code-Aufwand auch XML-Kommentare – diese müssen einfach nur in den Projekteinstellungen aktiviert sein – in die Ansicht integriert; und das sogar auf optisch wirklich hochwertige Weise (Bild 5).

Microsoft Kiota für Top-down-Entwicklung

Für Entwicklungsteams oder Dev-Leads, die den Top-down-Ansatz bevorzugen, beziehungsweise wenn dieser sich aufgrund des agilen Vorgehens besser eignet, stellt Microsoft mit dem Projekt Kiota [13] ein mächtiges Kommandozeilen-Tool zur Verfügung.

Kiota Installieren

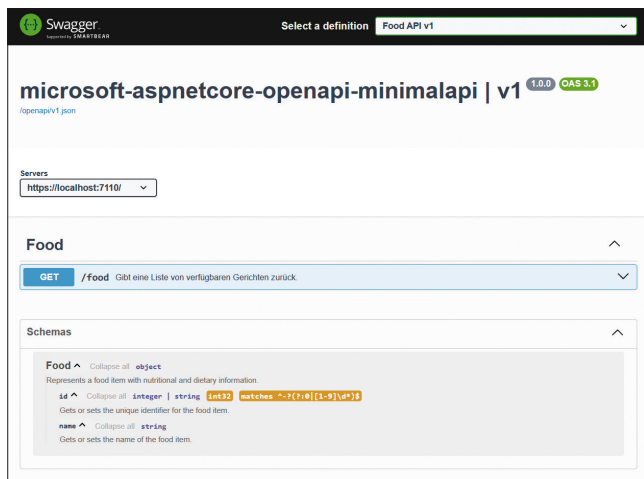
```
dotnet tool install --global Microsoft.OpenApi.Kiota
```

```
## C# Client-Code anhand einer openapi.yml generieren
kiota generate -l CSharp -c FoodClient -n Food.Client
-d ./sample-openapi..yaml -o ./Client
```



```
# TypeScript-Client-Code anhand einer openapi.yml
# generieren
kiota generate -l TypeScript -c FoodClient -n Feod.Client
-d ../../sample-openapi.yml -o ./TSClient
```

Kiota ermöglicht die Generierung von Client-Code für eine beeindruckende Vielfalt von Programmiersprachen und Plattformen. Zu den vollständig unterstützten Sprachen ge-



Swagger-Frontend mit neuer, ausführlicher Endpunktdokumentation in .NET 10 (Bild 5)

hören C#, Dart, Go, Java, PHP und Python. Darüber hinaus befinden sich Ruby und TypeScript zum Zeitpunkt der Abfassung dieses Artikels in der Preview-Phase.

Führt man die Generierung aus, werden die entsprechenden Dateien innerhalb des ausgewählten Verzeichnisses erstellt und Kiota antwortet, welche manuellen Schritte man selbst noch umsetzen muss:

```
kiota generate -l TypeScript -c FoodClient -n Food.Client
-d ../../sample-openapi.yml -o ./TSClient
Warning: the TypeScript language is in preview (Preview)
some features are not fully supported and source
breaking changes will happen with future updates.
warn: Kiota.Builder.KiotaBuilder[0]
OpenAPI warning: #/ - Multiple servers entries
# were found in the OpenAPI description. Only
# the first one will be used. The root URL can
# be set manually with the request adapter.
Generation completed successfully
Client base url set to https://api.example.com/v1
```

Kiota beschränkt sich dabei ausschließlich auf die Generierung von Client-Code; für die Generierung von Server-Code für das Top-down-Vorgehen muss weiterhin auf Open-Source-Projekte von Drittanbietern wie NSwag zurückgegriffen werden.

Besonders interessant ist der Einsatz von Kiota im CI/CD-Prozess, da sich auf diese Weise Konsistenz und Nachvoll-

ziehbarkeit im Team sichern lassen, sowohl für das Backend-Team als auch für das Frontend-Team. Eine potenzielle Best Practice ist es, die API-Definition im Repository abzulegen (zum Beispiel als *openapi.json*) und ein Skript mit dem Kiota-Befehl (*dotnet tool restore && kiota generate ...*) sowohl lokal als auch in der Pipeline zu verwenden. Die Ablage des Open-API kann entweder durch das Top-down-Vorgehen erfolgen oder dadurch, dass die OpenAPI-Definition zur Compile-Time erzeugt wird.

Die mit Kiota generierten Clients können anschließend als automatisch erzeugte Pakete für .NET, TypeScript oder andere Sprachen bereitgestellt und versioniert über Paket-Feeds wie NuGet, NPM und Co. zentral über Feeds an die jeweiligen Teams verteilt werden.

Scalar

Seit Swagger selbst in die OpenAPI Foundation übergegangen ist, wird mit der Bezeichnung Swagger eigentlich nur noch das Toolset bezeichnet, um mit OpenAPI Contracts zu interagieren; vor allem für die Ansicht der OpenAPI-Dokumente, zum Beispiel eben durch das SwaggerUI oder das Produkt SwaggerEditor.

Die Entwicklung rund um das Swagger-Tooling ist jedoch eingeschlafen: Die Kritik bezüglich fehlender Modernisierungen wurde zunehmend lauter. Es gab einen immer größer werdenden Bedarf im Hinblick auf moderneres, leistungsfähigeres und effizienteres Tooling. Sowohl Open-Source-Projekte wie ReDoc [14], Stoplight [15] und Bruno [16], aber auch kommerzielle Lösungen wie Postman [17] haben versucht sich in dieser Lücke zu platzieren; manche mit mehr, manche mit weniger Erfolg – aber alle mit guten Ideen.

Kein Ansatz jedoch wurde so extrem schnell adaptiert wie Scalar (Bild 6): Scalar [18] ist ein Open-Source-Projekt, genauer gesagt bezeichnet sich Scalar selbst als die Open Source API Platform. Im direkten Vergleich mit dem Swagger-Tooling bietet Scalar eine deutlich modernere, benutzerfreundlichere und kollaborativere Lösung für die Darstellung und Verwaltung von OpenAPI Contracts. Während Swagger in erster Linie ein technisches Werkzeug zur Visualisierung von OpenAPI-Spezifikationen darstellt und primär auf Entwickler ausgerichtet ist, positioniert sich Scalar als vollwertige Plattform für API-Dokumentationen. Scalar wurde dabei maßgeblich von Marc Laventure [19] gegründet, der sich mittlerweile auch im Board der OpenAPI Initiative befindet.

Scalar überzeugt durch ein ansprechendes, responsives UI-Design, das sowohl technische als auch nichttechnische Nutzer anspricht. Scalar bietet eine Vielzahl von Funktionalitäten, die man sich über Jahre vom Swagger-Tooling erhofft hat, darunter eine viel einfachere .NET-Konfiguration, die direkte Generierung von Client-Code im Scalar-UI für eine Vielzahl von Programmiersprachen und ein sehr viel einfacheres Handling des API-Testens im UI, inklusive Cookie-Support, Header-Werten und Co.

Scalar in ASP.NET Core

Die Integration von Scalar in ASP.NET Core erfolgt über das NuGet-Paket *Scalar.AspNetCore* [20]:

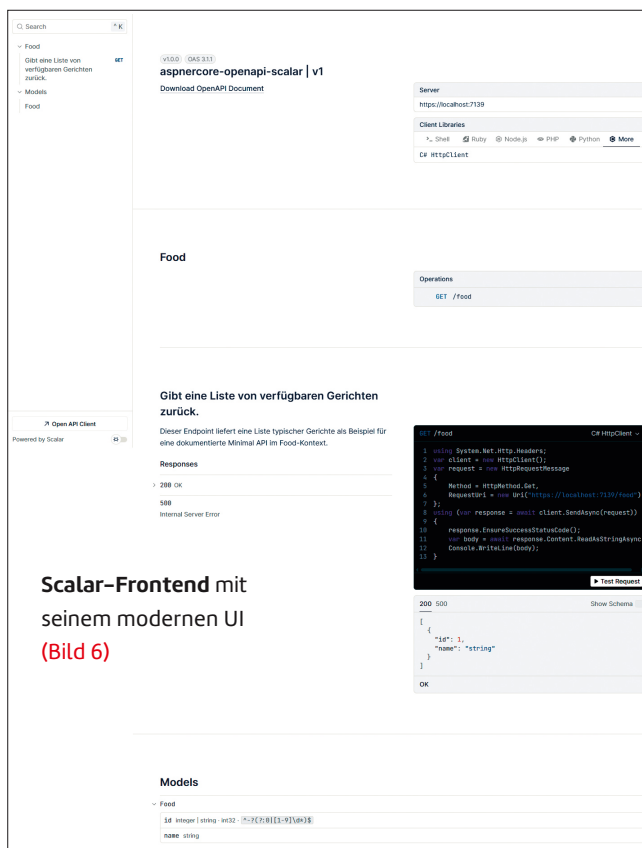
```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net10.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="
      "Microsoft.AspNetCore.OpenApi"
      Version="
        "10.0.0-preview.7.25380.108" />
    <PackageReference Include="Scalar.AspNetCore"
      Version="2.6.9" />
  </ItemGroup>

</Project>
```

Sobald das NuGet-Paket eingebunden, geladen und durch ein *using* bekannt gemacht wurde, wird durch *MapScalarApiReference* die Middleware der ASP.NET-Core-Anwendung hinzugefügt. Wie bei allen Middleware-Registrierungen [21] ist die Reihenfolge wichtig.

```
using Microsoft.AspNetCore.Http.HttpResults;
using Scalar.AspNetCore;
WebApplicationBuilder builder =
    WebApplication.CreateBuilder(args);
// Add OpenAPI
```



Scalar-Frontend mit seinem modernen UI (Bild 6)

```
builder.Services.AddOpenApi();
WebApplication app = builder.Build();
// add openapi
app.MapOpenApi( I* optional config here *I );
app.MapScalarApiReference( I* optional config here *I );
app.UseHttpsRedirection();
```

Scalar unterstützt dabei alle HTTP-basierten API Contracts, sei es Plain-Text-JSON, aber eben auch Standards wie OpenAPI. Für die Generierung der Inhalte greift die Scalar-Middleware auf die bewährten OpenAPI-Funktionalitäten des Pakets Microsoft.AspNetCore.OpenApi zurück, was im Umkehrschluss bedeutet, dass auch erst mit der Version 10 die Verfügbarkeit der XML-Dokumentation in das Scalar-UI kommen wird.

Scalar in .NET Aspire – und für alle anderen Proxies

Aspire [22] ist das absolute Lieblingskind des ASP.NET-Core-Teams – und das zu Recht. Es ist mit das schnellste und erfolgreichste Framework im .NET-Ökosystem und extrem beliebt – auch außerhalb der klassischen .NET-Welt. Klar, dass Scalar auch in dieser Umgebung funktionieren muss.

Prinzipiell ist die Integration von Scalar in eine Aspire-Umgebung nichts anderes wie die in eine normale ASP.NET-Core-Variante – es sind schließlich ASP.NET-Core-Apps. Bei Aspire handelt es sich jedoch um eine Proxy-Umgebung, weshalb Forwarding-Header [23] gesetzt werden müssen, damit die Scalar-Oberfläche den korrekten Endpunkt findet:

```
builder.Services.AddOpenApi();

// configure forwarding headers
builder.Services.Configure<ForwardedHeadersOptions>(<
    options =>
    {
        options.ForwardedHeaders =
            ForwardedHeaders.XForwardedFor |
            ForwardedHeaders.XForwardedProto;
    });
WebApplication app = builder.Build();

// register forwarding headers
app.UseForwardedHeaders();
app.MapOpenApi();
app.MapScalarApiReference( /* optional config here */ );
// scalar/v1
```

Qualitätsverbesserungen durch OpenAPI Linting

Wie bereits angedeutet, lässt die Qualität von automatisch generierten OpenAPI Contracts aus dem Code zu wünschen übrig; häufig fehlen Informationen zum Hostnamen, zur Authentifizierung oder ganz einfach Endpunktebeschreibungen.

Abhilfe schafft hier Spectral (24). Spectral ist ein Linter für OpenAPI und ermöglicht die automatische Prüfung von OpenAPI-Files auf Best Practices, Konsistenz und individuelle Teamregeln. Das Tool lässt sich unkompliziert per Yarn lokal in ►

das Projekt installieren und via CLI nutzen, und es kann problemlos in bestehende Build-Prozesse integriert werden.

Fazit

OpenAPI hat sich als Standard für die Beschreibung und Dokumentation von HTTP-basierten APIs etabliert und ist aus der modernen .NET-Entwicklung nicht mehr wegzudenken: Swagger ist erwachsen geworden.

Sowohl der Top-down- als auch der Bottom-up-Ansatz bieten jeweils Vor- und Nachteile, aber auch Herausforderungen: Während Top-down vor allem für externe oder langfristig gepflegte APIs eine hohe Qualität und Konsistenz sicherstellt, sind die Vorteile von Bottom-up Geschwindigkeit und einfache Integration in bestehende Tools, Anwendungen oder Projekte. Gerade beim Bottom-up-Vorgehen ist es jedoch essenziell, die Qualität der generierten Contracts nicht dem Zufall zu überlassen. Tools wie Spectral und Scalar, moderne Frameworks wie .NET Aspire sowie Best Practices rund um Dokumentation und Proxies helfen dabei, auch automatisch erzeugte OpenAPI Contracts qualitativ hochwertig umzusetzen. Die konsequente Nutzung dieser Werkzeuge und Methoden sorgt dafür, dass APIs nicht nur maschinenlesbar,

sondern auch für Entwickler verständlich, wartbar und sicher bleiben – und der Frust durch schlechte API-Dokumentationen deutlich sinkt und dadurch die Akzeptanz erhöht wird.

Die Entwicklung hochwertiger APIs ist heute kein Hexenwerk mehr – vorausgesetzt, die richtigen Standards, Tools und Prozesse werden von Anfang an berücksichtigt: angefangen bei einfachen Regeln wie der URL-Gestaltung, der Verwendung der korrekten HTTP-Verben und der Adaption von Standards.

Wer OpenAPI, Linting, Proxies und automatisierte Validierung in den Entwicklungsalltag integriert, legt das Fundament für nachhaltigen Erfolg in der API-Entwicklung mit .NET. ■

[1] *dotnet.REST*, <https://dotnet.rest>

[2] *Microsoft Learn, ASP.NET Core web API documentation with Swagger/OpenAPI*, <https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger>

[3] *OpenAPI Initiative*, www.openapis.org

[4] *Martin Fowler, Richardson Maturity Model*, <https://martinfowler.com/articles/richardsonMaturityModel.html>

● Spectral im .NET-Umfeld

Für die sinnvolle Nutzung von Spectral in .NET-Projekten ist zunächst das NuGet-Paket `Microsoft.Extensions.ApiDescription.Server` [25] erforderlich. Dieses Paket sorgt dafür, dass automatisch beim Build statt erst zur Laufzeit ein OpenAPI Contract als `openapi.json` im Projektverzeichnis generiert wird.

In der `.csproj`-Datei sind dazu die folgenden Properties zu ergänzen:

```
<ItemGroup>
  <PackageReference
    Include="Microsoft.AspNetCore.OpenApi"
    Version="10.0.0-preview.7.25380.108" />

  <PackageReference Include="
    Microsoft.Extensions.ApiDescription.Server"
    Version="10.0.0-preview.7.25380.108">
  </PackageReference>
  <PrivateAssets>all</PrivateAssets>
  <IncludeAssets>runtime; build; native;
    contentfiles; analyzers</IncludeAssets>
</ItemGroup>
```

Beispiel-Ausgabe
von spectral zur
OpenAPI-Contract-
Validierung (Bild 7)

```
> spectral lint sample-openapi.yml

c:/source/benabt/samples/aspnetcore-openapi-linter/sample-openapi.yml
2:6 warning info-contact Info object must have "contact" object.
14:9 warning operation-description Operation "description" must be present and non-empty string.
18:11 warning operation-tag-defined Operation tags must be defined in global tags.
30:10 warning operation-description Operation "description" must be present and non-empty string.
34:11 warning operation-tag-defined Operation tags must be defined in global tags.
61:9 warning operation-description Operation "description" must be present and non-empty string.
65:11 warning operation-tag-defined Operation tags must be defined in global tags.
77:9 warning operation-description Operation "description" must be present and non-empty string.
81:11 warning operation-tag-defined Operation tags must be defined in global tags.
101:12 warning operation-description Operation "description" must be present and non-empty string.
105:11 warning operation-tag-defined Operation tags must be defined in global tags.

info
paths./foods.get.tags[0]
paths./foods.get.tags[0]
paths./foods.post.tags[0]
paths./foods/{foodId}.get.tags[0]
paths./foods/{foodId}.put.tags[0]
paths./foods/{foodId}.delete.tags[0]
```

```
</ItemGroup>
<PropertyGroup>
  <OpenApiDocumentsDirectory>$(MSBuildProjectDirectory)
  </OpenApiDocumentsDirectory>
  <OpenApiGenerateDocuments>true
  </OpenApiGenerateDocuments>
</PropertyGroup>
```

Für die Standardregeln genügt eine `.spectral.yml` im Projektverzeichnis mit folgendem Inhalt:

```
# .spectral.yml
extends: ["spectral:oas"]
```

Ein typischer Output könnte wie in **Bild 7** gezeigt aussehen.

Auf diese Weise lässt sich die Qualität der OpenAPI Contracts kontinuierlich und automatisiert sicherstellen. Insbesondere in Teamumgebungen trägt dies maßgeblich zu einer höheren Konsistenz und Wartbarkeit der API-Beschreibungen bei.

- [5] 42Crunch, OpenAPI (Swagger) Editor, <https://marketplace.visualstudio.com/items?itemName=42Crunch.vscode-openapi>
- [6] Redocly OpenAPI, <https://marketplace.visualstudio.com/items?itemName=Redocly.openapi-vs-code>
- [7] NSwag, <https://github.com/RicoSuter/NSwag>
- [8] NuGet-Paket NSwag.AspNetCore, www.nuget.org/packages/NSwag.AspNetCore
- [9] Swashbuckle.AspNetCore auf GitHub, <https://github.com/domaindrivendev/Swashbuckle.AspNetCore>
- [10] NuGet-Paket Swashbuckle.AspNetCore, www.nuget.org/packages/Swashbuckle.AspNetCore
- [11] NuGet-Paket Microsoft.AspNetCore.OpenApi, www.nuget.org/packages/Microsoft.AspNetCore.OpenApi
- [12] Microsoft Learn, Native AOT deployment, <https://learn.microsoft.com/en-us/dotnet/core/deploying/native-aot/>
- [13] Microsoft Learn, Welcome to Kiota, <https://learn.microsoft.com/en-us/openapi/kiota/overview>
- [14] Redoc: Open source API documentation tool, <https://redocly.com/docs/redoc>
- [15] Stoplight, <https://stoplight.io>
- [16] Bruno, www.usebruno.com
- [17] Postman, www.postman.com
- [18] Scalar, <https://scalar.com>
- [19] Marc Laventure auf LinkedIn, www.linkedin.com/in/marc-laventure/
- [20] NuGet-Paket Scalar.AspNetCore, www.nuget.org/packages/Scalar.AspNetCore
- [21] Microsoft Learn, Middleware order, <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-9.0#middleware-order>
- [22] Microsoft Learn, Aspire overview, <https://learn.microsoft.com/en-us/dotnet/aspire/get-started/aspire-overview>
- [23] .NET Aspire: Scalar displays API instance url when accessing via proxy, Issue #4025, <https://github.com/scalar/scalar/discussions/4025>
- [24] Spectral auf GitHub, <https://github.com/stoplightio/spectral>
- [25] NuGet-Paket Microsoft.Extensions.ApiDescription.Server, www.nuget.org/packages/Microsoft.Extensions.ApiDescription.Server



Benjamin Abt

arbeitet als Principal Software Engineer bei der MediaLesson GmbH in Pforzheim, betreibt das C#-Forum myCSharp.de und tritt als Trainer sowie Sprecher auf Konferenzen und User Groups auf. Seit 2015 ist er Microsoft MVP für .NET.
mail@benjamin-abt.com

DWX ACADEMY

Hands-on Workshops
und Weiterbildung für
Software-Entwickler
und -Architekten

ARCHITEKTUR FÜR .NET-ANWENDUNGEN

- Prinzipien für Objektorientierung
- Metriken
- Komponenten
- Inversion of Control
- Multi-Layer-und Multi-Tier-Architektur
- Referenzarchitektur

3 TAGE
17.-19.11.2025



DAVID TIELKE



Deine Ansprechpartnerin: Susanne Herl
+49 731 88005-8835 • susanne.herl@ebnermedia.de

developer-world.de/dwx-academy

AUF IN DIE WELT MIT SIGNALR, RASPBERRY PI UND SECHS RÄDERN

SignalRC – in Echtzeit ans Steuer

Ein vernetztes Fahrzeug, gesteuert per Weboberfläche und LTE. SignalR sorgt dabei für millisekundenschnelle Kommunikation – direkt, stabil und skalierbar.

Wie beginnt man eine Reise? Normalerweise am Bahnsteig oder Flughafenterminal. Das machen wir heute nicht. Heute beginnen wir am Linux-Terminal. Statt mit dem Rucksack durch fremde Länder zu ziehen, entdecken wir mit unserem Fahrzeug, das sich per LTE von überall aus fernsteuern lässt, inklusive Livebild und Statusdaten, die Welt.

Der Maßstab mag klein sein, doch die Herausforderung ist groß. Wie klein? Uns reicht ein Maßstab 1:10, erst einmal. Die Idee dahinter ist simpel: Die Welt erkunden, ohne vor die Tür zu gehen.

Das klingt nach Spielzeug? Streng genommen ist das schon richtig. Doch so spielerisch das Ziel auch wirkt, die Anforderungen sind interessanter und herausfordernder, als man im ersten Moment denken mag. Das System muss in Echtzeit reagieren, zuverlässig auch über Mobilfunknetze funktionieren und dabei möglichst wenig Energie verbrauchen. Die Software soll auf einem Raspberry Pi laufen, mit möglichst wenig Overhead. Die Architektur basiert auf .NET, und im Zentrum steht SignalR. Warum ausgerechnet SignalR? Ist das nicht etwas für Webanwendungen?

In klassischen Webanwendungen zählt man selten die einzelnen Millisekunden. Doch bei einem ferngesteuerten Fahrzeug kommt es auf jede Verzögerung an, sei sie noch so klein. SignalR ermöglicht bidirektionale Kommunikation mit Web-Socket-Unterstützung, Fallback-Strategien und automatischer Reconnect-Logik und ist damit ideal für alle mobilen Szenarien. Außerdem ist SignalR der Inbegriff für eine stabile, bidirektionale Kommunikation zwischen zwei Anwendungen

im .NET-Universum. Also genau das, was wir brauchen, um ein Auto fernzusteuern.

Herausforderungen

Wer schon mal in Deutschland LTE benutzt hat, weiß, dass stabile Realtime-Kommunikation mit minimaler Latenz auf mobiler Hardware keine Selbstverständlichkeit ist. Weiterhin darf nicht vergessen werden, dass allein die Aufzeichnung und Umwandlung des Livestreams vom Fahrzeug den größten Teil der Kapazität eines Raspberry Pi und des Netzwerks auffrisst. Daher soll die Anwendung für die Steuerung sämtlicher Fahrzeugfunktionen möglichst ressourcensparend arbeiten. Ressourcensparen ist besonders wichtig im Batteriebetrieb, denn ein System, das den ganzen Tag im Feld stehen soll, darf den Akku nicht in zwei Stunden leer saugen. Was macht man nicht alles, um eine kleine digitale Mini-Weltreise möglich zu machen.

Aufbau

Auf einer echten Reise begegnet man sich nicht immer zur gleichen Zeit am gleichen Ort. Dieses Prinzip gilt auch für die Architektur unseres Projekts. Fahrzeug und Client sind voneinander entkoppelt, sie verbinden sich unabhängig voneinander mit dem Server. Das Fahrzeug stellt regelmäßige Verbindung zum Server her, und das möglichst dauerhaft, sofern die LTE-Verbindung stabil bleibt. Es meldet sich beim SignalR-Hub an, überträgt seine ID und hält einen Kanal für eingehende Befehle offen. Sensorwerte, Akkustand oder Statusänderungen sendet es aktiv über denselben Kanal zurück.

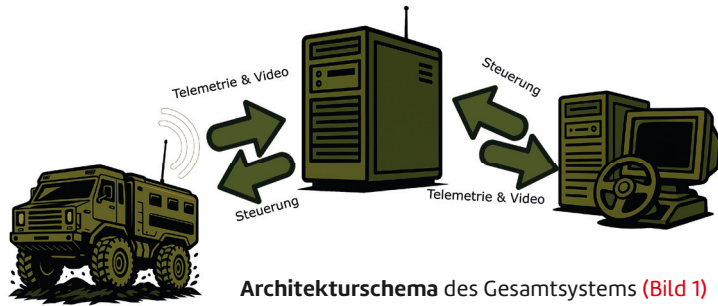
Auf der anderen Seite verbindet sich der Client, eine Webanwendung, die ebenfalls per SignalR mit dem Server verbunden ist. Auch hier gilt: Es gibt keinen festen Zeitpunkt und keinen Zwang zur Gleichzeitigkeit, der Benutzer kann sich jederzeit einloggen. Man öffnet die Steueroberfläche im Browser und bekommt sofort eine Übersicht über die verfügbaren Fahrzeuge, angefangen mit denen, zu denen der Client bereits eine Verbindung hatte. Der Server fungiert als neutraler Treffpunkt. Er verwaltet alle aktiven Verbindungen und hält sie vor.

Sobald beide Seiten verbunden sind, kann die Kommunikation beginnen. Der Client hat volle Kontrolle über das Fahrzeug, sofern er sich am Fahrzeug angemeldet hat und von diesem eine Session zurückbekommen hat. Aber selbst wenn

● Du möchtest gerne mehr erfahren?

Das freut uns, denn ab Erscheinen des vorliegenden Artikels veröffentlichen wir auf developer-world.de/insights weitere Beiträge, die den Artikel fortsetzen und ausschließlich online zur Verfügung stehen. Derzeit geplant sind folgende Beiträge:

- SignalR Webserver auf Azure
- Reactflow – Komponenten miteinander verbinden als Nodes
- Kontrolle über PI GPIO
- JANUS Server (Videostreaming/Videotelefonie)
- Warum SQLite als Datentransportformat einfach grandios ist



Architekturschema des Gesamtsystems (Bild 1)

das Fahrzeug gerade offline ist, bleibt die Steueroberfläche funktional: Der Server zeigt den letzten bekannten Zustand, informiert über Verbindungsabbrüche und bietet eine klare Statusanzeige, auf die wir später genauer eingehen werden. Um den Videostream zu übertragen, kommt Janus zum Einsatz. Das ist ein minimalistischer WebRTC-Server, der für diesen Artikel aber keine Rolle spielt. **Bild 1** verdeutlicht das Schema der Architektur.

Abfahrt

Dieses Modell erlaubt maximale Flexibilität und lehnt sich an das Look and Feel eines Open-World-Spiels an. Wir starten in der Lobby: Die Fahrzeuge stehen bereit, laufen im Hintergrund und senden ihre Daten. Der Benutzer kann jederzeit „einsteigen“. Es ist wie bei GTA: Das Fahrzeug steht da, die Tür ist offen, der Motor läuft. Der einzige Unterschied: Ohne Schlüssel passiert nichts. Dieser Schlüssel ist die aktive Verbindung über SignalR und eine Session. Wer sich einloggt, bekommt Zugriff auf Steuerung und Feedback. Die Kontrolle wechselt in Echtzeit, ohne Neustart, ohne manuelle Koppelung. Alle anderen können die Fahrt im Spectator-Modus verfolgen, wenn sie möchten.

Kontrolle wird aber immer nur von einem Benutzer ausgeübt, und der kann auch nur durch Time-out oder Log-out abgelöst werden. Clients steigen bei Bedarf ein, übernehmen die Steuerung und steigen auch wieder aus. Der Server sorgt dafür, dass dabei niemand dem anderen in Lenkrad greifen kann und jedes Kommando genau dort ankommt, wo es hinsoll. Einfach den Schlüssel, der im Fahrzeug konfiguriert ist, eingeben, und die Straße gehört dir. Wer komplexere Fahrzeuge hat, die mehrere Fahrer benötigen, kann den zweiten Platz als weitere Instanz laufen lassen, die sich dann als weiteres Fahrzeug am Server anmeldet. Nicht der häufigste Use Case, aber vielleicht lohnend zu erwähnen.

Der Server: Schaltzentrale mit SignalR

Im Zentrum des Systems steht der Server, ein ASP.NET Core Web API, das alle Kommunikationsströme zwischen Fahrzeugen und Steuer-Clients koordiniert. SignalR übernimmt dabei die Rolle der Transportebene: leichtgewichtig, bidirektional und echtzeitfähig. Die gesamte bisher beschriebene Logik basiert auf einem dedizierten Hub namens CarControlHub.

Ganz von vorne – Initialisierung von SignalR

Bereits beim Start der Serveranwendung wird SignalR in der *Program.cs* oder *Startup.cs* eingebunden. Die Konfiguration ist minimal, aber entscheidend; sie findet sich in **Listing 1**.

Der Hub wird im Routing definiert und unter einem festen Pfad verfügbar gemacht. Damit ist die Kommunikationsschnittstelle offen, sowohl für Fahrzeuge als auch für Webclients. In unserem Projekt gibt es natürlich mehr Hubs, die allerdings im Listing der Einfachheit halber fehlen.

Für mich hat es sich bewährt, die Pfade in Konstanten auszulagern. Bei der Konstellation unserer beteiligten Programme ist das Auslagern besonders lohnend, denn der Server und die Onboard-Software teilen sich eine Bibliothek. Auf diese Weise kann der Compiler schon mal checken, dass die Pfade passen. Damit werden keine Verbindungen versucht, die schon allein vom Mapping her nicht funktionieren können.

Der ControlHub als Vermittler

Der *CarControlHub* ist der zentrale Vermittler. Er verwaltet grundlegende Funktionen wie die Registrierung eines Fahrzeugs, das Anfordern einer Steuersitzung, die Freigabe dieser Kontrolle sowie die Kanalübertragung in Echtzeit. Eine gekürzte Version des Hubs findet sich in **Listing 2**. Zu beachten sind die beiden Interfaces. Der generische Parameter *ICarControlClient* gibt an, welche Funktionen der Server beim Client aufrufen kann. Andersherum wird das Interface *ICarControlServer* vom Hub implementiert. Das stellt sicher, dass der Client die entsprechenden Funktionen des Hubs aufrufen kann. Dazu gleich mehr; schauen wir uns zunächst den Code von *CarControlHub* an.

Das Herz des Fahrzeugs: der Onboard-Client

Sobald das Fahrzeug startet, nimmt es Kontakt mit dem Server auf. Es meldet sich an, stellt seine verfügbaren Steuerkanäle bereit und wartet auf Kommandos. Die Verbindung erfolgt ebenfalls über SignalR, also derselben Technologie wie auf der Serverseite, was Sinn ergibt, denn schließlich müssen sich beide verstehen. Auf dem Onboard-Teil geschieht das allerdings in Form eines Hub-Clients, der aktiv einen WebSocket-Kanal zum Server aufbaut.

Die Initialisierung beginnt mit dem Aufbau der Verbindung. Dazu wird in der Klasse *ServerConnectionService* ein *HubConnection*-Objekt erzeugt. Der Ziel-URL wird dynamisch auf Basis der Konfiguration generiert, einschließlich Hostname, Port und Protokoll. Die Verbindung nutzt eine automatische Wiederverbindung mit exponentiellem Backoff, um auch bei instabiler LTE-Verbindung zuverlässig online zu bleiben. Wichtig hierbei: Ist der Server bereits eine Weile ►

● Listing 1: SignalR initialisieren in Program.cs

```
builder.Services.AddSignalR();
// Alle anderen Services
app.MapHub<CarConnectionHub>(
    HubPaths.CarConnectionHub);
app.MapHub<CarControlHub>(HubPaths.CarControlHub);
app.MapHub<TelemetryHub>(HubPaths.TelemetryHub);
// Alle anderen Hubs
```

● Listing 2: Zentraler Verteiler

```
public interface ICarControlClient
{
    Task<string> AquireCarControl(string carSecret);
    Task ReleaseCarControl(string sessionId);
    Task UpdateChannel(string sessionId,
        string channelId, decimal value);
}

public interface ICarControlServer
{
    Task<string?> AquireCarControl(string carId,
        string carSecret);
    Task ReleaseCarControl(string carId,
        string sessionId);
    Task UpdateChannel(string carId, string sessionId,
        string channelId, decimal value);
    Task RegisterForControl(string carId);
}

public class CarControlHub : Hub<ICarControlClient>,
    ICarControlServer
{
    public async Task RegisterForControl(string carId)
    {
        // Fahrzeug meldet sich an
    }

    public async Task<string?> AquireCarControl(
        string carId, string? carSecret)
    {
        // Steuersitzung anfordern
        return await Clients.Client(
            carClientId).AquireCarControl(carSecret);
    }

    public async Task ReleaseCarControl(
        string carId, string sessionId)
    {
        // Steuersitzung beenden
        await Clients.Client(
            carClientId).ReleaseCarControl(sessionId);
    }

    public async Task UpdateChannel(string carId, string
        sessionId, string channelId, decimal value)
    {
        // Steuerdaten senden
        await Clients.Client(carClientId).UpdateChannel(
            sessionId, channelId, value);
    }
}
```

nicht erreichbar, nehmen die Verbindungsversuche ab. Das spart ein wenig Strom. Nach erfolgreichem Verbindungsaufbau ruft das Fahrzeug aktiv die Methode *OpenCarConnection* auf. Diese Übergabe enthält neben der Fahrzeug-ID auch eine Prüfsumme der lokalen Kanaldefinition, quasi ein Fingerabdruck der Steuerkanäle im Fahrzeug. Den Verbindungsaufbau und die Anmeldung zeigt Listing 3.

TypedSignalR: Starke Typisierung, weniger Fehler

Für die Kommunikation zwischen Server und Client wird TypedSignalR verwendet. Das ist eine Erweiterung, die stark typisierte Schnittstellen zwischen beiden Seiten ermöglicht. Statt auf lose *InvokeAsync("MethodName", ...)*-Aufrufe zu setzen, werden feste Interfaces definiert, wie in Listing 2 zu sehen ist. Dadurch entstehen auf beiden Seiten Klarheit und Typsicherheit über die erlaubten Methoden.

Zugleich greift dadurch das Fail-Fast-Prinzip: Fehler in der Schnittstellendefinition fallen bereits dem Compiler auf und nicht erst zur Laufzeit. Außerdem profitieren Entwickler von vollständigem IntelliSense-Support. Die Methoden, Parameter und Rückgabewerte sind sofort sichtbar und dokumentierbar. Das beschleunigt die Entwicklung und reduziert typische Fehlerquellen wie Tippfehler oder falsche Argumente.

Der Client erzeugt zur Laufzeit einen typisierten Proxy, der entsprechend alle Funktionen des Interfaces enthält, das der

Hub implementiert. Listing 3 zeigt, wie einfach eine Server-Instanz aus der HubConnection erzeugt werden kann.

Das Interface *ICarConnectionServer* definiert exakt, welche Methoden das Fahrzeug beim Server aufrufen darf. Der Server kennt im Gegenzug den Client mit dem entsprechenden Client-Interface *ICarConnectionClient*, um Kommandos an das Fahrzeug zu senden. Der Client registriert sich aktiv als solcher beim Hub, indem er die Service-Instanz an die Server-Events bindet. So entsteht eine klar definierte, typisierte Zwei-Wege-Kommunikation, die sicher und wartbar ist. Die Interfaces werden in einer eigenen *Shared*-Bibliothek gehalten, sodass abweichende Definitionen nicht passieren können. Der Compiler klärt auch hier direkt über Fehler auf.

Channel-Abgleich: Kontrolle mit Vertrag

Beim ersten Verbindungsaufbau wird die Kanalstruktur des Fahrzeugs in Form eines Hashwerts an den Server übergeben. Der Server prüft, ob dieser mit dem zuletzt bekannten Kanalplan übereinstimmt. Falls nicht, fordert er eine Aktualisierung an.

Damit ist sichergestellt, dass der Server stets über die aktuelle Steuerstruktur des Fahrzeugs informiert ist, ohne unnötig große Datenmengen zu übertragen. Änderungen an der Kanalstruktur, etwa durch neue Aktoren oder Sensoren, lassen sich so leicht erfassen und synchronisieren.

Anschließend übergibt das Fahrzeug seine Konfigurationsstruktur, die der Server speichert und für die Konfiguration verwendet. Diese enthält die Steuer- und Telemetrieinformationen, also welche Kanäle das Fahrzeug annehmen und welche Werte der Fahrer abonnieren kann. Sofern ein Kanal abonniert wird, sendet das Fahrzeug alle Updates an diesen Kanal. Sowohl Telemetrie als auch Kanäle für sämtliche Aktoren – also Lenkservos, Fahrtregler, Beleuchtung und vieles mehr – können zentral konfiguriert werden. Die Konfiguration übersetzt sich dann zur Laufzeit in Instanzen, die wie kleine Unterprogramme Informationen lesen oder Aktionen ausführen können. All dies erfolgt über eine einfache Schnittstelle, bestehend aus dem Namen des jeweiligen Aktors sowie seinem neuen Wert. Alle übrigen Informationen wurden zu Beginn bereits ausgehandelt und müssen nicht mehr übertragen werden.

Zellenwechsel

Damit die Steuerung des Fahrzeugs auch bei temporären LTE-Ausfällen oder IP-Adress-Wechseln zuverlässig funktioniert, implementiert der *ControlService* eine robuste Wiederverbindungs-Logik mit TypedSignalR. Sobald ein Nutzer mittels *AquireCarControl* die Steuerung des Fahrzeugs übernimmt, wird auf Basis von *ShortGuid* eine eindeutige Session-ID generiert. Sie ist kompakt, aber zufällig, denn auch hier gilt es Datensparsamkeit zu beachten. Diese Session-ID dient als Schlüssel zur Authentifizierung bei jeder Steueraktion. Wird die Verbindung zwischenzeitlich getrennt, bleibt die Session weiter bestehen, solange kein Time-out oder aktives

Listing 3: Als wäre der Hub eine lokale Klasse

```
_connection = new HubConnectionBuilder()
    .WithUrl(connectionHubEndpoint)
    .WithAutomaticReconnect(Enumerable.Range(0, 50)
        .Select(e => TimeSpan.FromMilliseconds(
            Math.Pow(1.25d, e) * 1000))
        .ToArray())
    .AddMessagePackProtocol()
    .Build();
// Hub Client Callbacks auf aktuelle Instanz mappen
_connection.Register<ICarControlClient>(this);
await _connection.StartAsync();
// Instanz für Hub-Aufrufe erstellen
_server =
    _connection.CreateHubProxy<ICarControlServer>();
// Am Hub anmelden, damit er weiß welches Fahrzeug
// online gekommen ist
var carId = Configuration.GetValue<string>("carId");
await _server.RegisterForControl(carId);
var channelMapHash =
    ChannelMapHashProvider.GenerateHash(_channelMap);
var config = await connectionServer.OpenCarConnection(
    carId, channelMapHash);
```

Listing 4: Einen Tick bereitstellen

```
await Task.Run(async () =>
{
    var telemetryService = serviceProvider
        .GetRequiredService<TelemetryService>();
    while (true)
    {
        await Task.WhenAll(telemetryService.Tick(),
            Task.Delay(100));
    }
});
```

ReleaseCarControl erfolgt. Ein weiteres zentrales Feature ist die Implementierung von *IHubConnectionObserver*, einer Schnittstelle aus TypedSignalR. Sie stellt drei Events bereit:

- *OnReconnecting(Exception?)*: Wird ausgelöst, wenn die Verbindung verloren geht. In diesem Moment werden alle Kanäle des Fahrzeugs auf *default* zurückgestellt (*Control.ReleaseControl()*), um nicht den letzten Status vor dem Verbindungsabbruch zu behalten, denn das könnte im Zweifel geradeaus Vollgas sein. Auf diese Weise kommt das Auto zumindest zum Stehen.
- *OnReconnected(string?)*: Wird nach erfolgreichem Reconnect aufgerufen. Der Client meldet sich mit *RegisterForControl* automatisch wieder beim Server an, sodass der Server die Fahrzeugverbindung erneut zuordnen kann. Gleichzeitig wird der Telemetrie-Service informiert, dass die Verbindung wieder aktiv ist.
- *OnClosed(Exception?)*: Signalisiert, dass die Verbindung endgültig getrennt wurde, zum Beispiel nach einem längeren Ausfall. Auch hier wird die Steuerung zurückgesetzt und die Trennung dokumentiert.

Telemetrie

Neben der Steuerung muss das Fahrzeug auch Rückmeldung über seinen Zustand beziehungsweise den seiner einzelnen Komponenten geben. Die Summe dieser Rückmeldungen bildet die Telemetrie, also Messwerte, die regelmäßig oder bei Bedarf an den Server gesendet werden. Jede Telemetriequelle ist als eigene Reader-Klasse implementiert, die von *TelemetryReaderBase* erbt und über einen Ticker regelmäßig ausgeführt wird. Der *TelemetryService* koordiniert alle aktiven Reader, verwaltet deren Intervalle und sendet die Messwerte über SignalR an den Server. Der Ablauf folgt einem klar festgelegten Schema: lesen, loggen, übertragen.

Die Kommunikation läuft dabei über einen separaten SignalR-Hub (*TelemetryHub*), zu dem das Fahrzeug beim Start seine eigene Verbindung aufbaut. Das bedeutet zugegebenermaßen ein wenig Mehrarbeit, ist das (kleine) Opfer für die „Separation of Concerns“ aber wert. Die Telemetriedaten selbst sind auch in der *ChannelMap* definiert und lassen sich dynamisch aktivieren und deaktivieren. Der Server kann dazu gezielt Kanäle abonnieren oder abwählen, und der ►

Onboard-Client erstellt dafür passende Reader-Instanzen mithilfe des Service Providers. Innerhalb der Anwendung läuft eine Service Loop, die den Telemetrie-Readern einen sogenannten Tick zukommen lässt. Diese Loop zeigt [Listing 4](#).

Der Einfachheit halber – und um das System nicht mit Telemetriedaten zu fluten – soll immer mindestens 100 ms zwischen den Ticks gewartet werden, denn kaum eine Telemetrie ist so zeitkritisch, dass sie das Risiko wert ist, die Video- oder Kontrollverbindung zu verstopfen. Einige Beispiele:

- Batterieladestand ändert sich um 1% alle ~1–10 Minuten (600 Ticks)
- GPS-Position 1-mal pro Sekunde (10 Ticks)
- LTE-Empfang ändert sich alle 50 m, also alle ~5 Sekunden (50 Ticks)
- Ausnahmen: Distanzmesser; aktueller Stromverbrauch; Gyroskop, wenn vorhanden

Außerdem werden die Werte nur dann gesendet, wenn sie sich auch innerhalb ihrer festgelegten Genauigkeit geändert haben.

MessagePack: straff gepackte Pakete

Wir haben gesehen, was alles übertragen werden soll. Kommen wir nun zum „Wie“. Für die Serialisierung der Kommunikationsdaten zwischen Fahrzeug und Server wird MessagePack verwendet. Dabei handelt es sich um ein binäres Format zur Darstellung strukturierter Objekte. Man kann es gut mit JSON vergleichen. Es ist nur deutlich kompakter und schneller. Anders als bei textbasierten Formaten wird jedes Feld typisiert und direkt binär codiert, was nicht nur die Übertragungsgröße, sondern auch die CPU-Last reduziert. Gerade bei Echtzeitkommunikation über LTE, wo Latenz und Paketgröße entscheidend sind, spielt MessagePack seine Stärken voll aus. Weiterhin lässt sich MessagePack besonders elegant in .NET einsetzen. Die Bibliothek ist tief in das ASP.NET-Ökosystem integriert, sodass einfache POJO-Klassen ohne weitere Konfiguration serialisiert werden können.

Noch mehr straffen

MessagePack punktet in diesem Projekt besonders, weil massenhaft *float*-Werte gesendet werden. Hier gewinnt MessagePack bereits nach zwei Nachkommastellen. Um jedoch maximale Effizienz zu erreichen, kann auf die Attribute *MessagePackObjectAttribute* und *KeyAttribute* zurückgegriffen werden. Letzteres erlaubt den Einsatz numerischer Schlüssel anstelle von Property-Namen. Das spart weiteres Übertragungsvolumen und sorgt für ein deterministisches Verhalten beim Serialisieren. Auch wenn das zwischen zwei .NET-Anwendungen durch die Typisierung nicht zwingend erforderlich ist, ist es genauso unnötig, den Property-Namen in jeder Nachricht noch einmal zu senden. Ist man also an dem Punkt angekommen, an dem man über MessagePack nachdenkt, kann man auch diese Attribute verwenden. Diese machen zwar den Code länger, dafür aber die gesendeten Nachrichten deutlich kürzer. Ich behaupte, dass es keinen Fall mit einem .NET-Client gibt, in dem man die Attribute weglassen sollte. Denn offenbar überwiegt das Interesse an Übertra-

● Listing 5: Kompakte Struktur mit MessagePackObject

```
[MessagePackObject]
public class ChannelUpdate
{
    [Key(0)]
    public string ChannelId { get; set; }

    [Key(1)]
    public float Value { get; set; }
}
```

gungsoptimierung schon bei der Entscheidung für MessagePack. [Listing 5](#) zeigt ein Beispiel für eine stark optimierte Struktur und [Listing 6](#) den Unterschied in der Darstellung.

Der Vergleich aus [Listing 6](#) klingt zunächst nach einem geringen Unterschied. Hält man sich allerdings vor Augen, dass etwa 15 bis 20 Nachrichten pro Sekunde über den Stream gehen, und das auch noch via LTE, summiert sich das zu einem erheblichen Wert. Nun mag man einwenden: „Neben dem Video macht das doch aber keinen wirklichen Unterschied!“ Doch! Den macht es. Es geht um eine Differenz von 17 MByte pro Stunde allein für die Steuerbefehle, unter Berücksichtigung nur eines Channels mit lediglich zehn Zeilen Code.

Weiterhin ist das Beispiel das mit Abstand kleinste Objekt, das in dem Projekt gesendet wird. Bei den Datenvolumenkosten in Deutschland ist die Einsparung schon eine Überlegung wert, und gleichzeitig veranschaulicht sie, wie effizient MessagePack wirken kann.

Stärker spürbar wird der Unterschied beispielsweise bei der Antwort für den Kanalabgleich an den Server. Ich erspare euch die Seite, die der JSON-MessagePack-Vergleich ausgeschrieben umfassen würde. Der Unterschied beim Abgleich beträgt 761 zu 417 Bytes, also nur etwas mehr als die Hälfte oder 54,79632063 Prozent, um genau zu sein.

● Listing 6: Vergleich in einem kleinen Modell

```
//JSON
{"ChannelId":"steering","Value":0.850000}

// MessagePack
82 // Map mit 2 Einträgen
00 // Key 0 (ChannelId)
A8 73 74 65 65 72 69 6E 67 // "steering" (8 Bytes
    String)
01 // Key 1 (Value)
CA 3F EB 33 33 // float32 für 0.850000 (5 Bytes
    inklusive Typ-Byte)

// 42 vs 17 bytes
```

Grandios! Das Ende für JSON?

Auch in TypeScript ist die Verwendung von MessagePack grundsätzlich möglich, bringt jedoch einige Einschränkungen mit sich. Die JavaScript-Runtime kennt keine echten Typinformationen, weshalb die Struktur der Daten strikt manuell eingehalten werden muss. Fehler bei der Reihenfolge oder Typen der Felder führen dort erst zur Laufzeit zu Problemen, was die Fehlersuche entsprechend erschwert. Zudem ist MessagePack nicht wirklich menschenlesbar, außer natürlich für all diejenigen, die mit dem Hex-Editor aufgewachsen sind. Dieser Mangel erschwert das Debugging und Logging ungemein. Tools oder Hilfsfunktionen zur Deserialisierung sind daher notwendig, aber auch verfügbar. Trotzdem überwiegen für dieses Projekt die Vorteile. Der reduzierte Overhead, die besonders kleinen Transportvolumina und die exzellente .NET-Integration machen MessagePack zur idealen Wahl, insbesondere für mobile, latenzkritische Anwendungen wie unser ferngesteuertes Fahrzeug.

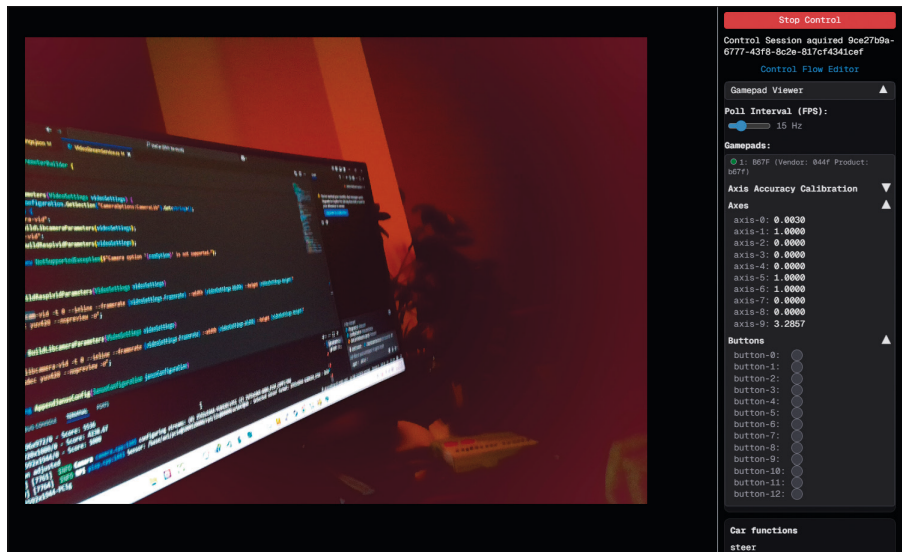
Weitere Optimierungen

Das gesamte Protokoll lässt sich allerdings auch noch weiter straffen, wenn man komplett auf die „Magic Strings“ verzichtet, denn im Grundsatz kennen sowohl der Server als auch das Fahrzeug die Namen der Kanäle. Das bedeutet, dass für eine weitere Verkleinerung der Pakete lediglich eine Antwort auf den Kanalabgleich nötig wäre, damit auch das Auto die IDs kennt, die der Server für die Kanäle nutzt. Der Haken daran: Das Nachverfolgen der Nachrichten auf dem Transportweg ist dann aber so gut wie gar nicht mehr möglich.

Cockpit und Maschinenraum

Wie bereits erwähnt, wäre die viele Kommunikation ohne eine Oberfläche, die der Fahrer nutzen kann, relativ witzlos. Daher gibt es ein UI mit lediglich zwei Ansichten – dem Cockpit und dem Maschinenraum. Das Cockpit ist die Ansicht, die während der Fahrt genutzt wird. Sie zeigt den Videostream, die Eingaben vom Lenkrad beziehungsweise Controller sowie die Telemetriedaten an. Weiterhin findet hier die Anmeldung am Fahrzeug statt. Alle Daten werden unverzüglich aktualisiert und anhand der Konfiguration im Maschinenraum an das ausgewählte Fahrzeug übertragen (siehe Bild 2).

Der Maschinenraum bietet die Option, die Kanäle des Lenkrads und die des Fahrzeugs zusammenzubringen. Dabei ermöglicht er es nicht nur, Key Bindings zu erstellen, sondern erlaubt es über einen Flow auch, das Verhalten der Inputs anzupassen. So ist der Server darauf vorbereitet, unterschiedliche Fahrzeuge zu unterstützen, die vor Funktionen nur so strotzen: Von einem simplen Remote-Auto mit lediglich Gas und Lenkung über Remote-Lkw mit zwei Achsen, zwei Blinkern und



Die Ansicht des Cockpits (Bild 2)

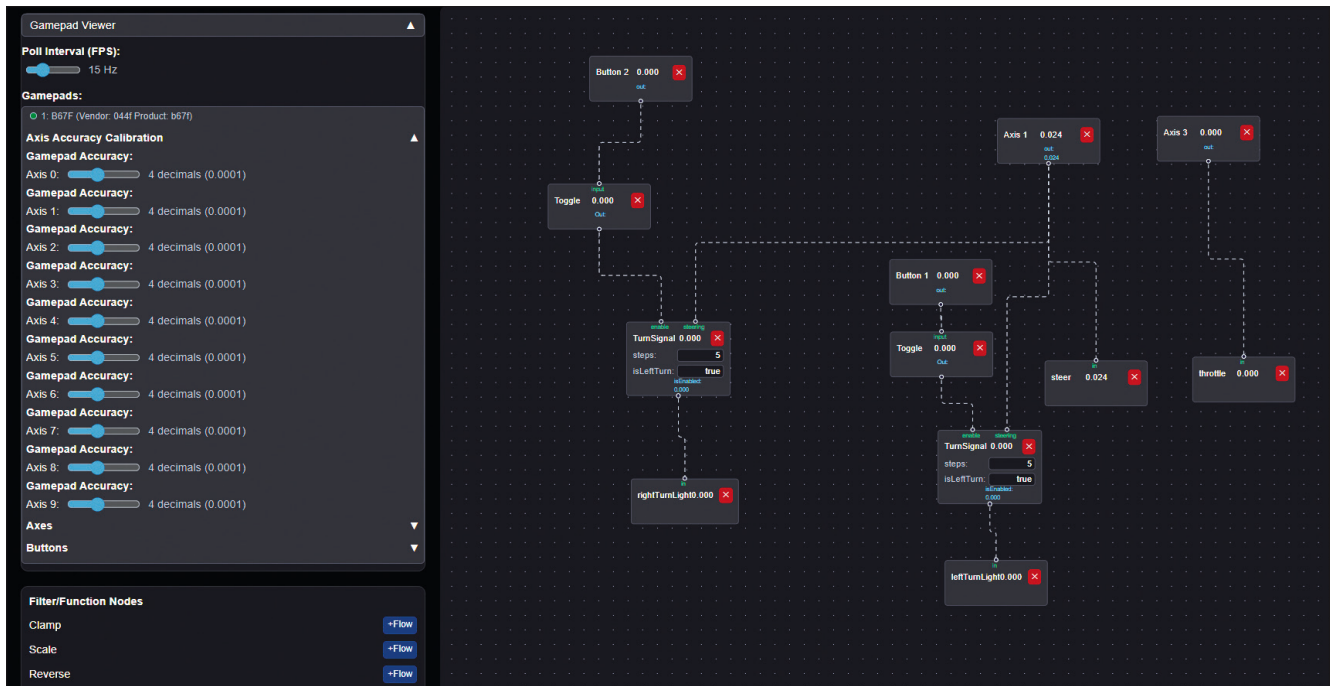
Abblendlicht bis hin zu einem Remote-Bagger mit zwei Ketten, zwei Armteilen, dreh- und schwenkbarer Kamera, einer Drehachse, ausfahrbaren Stützen und Beleuchtung an allen beweglichen Teilen. Kein Problem! Wenn die Buttons am Lenkrad nicht reichen, kann auch noch die Tastatur herhalten.

Die Kanäle sind dabei auf Werte aus dem *float*-Wertebereich beschränkt, was von einer normalen RC-Hardware-Fernbedienung abgekupfert ist. Demnach müssen Kanäle die 2D-Daten liefern und entsprechend auch zwei Kanäle aufmachen. Das gilt sowohl für die Telemetrie- als auch für die Steuerkanäle.

Bild 3 zeigt, wie die Konfiguration vonstattengeht. Hier findet sich das UI, das es uns erlaubt, mit React Flow eine komplexe Steuerung zu erstellen. Alle Flows werden dabei direkt im Browser ausgeführt, um auch hier wieder Daten zu sparen und den Raspberry Pi zu entlasten. Alle Inputs – auch die, die als Telemetriewerte angekommen sind – stehen damit dem Benutzer zur Verfügung. So kann man über einige Funktionsblöcke einfach die Eingaben anpassen, um beispielsweise die Steuerung zu kalibrieren oder zu invertieren. Ein Flow startet dabei immer mit einem Input. Ändert sich ein Inputwert eines Blocks, wird er entsprechend neu berechnet. Dieser Flow setzt sich über alle Verbindungen fort, bis entweder ein Output-Node kommt, der dann seinen Wert direkt an das Fahrzeug sendet, oder sich der Ergebniswert nicht mehr ändert. Ein Output kann dabei mit mehreren Inputs verbunden werden. Die Verzweigungen können hier beliebig groß werden.

Zusammengesteckt

Erst durch das Zusammenspiel aller Komponenten entsteht aus dem Projekt ein vollständiges, fernsteuerbares System. Der Server bildet mit SignalR das Rückgrat der Kommunikation, nimmt Verbindungen entgegen, koordiniert Steuerbefehle und verwaltet die Fahrzeug-Clients. Der *ControlHub* vermittelt dazu gezielt, stabil und latenzarm zwischen Fahrer und Fahrzeug. Der Client auf dem Fahrzeug verwaltet über den *ServerConnectionService* die Verbindung, regelt den ►



Blick in den Maschinenraum (Bild 3)

Kanalabgleich und setzt Steuerkommandos, die sofort vom Fahrzeug ausgeführt werden, direkt ab. Gleichzeitig liefert der *TelemetryService* alle wichtigen Statusdaten zurück. Durch die Verwendung typisierter Schnittstellen und die effiziente Datenübertragung entsteht eine Architektur, die sowohl robust als auch leichtgewichtig ist. Sie erlaubt es, mehrere Fahrzeuge parallel zu betreiben, Zustände live zu überwachen, Steuerverbindungen dynamisch zuzuweisen und selbst bei schwankender Netzqualität reaktiv zu bleiben.

Damit ist der Weg frei für das eigentliche Ziel: Ein Fahrzeug zu bauen, das sich von überall auf der Welt präzise steuern lässt, mit Echtzeitverhalten, Sichtkontakt per Video und voller Kontrolle über alles, was unter der Haube passiert.

Kein Auto-Fan? Realtime geht auch ohne Räder!

Wer mit Autos nicht so viel anfangen kann, muss das System trotzdem nicht links liegen lassen. Denn die Architektur, die hier für ein ferngesteuertes Fahrzeug entwickelt wurde, lässt sich eins zu eins auf andere Szenarien übertragen. Die Trennung von Steuerung, Rückmeldung und zentraler Koordination ist ebenso universell einsetzbar wie die Anforderungen an Echtzeitkommunikation, Zuverlässigkeit und Skalierbarkeit. Gleiches gilt für die einzelnen Softwarebausteine: Ob es sich um Smart-Home-Komponenten, Industrieroboter, Drohnen (Achtung: Latenzanforderung noch höher und niemals ohne Spotter, da sonst Ärger droht!), Fernwartung von Maschinen oder interaktive Dashboards handelt, spielt keine Rolle. Denn sobald ein Gerät auf Kommandos reagieren und gleichzeitig Statusdaten liefern soll, kann diese Architektur eine Lösung bieten.

SignalR übernimmt dabei weitestgehend die Rolle des Echtzeitkanals, TypedSignalR bringt Ordnung und Typensicherheit hinein und MessagePack sorgt dafür, dass selbst auf

schmaler Leitung schnelle Kommunikation möglich ist. Was bleibt, ist das Grundmuster:

- Eine klar definierte Steuerverbindung,
- eine asynchrone Telemetrie-Schnittstelle,
- ein leichtgewichtiges, loses Kopplungskonzept.

Das Projekt mag mit einem kleinen Fahrzeug begonnen haben, aber es endet erst, wo euch eure Vorstellungskraft verlässt. Wer die Bausteine kennt, kann damit alles bewegen.

Schon Schluss? Und nun?

Wer mit mir um die Welt fahren möchte, fühle sich dazu eingeladen, das Repository unter [1] zu klonen und dabei zu sein. Contributions sind jederzeit willkommen, denn zusammen zu entwickeln macht immer noch mehr Spaß als allein. Lasst mich gerne wissen, wenn ihr an etwas Eigenem bastelt. Wer der weiteren Entwicklung folgen möchte, kann gerne auf der Website der dotnetpro unter developer-world.de/insights vorbeischauen (siehe Infokasten am Artikelanfang), dort werden weitere Beiträge zu diesem Projekt erscheinen. ■

[1] Beispielcode zum Artikel,
<https://github.com/atomroflman/LteCar>



Georg Poweleit

ist Softwarearchitekt mit Fokus auf Microsoft und beschäftigt sich bereits seit über 15 Jahren mit der Softwareentwicklung, davon 12 Jahre im Umfeld der .NET-(Web-)Technologien – und anderen, die sich damit verknüpfen lassen.

DWX ACADEMY

Hands-on Workshops
und Weiterbildung für
Software-Entwickler
und -Architekten

REMOTE
ODER
INHOUSE

Hands-on Workshops .NET / C#

C#-Grundlagen	(3 Tage)
C# für Fortgeschrittene	(3 Tage)
ASP.NET Core und Web API	(3 Tage)
Datenbankentwicklung mit dem Entity Framework Core	(3 Tage)
Clean Code und Software Design	(3 Tage)
Async & Await in der Praxis	(2 Tage)
C#, .NET & SQL-Server-Update	(1 Tag)
Fix von .NET Framework zu .NET (CORE)	(1 Tag)
Einstieg in test- und behaviour-driven Development (TDD/BDD) mit .NET Core	(2 Tage)
Testen von .NET-Anwendungen	(3 Tage)
Einstieg in MVVM mit WPF	(2 Tage)
Modulare WPF-Anwendungen mit PRISM	(3 Tage)
Einstieg in NoSQL mit MongoDB für .NET-Entwickler	(3 Tage)
ASP.NET Blazor – SPA-Anwendungen mit C# und .NET	(3 Tage)



Weitere Informationen auf developer-world.de/dwx-academy
Preise auf Anfrage!

Deine Ansprechpartnerin: Susanne Herl • +49 731 88005-8835 • susanne.herl@ebnermedia.de





DATA ACCESS

Die Datenzugriffsberatung mit DR. HOLGER SCHWICHTENBERG

ENTITY FRAMEWORK CORE UND LINQ IN .NET 10.0 PREVIEW 1 BIS 6

Wenig Neues

Die Preview-Versionen 1 bis 6 von EF Core 10.0 bringen nur wenige Neuerungen, aber immerhin funktionieren zwei der vier neuen LINQ-Operatoren aus .NET 10 auch in EF Core.

Microsofts objektrelationaler Mapper Entity Framework Core wird in Version 10.0 im Rahmen von .NET 10.0 im November 2025 erscheinen. Entity Framework Core 10.0 läuft (anders als der Vorgänger Entity Framework Core 9.0, der mit .NET 8.0 und 9.0 zwei .NET-Versionen unterstützte) nur auf .NET 10.0.

Dieser Artikel stellt die Neuerungen vor, die in Preview 1 bis 6 von .NET 10.0 veröffentlicht wurden. Sollten in der Preview-Version 7 und den Release-Candidate-Versionen noch nennenswerte Neuerungen erscheinen, so werden diese in einer späteren Ausgabe der dotnetpro in der Datenzugriffskolumne thematisiert.

Neue LINQ-Operatoren LeftJoin() und RightJoin()

Wie in den vorangegangenen .NET-Versionen auch liefert Microsoft in .NET 10.0 wieder neue Operatoren für Language Integrated Query (LINQ), die bestehende Konstrukte vereinfachen.

Dieses Mal sind es mit *LeftJoin()* und *RightJoin()* zwei elementare Operatoren aus der Mengenlehre und den relationalen Datenbanken. Tatsächlich waren diese Operationen bisher in LINQ bereits möglich, allerdings umständlich mithilfe einer Gruppierung mit *GroupJoin()* und *SelectMany()* sowie *DefaultIfEmpty()*.

Die neuen Methoden *LeftJoin()* und *RightJoin()* vereinfachen die Verwendung, wie Listing 1 am Beispiel des Joins zwischen den Klassen *Compa-*

ny und *Website* zeigt. Listing 1 und Bild 1 veranschaulichen dies zunächst mit LINQ-to-Objects.

Die in .NET 10.0 neu eingeführten Operatoren *LeftJoin()* und *RightJoin()* werden in Entity Framework Core 10.0 bereits für Datenbankzugriffe unterstützt. Listing 2 und Listing 3 zeigen den Einsatz von *LeftJoin()* und *RightJoin()* bei EF Core 10.0 in Verbindung mit einer relationalen Datenbank. Auch in Entity Framework konnte man das Ergebnis von *RightJoin()* und *LeftJoin()* bisher schon erzielen: via *GroupJoin()* und *SelectMany()* mit *DefaultIfEmpty()*. Die Beispiele beginnen mit der alten Variante und zeigen dann die neue Variante.

Aus dem LINQ-Befehl mit *LeftJoin()* in Listing 2 entsteht dieser SQL-Befehl:

```
SELECT TOP(@p) [f].[FlightNo], [f].[Departure],
[f].[Destination], [f].[FlightDate], CASE
```

```
LeftRightJoin
--- Alle Firmen mit ggf. vorhandenen Websites via LeftJoin ALT (GroupJoin+SelectMany) seit .NET Framework 3.5 ---
www.IT-Visions.de -> www.IT-Visions.de
www.IT-Visions.de -> www.dotnet10.de
Software & Support -> www.entwickler.de
Heise Gruppe GmbH & Co. KG -> www.heise.de
Startup i.Gr. -> - keine URL -
--- Alle Firmen mit ggf. vorhandenen Websites via LeftJoin NEU ab .NET 10.0 ---
www.IT-Visions.de -> www.IT-Visions.de
www.IT-Visions.de -> www.dotnet10.de
Software & Support -> www.entwickler.de
Heise Gruppe GmbH & Co. KG -> www.heise.de
Startup i.Gr. -> - keine URL -
--- Alle Websites mit ggf. vorhandenen Firmen via RightJoin ALT (GroupJoin+SelectMany) seit .NET Framework 3.5 ---
www.IT-Visions.de -> www.IT-Visions.de
www.IT-Visions.de -> www.dotnet10.de
Software & Support -> www.entwickler.de
- keine Firma -> www.Microsoft.com
Heise Gruppe GmbH & Co. KG -> www.heise.de
--- Alle Websites mit ggf. vorhandenen Firmen via RightJoin NEU ab .NET 10.0 ---
www.IT-Visions.de -> www.IT-Visions.de
www.IT-Visions.de -> www.dotnet10.de
Software & Support -> www.entwickler.de
- keine Firma -> www.Microsoft.com
Heise Gruppe GmbH & Co. KG -> www.heise.de
--- Alle Firmen, die Websites haben, via InnerJoin seit .NET Framework 3.5 ---
www.IT-Visions.de -> www.IT-Visions.de
www.IT-Visions.de -> www.dotnet10.de
Software & Support -> www.entwickler.de
Heise Gruppe GmbH & Co. KG -> www.heise.de
```

Ausgabe von Listing 1 (Bild 1)



Listing 1: LeftJoin() und RightJoin() bei LINQ-to-Objects (Teil 1)

```

public void LeftRightJoin()
{
    CUI.Demo();

    Company[] companies =
    [
        new Company{ ID = 1, Name = "www.IT-Visions.de" },
        new Company{ ID = 2, Name = "Software & Support" },
        new Company{ ID = 3, Name =
            "Ebner Media Group GmbH & Co. KG" },
        new Company{ ID = 4, Name = "Startup i.Gr." }
        // hat noch keine Website
    ];

    Website[] websites =
    [
        new Website{ CompanyID = 1, URL =
            "www.IT-Visions.de" },
        new Website{ CompanyID = 1, URL =
            "www.dotnet10.de" },
        new Website{ CompanyID = 2, URL =
            "www.entwickler.de" },
        new Website{ URL = "www.microsoft.com" },
        // Diese kleine ;- ) Firma ist noch nicht
        // angelegt...
        new Website{ CompanyID = 3, URL =
            "www.ebnermedia.de" },
    ];

    // ### LeftJOIN ALT #####

    CUI.H2("--- Alle Firmen mit ggf. vorhandenen " +
        "Websites via LeftJoin ALT (GroupJoin+SelectMany)" +
        "seit .NET Framework 3.5 ---");

    var AllCompaniesWithWebsitesSetOld = companies
        .GroupJoin(websites,
            c => c.ID,
            w => w.CompanyID,
            (c, websites) => new { Company =
                c, Websites = websites })
        .SelectMany(
            x => x.Websites.DefaultIfEmpty(), // Falls keine
            // Webseite existiert, wird `null` verwendet

            (c, w) => new WebsiteWithCompany
            {
                Name = c.Company.Name,
                URL = w.URL,
                // Falls `w` null ist, bleibt URL null
                City = c.Company.City
            });

    foreach (var item in AllCompaniesWithWebsitesSetOld)
    {
        Console.WriteLine((item.Name != null ? item.Name +
            " " + item.City : "- keine Firma -").Trim() +
            " -> " + (item.URL ?? "- keine URL -"));
    }

    // ### LeftJOIN NEU #####

    CUI.H2("--- Alle Firmen mit ggf. vorhandenen " +
        "Websites via LeftJoin NEU ab .NET 10.0 ---");
    var AllCompaniesWithWebsitesSet = companies
        .LeftJoin(websites,
            e => e.ID,
            e => e.CompanyID,
            (c, w) => new WebsiteWithCompany {
                Name = c.Name, City = c.City,
                URL = w.URL }
            );

    foreach (var item in AllCompaniesWithWebsitesSet)
    {
        Console.WriteLine((item.Name != null ? item.Name +
            " " + item.City : "- keine Firma -").Trim() +
            " -> " + (item.URL ?? "- keine URL -"));
    }

    // ### RightJoin ALT #####

    CUI.H2("--- Alle Websites mit ggf. vorhandenen " +
        "Firmen via RightJoin ALT (GroupJoin+SelectMany)" +
        "seit .NET Framework 3.5 ---");
    var WebsiteWithCompanySetOLD = websites
        .GroupJoin(companies,
            w => w.CompanyID,
            c => c.ID,

```

```

    WHEN [f].[Pilot_PersonID] IS NULL THEN 'n/a'
    ELSE COALESCE(CONVERT(varchar(11),
        [f].[Pilot_PersonID]), '')
END AS [PilotId], [p0].[GivenName], [p0].[Surname]
FROM [Operation].[Flight] AS [f]
LEFT JOIN [People].[Pilot] AS [p] ON
    [f].[Pilot_PersonID] = [p].[PersonID]
LEFT JOIN [People].[Employee] AS [e] ON
    [p].[PersonID] = [e].[PersonID]
LEFT JOIN [People].[Person] AS [p0] ON
    [e].[PersonID] = [p0].[PersonID]
WHERE [p0].[Surname] IS NULL

```

Aus dem LINQ-Befehl mit *RightJoin()* in Listing 3 entsteht dieser SQL-Befehl:

```

SELECT [p0].[PersonID] AS [PilotId], [p1].[GivenName],
    [p1].[Surname], [f].[FlightNo], [f].[Airline],
    [f].[Departure], [f].[Destination], [f].[FlightDate],
    [f].[FreeSeats], [f].[Memo], [f].[NonSmokingFlight],

```

```

    [f].[Pilot_PersonID], [f].[Seats], [f].[Timestamp]
FROM [Operation].[Flight] AS [f]
RIGHT JOIN (
    SELECT TOP(@p) [p].[PersonID]
    FROM [People].[Pilot] AS [p]
    ORDER BY [p].[PersonID] DESC
) AS [p0] ON [f].[Pilot_PersonID] = [p0].[PersonID]
INNER JOIN [People].[Employee] AS [e] ON [p0].[PersonID]
    = [e].[PersonID]
INNER JOIN [People].[Person] AS [p1] ON [e].[PersonID]
    = [p1].[PersonID]
ORDER BY [p0].[PersonID]

```

Es sei zudem explizit darauf hingewiesen, dass man die neuen Operatoren *RightJoin()* und *LeftJoin()* einschließlich der vorher schon vorhandenen Operatoren *Join()* und *GroupJoin()* nur für die Verbindung von Tabellen braucht, für die es im Objektmodell keine Navigationsbeziehung gibt. So kann man statt mit dem aufwendigen *RightJoin()* bei einer vorhandenen Navigationsbeziehung im Objektmodell dasselbe

Listing 1: LeftJoin() und RightJoin() bei LINQ-to-Objects (Teil 2)

```

(w, companies) => new { Website = w,
    Companies = companies }

.SelectMany(
    x => x.Companies.DefaultIfEmpty(), // Falls
    // kein Unternehmen existiert, bleibt `null`
    (w, c) => new WebsiteWithCompany
    {
        Name = c.Name,
        // Falls `c` null ist, bleibt `Name` null
        City = c.City,
        // Falls `c` null ist, bleibt `City` null
        URL = w.Website.URL
    });

foreach (var item in WebsiteWithCompanySetOLD)
{
    Console.WriteLine((item.Name != null ? item.Name +
        " " + item.City : "- keine Firma -").Trim() +
        " -> " + (item.URL ?? "- keine URL -"));
}

// ### RightJoin NEU #####

CUI.H2("--- Alle Websites mit ggf. vorhandenen " +
    "Firmen via RightJoin NEU ab .NET 10.0 ---");
var WebsiteWithCompanySet = companies.
    RightJoin(websites,
        e => e.ID,
        e => e.CompanyID,
        (c, w) => new WebsiteWithCompany {
            Name = c.Name, City = c.City,

```

```

        URL = w.URL
    });

foreach (var item in WebsiteWithCompanySet)
{
    Console.WriteLine((item.Name != null ? item.Name +
        " " + item.City : "- keine Firma -").Trim() +
        " -> " + (item.URL ?? "- keine URL -"));
}

// ### Zum Vergleich: Inner Join, den es seit .NET
// Framework 3.5 gibt
CUI.H2("--- Alle Firmen, die Websites haben, via " +
    "InnerJoin seit .NET Framework 3.5 ---");
var CompaniesWithWebsitesSet = companies
    .Join(websites,
        c => c.ID,
        w => w.CompanyID,
        (c, w) => new WebsiteWithCompany
        {
            Name = c.Name,
            URL = w.URL,
            City = c.City
        });

foreach (var item in CompaniesWithWebsitesSet)
{
    Console.WriteLine((item.Name != null ? item.Name +
        " " + item.City : "- keine Firma -").Trim() +
        " -> " + (item.URL ?? "- keine URL -"));
}

```

Listing 2: Einsatzbeispiel für LeftJoin() in einer Datenbank mit den Tabellen „Flight“ und „Pilot“

```

/// <summary>
/// LeftJoin(): Suche alle Flüge, zu denen es keinen
/// Piloten gibt
/// </summary>
public void EFC10_LeftJoin()
{
    CUI.Demo(nameof(EFC10_LeftJoin));
    var ctx = new DA.WWWings.WwwingsV1EnContext();

    #region ----- ALT
    CUI.H2("ALT: Suche alle Flüge, zu denen es keinen " +
        "Piloten gibt via GroupJoin() und SelectMany()");
    var fluegeOhnePilotAlt = ctx.Flights
        .GroupJoin(
            ctx.Pilots,
            f => f.PilotPersonId,
            p => p.PersonId,
            (f, pilots) => new { Flight = f,
                Pilots = pilots.DefaultIfEmpty() }
        )
        .SelectMany(
            fp => fp.Pilots,
            (fp, p) => new
            {
                fp.Flight.FlightNo,
                fp.Flight.Departure,
                fp.Flight.Destination,
                fp.Flight.FlightDate,
                PilotId = fp.Flight.PilotPersonId == null ?
                    "n/a" : fp.Flight.PilotPersonId.ToString(),
                GivenName = p.Employee.Person.GivenName,
                Surname = p.Employee.Person.Surname
            }
        )
        .Where(x => x.Surname == null)
        .Take(20)
        .ToList();

    Console.WriteLine("Gefundene Flüge: " +
        fluegeOhnePilotAlt.Count);
    foreach (var item in fluegeOhnePilotAlt)
    {
        Console.WriteLine($"{item.FlightNo} " +
            $"{item.Departure}->{item.Destination} am " +
            $"{item.FlightDate}: Pilot {item.PilotId} " +
            $"{item.GivenName} {item.Surname}");
    }
}

#endregion

#region ----- NEU
CUI.H2("NEU: Suche alle Flüge, zu denen es " +
    "keinen Piloten gibt, via LeftJoin()");
var fluegeOhnePilotNeu = ctx.Flights
    .LeftJoin(
        ctx.Pilots,
        f => f.PilotPersonId,
        p => p.PersonId,
        (f, p) => new
        {
            f.FlightNo,
            f.Departure,
            f.Destination,
            f.FlightDate,
            PilotId = f.PilotPersonId == null ? "n/a" :
                f.PilotPersonId.ToString(),
            p.Employee.Person.GivenName,
            p.Employee.Person.Surname,
        })
        .Where(x => x.Surname ==
            null)
        .Take(20)
        .ToList();

    Console.WriteLine("Gefundene Flüge: " +
        fluegeOhnePilotNeu.Count);
    foreach (var item in fluegeOhnePilotNeu)
    {
        Console.WriteLine($"{item.FlightNo} " +
            $"{item.Departure}->{item.Destination} am " +
            $"{item.FlightDate}: Pilot {item.PilotId} " +
            $"{item.GivenName} {item.Surname}");
    }
}

#endregion

#region ----- Kontrolle
CUI.H3("Zur Kontrolle:");
// Zur Kontrolle:
if (fluegeOhnePilotNeu.Count() > 0)
{
    var f = ctx.Flights.Find(
        fluegeOhnePilotNeu[0].FlightNo);
    Console.WriteLine(f.ToNameValueString());
}
}

#endregion

```

Ausgabeergebnis mit einem *Include()* erreichen (Listing 4). In diesem Fall erhält man allerdings keine flache Liste mit Daten aus *Pilot* und *Flug*, sondern eine Objekthierarchie, daher zwei verschachtelte *foreach*-Schleifen.

Entity Framework Core führt dabei nur *INNER JOINS* aus:

```

SELECT [p0].[PersonID], [p0].[FlightHours], [p0].[
    FlightSchool], [p0].[LicenseDate],

```


● Listing 3: Einsatzbeispiel für RightJoin() in einer Datenbank mit den Tabellen „Flight“ und „Pilot“

```

/// <summary>
/// Gibt zu den letzten drei angelegten Piloten alle
/// Flüge aus
/// </summary>
public void EFC10_RightJoin()
{
    CUI.Demo(nameof(EFC10_RightJoin));
    var ctx = new DA.WWWings.WwwingsV1EnContext();

    CUI.H2("Alt: Gibt zu den letzten drei angelegten " +
        "Piloten alle Flüge aus via GroupJoin() und " +
        "SelectMany()");

    var ctx2 = new DA.WWWings.WwwingsV1EnContext();

    #region ----- ALT
    var pilotenMitFlugAlt = ctx.Pilots
        .OrderByDescending(x => x.PersonId)
        .Take(3)
        .GroupJoin(
            ctx.Flights,
            p => p.PersonId,
            f => f.PilotPersonId,
            (p, flights) => new { Pilot = p,
                Flights = flights.DefaultIfEmpty() }
        )
        .SelectMany(
            pf => pf.Flights,
            (pf, f) => new
            {
                PilotId = pf.Pilot.PersonId,
                pf.Pilot.Employee.Person.GivenName,
                pf.Pilot.Employee.Person.Surname,
                Flight = f,
                f.Departure,
                f.Destination,
                // Date = f == null || f.FlightDate ==
                // DateTime.MinValue ? "n/a" :
                // f.FlightDate.ToShortDateString()
            }
        )
        .OrderBy(x => x.PilotId)
        .ToList();

    foreach (var p in pilotenMitFlugAlt)
    {
        Console.WriteLine($"Pilot #{p.PilotId} " +
            "{p.GivenName} {p.Surname} fliegt " +
            (p.Flight != null ? $"Flug #{p.Flight?.FlightNo} " +
            {p.Flight.Departure}->{p.Flight.Destination} am " +
            {p.Flight.FlightDate}" : "bisher keinen Flug"));
    }

    Console.WriteLine("Anzahl: " +
        pilotenMitFlugAlt.Count);
    #endregion

    #region ----- NEU
    CUI.H2("Neu: Gibt zu den letzten drei angelegten " +
        "Piloten alle Flüge aus via RightJoin()");
    var pilotenMitFlugNeu = ctx.Flights
        .RightJoin(
            ctx.Pilots.OrderByDescending(x =>
                x.PersonId).Take(3),
            f => f.PilotPersonId,
            p => p.PersonId,
            (f, p) => new

```

● Listing 4: Include() über Navigationsbeziehung statt RightJoin()

```

CUI.H2("Gibt zu den letzten drei angelegten Piloten " +
    "alle Flüge aus via Navigation Property");

var pilotenMitFlug2 = ctx.Pilots.Include(p =>
    p.Employee).ThenInclude(p => p.Person).Include(
    p => p.Flights).OrderByDescending(x => x.PersonId)
    .Take(3).OrderBy(x => x.PersonId).ToList();

foreach (Pilot p in pilotenMitFlug2)
{
    if (p.Flights.Count == 0)
    {
        Console.WriteLine($"Pilot #{p.PersonId} " +
            "{p.Employee.Person.GivenName} " +
            "{p.Employee.Person.Surname} fliegt bisher " +
            "keinen Flug");
    }
    else
    {
        foreach (var f in p.Flights)
        {
            Console.WriteLine($"Pilot #{p.PersonId} " +
                "{p.Employee.Person.GivenName} " +
                "{p.Employee.Person.Surname} fliegt Flug " +
                "#{f.FlightNo} {f.Departure}->" +
                "{f.Destination} am {f.FlightDate}");
        }
    }
}

```

```

{
    PilotId = p.PersonId,
    p.Employee.Person.GivenName,
    p.Employee.Person.Surname,
    Flight = f,
    f.Departure,
    f.Destination,
    // Date = f.FlightDate == DateTime.MinValue ?
    // "n/a" : f.FlightDate.ToShortDateString(),
    // TODO: LAUFZEITFEHLER. Prüfen in RTM!

}).OrderBy(x => x.PilotId).ToList();

foreach (var p in pilotenMitFlugNeu)
{
    Console.WriteLine($"Pilot #{p.PilotId} " +
        "{p.GivenName} {p.Surname} fliegt " +
        (p.Flight != null ? $"Flug #{p.Flight?.FlightNo} " +
        {p.Flight.Departure}->{p.Flight.Destination} am " +
        {p.Flight.FlightDate}" : "bisher keinen Flug"));
}
Console.WriteLine("Anzahl: " +
    pilotenMitFlugNeu.Count);
#endregion

#region ----- Kontrolle
CUI.H2("Nur zur Kontrolle: Gibt zu den letzten " +
    "drei angelegten Piloten alle Flüge aus via " +
    "Navigation Property");

var pilotenMitFlug2 = ctx.Pilots.Include(p =>
    p.Employee).ThenInclude(p => p.Person).Include(

```

```

p => p.Flights).OrderByDescending(x =>
    x.PersonId).Take(3).OrderBy(x =>
    x.PersonId).ToList();
int count = 0;
foreach (Pilot p in pilotenMitFlug2)
{
    if (p.Flights.Count == 0)
    {
        count++;
        Console.WriteLine($"Pilot #{p.PersonId} " +
            "{p.Employee.Person.GivenName} " +
            "{p.Employee.Person.Surname} fliegt " +
            "bisher keinen Flug");
    }
    else
    {
        foreach (var f in p.Flights)
        {
            count++;
            Console.WriteLine($"Pilot #{p.PersonId} " +
                "{p.Employee.Person.GivenName} " +
                "{p.Employee.Person.Surname} fliegt " +
                "Flug #{f.FlightNo} {f.Departure}->" +
                "{f.Destination} am {f.FlightDate}");
        }
    }
}
Console.WriteLine("Anzahl: " + count);
#endregion
}

```

```

[p0].[LicenseType], [e].[PersonID], [e].[EmployeeNo],
[e].[HireDate], [e].[Supervisor_PersonID],
[p1].[PersonID], [p1].[Birthday], [p1].[City],
[p1].[Country], [p1].[EMail], [p1].[GivenName],
[p1].[Memo], [p1].[Photo], [p1].[Surname],
[f].[FlightNo], [f].[Airline], [f].[Departure],
[f].[Destination], [f].[FlightDate], [f].[FreeSeats],
[f].[Memo], [f].[NonSmokingFlight],
[f].[Pilot_PersonID], [f].[Seats], [f].[Timestamp]
FROM (
    SELECT TOP(@p) [p].[PersonID], [p].[FlightHours],
        [p].[FlightSchool], [p].[LicenseDate],
        [p].[LicenseType]
    FROM [People].[Pilot] AS [p]
    ORDER BY [p].[PersonID] DESC
) AS [p0]
INNER JOIN [People].[Employee] AS [e] ON [p0].[PersonID]
    = [e].[PersonID]

```

```

INNER JOIN [People].[Person] AS [p1] ON [e].[PersonID]
    = [p1].[PersonID]
LEFT JOIN [Operation].[Flight] AS [f] ON [p0].[PersonID]
    = [f].[Pilot_PersonID]
ORDER BY [p0].[PersonID], [e].[PersonID],
    [p1].[PersonID]

```

Neuer LINQ-Operator Reverse()

Der neue LINQ-Operator *Reverse()* kehrt eine Menge um. *Reverse()* liefert *void* zurück und ändert die Menge, auf die es angewendet wird, siehe [Listing 5](#) und seine Ausgabe in [Bild 2](#) sowie [1].

Dieser neue Operator *Reverse()* funktioniert aber nur im RAM bei LINQ-to-Objects. Es gibt keine Übersetzung in SQL bei Entity Framework Core.

Den nachfolgenden Versuch quittiert Entity Framework Core deshalb mit dem Laufzeitfehler „The LINQ expression ►

Listing 5: Beispiele zum Einsatz von Reverse()

```
public void Reverse()
{
    CUI.Demo();

    #region Beispiel 1
    CUI.H2("Umkehrung der Liste der Zahlen von 1 bis 20");
    // Erzeuge eine Liste mit den Zahlen von 1 bis 20
    var numbers = Enumerable.Range(1, 20).ToList();

    CUI.H3("Ausgangslage");
    // Gib die Zahlen aus
    Console.WriteLine(string.Join(", ", numbers));

    CUI.H3("Reverse()");
    // Nutze Reverse aus .NET 10.0
    numbers.Reverse();

    // Gib die Zahlen aus
    Console.WriteLine(string.Join(", ", numbers));
    #endregion

    #region Beispiel 2
    CUI.H2("Umkehrung der der Wörter in einem Satz");
    var satz = "Geld macht nicht glücklich";
    List<string> words = satz.Split(" ").ToList();

    CUI.H3("Ausgangslage");
    Console.WriteLine(string.Join(" ", words));
    CUI.H3("Reverse()");
    words.Reverse();
    Console.WriteLine(string.Join(" ", words));
    #endregion
}
```

could not be translated. Additional information: 'Reverse' could not be translated":

```
var flights = ctx.Flights.Where(x => x.Departure ==
    "Berlin").Reverse().ToList();
```

Möglich ist aber, *Reverse()* auf einer Objektmenge im RAM anzuwenden, auch wenn diese Menge per Entity Framework Core materialisiert wurde:

```
var ctx = new DA.WWWings.WwwingsV1EnContext();
var flights = ctx.Flights.Where(x => x.Departure ==
    "Berlin").ToList();
flights.Reverse();
```

Neuer LINQ-Operator Shuffle()

Der neue LINQ-Operator *Shuffle()* mischt eine Menge zufällig. Diese Methode ändert die Ausgangsmenge nicht, sondern liefert das gemischte Ergebnis als *IEnumerable<T>* zurück, siehe [Listing 6](#) und dessen Ausgabe in [Bild 3](#) sowie [2].

Die neue LINQ-Operation *Shuffle()* verwendet unter der Haube die Klasse *System.Random*. Die Randomisierung erfolgt daher also mithilfe eines kryptografisch nicht sicheren Zufallszahlengenerators.

Auch dieser neue Operator *Shuffle()* funktioniert aber nur im RAM bei LINQ-to-Objects. Es gibt keine Übersetzung in SQL bei Entity Framework Core. Wiederum möglich ist aber, *Shuffle()* auf eine Objektmenge im RAM anzuwenden, auch wenn diese von Entity Framework Core materialisiert wurde:

Listing 6: Beispiele zum Einsatz von Shuffle()

```
public void Shuffle()
{
    CUI.Demo();

    CUI.H2("Zufällige Mischung der Liste der Zahlen " +
        "von 1 bis 20");
    // Erzeuge eine Liste mit den Zahlen von 1 bis 20
    var numbers = Enumerable.Range(1, 20).ToList();
    CUI.H3("Ausgangslage");
    // Gib die Zahlen aus
    Console.WriteLine(string.Join(", ", numbers));

    CUI.H3("Shuffle()");
    // Nutze Shuffle aus .NET 10.0
    var numbersShuffled = numbers.Shuffle();

    // Gib die Zahlen aus
    Console.WriteLine(string.Join(", ",
        numbersShuffled));

    CUI.H2("Zufällige Mischung der der Wörter in " +
        "einem Satz");
    var satz = "Geld macht nicht glücklich";
    List<string> words = satz.Split(" ").ToList();

    CUI.H3("Ausgangslage");
    Console.WriteLine(string.Join(" ", words));
    CUI.H3("Shuffle()");
    IEnumerable<string> wordsShuffled = words.Shuffle();
    Console.WriteLine(string.Join(" ", wordsShuffled));
}
```

Listing 7: Festlegen der Constraints–Namen bei Standardwerten

```
protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    modelBuilder.Entity<Flight>(entity =>
    {
        entity.Property(e=>e.NonSmokingFlight)
            .HasDefaultValue(true,
                "Default_Flight_NonSmokingFlight");
        entity.Property(e=>e.Created)
            .HasDefaultValueSql("GETDATE()",
                "Default_Flight_Created");
        ...
    }
}
```

Diese Namen erscheinen dann im CREATE TABLE-Befehl:

```
CREATE TABLE [Operation].[Flight] (
    [FlightNo] int NOT NULL,
```

```
[Airline] nvarchar(3) NULL,
[Departure] nvarchar(30) NOT NULL,
[Destination] nvarchar(30) NOT NULL,
[FlightDate] datetime NOT NULL CONSTRAINT [Default_
Flight_FlightDate] DEFAULT (GETDATE()),
[NonSmokingFlight] bit NOT NULL CONSTRAINT [Default_
Flight_NonSmokingFlight] DEFAULT CAST(1 AS bit),
[Seats] smallint NOT NULL,
[FreeSeats] smallint NOT NULL,
[Pilot_PersonID] int NULL,
[Memo] varchar(max) NULL,
[Timestamp] rowversion NULL,
CONSTRAINT [PK_Flight] PRIMARY KEY ([FlightNo]),
CONSTRAINT [FK_FL_Flight_PI_Pilot] FOREIGN KEY
([Pilot_PersonID]) REFERENCES [People].[Pilot]
([PersonID])
);
```

```
var ctx = new DA.WWWings.WwwingsV1EnContext();
// var flights = ctx.Flights.Where(x => x.Departure ==
// "Berlin").Shuffle().ToList(); "The LINQ expression
// could not be translated. Additional information:
// 'Shuffle' could not be translated".
var flights = ctx.Flights.Where(x => x.Departure ==
    "Berlin").ToList();
var flightsShuffled = flights.Shuffle();
```

Vereinfachte Syntax für ExecuteUpdate() und ExecuteUpdateAsync()

Bei den Methoden *ExecuteUpdate()* und *ExecuteUpdateAsync()* zur Massenaktualisierung von Datensätzen, die Microsoft mit Version 7.0 von Entity Framework Core eingeführt hatte, bietet Entity Framework Core in der Version 10.0 nun eine bessere Übersichtlichkeit bei komplexeren Ausdrücken, indem nicht nur Expression Lambdas, sondern auch Statement Lambdas erlaubt sind.

Anstelle von

```
var c0 = ctx.Flights.Where(f =>
    f.FlightNo <= 100).ExecuteUpdate(s =>
```

```
s.SetProperty(f => f.FreeSeats, f => f.FreeSeats - 1)
    .SetProperty(f => f.FlightDate, f => (minutes > 0 ?
        f.FlightDate.AddMinutes(minutes) : f.FlightDate))
    .SetProperty(f => f.Memo, f => (setMemo ?
        "Freie Plätze geändert am " +
        DateTime.Now : f.Memo))
);
```

dürfen Entwicklerinnen und Entwickler nun auch schreiben:

```
var c1 = ctx.Flights.Where(f =>
    f.FlightNo <= 100).ExecuteUpdate(s =>
{
    s.SetProperty(f => f.FreeSeats, f => f.FreeSeats - 1);
    if (minutes > 0)
    {
        s.SetProperty(f => f.FlightDate, f =>
            f.FlightDate.AddMinutes(minutes));
    }
    if (setMemo)
    {
        s.SetProperty(f => f.Memo,
```



```
Reverse
Umkehrung der Liste der Zahlen von 1 bis 20
Ausgangslage
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
Reverse()
20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
Umkehrung der der Wörter in einem Satz
Ausgangslage
Geld macht nicht glücklich
Reverse()
glücklich nicht macht Geld
```

Ausgabe von Listing 5 (Bild 2)

```
Shuffle
Zufällige Mischung der Liste der Zahlen von 1 bis 20
Ausgangslage
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
Shuffle()
4, 16, 13, 17, 3, 15, 11, 19, 6, 8, 10, 20, 1, 9, 14, 12, 2, 7, 5, 18
Zufällige Mischung der der Wörter in einem Satz
Ausgangslage
Geld macht nicht glücklich
Shuffle()
nicht glücklich Geld macht
```

Ausgabe von Listing 6 (Bild 3)


```

EFC10_QueryFilterNames
OnModelCreating
dbug: 23.07.2025 19:39:00.712 RelationalEventId.CommandExecuting[20100] (Microsoft.EntityFrameworkCore.Database.Command)
      Executing DbCommand [Parameters=[], CommandType='Text', CommandTimeout='300']
      SELECT [f].[FlightNo], [f].[Airline], [f].[Departure], [f].[Destination], [f].[FlightDate], [f].[FreeSeats], [f].[Memo],
      [f].[NonSmokingFlight], [f].[Pilot_PersonID], [f].[Seats], [f].[Timestamp]
      FROM [Operation].[Flight] AS [f]
      WHERE [f].[NonSmokingFlight] = CAST(1 AS bit) AND [f].[FlightDate] > GETDATE() AND [f].[FreeSeats] > CAST(0 AS smallint)
      AND [f].[Departure] = N'Berlin'
Anzahl gefundene Flüge: 200
dbug: 23.07.2025 19:39:00.813 RelationalEventId.CommandExecuting[20100] (Microsoft.EntityFrameworkCore.Database.Command)
      Executing DbCommand [Parameters=[], CommandType='Text', CommandTimeout='300']
      SELECT [f].[FlightNo], [f].[Airline], [f].[Departure], [f].[Destination], [f].[FlightDate], [f].[FreeSeats], [f].[Memo],
      [f].[NonSmokingFlight], [f].[Pilot_PersonID], [f].[Seats], [f].[Timestamp]
      FROM [Operation].[Flight] AS [f]
      WHERE [f].[NonSmokingFlight] = CAST(1 AS bit) AND [f].[Departure] = N'Berlin'
Anzahl gefundene Flüge: 1029

```

Ausgabe für das Angeben eines Arrays der Filternamen als Parameter (Bild 4)

```

        "Freie Plätze geändert am " + DateTime.Now);
    }
});

```

Auch wenn die Release Notes zu Entity Framework Core 10.0 Preview 1 [3] explizit nur von der asynchronen Methode *ExecuteUpdateAsync()* sprechen, funktioniert das Feature auch mit der synchronen Variante *ExecuteUpdate()*.

Benannte Query-Filter in Entity Framework Core

Globale Query-Filter gibt es seit Entity Framework Core 2.0. Man erzeugt sie in der Methode *OnModelCreating()* in der Kontextklasse. Sie wirken dann auf alle LINQ-Abfragen, die sich auf die Kontextklasse beziehen. Entwicklerinnen und Entwickler hatten aber immer schon die Möglichkeit, mit dem Aufruf *IgnoreQueryFilters()* in einer LINQ-Abfrage die globalen Query-Filter zu deaktivieren.

Seit Entity Framework Core 10.0 Preview 6 können Entwicklerinnen und Entwickler nun bei globalen Query-Filtern diesen dedizierte Namen geben und globale Query-Filter anhand ihres Namens einzeln in LINQ-Abfragen deaktivieren.

Im folgenden Beispiel werden drei benannte globale Query-Filter erzeugt:

```

modelBuilder.Entity<Flight>()
    .HasQueryFilter("IsNonSmokingFlight",
        b => b.NonSmokingFlight == true)
    .HasQueryFilter("IsFuture",
        b => b.FlightDate > DateTime.Now)
    .HasQueryFilter("HasFreeSeats", b => b.FreeSeats > 0);

```

Der Aufruf *IgnoreQueryFilters()* hat bisher immer alle globalen Query-Filter für eine Abfrage ausgeschlossen. Nunmehr können Entwicklerinnen und Entwickler einzelne oder mehrere Filter ausschließen, indem sie ein Array der Filternamen als Parameter angeben:

```

var ctx = new DA.WWWings.WwwingsV1EnContext();

var flightsWithAllQueryFilters =
    ctx.Flights.Where(x=>x.Departure=="Berlin").ToList();

```

```

Console.WriteLine("Anzahl gefundene Flüge: " +
    flightsWithAllQueryFilters.Count);

```

```

var flightsWithSomeQueryFilters = ctx.Flights.Where(
    x => x.Departure == "Berlin").IgnoreQueryFilters(
    ["IsFuture", "HasFreeSeats"]).ToList();
Console.WriteLine("Anzahl gefundene Flüge: " +
    flightsWithSomeQueryFilters.Count);

```

Bild 4 zeigt, dass die zweite Abfrage weniger Filterkriterien im SQL-Befehl besitzt und damit eine größere Ergebnismenge liefert.

Nicht in der Dokumentation von Microsoft steht, was passiert, wenn man bei *IgnoreQueryFilters()* einen Namen falsch schreibt. Der Autor dieser Kolumne hat dies natürlich getestet und wundert sich darüber, dass falsch geschriebene Namen nicht moniert, sondern einfach ignoriert werden. Das kann zu unentdeckten Fehlern führen, wenn bei den Unit-Tests nicht gut getestet wird.

Explizite Namen für Constraints bei Standardwerten

Man kann in Entity Framework Core 10.0 nun beim Anlegen von Standardwerten im Rahmen des Forward Engineering (also beim Code-First-Ansatz) festlegen, wie die entsprechenden Constraints in der Datenbank heißen sollen (Listing 7). Dazu erhalten die Methoden *HasDefaultValue()* und *HasDefaultValueSql()* weitere Überladungen mit einem zusätzlichen Zeichenkettenparameter.

Entwicklerinnen und Entwickler können auch wie bisher die Namen der Constraints weglassen. Dann gibt es bei *CREATE TABLE* keine Namensangabe, und das Datenbankmanagementsystem vergibt die Namen selbst, siehe Bild 5:

```

CREATE TABLE [Operation].[Flight] (
    [FlightNo] int NOT NULL,
    [Airline] nvarchar(3) NULL,
    [Departure] nvarchar(30) NOT NULL,
    [Destination] nvarchar(30) NOT NULL,
    [FlightDate] datetime NOT NULL DEFAULT (GETDATE()),
    [NonSmokingFlight] bit NOT NULL DEFAULT CAST(1 AS bit),

```

```
[Seats] smallint NOT NULL,
[FreeSeats] smallint NOT NULL,
[Pilot_PersonID] int NULL,
[Memo] varchar(max) NULL,
[Timestamp] rowversion NULL,
CONSTRAINT [PK_Flight] PRIMARY KEY ([FlightNo]),
);
```

Eine dritte Variante gibt es beim der Aufruf der neuen Methode `UseNamedDefaultConstraints()` im Rahmen von `OnModelCreating()`, siehe [Listing 8](#). Dann vergibt Entity Framework Core automatisch Namen, die mit `DF_` beginnen, aber anders als bei den durch Microsoft SQL Server vergebenen Namen den ganzen Spaltennamen und keine Nummer umfassen.

```
CREATE TABLE [Operation].[Flight] (
    [FlightNo] int NOT NULL,
    [Airline] nvarchar(3) NULL,
    [Departure] nvarchar(30) NOT NULL,
    [Destination] nvarchar(30) NOT NULL,
    [FlightDate] datetime NOT NULL
    CONSTRAINT [DF_Flight_FlightDate]
    DEFAULT (GETDATE()),
    [NonSmokingFlight] bit NOT NULL
    CONSTRAINT
    [DF_Flight_NonSmokingFlight] DEFAULT CAST(1 AS bit),
    [Seats] smallint NOT NULL,
    [FreeSeats] smallint NOT NULL,
    [Pilot_PersonID] int NULL,
    [Memo] varchar(max) NULL,
    [Timestamp] rowversion NULL,
    CONSTRAINT [PK_Flight] PRIMARY KEY ([FlightNo]),
    CONSTRAINT [FK_FL_Flight_PI_Pilot] FOREIGN KEY
    ([Pilot_PersonID]) REFERENCES [People].[Pilot]
    ([PersonID])
);
```

Weitere Verbesserungen für LINQ

EF Core bietet in Version 10.0 darüber hinaus kleinere Verbesserungen bei der Übersetzung von LINQ in SQL, siehe [4]:

- Übersetzung von `DateOnly.ToDateTime()`.
- Übersetzung von `DateOnly.DayNumber` und Subtraktion von `DayNumber` für SQL Server und SQLite.
- Übersetzung von mehreren aufeinanderfolgenden `LIMIT`-Anweisungen.

- Übersetzung von `COALESCE` als `ISNULL` auf SQL Server in den meisten Fällen.
- Übersetzung von einigen Zeichenkettenfunktionen mit `char`-Argumenten.
- Übersetzung von Datums-/Zeitfunktionen mit den Argumenten `DatePart.Microsecond` und `DatePart.Nanosecond`.
 - Optimierung der Verwendung der `Count`-Operation auf `ICollection<T>`.
 - Optimierung von `MIN/MAX` über `DISTINCT`.
 - Vereinfachte Parameternamen (zum Beispiel von `@__xy_0` zu `@xy`).
 - Verwendung von `MAX/MIN/ORDER BY` mit `decimal` auf SQLite.
 - Projizieren unterschiedlicher Navigations-eigenschaften (aber gleichen Typs) über den bedingten Operator (`?:`).

Verbesserte Suche mit CosmosDB

Entity Framework Core 10.0 unterstützt seit Preview 4 in Verbindung mit Azure CosmosDB zwei neue Suchmodi, die es dort seit März 2025 als Vorschauversion gibt: Volltextsuche [5] und hybride Suche, die die Vektorsuche und die Volltextsuche mit einem RRF-Algorithmus (Reciprocal Rank Fusion) kombiniert [6]. Microsoft hat zudem angekündigt, dass die in Entity Framework Core Version 9.0 eingeführte Vektorsuche mit CosmosDB nun stabil ist [7].

Weitere geplante Neuerungen in Entity Framework Core 10.0

Wie bei Entity Framework Core 9.0 hat das Entity-Frame-work-Core-Entwicklungsteam auch dieses Mal wieder keinen Plan veröffentlicht, welche Features kommen sollen. Der letzte öffentliche Plan war zu Entity Framework Core 8.0 bekannt gegeben worden, siehe [Bild 6](#).

Man kann natürlich in das GitHub-Repository [8] schauen und sich dort die Einträge zusammensuchen. Ein guter Start für diese Forschungsarbeiten ist die Abfrage unter [9]. Dabei findet man zum Beispiel:

- Der neue SQL-Server Datentyp `VECTOR` und die zugehörigen Funktionen `VECTOR_DISTANCE()` und `VECTOR_SEARCH()` sollen unterstützt werden. Dafür können Entwicklerinnen und Entwickler in Entitätsklassen den .NET-Typ `Microsoft.Data.SqlTypes.SqlVector<T>` und die Funktion `EF.Functions.VectorDistance()` nutzen. Ein Teil ist ►

dbo.Flight
Columns
FlightNo (PK, int, not null)
Airline (nvarchar(3), null)
Departure (nvarchar(30), not null)
Destination (nvarchar(30), not null)
FlightDate (datetime, not null)
NonSmokingFlight (bit, not null)
Seats (smallint, not null)
FreeSeats (smallint, not null)
Pilot_PersonID (int, null)
Memo (varchar(max), null)
Timestamp (timestamp, null)
Keys
Constraints
DF_Flight_FlightDa_286302EC
DF_Flight_NonSmoki_29572725

Vom Microsoft SQL Server automatisch vergebene Namen für die Constraints ([Bild 5](#))

Listing 8: Automatische Namensvergabe für Constraints-Namen bei Standardwerten aktivieren

```
protected override void OnModelCreating(
    modelBuilder.UseNamedDefaultConstraints();
    modelBuilder
    {
```

bereits am 17. Juli 2025 in die Codebasis eingeflossen [10] und wird damit voraussichtlich in der Preview 7 erscheinen. Hingegen fehlt für `VECTOR_SEARCH()` eine Umsetzung bisher noch [11].

- Komplexe Typen sollen zukünftig optional sein können [12] und ein JSON-Mapping unterstützen [13]. Damit wird es dann auch möglich, Mengen komplexer Typen zu haben [14].

- Für SQL Server 2025 wird das neue Schlüsselwort `RETURNING` in Verbindung mit `JSON-VALUE()` unterstützt [15].
- Entity Framework Core soll zukünftig seine Versionsnummer an den SQL Server als Zusatzinformation im Connection-String übermitteln [16].
- Beim Zusammenbau von SQL-Zeichenketten via Zeichenketten-Konkatenation soll es künftig eine Compiler-Warnung geben, da die Gefahr eines SQL-Injektionsangriffs besteht [17].

Keinerlei Verbesserungen gibt es leider bei der bisher sehr mangelhaften Integration von Entity Framework Core in den NativeAOT-Compiler; die Datenzugriffskolumne berichtete in Ausgabe 6-7/2025 [18] über den Stand in Entity Framework Core 9.0.

- [1] API Proposal: `Enumerable.Reverse<T>(T[])`, <https://github.com/dotnet/runtime/issues/107723>
- [2] API Proposal: `System.Linq.Shuffle()`, <https://github.com/dotnet/runtime/issues/111221>
- [3] Entity Framework Core 10 Preview 1 – Release Notes, <https://github.com/dotnet/core/blob/main/release-notes/10.0/preview/preview1/efcore.md#executeupdateasync-now-accepts-a-regular-non-expression-lambda>
- [4] Microsoft Learn, What's New in EF Core 10, <https://learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-10.0/whatsnew>
- [5] Microsoft Learn, Full-text search in Azure Cosmos DB for NoSQL, <https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/full-text-search>
- [6] Microsoft Learn, Hybrid search in Azure Cosmos DB for NoSQL, <https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/hybrid-search>
- [7] Entity Framework Core 10 Preview 4 – Release Notes, <https://github.com/dotnet/core/blob/main/release-notes/10.0/preview/preview4/efcore.md#vector-similarity-search-exits-preview>

EF Core-Releases und Planung

14.11.2024

Stabile Releases

Freigabe	Zielframework	Support bis zum	Verknüpfungen
EF Core 9.0 ↗	.NET 8	12. Mai 2026	Neuerungen / Breaking Changes
EF Core 8.0 ↗	.NET 8	10. November 2026	Neuerungen / Breaking Changes
EF Core 7.0 ↗	.NET 6	Abgelaufen: 14. Mai 2024	Neuerungen / Breaking Changes
EF Core 6.0 ↗	.NET 6	Abgelaufen am 12. November 2024	Neuerungen / Breaking Changes

Kein Plan zu Entity Framework Core 9.0 und 10.0 (Bild 6)

- [8] Microsoft: GitHub-Repository für Entity Framework Core, <https://github.com/dotnet/efcore>
- [9] Abfrage offener Issues in EF Core 10, <https://github.com/dotnet/efcore/issues?q=is%3Aopen%20is%3Aissue%20milestone%3A10.0.0%20type%3AFeature>
- [10] Implement support for SQL Server vector search, <https://github.com/dotnet/efcore/pull/36383>
- [11] Implement support for SQL Server `VECTOR_SEARCH()`, <https://github.com/dotnet/efcore/issues/36384>
- [12] Allow mapping optional complex properties, <https://github.com/dotnet/efcore/issues/31376>
- [13] Add relational JSON mapping support for complex types, <https://github.com/dotnet/efcore/issues/31252>
- [14] Complex type collection support, <https://github.com/dotnet/efcore/issues/31237>
- [15] Enable the JSON data type and `JSON_VALUE()` `RETURNING` clause for Azure SQL, <https://github.com/dotnet/efcore/issues/36460>
- [16] SQL Server: Introduce EF and its version into the connection string's Application Name, <https://github.com/dotnet/efcore/issues/35730>
- [17] Analyzer warnings for string concatenation in raw SQL APIs, <https://github.com/dotnet/efcore/issues/35735>
- [18] Holger Schwichtenberg, Stark beschränkt, *dotnetpro* 6-7/2025, Seite 50 ff.
- [19] EF Core-Releases und Planung, <https://learn.microsoft.com/de-de/ef/core/what-is-new/>

Dr. Holger Schwichtenberg

gehört zu den bekanntesten .NET- und Webexperten in Deutschland. Microsoft zeichnet ihn seit 22 Jahren als MVP aus. Mit dem 54-köpfigen Expertenteam bei www.IT-Visions.de bietet er Beratungen, Schulungen und Softwareentwicklung an.

www.dotnet-doktor.de

Bildquelle: Microsoft [19]

.NET DEVELOPER CONFERENCE

Die Konferenz für .NET-Entwickler:innen

**24. BIS 27.
NOVEMBER 25**
Pullman Cologne Hotel

**10%
Rabatt**
mit dem Code
DNPAbospecial

Software Design | Software Architecture | Software Quality
Software Testing | Frontend Development | GenAI & Agentic
Data & Analytics | .NET | Approaches | Backend Technologies
Cloud & Infrastructure



WORKSHOPS – MONTAG, 24. NOVEMBER 2025

Uhrzeit	Workshop 1				Workshop 2				Workshop 3				Workshop 4			
																
09.00 – 18.00	Moderne Softwarearchitekturen <i>David Tielke</i> In Zeiten, in denen Anwendungen mit Cloud, künstlicher Intelligenz und ähnlichen Technologien erstellt werden, ist ein Qualitätsmerkmal ihrer Architektur wichtiger denn je: Flexibilität. Eine moderne Architektur sollte daher nicht nur die gegenwärtige Anforderungen abdecken, sondern auch künftige Einsatzszenarien berücksichtigen. Im Workshop befassen wir uns mit der Frage, wie einer Architektur die notwendige Flexibilität verliehen werden kann, um alle zukünftigen Anforderungen zu erfüllen und wirtschaftlich optimal betrieben zu werden, und lernen Wege und Methoden kennen, mit denen sich dies sicherstellen lässt.				Autonom und Intelligent: AI-Agentensysteme mit .NET entwickeln <i>Jörg Neumann</i> AI-Agenten bilden die Basis einer völlig neuen Anwendungsgeneration – Apps, die mit Hilfe von AI nicht nur Fragen beantworten, sondern völlig autonom Aufgaben erledigen können. Hierbei generieren sie nicht nur Text, sondern haben auch Zugriff auf Tools und Daten. Dies lässt sie mit anderen Systemen kommunizieren und so ganze Geschäftsprozesse abarbeiten. Und das Ganze allein oder zusammen in cross-funktionalen Agenten-Teams! Jörg Neumann führt in das Thema ein und zeigt euch, wie ihr intelligente Agentensysteme mit .NET und Semantic Kernel entwickelt.				SPAs mit Blazor 10.0 und C# 14.0 <i>Dr. Holger Schwichtenberg</i> Microsoft hat die Einsatzszenarien von Blazor durch Blazor Static-Server-Rendering und Auto-Modus mit Umschaltung zwischen Blazor Server und Blazor WebAssembly erheblich erweitert. Im Workshop präsentiert DOTNET-DOKTOR Holger Schwichtenberg den Stand von Blazor in Version 10.0 und stellt die verschiedenen Blazor-Architekturmodelle mit ihren Vor- und Nachteilen gegenüber. Danach wird er die Teilnehmer:innen anleiten, selbst eine Blazor 10.0-Anwendung zu entwickeln, die serverseitig vorrendert, dann Blazor Server nutzt und im Hintergrund automatisch die Dateien für Blazor WebAssembly herunterlädt (Progressive Enhancements).				EF Core 10 – Das Datenmodell im Griff <i>Christian Giesswein</i> EF Core 10 bringt wieder einige spannende Neuerungen mit – und dieser Workshop zeigt dir, wie du sie in der Praxis sinnvoll einsetzt. Statt oberflächlichem Feature-Hopping gehen wir tief rein: Modellierung, Migrationen, Performance, komplexe Abfragen, Multi-Tenancy, Interceptors, JSON-Spalten, Testing – der Workshop bietet nahezu alles, was du brauchst, um EF Core produktionsreif zu beherrschen.			
	Workshop 5  C# als Fremdsprache <i>Thorsten Kansy</i> Der Workshop richtet sich an all diejenigen, die modernes C# schnell und effizient lernen wollen und bereits in anderen Programmiersprachen zu Hause sind. Also Entwickler:innen, denen die Konzepte der Programmierung bekannt sind und nun die Erklärungen von for-Schleifen, if-Statements und vieles andere mehr kennenlernen wollen. Thorsten Kansy stellt bei dieser kompakten Druckbetankung alles wichtige vor, ohne sich in Details zu verlieren und ohne die Praxis aus den Augen zu verlieren.				Workshop 6  CI/CD-Pipelines mit Azure DevOps meistern <i>Marc Müller</i> Beginnend mit modularisierten YAML-Pipelines und Configuration-as-Code zur Optimierung von Workflows lernst du, Docker-Container für effiziente Build- und Laufzeitumgebungen zu integrieren, um Portabilität und Konsistenz zu gewährleisten. Automatisierte Sicherheitsprüfungen, bei denen Code und Abhängigkeiten analysiert werden, um Schwachstellen zu beheben, verbessern den CI-Prozess. Weitere Themen: Automatisierte Bereitstellung von Artefakten, Einbettung von Sicherheitspraktiken für die Agentenkonfiguration, Verwaltung von Anmeldeinformationen, Blue-Green- und Canary-Deployments für nahtlose Updates mit minimalen Ausfallzeiten.				Workshop 7  Von Doing Agile zu Being Agile <i>Ina Einemann</i> In vielen Teams läuft Scrum, aber niemand redet offen. Jira ist voll, aber keiner fühlt sich verantwortlich. Dailys passieren, aber bewegen nichts. Kommt dir bekannt vor? Dann ist dieser Workshop für dich. Denn Agilität ist mehr als Sprints und Dailys. Sie zeigt sich vor allem in unserer Haltung. Denn machen wir uns nichts vor: Jira ersetzt keine Haltung. Und „Scrum machen“ reicht nicht, wenn Mut, Offenheit oder echte Zusammenarbeit fehlen. Wir schauen uns an, was echte Agilität bedeutet und du als Developer dafür tun kannst. Ohne auf den großen Kulturwandel zu warten. Sondern einfach bei dir anzufangen.				Workshop 8  Raus aus .NET-Brownfield. Migrationsstrategien für Entwickler:innen und Architekt:innen <i>Oliver Sturm</i> Alte Software vollständig zu ersetzen ist selten realistisch. Doch am Status quo festzuhalten bringt uns auch nicht weiter. Oliver Sturm zeigt, wie eine neue, moderne Architektur und ein zukunftsfähiger Entwicklungsansatz entstehen können – trotz bestehender Abhängigkeiten, eingeschränkter Ressourcen und zahlreicher Altlasten. Es geht nicht um kosmetische Reparaturen am Bestandscode, sondern um fundierte Neuentwicklung – mit realistischer Rücksicht darauf, was vorhanden ist und mit dem auch eine neue Lösung umgehen können muss.			



Die Programmdetails sind zu finden unter:
developer-world.de/ddc/ddc-programm

.NET DEVELOPER CONFERENCE

KONFERENZ – DIENSTAG, 25. NOVEMBER 2025

UHRZEIT	FRONTEND DEVELOPMENT	GENAI & AGENTIC	DATA & ANALYTICS	SOFTWARE ARCHITECTURE, TESTING, AND QUALITY
08.00 – 09.00	Registrierung, Kaffee, Networking			
09.00 – 10.00	Keynote: Agentic AI: Die nächste Evolutionsstufe in der Softwareentwicklung <i>Daniel Meixner, GitHub</i>			
10.00 – 10.30	Kaffeepause und Besuch der Expo			
10:30 – 11:30	Moderne Cross-Plattform-Applikationen mit .NET und Avalonia UI <i>Roland König</i>	In geheimer Mission: AI Agents mit .NET und OpenAI entwickeln <i>Jörg Neumann</i>	SQL Server 2025 – eine kritische Würdigung <i>Thorsten Kansy</i>	Architektur per Knopfdruck <i>David Tielke</i>
Raumwechsel				
11.40 – 12.40	SaaS-Startup mit .NET, ASP.NET Core & Blazor – Der Traum, der Stack und die Realität <i>Christian Giesswein</i>	Building an Agentic RAG Pipeline <i>Francesco Esposito</i> 	Vektordatenbanken verstehen und optimal einsetzen: Was hinter dem Hype der semantischen Suche steckt <i>Gregor Biswanger</i>	Webanwendungen testen mit Microsoft Playwright <i>Dr. Holger Schwichtenberg</i>
12.40 – 14.00	Mittagspause und Besuch der Expo			
14.00 – 15.00	Auf welche Web-UI Technologie setzt Microsoft? Ist das React? <i>Harald Binkle</i>	Getting Started with MCP in .NET <i>Michael Staib</i>	Autoincrement (identity) is not the only option for primary keys <i>Jiří Činčura</i> 	Alles nur CI-Theater? <i>Konstantin Diener</i>
Raumwechsel				
15.10 – 16.10	GUI-Frameworks für .NET: Die Qual der Wahl <i>Dr. Holger Schwichtenberg</i>	Humanoider Roboter mit Gen AI als Guide? <i>Marc Schuh, Daniel Klingmann</i>	Unleashing the Power of Vector Search with Azure SQL Database <i>Taswar Bhatti</i> 	Telemetrie und Distributed Tracing mit OpenTelemetry für ASP.NET Core Microservices <i>Marc Müller</i>
16.10 – 16.40	Kaffeepause und Besuch der Expo			
16.40 – 17.40	Raus aus der Wartungsfalle: WinForms- und WPF-Legacy-Anwendungen modernisieren <i>André Krämer</i>	RAG-tastisch: Der neue Standard in der intelligenten Informationssuche <i>Thomas Tomow</i>	Next-Gen AI Insights: Native Vektorsuche mit dem SQL Server 2025 <i>Torsten Strauß</i>	Build event-driven systems with Azure <i>Sofía Platas, Andrea Díez</i> 
Raumwechsel				
17.50 – 18.50	Creating Data-Driven .NET MAUI Apps <i>Mark Allibone</i>	Build AI with Azure AI Foundry <i>Codrina Merigo</i> 	KI für Entwickler in SQL Server 2025 und Microsoft-Datenbanken <i>Constantin „Kostja“ Klein</i>	Taktisches Refactoring von .NET Legacy Code <i>Gregor Biswanger</i>
Raumwechsel				
19.00 – 20.00	Night Session: SignalRC – Wenn WebSockets auf Allrad treffen			
20.00 – 22.00	Networking, Fingerfood & Drinks			

KONFERENZ – MITTWOCH, 26. NOVEMBER 2025

UHRZEIT	.NET FUNDAMENTALS & BEST PRACTICES	APPROACHES	BACKEND TECHNOLOGIES	CLOUD DEVELOPMENT
08.00 – 09.00	Registrierung, Kaffee, Networking			
09.00 – 10.00	Keynote: Agilität zwischen Haltung und Handeln Ina Einemann, Agile Coach			
10.00 – 10.30	Kaffeepause und Besuch der Expo			
10.30 – 11.30	.NET 10 – Zwei Leute lieben's, der Rest wartet auf .NET 11 Christian Giesswein	Meet Your New Teammates: Wie GitHub Copilot Agents die Softwareentwicklung neu definiert Michael Kaufmann	.NET MAUI and . NET Aspire: a beautiful combination Johan Smarius	.NET 10 für verteilte Cloud- & Enterprise-Anwendungen Marc Müller
Raumwechsel				
11.40 – 12.40	AI Driven Development (ADD) in .NET David Tielke	How to become a software hero? – Tips and tricks from the trenches Ivo Verburgh	Minimal API 2.0: Advanced Patterns and Best Practices for Lightweight Web Services Kajetan Duszyński	Next-Gen Cloud Native .NET Apps: Aspire and Dapr Unleashed Florian Lenz
12.40 – 14.00	Mittagspause und Besuch der Expo			
14.00 – 14.45	DDC Panel: Developer Productivity Moderation: Fernando Schneider, Chefredakteur dotnetpro Teilnehmer u.a. Daniel Meixner, GitHub Marc Schuh, TNG Technology Consulting			
Raumwechsel				
15.00 – 16.00	How to build well-designed and reusable NuGet packages Dennis Doomen	Scrum Master 2.0: Ist KI der bessere Facilitator? Miriam Hunger	Web-Security, up-to-date: Die OWASP Top Ten 2025 Christian Wenz	Building next-gen applications with event-driven architectures Teena Idnani
16.00 – 16.30	Kaffeepause und Besuch der Expo			
16.30 – 17.30	Mono Repo in .NET: Should You Use It or Avoid It? Emanuele Bartolesi	Migrating from a monolith to a new service Connell Sharp	Building Ultra-Fast, Globally Distributed Apps with Serverless at the Edge Florian Lenz	Cosmos DB and Azure Functions Johan Smarius
Raumwechsel				
17.40 – 18.40	Writing a .NET profiler in C# with Silhouette Kevin Gosse	Beyond the Code: Can GitHub Copilot Be Your Co-Architect? Yuliya Khadasevich, Liuba Gonta	Taking Entity Framework Core data seeding to the next level with Bogus Sander ten Brinke	The 7 Deadly Cloud Sins (and How to Avoid Them) Ruby Jane Cabagnot
18.40 – 20.00	Feierabendbier			



Die Programmdetails sind zu finden unter:

developer-world.de/ddc/ddc-programm

.NET DEVELOPER CONFERENCE

DEVSESSIONS – DONNERSTAG, 27. NOVEMBER 2025

Uhrzeit	DevSession 1	DevSession 2	DevSession 3	DevSession 4
09.00 – 13.00	 Mein eigener Kubernetes-Cluster mit ASP.NET Core & Blazor <i>Christian Giesswein</i> Christian zeigt dir, wie du deinen eigenen Kubernetes-Cluster ohne teure Cloud-Abos aufsetzt, darauf produktionsreif eine moderne Webanwendung mit ASP.NET Core & Blazor hostest und warum du das überhaupt willst. Themen sind unter anderem: Kubernetes selbst hosten Load Balancing, TLS & Ingress CI/CD für .NET & Blazor ASP.NET Core & Blazor Server/ WebAssembly im Cluster Logging, Monitoring, Updates Die DevSession richtet sich an all diejenigen, die keine Lust auf Black-boxes in der Cloud haben und verstehen wollen, was unter der Haube passiert.	 Moderne Datenzugriffsschicht mit GraphQL – ein pragmatischer Einstieg <i>Oliver Sturm</i> Die DevSession knüpft inhaltlich an den Workshop „Raus aus dem .NET-Brownfield“ vom Montag an. Ihr Fokus liegt auf der konkreten Umsetzung: Oliver zeigt, wie sich mit GraphQL eine flexible, moderne Daten-schnittstelle über bestehende .NET-Datenmodelle legen lässt – als strategisches Bindeglied zwischen Alt und Neu. Wir entwickeln ein schlankes und robustes Setup, das sich für erste Modernisierungsschritte im Brownfield eignet und Grundlage für größere Architekturvorhaben ist. Keine Framework-Schlachten, keine Komplettlösungen – einfach nur praktikable Techniken, die sich in echten Projekten bewährt haben.	 Automatisiertes Testen – die schwierigen Teile <i>Florian Winkelbauer</i> Einführende Tutorials für automatisierte Testframeworks lassen es einfach aussehen: Man schreibt etwas Code, erstellt ein paar Tests, und fertig. Wenn man sich allerdings mit Produktionscode befasst, stößt man nicht selten auf größere Herausforderungen und lernt, dass die Realität anders aussieht. Wir lernen solche Herausforderungen kennen und erarbeiten Lösungen für die Erstellung einer Reihe automatisierter Tests in C#. Lerne auf diese Weise neue Aspekte der Denkweise in Bezug auf Code kennen, probiere verschiedene Teststrategien aus und entdecke Techniken, um Tests lesbar und unterhaltsam zu gestalten.	 Auf dem Weg zur AI Software Factory – Vision trifft Praxis <i>Jana Obernosterer</i> Wie sieht die Softwareentwicklung von morgen aus? In diesem Workshop tauchen wir ein in die Welt der agenten-basierten Softwareentwicklung und entwickeln eine „AI Software Factory“ – ein System, das in der Lage ist kleine Entwicklungsprojekte weitgehend automatisiert umzusetzen. Im Rahmen dieser DevSession erarbeiten wir hierfür in Konzeptrunden, Diskussionen etc. – in denen Du Deine eigenen Ideen und Vorschläge einbringst – gemeinsam die Architektur, Workflows, Prompts und Informationsgrundlagen für die Software Factory.
14.00 – 18.00	 Azure meets Containers: Maximiere deine Cloud-Performance <i>Florian Bader</i> Möchtest du erfahren, wie du deine Anwendungen in Azure mit Containern betreiben kannst? In dieser DevSession lernst du verschiedene Möglichkeiten kennen, Container in Azure zu deployen und zu skalieren. Es werden die Vor- und Nachteile von Containern in der Cloud erläutert und die Frage beantwortet, welcher Azure Service sich besonders gut für welche Anforderungen eignet. Außerdem stelle ich dir verschiedene Anwendungsszenarien vor, in denen der Einsatz von Containern sinnvoll sein kann. Nimm teil und erfahre so mehr über die Möglichkeiten, die Container in Azure bieten.	 .NET Aspire in Action – Jenseits von „Hello World“ <i>Florian Lenz</i> Florian Lenz zeigt dir, wie du mit .NET Aspire moderne, cloud-native .NET-Anwendungen entwickelst, die weit über einfache Demo-Beispiele hinausgehen. Statt „Hello World“ warten echte Architekturmuster, praktische Tipps aus dem Projektalltag und Insights auf Dich, die dir helfen, Aspire produktiv einzusetzen. Dabei nimmt dir .NET Aspire komplexe Infrastrukturthemen ab, bringt Observability, Cloud-Readiness und modulare Architektur out of the box mit und sorgt so dafür, dass du dich auf die Entwicklung von Business-Logik und schnelle Ergebnisse konzentrieren kannst.	 Von WPF zum Cross-Plattform UI Framework Avalonia <i>Roland König</i> WPF galt lange Zeit als Standard, wenn es um die Entwicklung von (Windows) Desktop-Applikationen mit C# ging. Selbst heute ist das Framework trotz der geringen Weiterentwicklung durch Microsoft und der Plattformbindung an Windows noch sehr beliebt. Dabei ist es höchste Zeit, über mögliche alternative Frameworks nachzudenken. Eine dieser Alternativen ist Avalonia. Es weist viele Ähnlichkeiten zu WPF auf, setzt auf XAML, bevorzugt MVVM, bringt einen visuellen Designer mit, läuft stabil auf allen Plattformen und bietet unter anderem ein Compiled-Binding.	 Architecting and Developing Custom Agents and Copilots <i>Dr. Damir Dobric</i> Als Software Architect oder Developer bist du gefordert, Anwendungen zu entwickeln, die Large Language Modelle (LLM) nutzen. Ich zeige dir, wie LLMs und GenAI unter der Haube funktionieren. Wir beginnen dazu mit wesentlichen Grundlagen und tauchen anschließend tief in Token-Generierung, Vector Similarity, Embeddings, Modelltypen, Plugins, Agents & Co. ein. Du erfährst, wie sich AI-Agents erstellen lassen, die intelligenten Code in jeder Anwendung ausführen, lernst Semantic Kernel kennen und integrierst AI in die Azure-Plattform, um leistungsfähige Anwendungen und mächtige Backends zu erstellen.

DAS FRAMEWORK SEMANTIC FLOW

Pizza bestellen mit Struktur

Komplexe KI-Prozesse im Griff mit Semantic Kernel.

Du stehst vor der Aufgabe, generative KI sauber in echte Softwareprozesse einzubetten, stößt dabei jedoch schnell an Grenzen: überlange System Prompts, explodierende Token-Kosten, Fokusverluste und ein überdimensioniertes Standardmodell, das alles regelt und doch nie perfekt passt.

Genau hier setzt Semantic Flow an. Das Open-Source-Framework zerlegt komplexe Abläufe in kleine, klar definierte Activities, die jeweils mit eigenem Prompt, passender Modellwahl und minimalem Kontext arbeiten. Dadurch reduzierst du Kosten, erhöhst die Präzision und behältst die volle Kontrolle über jede Prozessstufe.

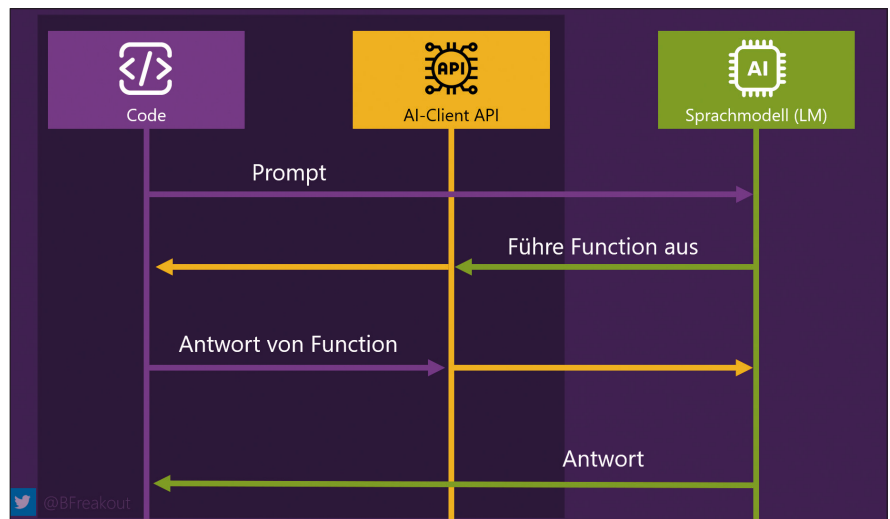
Der Artikel zeigt dir, warum klassische Agent-Frameworks oft nur hübsche Theorie sind, wo Function Calling seine Schattenseiten hat und wie Semantic Flow dank kontrolliertem Kontext und Activity-basiertem Modell-Routing echte Skalierbarkeit erreicht. Kurze, aber aussagekräftige Praxisbeispiele wie ein Pizza-Bestell-Workflow illustrieren, wie du von chaotischen Monolith-Prompts zu schlankem, wartbarem KI-Design wechselst. Nach dieser Einführung weißt du, warum ein „Prompt auf Diät“ und modulare Orchestrierung keine Nice-to-have-Ideen, sondern zwingende Erfolgsfaktoren für jedes ambitionierte KI-Projekt sind.

Die Geburt von Semantic Flow

Das Thema Generative KI ist noch jung, frisch und voller Dynamik. Wir haben die menschliche Sprache als API geknackt. Plötzlich ist sie nicht nur ein Kommunikationsmittel, sondern eine vollwertige Schnittstelle für Software. Was vor Kurzem noch nach Science-Fiction klang, ist heute ein massiver Effizienz- und Qualitätsbooster für unsere Projekte.

Kein Wunder also, dass sich die Entwickler-Community mit großer Begeisterung darauf stürzt. Ich selbst hatte das Glück, eng mit dem Microsoft-Produktteam rund um GitHub Copilot für Visual Studio Code zusammenzuarbeiten. Parallel dazu durfte ich unterschiedliche Kundenprojekte begleiten, in denen generative KI eine zentrale Rolle spielte – meist auf Basis von .NET und mit dem Microsoft Semantic Kernel.

Schon beim ersten echten Projekt wurde mir allerdings klar: So einfach, wie es klingt, ist die Integration generativer KI in reale Softwareprozesse nicht. Gerade wenn es um komplexe Abläufe geht, stößt man schnell an Grenzen. Der erste



Function Calling im Hintergrund, AI-Client-API ist hierbei der Semantic Kernel (Bild 1)

Stolperstein: System Prompts. Die waren so lang, dass regelmäßig das Token-Limit erreicht wurde. Noch schlimmer: Nicht alle Inhalte darin waren relevant für den aktuellen Prozessschritt. Das Sprachmodell verlor oft den Fokus, wurde unpräzise oder einfach zu teuer, denn für alles wurde das leistungsstärkste (und teuerste) GPT4-Sprachmodell genutzt, obwohl das nur an bestimmten Stellen wirklich nötig gewesen wäre.

Ich habe KI-Lösungen gebaut, die für mehrere Tausend Nutzer pro Tag skalieren müssen. Da kann man sich keine verschwenderischen Prompts leisten, weder inhaltlich noch finanziell. Und als dann bei einem zweiten Kundenprojekt mit völlig anderem Use Case dieselben Probleme wieder auftraten, wusste ich: Das ist kein Einzelfall. Das ist ein strukturelles Problem in unserer generativen KI-Welt.

Natürlich habe ich mich im Anschluss durch unzählige Beiträge zu sogenannten „Agenten“ oder sogar „Multi-Agenten“ gearbeitet, aber über 90 Prozent dieser Frameworks sind am Ende nichts anderes als gut klingende Verpackungen ohne echten Kern. Viele lösen keines der tatsächlichen Probleme, die in der Praxis entstehen. Auch im Semantic-Kernel-Umfeld gibt es zwei offizielle Ansätze: das Agent Framework und das Process Framework. Beide klingen auf dem Papier interessant, aber auch sie verfolgen hauptsächlich theoretische Konzepte. In keinem dieser Ansätze habe ich die Antwort auf meine zentrale Frage gefunden:

Wie orchestriere ich einen komplexen KI-gestützten Prozess so, dass jedes Sprachmodell zur richtigen Zeit nur genau die Informationen bekommt, die es wirklich braucht?

Ich wollte keine abstrakte Framework-Spielwiese. Ich wollte eine wiederverwendbare Lösung, die in der Realität funktioniert. Also habe ich mir selbst eine gebaut, pragmatisch, mit echtem Fokus auf Entwicklerfreundlichkeit und Wiederverwendbarkeit. So ist Semantic Flow entstanden: ein Open-Source-Projekt, das komplexe KI-Prozesse in kleine, modulare Activities zerlegt. Jedes Modul arbeitet mit eigenem Prompt, eigener Modellwahl und eigenem Funktionsumfang. Kein Overhead, keine unnötigen Tokens, keine Chaos-Architektur.

Was ist Semantic Flow?

Wenn du in einem realen Projekt mit generativer KI arbeitest, kommst du schnell an einen Punkt, an dem die klassischen Ansätze einfach nicht mehr ausreichen. Ein Sprachmodell wie GPT ist unglaublich leistungsfähig, aber es ist nicht dafür gemacht, ohne Struktur durch einen langen, komplexen Geschäftsprozess zu navigieren. Genau hier setzt Semantic Flow an.

Semantic Flow ist eine modulare Orchestrierungsarchitektur für generative KI-Prozesse. Sie wurde entwickelt, um große, dynamische Abläufe in kleine, überschaubare Einheiten zu zerlegen, sogenannte Activities. Jede dieser Activities übernimmt genau einen konkreten Schritt im Prozess. Dabei bringt jede Activity ihr eigenes Setup mit: einen eigenständigen Prompt, ein passendes Sprachmodell, spezifische Parameter und exakt die Funktionen, die in diesem Moment gebraucht werden. Nicht mehr und nicht weniger.

Warum das so entscheidend ist? In klassischen Setups, auch bei Semantic Kernel selbst, wird oft mit einem überladenen System Prompt gearbeitet. Darin steckt dann gleich der gesamte Prozess: alle Eventualitäten, alle Anforderungen, alle Funktionen. Das wirkt wie ein One-Pager für das Sprachmodell, aber es ist in Wahrheit ein Token-Fresser, ein Performance-Killer und ein Wartungs-Albtraum. Semantic Flow

dreht dieses Prinzip radikal um. Anstelle eines System Prompts für alles gibt es spezialisierte Prompts für jeden Schritt.

Dazu kommt: Die meisten Frameworks gehen davon aus, dass ein einziges Sprachmodell für den kompletten Workflow zum Einsatz kommt. Aber das ist weder effizient noch kostengünstig. Warum solltest du GPT-4 durchgehend nutzen, wenn für einfache Klassifikationen auch Mistral oder Qwen ausreichen? Semantic Flow gibt dir für jede Activity die Freiheit, genau das Sprachmodell zu wählen, das zum Kontext passt. Kurz gesagt: Semantic Flow bringt Ordnung in das KI-Chaos.

Kernkonzepte: Activities, kontrollierter Kontext und die Schattenseite von Function Calling

In klassischen KI-Setups arbeitest du meist mit

- einem großen System Prompt, der alle Eventualitäten eines gesamten Prozesses enthält,
- einem Hauptmodell, das alles abdecken soll,
- und global registrierten Funktionen, die jederzeit zur Verfügung stehen, unabhängig davon, ob sie wirklich gebraucht werden.

Das klingt bequem, führt in der Praxis aber zu mehreren echten Problemen, die vor allem dann auftreten, wenn du skalierende, robuste KI-Features implementieren willst.

Wie Function Calling funktioniert und wo die Herausforderungen liegen

Im Semantic Kernel wird jede Plug-in-Methode oder Kernel Function technisch als Function Call gegenüber dem Sprachmodell registriert. Der Ablauf umfasst fünf Schritte und funktioniert dabei wie folgt, **Bild 1** und **Bild 2** visualisieren es passend:

- Die entsprechende C#-Methode wird automatisch in ein JSON-Schema übersetzt, das die Signatur, Parameter und Beschreibung der Methode enthält. ►

```

Request
Pretty Raw Hex
{
  "tools": [
    {
      "function": {
        "description": "Retrieves the current time and day in CET from germany.",
        "name": "TimeInformationPlugin-GetCurrentTimeAndDay",
        "strict": false,
        "parameters": {
          "type": "object",
          "required": [],
          "properties": {}
        }
      }
    }
  ],
  "type": "function"
},
{
  "role": "user",
  "content": "Welchen Tag haben wir heute?"
}
],
"model": "gpt-4o-mini",
"stream": true,
"stream_options": {
  "include_usage": true
},
"tool_choice": "auto"
}

Response
Pretty Raw Hex Render
21 data: {
  "choices": [
    {
      "content_filter_results": {},
      "delta": {
        "content": null, "refusal": null, "role": "assistant", "tool_calls": [
          {
            "function": {
              "arguments": "", "name": "TimeInformationPlugin-GetCurrentTimeAndDay"
            },
            "id": "call_sGdhG8eKontB7QrWf8mg9V4R", "index": 0, "type": "function"
          }
        ],
        "finish_reason": null, "index": 0, "logprobs": null
      },
      "created": 1753665823, "id": "chatcmpl-By6sZHzYi55ezn0pKXpN0MZgjdyHy", "model": "gpt-4o-mini-2024-07-18", "object": "chat.completion.chunk", "system_fingerprint": "fp_efad92c60b", "usage": null
    }
  ]
},
22
23 data: {
  "choices": [
    {
      "content_filter_results": {},
      "delta": {
        "tool_calls": [
          {
            "function": {
              "arguments": ""
            }
          }
        ]
      }
    }
  ]
}

```

Function Calling nativ in HTTP (Bild 2)

- Dieses JSON-Schema wird bei jedem Prompt zusammen mit dem User-Input an das Sprachmodell übergeben.
- Das Sprachmodell analysiert den Prompt und entscheidet eigenständig, ob und welche Funktion aufgerufen werden soll.
- Falls das Modell eine Funktion für erforderlich hält, erzeugt es einen strukturierten Funktionsaufruf, der dem .NET-Backend übergeben wird.
- Das Backend führt diesen Funktionsaufruf aus und liefert das Ergebnis an das Modell zurück, welches die Antwort damit vervollständigt.

Durch diesen Ablauf wird das Modell mit aktuellen und verlässlichen Informationen versorgt. Das reduziert Halluzinationen deutlich und macht das Sprachmodell zu einem Wissensmodell statt zu einem rein textbasierten Generator.

So weit, so smart. Doch genau an dieser Stelle liegen unsichtbare Kosten, die dich überraschen können:

- **Token-Kosten für das Register-Schema:** Schon die bloße Erklärung jeder Funktion fließt in den Prompt und verursacht laufend Kosten, auch wenn die Funktion nicht ausgelöst wird.
- **Kontextüberlastung:** Jede registrierte Funktion bläht das Kontextfenster auf, auch wenn Nutzeranfragen nur kurz sind.
- **Qualitätsverfall:** Modelle müssen relevanten Kontext finden, und bei langen Prompts mit vielen Funktionen verlieren sie oft den Fokus.

Studien belegen: Mehr Kontext hilft nur begrenzt

Drei aktuelle Benchmarks zeigen klar, dass zusätzliche Tokens schnell zur Last werden. Im CallNavi-Datensatz mit 729 API-Aufgaben fällt selbst GPT 4o von 94 Prozent Trefferquote bei einfachen Ein-Schritt-Calls auf nur noch 72 Prozent, sobald zwölf verschachtelte Aufrufe aus über 100 möglichen Funktionen geroutet werden müssen. Größere offene Modelle wie Llama 3 mit 70 Milliarden Parametern erreichen in komplexeren Fällen teils weniger als 50 Prozent Genauigkeit. Allein die hohe Dichte an registrierten Funktionen bringt das Modell zunehmend ins Straucheln.

ComplexFuncBench liefert weitere belastbare Zahlen. Bei 1000 realen Multi-Step-Szenarien mit bis zu 128 000 Tokens schafft GPT 4o im Schnitt nur rund 60 Prozent Erfolgsquote. Besonders auffällig: Bei klar abgegrenzten Aufgaben wie der Hotelbuchung liegt die Genau-

igkeit noch bei etwa 70 Prozent, fällt bei bereichsübergreifenden Prozessen jedoch unter 45 Prozent. Kleinere Modelle brechen sogar auf unter 10 Prozent ein. Das zeigt, dass lange Kontexte alleine nicht helfen. Die Modelle müssen erst die relevanten Informationen im Prompt finden und korrekt gewichten, was sie offensichtlich nicht zuverlässig beherrschen.

Auch IBM Research bestätigt diese Schwächen mit Long-FuncEval. In dieser Studie verschlechtert sich die Treffergenauigkeit um bis zu 85 Prozent, sobald die Zahl der verfügbaren Tools steigt. Werden zusätzlich die Toolantworten länger oder zieht sich die Konversation über viele Schritte, sinkt die Erfolgsquote weiter deutlich. Die Forscher sprechen von einem kumulativen Effekt, bei dem sich Modelle regelrecht im Kontext verlieren.

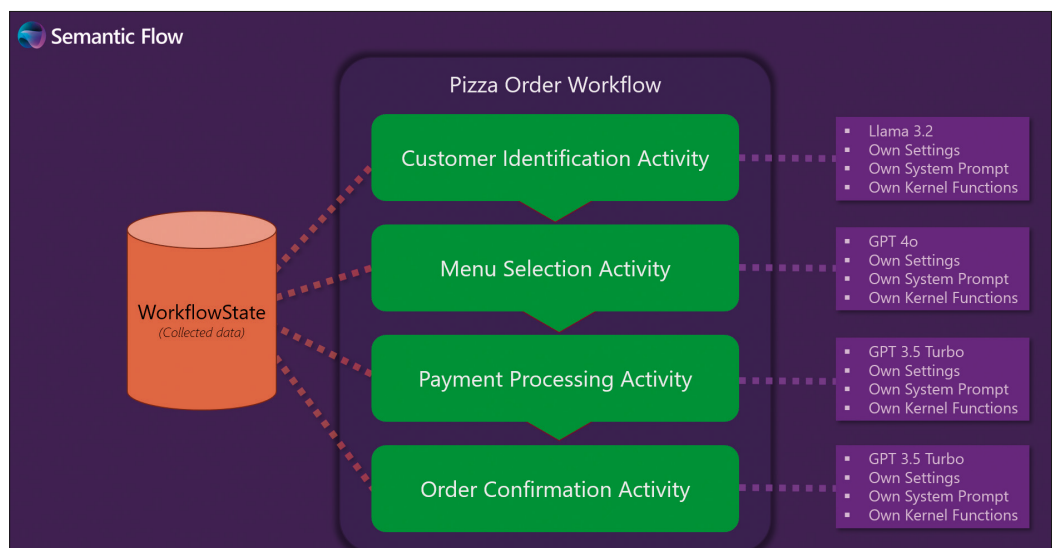
Für die Praxis bedeutet das: Größere Modelle und längere Kontextfenster lösen das Problem nicht. Entscheidend ist, irrelevante Informationen gar nicht erst ins Modell zu laden. Semantic Flow adressiert genau das, indem es Prozesse modularisiert, nur relevante Prompts und Funktionen übergibt und so Fokus, Kostenkontrolle und Modellleistung in Einklang bringt.

Warum große Prompts trotz hoher Kapazität schlecht sind

Ein leistungsfähiges Modell mag 1 Million Tokens verarbeiten können, die Qualität der Antwort leidet trotzdem:

- **Relevanzverfall:** Modelle gewichten frühen und späteren Kontext höher, die Mitte wird oft weniger beachtet.
- **Chunking-Fallen:** Längere Prompts führen bei Token-Limit-Überschreitung zu unvorhersehbarem Verhalten.
- **Komplexitätsparadoxon:** Je mehr Funktionen, desto mehr mentale „Auswahl“ für das Modell, das führt zu Fehlentscheidungen selbst bei guten Modellen.

Dieses Phänomen bestätigt, was viele Entwickler intuitiv spüren: Mehr Kontext ist nicht gleich mehr Qualität, auch wenn es von einem Sprachmodell unterstützt wird.



Der Pizza-Order-Workflow mit Semantic Flow (Bild 3)

Semantic Flow als Architekturantwort

Semantic Flow löst diese Herausforderungen durch eine gezielte Architektur:

- **Isolierte Activities:** Jede Activity arbeitet für sich mit eigenem Prompt, eigenem Modell und nur relevanten Funktionen.
- **Activity-spezifisches Model-Management:** Du kannst zum Beispiel Mistral für einfache Klassifikationen, GPT4o Mini für entscheidungsrelevante Schritte oder externe Modelle wie Llama einsetzen. Auch Einstellungen zu Temperature, TokenLimit, FunctionChoice et cetera sind pro Activity individuell möglich.
- **Minimales Kontextfenster:** Indem nur notwendige Prompts und Function Calls gestellt werden, bleibt der Kontext klein – und die Kosten überschaubar und die Modellleistung zuverlässig.

Ohne Semantic Flow:

- Globale Funktionen
=> hohe Tokenkosten, schwer zu debuggen
- Ein großes Modell => überdimensioniert oder ineffizient
- Langer Prompt => Kontextinflation

Mit Semantic Flow:

- Funktionen nur, wenn relevant
=> geringere Kosten, bessere Trefferquote
- Modell pro Schritt => gezielt und günstig
- Prompt pro Activity => klar, kompakt, wartbar

Vom echten Prozess zur modularen KI-Architektur

Bevor wir mit dem Coden loslegen, nehmen wir uns einen Moment Zeit: Eine gute Architektur beginnt mit dem Verständnis der Realität. Semantic Flow lebt davon, dass Prozesse modularisiert werden. Aber das funktioniert nur dann gut, wenn du weißt, wo und warum du schneiden musst.

Stell dir vor, du sprichst mit einem Kunden, in unserem Fall einem Pizzeria-Betreiber, der seinen Bestellprozess gerne automatisieren möchte. Statt direkt loszuprogrammieren, hörst du erst mal genau zu, wie es in seinem Alltag abläuft.

„Also pass auf“, sagt er, „bei uns beginnt alles mit der Frage: Wer ist der Kunde? Wenn uns jemand anruft, schauen wir als Erstes, ob wir ihn kennen. Ist er ein Bestandskunde, haben wir seine Daten parat: Adresse, Telefonnummer, Lieblingspizzen, sogar Extrawünsche. In dem Fall reicht oft ein einfaches ‚Das Übliche?‘, und wir sind fast durch.“

Wenn es aber ein Neukunde ist, brauchen wir alle Angaben: vollständiger Name, Adresse, Telefonnummer, ohne das geht keine Pizza raus. Da darf nix fehlen. Und wenn was fehlt, fragen wir aktiv nach. Wenn das erledigt ist, geht es ans Menü. Der Kunde wählt, was er will: Standardpizza, extra Belag, glutenfrei, keine Oliven. Du kennst das. Manche wissen genau, was sie wollen, andere lassen sich beraten. Wichtig ist, dass am Ende klar ist, was wir backen sollen.

Dann kommt das Thema Bezahlung. Wir bieten PayPal, Kreditkarte oder Barzahlung an. Manche Kunden fragen nach Apple Pay, das kommt auch vor. Wir fragen einfach gezielt nach der bevorzugten Zahlungsart.

Listing 1: Die CustomerIdentificationActivity.cs

```
using System.ComponentModel;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Connectors
    .AzureOpenAI;
using SemanticFlow.DemoConsoleApp.Services;
using SemanticFlow.DemoConsoleApp.Workflow.DTOS;
using SemanticFlow.Interfaces;
using SemanticFlow.Services;

namespace SemanticFlow.DemoConsoleApp.Workflow;

public class CustomerIdentificationActivity(
    WorkflowService workflowService,
    SessionService sessionService,
    Kernel kernel) : IActivity
{
    public string SystemPrompt { get; set; } =
        File.ReadAllText("../Workflow/
            CustomerIdentificationActivity.SystemPrompt.txt");

    public PromptExecutionSettings
        PromptExecutionSettings { get; set; } =
        new AzureOpenAIPromptExecutionSettings
        {
            ModelId = "gpt-35-turbo",
            FunctionChoiceBehavior =
                FunctionChoiceBehavior.Auto(),
            Temperature = 0.7,
            MaxTokens = 256
        };

    [KernelFunction]
    [Description("Confirms the customer's data for the
        pizza delivery.")]
    public string CustomerDataApproved(
        Customer customer)
    {
        var id = sessionService.GetId();
        var nextActivity =
            workflowService.CompleteActivity(
                id, customer, kernel);

        return @$"(nextActivity.SystemPrompt) ###
            {workflowService.WorkflowState.DataFrom(
                id).ToPromptString()}";
    }
}
```

Und am Ende? Da fassen wir alles nochmal zusammen: Name, Adresse, bestellte Pizza, Preis, Zahlungsart. Dann fragen wir: ‚Passt das so?‘ Und erst wenn der Kunde bestätigt, geht es ab in die Küche. ▶

● Listing 2: Die MenuSelectionActivity.cs

```

using System.ComponentModel;
using System.Text.Json;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Connectors.AzureOpenAI;
using SemanticFlow.DemoConsoleApp.Services;
using SemanticFlow.DemoConsoleApp.Workflow.DTOS;
using SemanticFlow.Interfaces;
using SemanticFlow.Services;

namespace SemanticFlow.DemoConsoleApp.Workflow;

public class MenuSelectionActivity(WorkflowService workflowService,
    SessionService sessionService,
    Kernel kernel) : IActivity
{
    private static readonly List<MenuItem> MenuItems =
    [
        new() { Title = "Margherita", Ingredients =
            "Tomato, Mozzarella, Basil", Price = 8.50 },
        new() { Title = "Pepperoni", Ingredients =
            "Tomato, Mozzarella, Pepperoni", Price = 9.50 },
        new()
        {
            Title = "Quattro Stagioni", Ingredients =
                "Tomato, Mozzarella, Mushrooms, Ham,
                Artichokes, Olives",
            Price = 11.00
        },
        new() { Title = "Marinara", Ingredients =
            "Tomato, Garlic, Oregano", Price = 7.00 },
        new() { Title = "Carbonara", Ingredients = "Bacon,
            Mozzarella, Eggs, Black Pepper", Price = 10.00 }
    ];

    public string SystemPrompt { get; set; } =
        File.ReadAllText("./Workflow/MenuSelectionActivity
            .SystemPrompt.txt");
    public PromptExecutionSettings
        PromptExecutionSettings { get; set; } =
        new AzureOpenAIPromptExecutionSettings
    {
        ModelId = "gpt-4o-mini",
        FunctionChoiceBehavior =
            FunctionChoiceBehavior.Auto(),
        Temperature = 0.1,
        MaxTokens = 256
    };

    [KernelFunction, Description("Checks if a specific
        menu item is available in the restaurant's menu,
        including handling unknown words.")]
    [return: Description("Returns a confirmation message
        if the menu item is available, or a suggestion for
        alternatives if it is not.")]
    public string CheckMenuAvailability(
        [Description("The name of the menu item that the
            customer wants to order.")] string menu)
    {
        var menuItem = MenuItems.FirstOrDefault(item =>
            item.Title.Contains(menu,
                StringComparison.OrdinalIgnoreCase) ||
            menu.Contains(item.Title,
                StringComparison.OrdinalIgnoreCase));
        if (menuItem != null)
        {
            var menuItemJson =
                JsonSerializer.Serialize(menuItem);

            return menuItemJson;
        }

        return "The requested menu item is not available.
            Please choose from our available options.";
    }

    [KernelFunction, Description("Finalizes the
        customer's order by confirming their complete
        menu selection.")]
    [return: Description("A confirmation message
        indicating that the customer's menu selection
        has been approved.")]
    public string MenuSelectionApproved(
        [Description("The full menu selection as
            confirmed by the customer, including all items
            they want to order.")] string fullMenuSelection)
    {
        var id = sessionService.GetId();
        var nextActivity = workflowService.CompleteActivity(
            id, fullMenuSelection, kernel);

        return @$"{nextActivity.SystemPrompt} ###
            {workflowService.WorkflowState.DataFrom(id)
                .ToPromptString()}";
    }
}

```

So erkennst du die logischen Schnittstellen im Prozess

Was der Pizzabäcker hier beschreibt, ist ein natürlicher, menschlicher Ablauf: linear, aber in klaren Etappen. Und ge-

nau hier liegt die Stärke von Semantic Flow: Du kannst jede dieser Etappen im Bestellvorgang als Activity modellieren, die unabhängig arbeitet, aber trotzdem Teil eines durchgehenden Gesprächs ist.

● Listing 3: Die MenuSelectionActivity.SystemPrompt.txt

```
**You are Leonardo, the assistant for La Bella
Pizza.**
The customer has already been successfully identified,
and all necessary contact details are on file.
Your sole task now is to take the customer's order.

### **Your Objectives:**
1. Record the customer's desired items (dishes,
drinks, etc.).
2. Verify the availability of each requested item
using the function **CheckMenuAvailability**.
3. Summarize the entire order at the end and ask if
that's everything the customer would like.
4. Once the customer confirms their order, call the
**MenuSelectionApproved** function with the
complete order details.
```

```
### **Important Notes:**
- Your only task is to accurately take and confirm the
customer's order.
- If an item is unavailable, politely apologize and
offer suitable alternatives.
- Always repeat the order clearly at the end and ask
for confirmation.
- Always maintain the language and form of address
(e.g., informal "you" or formal "you") used earlier
in the conversation. Adapt to the customer's
preferred communication style as established in the
dialogue so far.

### **Example for Starting the Conversation:**
"Alright, let's get started with your order! What can
I note down for you today?"
```

Schauen wir uns an, wie wir diesen echten Prozess in vier Activities überführen:

● CustomerIdentificationActivity

Ziel: Kundendaten vollständig erfassen oder aus bestehendem Profil laden.

Bedeutung: Der Schlüssel für alle Folgeentscheidungen.

● MenuSelectionActivity

Ziel: Auswahl der Speisen, Sonderwünsche, gegebenenfalls Rückfragen.

Bedeutung: Individueller Teil, je nach Kundenbedarf.

● PaymentProcessingActivity

Ziel: Zahlungsart klären und erfassen.

Bedeutung: Verbindlichkeit der Bestellung sicherstellen.

● OrderConfirmationActivity

Ziel: Gesamten Vorgang bestätigen lassen.

Bedeutung: Letzte Möglichkeit für Korrekturen oder Abbruch.

Jede Activity steht für einen eigenständigen Kontextabschnitt, mit eigenen Prompts, Datenanforderungen und möglicher Modellwahl, wie in **Bild 3** gezeigt wird. Damit erreichst du:

- Trennschärfe im Dialog,
- gezielte Modellkonfiguration
- und minimale Komplexität im jeweiligen Prompt.

Semantic Flow in einer .NET-Console-App

Jetzt, wo wir wissen, wie unser Workflow fachlich und logisch aufgebaut ist, starten wir mit der technischen Umsetzung. Und zwar ganz ohne ASP.NET Core, stattdessen setzen wir auf eine schlanke .NET-8-Console-App, perfekt zum Lernen, Testen und Verstehen.

Neues Projekt anlegen

```
dotnet new console -n PizzaFlow.ConsoleApp
cd PizzaFlow.ConsoleApp
```

Notwendige NuGet-Pakete installieren

```
dotnet add package SemanticFlow
dotnet add package Microsoft.SemanticKernel
```

Die erste Activity: CustomerIdentificationActivity

Nachdem das Projekt steht und die grundlegenden Konzepte klar sind, geht es jetzt an die erste echte Implementation. Wir starten mit dem ersten Prozessschritt: Kundenerkennung. Und das ist nicht nur ein formaler Auftakt; dieser Schritt entscheidet darüber, wie der gesamte Rest des Workflows verläuft:

- Ist es ein Bestandskunde?
=> Wir greifen auf bekannte Daten zurück.
- Ist es ein Neukunde?
=> Wir sammeln die benötigten Informationen vollständig.

In dieser Phase geht es darum, dem Sprachmodell genau die Werkzeuge an die Hand zu geben, die für diesen einen Schritt erforderlich sind – nicht mehr, nicht weniger.

Wir erstellen jetzt einen neuen Workflow-Ordner im Projekt. Darin legen wir die *CustomerIdentificationActivity.cs* ebenso ab wie eine separate Textdatei für den Prompt mit dem Namen *CustomerIdentificationActivity.SystemPrompt.txt*. Die Abtrennung des Prompts in eine Textdatei ist nicht zwingend, aber sie ist in der Praxis bewährt. Denn so kannst du Prompts unabhängig vom Code bearbeiten, anpassen oder versionieren, ohne dass dein Editor sich mit C#-Syntaxprüfungen meldet.

Die Prompt-Strategie

Der Prompt selbst enthält genau das, was für diesen Teilprozess nötig ist. Daher gibt es kein globales Prompt-Wirrwarr. Das Herzstück steht in *CustomerIdentificationActivity.SystemPrompt.txt*. Eine vereinfachte Variante aus einem echten Projekt könnte so aussehen. Er ist stark verkürzt, um den ►

Listing 4: Der gesamte Code der Console-App

```

using Azure.Identity;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using OllamaApiFacade.Extensions;
using SemanticFlow.DemoConsoleApp.Services;
using SemanticFlow.DemoConsoleApp.Workflow;
using SemanticFlow.Extensions;
using SemanticFlow.Services;

var azureOpenAiEndpoint =
    "https://ai-rag-workshop.openai.azure.com";
var azureOpenAiDeploymentNameGpt4oMini =
    "gpt-4o-mini";
var azureOpenAiDeploymentNameGpt35 = "gpt-35-turbo";

var builder = Kernel.CreateBuilder();

builder.Services.AddKernel()
    .AddAzureOpenAIChatCompletion(
        azureOpenAiDeploymentNameGpt4oMini,
        azureOpenAiEndpoint, new DefaultAzureCredential(),
        modelId: "gpt-4o-mini")
    .AddAzureOpenAIChatCompletion(
        azureOpenAiDeploymentNameGpt35,
        azureOpenAiEndpoint, new DefaultAzureCredential(),
        modelId: "gpt-35-turbo");

builder.Services.AddLogging();
builder.Services.AddTransient<SessionService>();

builder.Services.AddKernelWorkflow()
    .StartWith<CustomerIdentificationActivity>()
    .Then<MenuSelectionActivity>()
    .Then<PaymentProcessingActivity>()
    .EndsWith<OrderConfirmationActivity>();

// Analyze with Burp Suite the Semantic Kernel backend
// communication
// builder.Services.AddProxyForDebug();

var kernel = builder.Build();

var sessionService =
    kernel.GetRequiredService<SessionService>();
var id = sessionService.GetId();

var workflowService =
    kernel.GetRequiredService<WorkflowService>();
var chatHistory =
    new ChatHistory("Init system prompt");

while (true)
{
    // Necessary for this stateful example
    var kernelClone = kernel.Clone();

    var currentActivity = workflowService
        .GetCurrentActivity(id, kernelClone);

    var systemPrompt = currentActivity.SystemPrompt +
        " ### " + workflowService.WorkflowState
        .DataFrom(id).ToPromptString();

    var systemChatMessage = new ChatMessageContent(
        AuthorRole.System, systemPrompt);
    chatHistory[0] = systemChatMessage;

    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.Write("You: ");
    string userInput = Console.ReadLine();
    chatHistory.AddUserMessage(userInput);
    Console.ResetColor();

    if (userInput?.ToLower() == "exit")
        break;

    var chatCompletions = await kernelClone
        .GetChatCompletionForActivity(currentActivity)
        .GetChatMessageContentsAsync(new ChatHistory(
            chatHistory), currentActivity
            .PromptExecutionSettings, kernelClone);

    var chatResponse =
        chatCompletions.First().ToChatResponse();
    chatHistory.AddAssistantMessage(
        chatResponse.Message.Content);

    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine(
        $"AI Pizza dealer: {chatResponse.Message.Content}");
    Console.ResetColor();
}

```

Rahmen des Artikels hier nicht zu sprengen, und es fehlen viele Details, aber nur zur Demonstration:

"Du bist Leonardo, ein KI-Assistent der Pizzeria La Bella Pizza.

Wenn der Kunde bekannt ist, begrüße ihn mit Namen und lade vergangene Vorlieben aus dem Speicher.
Wenn der Kunde neu ist, erfasse folgende Informationen:
Vollständiger Name (Vor- und Nachname)
Lieferadresse (Straße, Hausnummer, Postleitzahl, Ort)

Telefonnummer
Bestätige am Ende immer die Daten und frage: Sind diese
Angaben korrekt?
Wenn bestätigt, rufe die Funktion `CustomerDataApproved`
auf und fahre fort.“

In einer echten Anwendung könnte hier auch ein RAG-Call auf ein Kunden-CRM eingebunden sein, Semantic Flow lässt diese Erweiterung später problemlos zu. Es wäre außerdem ebenfalls möglich, einen globalen System Prompt festzulegen, der prozessübergreifend die wichtigsten Aspekte beinhaltet. Die Activity-Prompts können in diesem Fall als Schnipsel eingewoben werden. Semantic Kernel unterstützt uns hierbei mit einer Prompt-Template-Engine. In unserem vereinfachten Beispiel hat die Activity bereits alle notwendigen Anweisungen.

In Semantic Flow dreht sich alles um den Begriff Activity, und mit dem Interface `IActivity` wird eine einfache C#-Klasse zu einem vollwertigen Baustein im orchestrierten KI-Prozess. Dieses Interface verlangt von dir zwei Dinge: einen Prompt und Modelleinstellungen. Damit ist sichergestellt, dass jede Activity für sich stehen kann, unabhängig von anderen Prozessschritten.

Die gesamte Implementierung der `CustomerIdentificationActivity`-Klasse findest du in [Listing 1](#).

Was genau macht eine Klasse zur Semantic Flow Activity?

Durch das Interface `IActivity` definierst du:

- **SystemPrompt:** ein String, der den Prompt-Text für genau diesen Prozessabschnitt enthält.
- **PromptExecutionSettings:** Hier wird gesteuert, welches Sprachmodell, welche Temperatur und wie viele Tokens verwendet werden sollen und wie das Function Calling für genau diesen Step aussehen soll.

So entsteht ein kontrollierter, modularer Prozess. Semantic Flow selbst kümmert sich dann zur Laufzeit darum, immer genau die richtige Activity mit den passenden Einstellungen aufzurufen.

Wie wählt Semantic Flow das richtige Sprachmodell?

In `PromptExecutionSettings` gibst du unter anderem die `ModelId` an. Diese `ModelId` ist der Name des Modells, das du zuvor im Semantic Kernel registriert hast, etwa mit Azure OpenAI oder einem Ollama-Backend. Das funktioniert zum Beispiel so:

```
builder.Services.AddKernel().AddAzureOpenAIChatCompletion(
    "gpt-35-turbo", endpoint, new DefaultAzureCredential(),
    modelId: "gpt-35-turbo").AddAzureOpenAIChatCompletion(
    "GPT-4o-mini", endpoint, new DefaultAzureCredential(),
    modelId: "GPT-4o-mini");
```

Wenn du dann `"gpt-35-turbo"` in deiner Activity hinterlegst, weiß Semantic Kernel genau, welches Modell gemeint ist.

Wichtig zu wissen: Wenn du keine `ModelId` explizit angibst, verwendet Semantic Kernel automatisch den zuletzt registrierten Connector. Das ist ein Standardverhalten des Semantic Kernel, nicht von Semantic Flow. Semantic Flow macht nur sichtbar, was sonst implizit geschieht, und gibt dir die Kontrolle zurück, ohne dass du dich zusätzlich darum kümmern musst.

Und was ist mit Funktionen? Die Kernel Functions

Optional kannst du in jeder Activity eigene Methoden definieren, die mit dem `[KernelFunction]`-Attribut gekennzeichnet sind. Diese Methoden erscheinen dann für das Sprachmodell als „callable tools“, also Function Calls, die es bei Bedarf vorschlagen kann. Beispiel:

```
[KernelFunction]
[Description("Confirms the customer's data for the pizza
    delivery.")]
public string CustomerDataApproved(Customer customer)
{
    // ...
}
```

Aber: Das Modell entscheidet nicht automatisch, wann es diese Funktion ausführen soll. Du musst es im Prompt dazu anleiten. Das Sprachmodell muss also selbst den Punkt erkennen: „Okay, alle Daten sind vollständig. Jetzt kann ich die Funktion `CustomerDataApproved` aufrufen.“

Genau das steuerst du über das `Description`-Attribut und einen gut gestalteten Prompt, also eine klare Anweisung an das Sprachmodell, wann und unter welchen Bedingungen der Prozess als abgeschlossen gilt.

Die Rolle des WorkflowService, das Herzstück im Ablauf

Sobald das Sprachmodell diesen Punkt erreicht hat und die Funktion ausgelöst wird, ruft Semantic Kernel deine C#-Methode auf, in unserem Fall die `CustomerDataApproved`.

Hier übergibst du dem `workflowService` die Verantwortung, die Activity offiziell zu beenden:

```
var nextActivity = workflowService.CompleteActivity(
    id, customer, kernel);
```

Was passiert hier?

- Der aktuelle Schritt wird abgeschlossen.
- Die übergebenen Kundendaten werden im internen State Manager von Semantic Flow gespeichert.
- Die nächste Activity im Workflow wird geladen.
- Du bekommst ihren Prompt und alle zugehörigen Einstellungen zurück.

Dieses zentrale State-Management erlaubt dir, über mehrere Activities hinweg ein konsistentes Bild zu halten, ohne dass du selbst mühsam Kontext aufbauen oder in Session-Objekten mitführen musst. ►

Bonus: Few-Shot Learning und RAG mit WorkflowState

Das wahre Potenzial zeigt sich dann im weiteren Verlauf: Alle Daten, die bei *CompleteActivity* übergeben wurden, kannst du später wieder abrufen und dem nächsten Prompt als zusätzlichen Datenkontext mitgeben.

Das fühlt sich für das Sprachmodell so an, als würde es sich an den bisherigen Dialogverlauf erinnern, dabei handelt es sich um eine bewusst gesteuerte Form von Few-Shot Learning.

Die zweite Activity: Die Menüauswahl

Nachdem die Kundenerkennung erfolgreich abgeschlossen ist, geht es direkt weiter in den nächsten Schritt unseres Pizza-Workflows: die Bestellung. Wenn du mit einem echten Pizzadealer sprechen würdest, würde er wahrscheinlich etwas sagen wie:

„Alles klar, was darf es heute sein? Soll es wieder die Quattro Stagioni mit extra Oliven sein, oder probierst du mal was Neues?“ Genau dieses Verhalten bildet Semantic Flow jetzt im Code ab. Denn nachdem klar ist, wer bestellt, ist der nächste logische Schritt: Was wird bestellt?

Im Beispiel haben wir eine eigene Activity erstellt, die genau diesen Bestellprozess abdeckt. Der Prompt ist darauf fokussiert, die Wünsche des Kunden abzufragen, die Verfügbarkeit dieser Optionen zu prüfen und die finale Auswahl zu bestätigen.

Was in dieser Activity besonders interessant ist:

- Die Speisekarte liegt direkt in der Klasse als Liste von *MenuItem*-Objekten vor.
- Über die KernelFunction *CheckMenuAvailability* kann das Modell prüfen lassen, ob ein gewünschtes Gericht verfügbar ist.
- Sobald die Bestellung vollständig ist, ruft das Modell *MenuSelectionApproved* auf und übergibt die finale Auswahl.
- Der aktuelle Prozessabschnitt wird damit abgeschlossen, und alle Bestelldaten werden in den State von Semantic Flow geschrieben.

Das ermöglicht, dass der nächste Schritt, etwa die Zahlungsabwicklung, genau auf diesem Ergebnis aufbaut.

Die System-Prompt-Datei dieser Activity beschreibt dabei ganz klar, wie Leonardo, unser virtueller Pizzaassistent, den Dialog führen soll: freundlich, professionell und mit klarer Bestelllogik. Ist ein Menüpunkt nicht verfügbar, wird eine Alternative vorgeschlagen. Und erst wenn alles stimmt, geht es weiter.

Das Modell, das hier verwendet wird, ist bewusst GPT-4o-mini. Für einen Bestelldialog reicht das völlig aus und spart im Vergleich zu GPT-4 bares Geld, ohne Qualitätseinbußen zur Folge zu haben.

So fügen wir schrittweise einen strukturierten KI-Prozess zusammen, bei dem jede Phase sauber getrennt ist, fachlich wie technisch. Die Menüauswahl ist abgeschlossen, wenn der Kunde zufrieden ist und alles bestätigt hat. In [Listing 2](#) und [Listing 3](#) sind die Implementierung und der System Prompt unserer *MenuSelectionActivity* zu sehen. [Listing 4](#) enthält den

gesamten Code der *Program.cs*, dort erkennt man mit der *AddKernelWorkflow*-Methode, dass hier Semantic Flow eingebunden wird, und via *Fluent-Interfaces* kann entschieden werden, welche Activities zu welchem Zeitpunkt vom Workflow aufgerufen werden sollten.

Fazit: Warum Semantic Flow den Unterschied macht

Wir haben gemeinsam gesehen, wie sich ein scheinbar einfacher Pizza-Bestellprozess in vier klar umrissene Activities zerlegen lässt und wie dir Semantic Flow dabei hilft, jede Phase mit dem exakt passenden Prompt, Modell und Funktionsset auszustatten. Dadurch

- reduzierst du TokenKosten und Laufzeiten,
- behältst jederzeit die volle Kontrolle über Kontext und Zustandsverwaltung,
- und bekommst einen Workflow, der nicht irgendwann in Prompt-Chaos versinkt, sondern sauber wartbar bleibt.

Das wichtigste Learning dabei: Orchestrierung ist keine Magie. Sobald du fachliche Schnittstellen erkennst und sie in eigenständige Activities überführst, spielt generative KI ihre Stärken aus, ganz ohne überladene System Prompts oder globale Funktionswüsten.

Alle Listings aus dem Artikel findest du in vollständiger Form in einem öffentlichen Repository [1]. Die Console-App liegt unter *samples/SemanticFlow.DemoConsoleApp*. Dort kannst du das Projekt sofort klonen, *dotnet restore* ausführen und loslegen.

Falls du statt der Konsole lieber gleich eine komfortable Chat-Oberfläche nutzen möchtest, gibt es im Repo auch ein ASP.NET Core Web API inklusive Ollama API Facade [2]. Damit koppelst du Semantic Flow in Sekunden an Tools wie Open WebUI an, ohne selbst ein UI bauen zu müssen. Das komplette Beispiel findest du im Repo unter *samples/SemanticFlow.DemoWebApi*.

Mein Rat zum Abschluss: Fang klein an. Bau erst eine Activity, teste sie und füge dann Schritt für Schritt neue Bausteine hinzu. So bleibt dein Projekt schlank, nachvollziehbar und kosteneffizient. Sobald du merkst, wie geschmeidig Semantic Flow große Prozesse handhabt, willst du nie wieder einen monolithischen Super-Prompt schreiben. ■

[1] Beispiel-Repository zum Artikel,

<https://github.com/GregorBiswanger/SemanticFlow>

[2] Gregor Biswanger, LLM-API – Ready to go!,

<https://youtu.be/o8qbaQtrqHA>



Gregor Biswanger

ist Microsoft MVP und freier Berater, Trainer, Autor und Sprecher. Er berät Unternehmen, Organisationen und Agenturen rund um die Themen Softwarearchitektur, Cloud-, Web- und Cross-Platform-Entwicklung mit C# und JavaScript/TypeScript.

KI-GESTÜTZTE DEVELOPMENT-WORKFLOWS

Vom Co-Pilot zum Co-Worker

KI-Agenten sind längst keine Zukunftsmusik mehr: Ein detaillierter Blick auf den aktuellen Stand der Tools und Technologien von Microsoft für KI-gestützte Softwareentwicklung.

Künstliche Intelligenz (KI) ist seit Jahren ein zentrales Thema in der Informatik. Allerdings ist sie auch mit enormem Hype und Buzzwords verbunden – vieles klingt dabei besser, als es ist. So gibt es auch zahlreiche kritische Stimmen in der Entwickler-Community, die ein großes Risiko bei KI-generiertem Code sehen. Andererseits warnen einige, die zukünftige Rolle des Softwareentwicklers sei gefährdet, und glauben, alles werde bald nur noch durch KI entwickelt. Vor diesem Hintergrund werden wir uns in diesem Artikel mit dem aktuellen Stand der KI-basierten Softwareentwicklung im Microsoft-Technologie-Umfeld auseinandersetzen und Chancen sowie Risiken identifizieren.

GitHub als Technologie-Träger

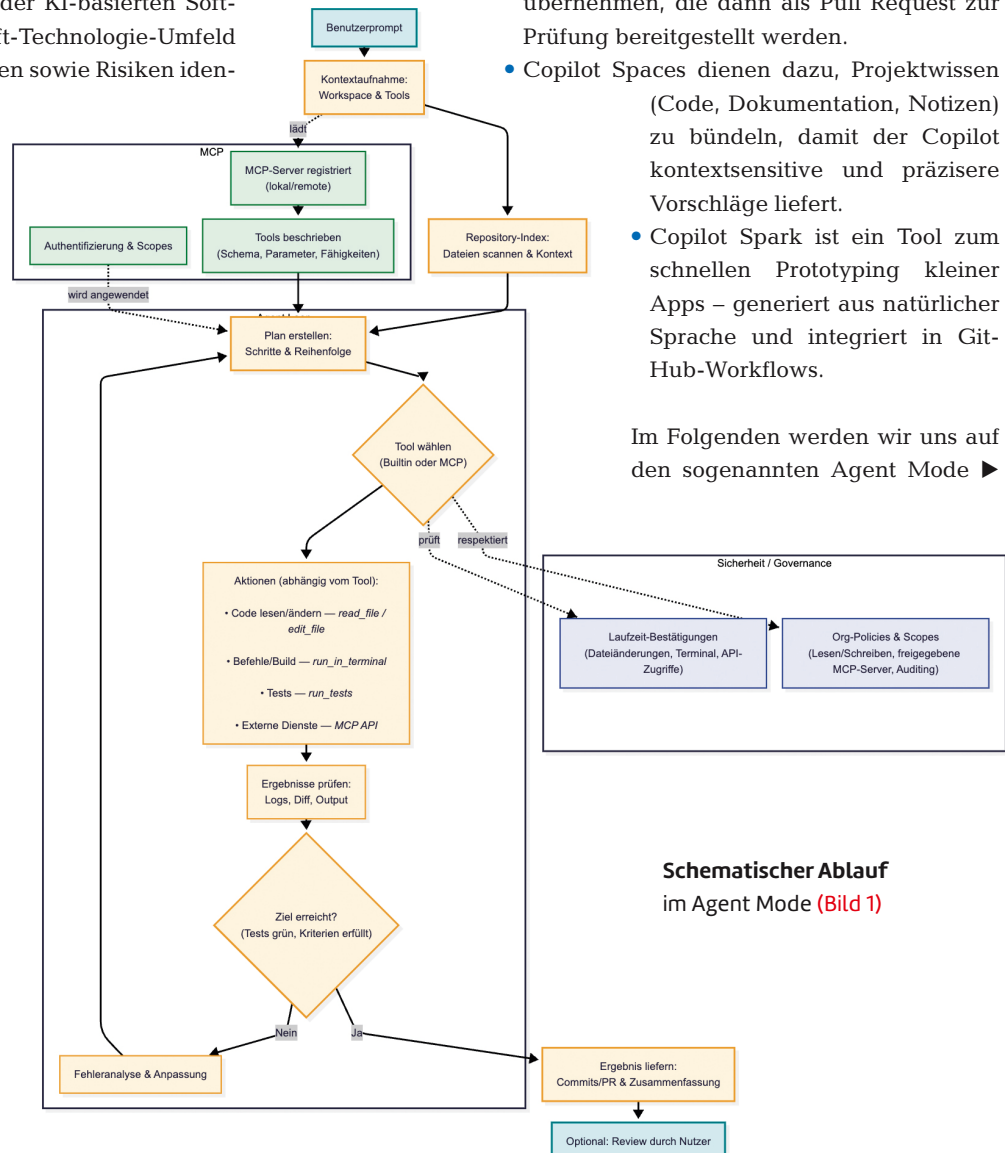
Microsoft hat bei nahezu all seinen Produkten und Services den Zusatz „Copilot“ eingebracht – von der Marketingseite wird Copilot also fast schon überstrapaziert. Die Ursprünge liegen beim ersten und bis heute erfolgreichsten Co-Piloten: GitHub Copilot. GitHub hat sich seit der Übernahme durch Microsoft stark gewandelt – vom reinen Code-Hosting mittels Git hin zu einer umfassenden DevOps-Umgebung. Dabei wurden zwei Schwerpunkte kontinuierlich ausgebaut: Security und Künstliche Intelligenz.

GitHub Copilot ist ein cloubasierter KI-Dienst, der auf mehrere Modelle zugreift und in verschiedenen Abo-Modellen angeboten wird. Der Funktionsumfang hat sich stark erweitert, sodass eine kurze Einordnung der Begriffe nötig ist:

- Copilot ist der Überbegriff für alle KI-Features, die GitHub mittels cloubasierter KI-Modelle bereitstellt.
- Der häufigste Einsatz erfolgt über die Integration in IDEs beziehungsweise Editoren. GitHub bietet Erweiterungen für VS Code, Visual Studio und JetBrains-IDEs. Von Code-Completion über Fragen zur Codebasis bis hin zur Ausführung von Entwicklungsaufgaben im Agent Mode ist vieles möglich.
- Mit der Coding-Agent-Funktionalität (Preview) kann ein Agent autonom und asynchron Aufgaben übernehmen, die dann als Pull Request zur Prüfung bereitgestellt werden.

- Copilot Spaces dienen dazu, Projektwissen (Code, Dokumentation, Notizen) zu bündeln, damit der Copilot kontextsensitive und präzisere Vorschläge liefert.
- Copilot Spark ist ein Tool zum schnellen Prototyping kleiner Apps – generiert aus natürlicher Sprache und integriert in GitHub-Workflows.

Im Folgenden werden wir uns auf den sogenannten Agent Mode ►



Schematischer Ablauf
im Agent Mode (Bild 1)

respektive den Coding Agent fokussieren, bei dem GitHub Copilot in der Lage ist, Aufgaben selbstständig zu erledigen und Ergebnisse zu verifizieren. Agentenbasierte Entwicklungsfunktionen gibt es bei verschiedenen Herstellern und Open-Source-Projekten. Microsoft hat kürzlich einen wichtigen Schritt vollzogen: Die „Copilot Chat Extension“ für VS Code wurde unter einer MIT-Lizenz als Open Source bereitgestellt, und relevante Komponenten sollen zudem in den Kern von VS Code integriert werden. Obwohl einige Wettbewerber-Lösungen auf VS Code basieren oder sogar VS Code geforkt haben, sichert sich Microsoft durch diesen Schritt seine starke Marktposition – insbesondere im Bereich der KI-Unterstützung für Entwicklerwerkzeuge.

GitHub Copilot Agent Mode: Vom Prompt zum fertigen Ergebnis

Der Agent Mode in GitHub Copilot hebt die Arbeit mit KI-Assistenten auf eine neue Stufe. Während der klassische Copilot vor allem Code-Vervollständigungen vorschlägt oder kleinere Abschnitte editiert, agiert der Agent Mode wie ein kleiner „Autopilot“ für komplexere Entwicklungsaufgaben. Statt lediglich einen Funktionsrumpf zu generieren, plant der Agent eigenständig die nötigen Schritte, führt sie iterativ aus und überprüft die Ergebnisse – bis das gewünschte Ziel erreicht ist.

Der Prozess beginnt mit einem Benutzerprompt: Entwicklerinnen oder Entwickler beschreiben, was sie erreichen möchten – etwa „Füge eine Suchfunktion hinzu“ oder „Führe ein Refactoring dieses Moduls durch“. GitHub Copilot Agent Mode sammelt daraufhin Kontextinformationen aus dem aktuellen Workspace und indexiert die Repository-Dateien, um zu verstehen, welche Stellen des Codes betroffen sind. Mittlerweile unterstützen GitHub und Azure DevOps das Remote-Indexing der Source-Code-Repositories. Somit muss das Indexieren nur einmal zentral auf dem Server erfolgen und nicht auf jedem Client. Zudem kann dem Agent unter die Arme gegriffen werden, und der Prompt kann bereits Verweise auf Code-Dateien mit der Hash-Tag-Referenz enthalten.

Nachdem der Kontext definiert ist, erstellt der Agent einen Aktionsplan, der aus mehreren Teilschritten besteht: Dateien

● Du möchtest gerne mehr erfahren?

Das freut uns, denn ab Erscheinen des vorliegenden Artikels veröffentlichen wir auf developer-world.de/insights weitere Beiträge, die den Artikel fortsetzen und ausschließlich online zur Verfügung stehen. Derzeit geplant sind folgende Beiträge:

- CoPilot Agent Mode – Konfiguration und Best Practices
- MCP – Integration und Nutzung
- Eigener MCP-Server mit .NET

lesen und analysieren, Code-Änderungen durchführen, Build- oder Testläufe starten und Ergebnisse auswerten.

Dieser „Agent Loop“ ist besonders spannend: Nach jedem Schritt prüft der Agent, ob das Ziel erreicht wurde – beispielsweise ob alle Tests erfolgreich durchlaufen oder Build-Fehler behoben sind. Falls nicht, passt er den Plan an, führt weitere Änderungen durch oder schlägt der Benutzerin beziehungsweise dem Benutzer Alternativen vor. Dieser Zyklus aus Planen, Ausführen, Prüfen und Korrigieren läuft so lange, bis das definierte Ziel erfüllt ist oder der Nutzer eingreift.

Ein wichtiger Baustein ist der Tool-Zugriff. GitHub Copilot nutzt hierfür interne Funktionen (zum Beispiel Dateien lesen/schreiben, Terminal-Befehle ausführen) und kann über das Model Context Protocol (MCP) auch Programme ausführen oder externe Datenbankquellen beziehungsweise APIs einbinden – etwa um Issues aus GitHub abzurufen oder Ergebnisse aus einem Test-Framework zu verarbeiten. MCP ist dabei ein offener Standard, der beschreibt, wie KI-Modelle mit Tools interagieren und welche Parameter und Berechtigungen dabei gelten sollen.

Selbstverständlich spielt die Sicherheit dabei eine große Rolle. Bevor GitHub Copilot kritische Aktionen wie zum Beispiel Terminalkommandos ausführt, erhalten Entwicklerinnen und Entwickler eine Vorschau und können den Schritten zustimmen oder Änderungen an ihnen vornehmen. Auf Organisationsebene lassen sich darüber hinaus Policies und

● **Tabelle 1: Agent Mode versus Coding Agent**

Merkmal	Agent Mode	Coding Agent
Wo läuft er?	Direkt im Editor oder in der lokalen Entwicklungsumgebung (zum Beispiel VS Code).	Vollständig in der Cloud, mittels GitHub-Actions-Workflows.
Wie interaktiv?	Synchron: Der Entwickler erstellt einen Prompt, und der Agent macht Vorschläge und Änderungen. Man kann direkt Einfluss auf die Änderungen nehmen, so wie in einer Pair-Programming-Session.	Asynchron: Dem Coding Agent wird eine Aufgabe zugeteilt. Dies kann über die Zuweisung eines Issues erfolgen oder über einen Kommentar mit der entsprechenden Referenzierung (@copilot). Der Agent arbeitet im Hintergrund, erzeugt Pull Requests und reagiert auf Pull-Request-Kommentare.
Parallelität	Es wird an einer Aufgabe gearbeitet; die IDE / der Editor ist damit belegt.	Es können beliebig viele Aufgaben parallel gelöst werden (dedizierte Branches / Pull Requests).
Typ der Ausgabe	Direktes Bearbeiten der Dateien, Ausführen von Befehlen im Editor, Tests und anderes mehr.	Pull Request mit Commits, Beschreibung, Review-Kommentaren und Überarbeitung durch Feedback.
Beste Aufgabenart	Für interaktive und iterative lokale Änderungen – ideal für komplexe Tasks, bei denen kurze Zyklen für Rückfragen und Änderungen nötig sind.	Für „Low-to-medium Complexity“-Aufgaben, die man delegieren kann, während man sich auf größere Dinge konzentriert: Bugfixes, Refactorings, Test-Erweiterungen, Dokumentation, technische Schulden reduzieren etc.

Scopes festlegen: Administratoren entscheiden beispielsweise, ob der Agent nur lesen oder auch schreiben darf, welche MCP-Server genutzt werden können und ob Aktionen protokolliert werden müssen.

Das Ergebnis ist ein intelligenter, kontrollierbarer Workflow: Entwicklerinnen und Entwickler formulieren ein Ziel, Copilot plant und arbeitet es selbstständig ab. Dieser Prozess ist in **Bild 1** veranschaulicht. Wichtig hierbei ist, dass der Benutzer immer die Kontrolle darüber behält. Zudem können einzelne Schritte rückgängig gemacht werden, und wer für jede Änderung einen Topic- respektive Feature-Branch verwendet, behält im eigenen Entwicklungsprozess auch die Kontrolle über die Integration dieser Änderungen.

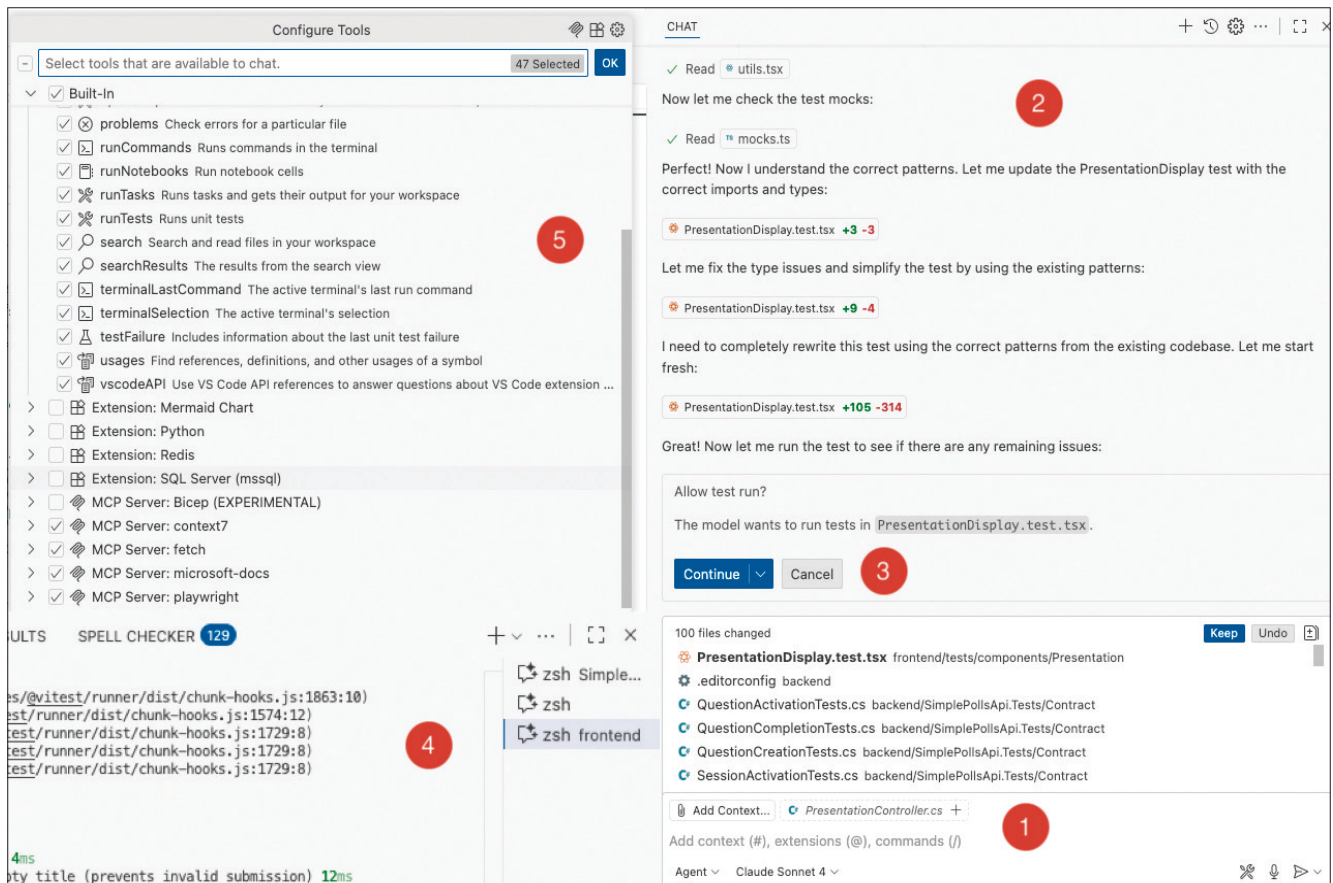
In **Bild 2** sind die typischen Agent-Mode-Funktionen in VS Code ersichtlich. In der Eingabeaufforderung (1) können neben dem Prompt auch zusätzlicher Kontext, Extensions und Commands angegeben werden. Im Chat-Verlauf (2) sind die Chat-Diskussion sowie die Interaktionen mit Dateien, Verzeichnissen und Kommandos ersichtlich. Soll ein Tool oder ein Built-in-Kommando ausgeführt werden, so muss dies erst durch den Benutzer freigegeben werden (3). Ausführbare Aktionen werden in einer eigenen Shell-Session ausgeführt (4). Der Agent ist über die Shell-Integration in der Lage, das Ausführen des Kommandos abzuwarten und den Output zu analysieren. Die Entwicklerin oder der Entwickler hat die volle Kontrolle darüber, welche Tools eingesetzt werden dürfen. Diese Einstellung ist unter Punkt (5) ersichtlich.

GitHub Coding Agent: dein asynchroner Teamkollege

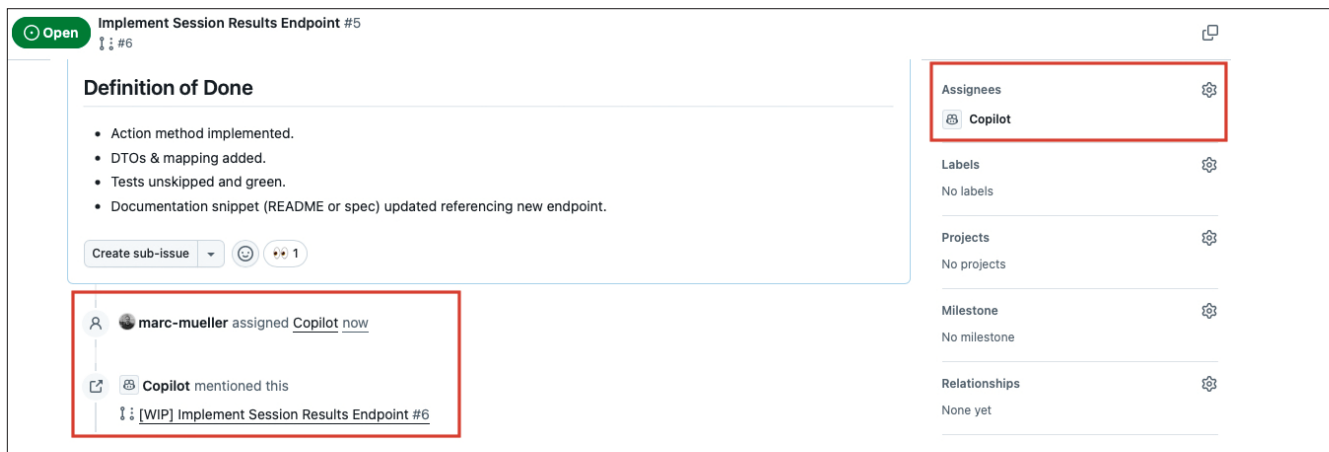
Während sich der Agent Mode im vorherigen Beispiel wie ein Echtzeit-Copilot verhält, der direkt im Editor arbeitet und einen unterstützt, ist der Coding Agent eher wie ein Teammitglied zu verstehen, das in der Zwischenzeit Aufgaben übernimmt und eigenständig über GitHub arbeitet. **Tabelle 1** veranschaulicht dies in einer Gegenüberstellung.

Bevor wir weiter in die Funktionen des Coding Agents eintauchen, möchte ich erst eine ganz wichtige Implikation des Coding Agents verdeutlichen. Bislang waren wir mehr oder weniger frei in der Entscheidung, welchen Editor und welche DevOps-Lösung wir verwenden, da die GitHub-Copilot-Extension im Editor läuft und mit den Cloud-Services kommuniziert. Der Coding Agent läuft im GitHub-Actions-Context und erstellt Pull Requests in GitHub. Dies bedeutet zwangsläufig, dass das Code-Repository in GitHub gehostet werden muss. Somit haben Azure-DevOps-Benutzerinnen und -Benutzer das Nachsehen.

Microsoft arbeitet seit Jahren an der Unternehmensstrategie „Better Together“, die Azure DevOps und GitHub mit Integrationen näher zusammenbringt. Wer Azure DevOps weiterhin nutzen möchte, jedoch nicht auf das Coding-Agent-Feature verzichten kann, muss das entsprechende Git-Repository nach GitHub migrieren. Das bedeutet auch, dass Pull Requests neu in GitHub durchgeführt werden. Work Items, Pipelines et cetera können weiterhin in Azure DevOps ►



Agent Mode in VS Code (**Bild 2**)



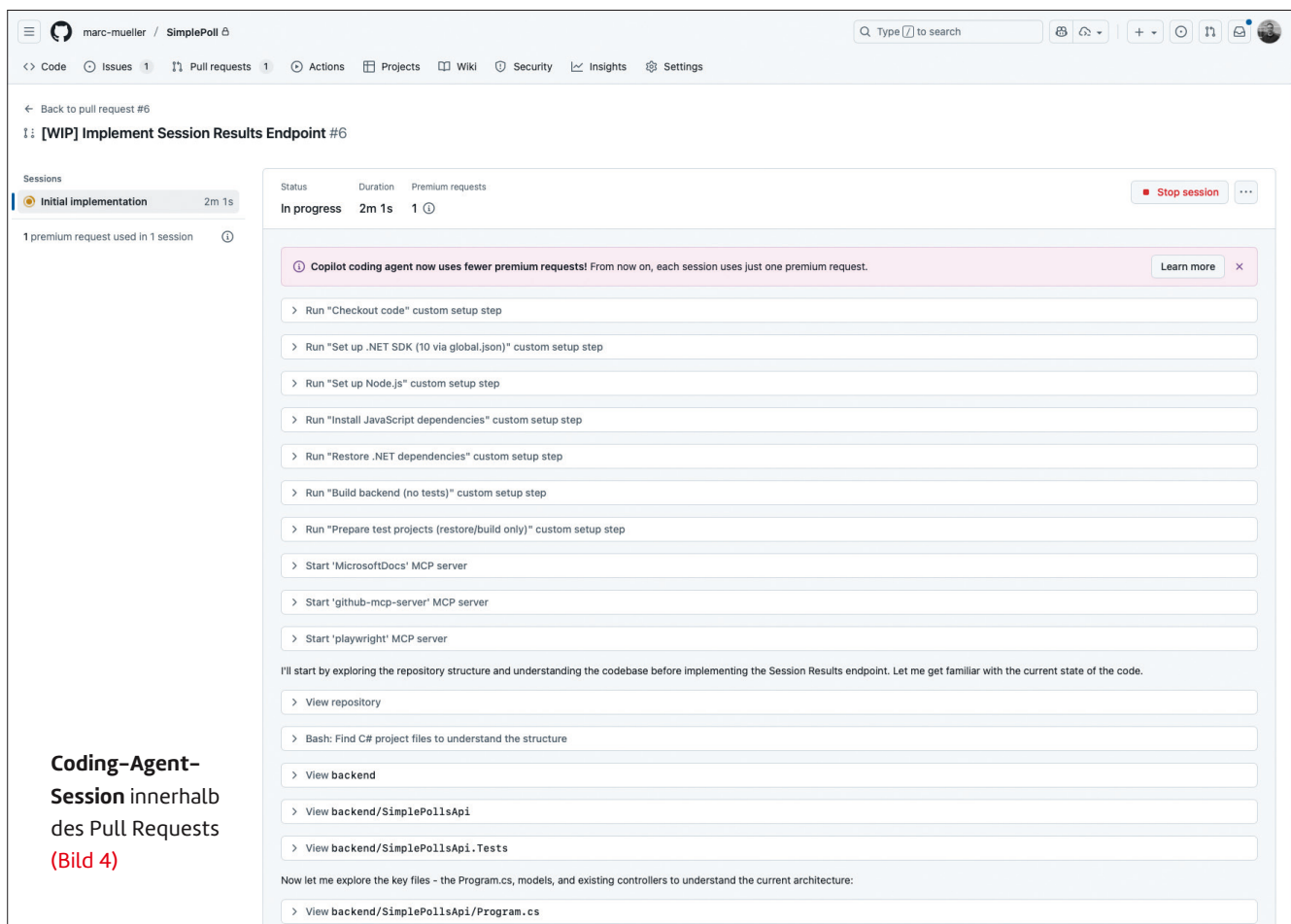
Issue an Copilot zuweisen (Bild 3)

verwendet werden. Über die Extensions ist alles ineinander integriert, Commits referenzieren Work Items, und Pipeline Runs werden aus GitHub getriggert. Leider müssen trotz aller Integrationen zwei Umgebungen verwaltet werden. Dafür bekommt man das Beste aus beiden Welten.

Wenn man nun das eigene Repository auf GitHub betreibt und einen entsprechenden GitHub-Copilot-Plan hat, kann der Coding Agent genutzt werden. Der Coding Agent ist grundsätzlich ein bezahlter Dienst, dessen Währung die so-

genannten Premium Requests sind. Eine gewisse Anzahl an Premium Requests steht einem je nach GitHub-Copilot-Plan zur Verfügung. Sind diese aufgebraucht, können weitere Premium Requests dazugekauft werden. Keine Angst: Auch hier behält man die volle Kontrolle mittels Freigabe und Budget.

Im Unterschied zu den Anfängen des Coding Agents, wo eine einzelne Session 30 bis 50 Premium Requests „verbraten“ hat, zeigt sich Microsoft mittlerweile von der spendablen Seite, und es wird pro Session nur noch ein Premium Request



Coding-Agent-Session innerhalb des Pull Requests (Bild 4)

verrechnet. Mit dieser Änderung sind wir in einem sehr lukrativen Bereich angekommen, da viel Arbeit für wenig Geld verrichtet wird. Die Kosten für einen Premium-Request liegen derzeit bei 0,04 US-Dollar. Streng genommen sind das aber nicht alle Kosten, denn der Coding Agent läuft in einer GitHub-Actions-Umgebung, und entsprechend kostet dies zusätzliche Build-Minuten. Der Vollständigkeit halber sei hier ebenfalls erwähnt, dass auch im Agent Mode in der IDE oder im Editor Premium Requests verrechnet werden können – je nach verwendetem Modell.

Ein weiterer Unterschied zwischen Agent Mode und Coding Agent liegt in der Selektion der KI-Modelle. Im Editor muss die Entwicklerin oder der Entwickler das Modell für die Aufgabe selbst auswählen. Beim Coding Agent ist dies automatisiert, und man kann keinen Einfluss auf das Modell nehmen.

Das Verhalten des Copilot Coding Agents ist dem des Agent Modes im Editor sehr ähnlich, jedoch nicht so interaktiv. Getriggert wird die Coding-Agent-Session über ein GitHub Issue. Sobald das Issue dem Benutzer *@copilot* zugewiesen wird, übernimmt der Coding Agent die Aufgabe. Der Prompt ist also der Inhalt des Issues. Es wird ein neuer Branch erstellt und basierend darauf ein Pull Request angelegt.

In **Bild 3** ist das Issue ersichtlich, das dem Benutzer *@copilot* zugewiesen wurde. Mit dem Augen-Emoji unterhalb der Issue Description signalisiert Copilot, dass er das Issue angenommen hat und sich darum kümmert. Nur wenig später wird der Draft Pull Request im Issue referenziert.

Im Pull Request ist die Coding-Agent-Session ersichtlich und kann entsprechend in Echtzeit überwacht werden. **Bild 4** zeigt die Live-View der Coding-Agent-Session. Ähnlich wie im lokalen Agent Mode sehen wir auch hier die einzelnen Schritte, die der Copilot unternimmt, um die Aufgabe zu lösen. In den ersten Schritten ist auch ersichtlich, dass die Coding-Agent-Umgebung eingerichtet wird. Die Session wird in einer GitHub-Actions-Umgebung durchgeführt, und über eine spezielle Workflow-Definition können wir auf dieses Environment Einfluss nehmen – dazu später mehr. Nach beendeter Session sind im Branch die entsprechenden Commits vorhanden, und der Pull Request ist aktualisiert. **Bild 5** zeigt diesen Zustand. Wie ein Arbeitskollege kommentiert Copilot den Pull Request und gibt eine Übersicht über die ausgeführten Arbeiten.

Ab hier geht es grundsätzlich so weiter wie bei einem klassischen Pull Request mit menschlichen

Autoren. Wir können Checks validieren und unser Feedback über Kommentare kundtun. Nur wenn der Benutzer *@copilot* im Kommentar erwähnt wird, wird eine weitere Coding-Agent-Session getriggert, in der die Beanstandungen oder Änderungswünsche abgearbeitet werden. **Bild 6** zeigt ein weiteres Sicherheitsfeature im Rahmen des Coding Agents: Werden bei einem regulären Pull Request die CI-Workflows automatisch getriggert, muss dies hier erst freigegeben werden. Somit kann überprüft werden, dass kein Schadcode oder Fehler in die Workflows eingeschleust wurde. Sind wir mit der Arbeit zufrieden, dann können wir den Pull Request integrieren.

Eine weitere interessante Pull-Request-Funktionalität von Copilot ist, dass dieser auch als Reviewer hinzugezogen werden kann. Somit kann ich zum Beispiel meinen Pull Request erst durch den Copilot reviewen lassen, bevor ein Arbeitskollege zugewiesen wird. Dies trägt ebenfalls zur Effizienz bei, da die kleinen Beanstandungen durch den Copilot identifiziert werden und somit der Pull Request für den Review durch die Arbeitskollegin oder den Arbeitskollegen in einer besseren Qualität vorliegen wird.

Konfiguration des Coding Agents

Damit wir mit dem Coding Agent loslegen können, benötigen wir erst einige Einstellungen. Einerseits müssen wir in unserem Account das Repository für den Coding Agent freischalten. Ist dies erledigt, können wir in den Repository-Einstellungen Sicherheitsfunktionen konfigurieren und Werkzeuge ►

The screenshot shows a GitHub Pull Request (PR) titled "Implement Session Results Endpoint #6". The PR is in "Draft" state and is assigned to "marc-mueller" and "Copilot". The PR description includes "Key Features" and "Response Structure". The "Response Structure" section shows a JSON response for a session. The PR is successfully merged and closed.

Key Features:

- Complete endpoint implementation with proper error handling (404 for non-existent sessions, route constraint handling for invalid GUIDs)
- Vote aggregation logic for multiple choice questions with vote count
- Text response collection for text-type questions with timestamps
- Visibility rules enforcement - active/draft questions show empty results, completed questions show full results
- Consistent API patterns following existing codebase conventions

Response Structure

The endpoint returns session metadata along with question-level results

```

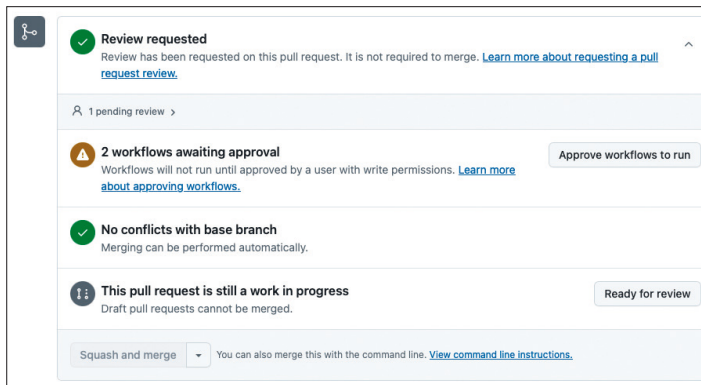
{
  "sessionId": "guid",
  "title": "Session Title",
  "totalParticipants": 42,
  "totalVotes": 128,
  "questions": [
    {
      "questionId": "guid",
      "questionText": "What is your favorite color?",
      "questionType": "MultipleChoice",
      "status": "Completed",
      "totalVotes": 10,
      "results": [
        {
          "optionId": "guid",
          "optionText": "Blue",
          "voteCount": 6,
          "percentage": 60.0
        }
      ]
    }
  ]
}

```

Test Coverage

Pull request successfully merged and closed
You're all set — the `copilot/fix-5` branch can be safely deleted.

Pull Request mit Arbeitsergebnissen (Bild 5)



Approvals für Workflows (Bild 6)

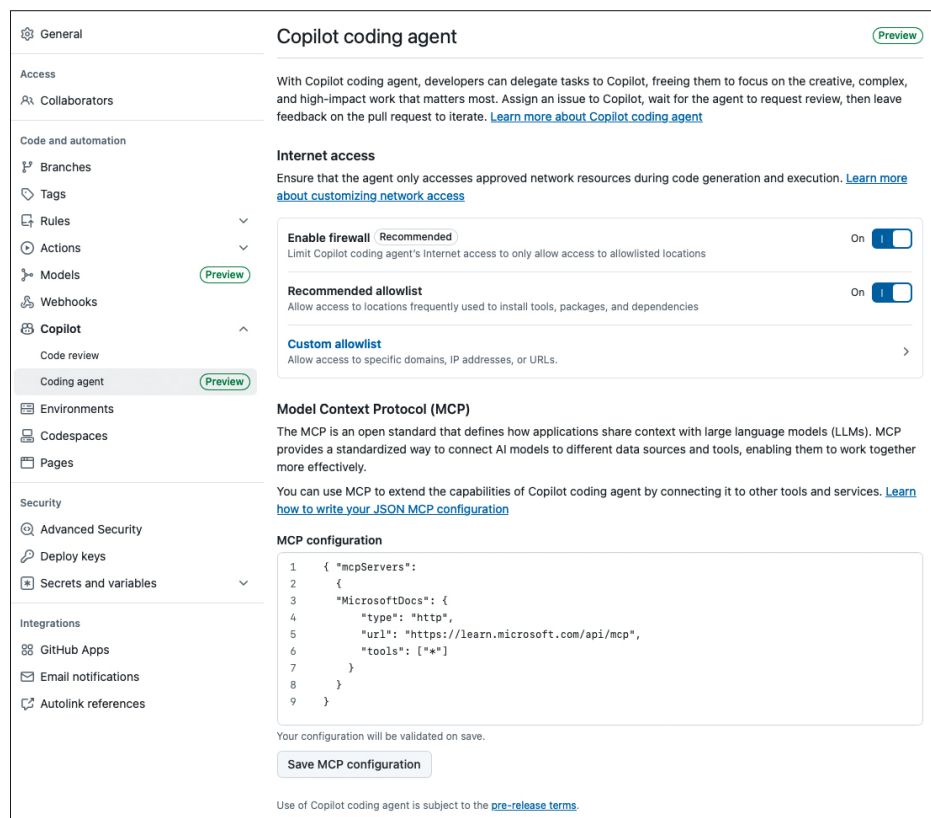
freischalten. Da der Coding Agent sehr autonom agiert, ist es wichtig, die Schranken klar zu definieren. Bild 7 zeigt die Einstellungen auf Repository-Stufe. Per Default ist eine Firewall aktiv. Diese können wir über eine eigene Allow-List feingranular konfigurieren. Wir sollten also die externe Kommunikation auf ein absolutes Minimum einschränken, damit keine sensiblen Daten nach außen gelangen können. Gründe dafür könnten Prompt Injections oder böswillig veränderte Tools sein. Als zusätzliche Werkzeuge können ebenso wie im Editor MCP-Server konfiguriert werden. Diese Syntax ist leicht abweichend, da wir hier auch die einzelnen Tools des MCP-Servers freigeben können. Der Coding Agent hat zwei MCP-Server per Default aktiviert: den GitHub MCP, um mit dem Repository zu interagieren, und den Playwright MCP, damit mit Webseiten interagiert werden kann. Gerade bei der Entwicklung von Webapplikationen ist dies sehr hilfreich, da der Coding Agent so auch Screenshots erstellen und verwenden oder das DOM analysieren kann. Es ist auch möglich, MCP-Server zu verwenden, die eine Authentisierung voraussetzen. Über die Secrets in den Repository-Settings können diese – unter Verwendung eines vordefinierten Prefixes – hinterlegt werden. Ebenso lässt sich der Coding-Agent-Workflow zur Konfiguration von Tools sowie zur Authentisierung gegen über Diensten verwenden.

Die Coding-Agent-Session läuft in einer GitHub-Actions-Umgebung. Entsprechend können wir auf diese Session mittels einer Workflow-Definition Einfluss nehmen. Wir können die Größe des Runners definieren, Tools und Abhängigkeiten vorinstallieren, Environment-Variablen aufsetzen, GIT LFS aktivieren oder Firewall-Einstellungen des Agents ändern.

Um dies durchzuführen, muss eine Workflow-Definition unter `.github/workflows/copilot-setup-steps.yml` erstellt werden. Die Workflow-Definition muss einen Job mit dem Namen `copilot-setup-steps` beinhalten. Dieser Job wird also zur Initialisierung des Coding-Agent-Environments ausgeführt. Listing 1 zeigt eine solche Workflow-Definition für unsere Beispielapplikation. Wir installieren die Dev-Tools für .NET und Node und initialisieren die Umgebung.

Context, Instructions, Tools und was dies für die Qualität bedeutet

Natürlich hört es sich erst einmal faszinierend an, eine Aufgabe für 0,04 US-Dollar durch den Coding Agent erledigen zu lassen. Doch was schlussendlich zählt, ist die Qualität der Arbeit und ob zusätzliche technische Schulden entstanden sind, die einen dann zu einem späteren Zeitpunkt wieder einholen. Gerade die Top-LLMs wie GPT 5 oder Claude Sonnet 4 liefern gute Analysen und schreiben einwandfreien Code. Je mehr Datengrundlage ein Modell für eine Aufgabe besitzt und je klarer die Rahmenbedingungen und Ziele definiert werden, desto besser ist das Resultat. Weitere wichtige Faktoren sind die Komplexität und der Umfang der Aufgabe. Zum aktuellen Zeitpunkt eignen sich der Agent Mode und der Coding Agent vor allem für Refactorings, kleine Features oder Bugfixing – alles Aufgaben mit einer überschaubaren Komplexität, guten Rahmenbedingungen und klaren Zielen. Hier entstehen durchaus sehr gute Resultate, und die Entwicklerin oder der Entwickler wird maßgeblich



Einstellungen für den Coding Agent (Bild 7)

entlastet. Eine komplexe Applikation aus einem Prompt als Greenfield-Entwicklung in Auftrag zu geben, wird dagegen wohl oder übel in einer Enttäuschung enden.

Wie kommen wir nun zu guten Resultaten? Wichtig ist, dass der Prompt die Aufgabe und Ziele klar definiert und je nachdem den Kontext bereits unter Angabe von Datei-Referenzen einschränkt. Der Prompt sollte die Dos und Don'ts beinhalten, wie wir unseren Code gerne geschrieben hätten und welche Architekturmuster und Abhängigkeiten wir erwarten. Entsprechend kann das Prompting sehr komplex ausfallen, und wir würden uns abermals wiederholen, was die Gültigkeit von Guidelines und Rahmenbedingungen angeht.

Alles, was allgemeingültig ist, können wir deshalb bei GitHub Copilot mittels Instructions hinterlegen. Instructions sind ähnlich zu verstehen wie ein System Prompt, also ein Prompt, der zusätzlich zum eigentlichen Prompt eingefügt und vom KI-Modell verarbeitet wird. Im Repository können wir unter `.github/copilot-instructions.md` eine allgemeingültige Instruction hinterlegen. Diese wird dem eigentlichen Prompt ange-

fügt. Darin regeln wir alles, was der Copilot bei der Implementierung berücksichtigen soll: Architekturrichtlinien, Coding Guidelines, Code-Struktur und wie automatisierte Tests geschrieben und ausgeführt werden sollen.

Hierbei handelt es sich um eine Markdown-Datei, die in natürlicher Sprache geschrieben wird. Im Verzeichnis `.github/instructions` können weitere Markdown-Dateien mit dem Namensschema `<meinname>.instructions.md` hinterlegt werden. Diese Dateien werden mit einem Frontmatter-Block ausgestattet, der ein Filterkriterium enthält. Somit werden diese Instructions nur dann eingefügt, wenn die Bedingung erfüllt ist. Auf diese Weise können für bestimmte Source-Dateien oder Verzeichnisse spezifische Instructions hinterlegt werden. [Listing 2](#) zeigt einen Auszug aus einer pfadspezifischen Instruction.

Eine umfangreiche Sammlung von Instructions und Prompts ist im GitHub Repository unter [1] verfügbar.

Copilot verwendet eine Reihe von Tools wie den Dateizugriff, die Shell-Integration oder die Testausführung. Erst ►

● Listing 1: Einstellungen für den Coding Agent

```
name: "Copilot Setup Steps"

on:
  workflow_dispatch:

jobs:
  # The job MUST be called `copilot-setup-steps` or it
  # will not be picked up by Copilot.
  copilot-setup-steps:
    runs-on: ubuntu-latest

    # Set the permissions to the lowest permissions
    # possible needed for your steps. Copilot will be
    # given its own token for its operations.
    permissions:
      # If you want to clone the repository as part of
      # your setup steps, for example to install
      # dependencies, you'll need the `contents: read`
      # permission. If you don't clone the repository
      # in your setup steps, Copilot will do this for
      # you automatically after the steps complete.
      contents: read

    # You can define any steps you want, and they will
    # run before the agent starts. If you do not check
    # out your code, Copilot will do this for you.
    steps:
      - name: Checkout code
        uses: actions/checkout@v5

      - name: Set up .NET SDK (10 via global.json)
        uses: actions/setup-dotnet@v5

      with:
        global-json-file: global.json
        dotnet-quality: preview

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: "20"
          cache: "npm"
          cache-dependency-path:
            frontend/package-lock.json

      - name: Install JavaScript dependencies
        run: npm ci
        working-directory: frontend

      - name: Restore .NET dependencies
        run: |
          dotnet restore SimplePoll.sln

      - name: Build backend (no tests)
        run: |
          dotnet build backend/SimplePollsApi/
            SimplePollsApi.csproj -c Debug --no-restore

      - name: Prepare test projects (restore/build
        only)
        run: |
          dotnet build backend/SimplePollsApi.Tests/
            SimplePollsApi.Tests.csproj -c Debug
            --no-restore
```

so können zum Beispiel Kompilationsfehler erkannt werden oder Testresultate ausgewertet werden. Diese Built-in-Tools sind immer noch sehr generisch, aber wichtig für die Grundfunktionalität. Damit wir dem Agenten weitere Tools zur Verfügung stellen können, können wir eigene Werkzeuge als MCP-Server bereitstellen. Dies beinhaltet nicht nur die Kommunikation mit externen Services, sondern wir können auch domänenspezifische Informationen oder Unternehmensdaten zugänglich machen. Das sind also die Kontextinformationen, die ein generisches KI-Modell nicht kennen kann. Über das MCP-Protokoll können wir diese Informationen strukturiert zur Verfügung stellen.

Doch auch mit allen Instruktionen, Tools und dem richtigen Kontext können wir den KI-Agenten nicht einfach drauflosprogrammieren lassen und die Änderungen blind integrieren. Der lokale Agent Mode arbeitet uns in unserer Entwicklungsumgebung lediglich zu, und wir als Entwickler sind dafür verantwortlich, diese Änderungen zu verifizieren. Der Coding Agent zeigt dies ebenfalls ganz klar auf. Er arbeitet immer in einem dedizierten Branch und erstellt immer einen Pull Request. Generell sollte der Pull Request in jedem Softwareentwicklungsteam immer das Quality Gate darstellen, in dem Code-Änderungen überprüft werden und wo mittels automatisierten Workflows/Pipelines sichergestellt wird, dass wir unsere Qualitäts- und Sicherheitsstandards erreicht haben.

MCPs in der Praxis – Warum sie für Copilot Agent und Coding Agent relevant sind

Das Model Context Protocol (MCP) ist ein noch junges, aber extrem wichtiges Konzept in der Welt der KI-gestützten Entwicklung. Entwickelt wurde es von Anthropic zusammen mit Microsoft und weiteren Partnern, um eine einheitliche Schnittstelle zwischen Agenten und externen Datenquellen oder Tools zu schaffen. Damit kann ein Agent nicht nur auf den aktuellen Code im Workspace zugreifen, sondern sich auch mit Datenbanken, API-Katalogen, Testsystemen oder internen Policies verbinden. Für Entwickler, die mit GitHub Copilot im Agent Mode oder dem Copilot Coding Agent arbeiten, ist MCP die Brücke zwischen „Code generieren“ und „Code im Kontext der Organisation ausführen und validieren“.

Der große Vorteil von MCP-Servern ist, dass sie nicht statisch sind. Ein Instructions-File kann zwar erklären, welche Auth-Library oder welcher Logging-Standard verwendet werden soll, aber es wird schnell veralten und bleibt immer rein beschreibend. Ein MCP-Server dagegen kann zur Laufzeit aktuelle Informationen liefern, Fragen beantworten und sogar strukturierte Antworten geben, die der Agent sofort in Code oder Testfälle umsetzen kann. Gerade in dynamischen Projekten, in denen sich API-Versionen oder Deployment-Ziele regelmäßig ändern, ist dieser Live-Kontext entscheidend.

Technisch unterstützt MCP mehrere Transportarten. Für lokale Tools oder Build-Skripte eignet sich der *stdio*-Transport, der einfach über Standard-Ein- und -Ausgabe kommuniziert. Wer einen zentralen MCP-Server im Unternehmen bereitstellen will, greift häufig auf HTTP oder SSE (Server-Sent Events) zurück. SSE ist vor allem interessant, wenn der Server dem Client Zwischenergebnisse oder Fortschrittsinformationen

● Listing 2: Pfadspezifische Instruktionen

```
---
applyTo: "**/*.ts,**/*.tsx"
---

## Conventions
- Use functional components + hooks. Avoid class components.
- Centralize network calls in `frontend/src/services/` (create if missing) with thin wrappers returning typed responses.
- Define shared API types in `frontend/src/types/api.ts` (create if absent) mirroring backend DTO shapes (PascalCase enums as string unions: `MultipleChoice` | `Text`).
```

schicken soll – etwa während ein lange laufender Test ausgeführt wird oder eine Build-Pipeline Feedback liefert.

Die Anwendungsfälle sind vielfältig. MCPs können externe Systeme abfragen, strukturierte Daten zurückgeben oder auch aktive Aktionen anstoßen. In der Praxis bedeutet das: Ein Agent kann eine Datenbank nach einem Schema fragen, einen Pull Request im Repository erstellen, eine Pipeline triggern oder einen UI-Test ausführen und das Ergebnis direkt zurückmelden. Microsoft hat in mehreren Beiträgen sehr anschaulich gezeigt, wie der Playwright-MCP-Server in der Praxis eingesetzt werden kann. So wird unter [2] die Vision eines vollständig KI-gestützten Test-Workflows beschrieben: von der Generierung der Tests über deren Ausführung bis hin zur automatischen Verifikation der Ergebnisse. Besonders spannend ist der konkrete Praxisbericht unter [3], in dem das Azure-DevOps-Team von Microsoft selbst schildert, wie sie manuelle Testfälle durch einen MCP-gestützten Workflow ersetzt haben. Dabei erstellt der Copilot Coding Agent selbstständig Playwright-Skripte, führt sie in der Testumgebung aus und liefert Screenshots und Statusmeldungen zurück. Der Test gilt nur dann als bestanden, wenn der visuelle Zustand exakt dem erwarteten Ergebnis entspricht. Das Ergebnis: drastisch verkürzte Testzyklen, weniger manuelle Klickarbeit und deutlich schnelleres Feedback für Entwicklerteams – ein Paradebeispiel für „Self-Verifying Development“ in der Praxis.

Für .NET-Entwickler sind inzwischen eine Reihe nützlicher MCPs verfügbar, die sich nahtlos in die eigene Entwicklungsumgebung integrieren lassen. Neben GitHub- und Azure-DevOps-MCPs gibt es Anbindungen an Microsoft Docs, die offizielle API-Referenzen liefern, sowie Context7, das dafür sorgt, dass Codebeispiele immer zur verwendeten Version passen. Playwright MCP ist insbesondere für Teams interessant, die End-to-End-Tests schreiben oder bestehende UIs automatisiert verifizieren wollen.

Trotz dieser großen Auswahl kann es sinnvoll sein, einen eigenen MCP-Server zu bauen. Gerade wenn interne Systeme angebunden werden sollen – etwa ein API-Katalog mit

projektspezifischen Endpunkten, eine interne Event-Schema-Registry oder eine firmeneigene Wissensdatenbank –, stößt man mit generischen Lösungen schnell an Grenzen. Ein eigener MCP-Server kann genau diese Domäneninformationen bereitstellen und so sicherstellen, dass ein Agent nicht nur syntaktisch korrekten, sondern auch fachlich passenden Code erzeugt. Das .NET-MCP-SDK macht die Implementierung dabei angenehm unkompliziert: Mit wenigen Zeilen Code kann ein eigener Server erstellt werden, der Tools bereitstellt, Anfragen entgegennimmt und Antworten im gewünschten Format zurückgibt.

Auch für Sicherheit ist gesorgt: MCP unterstützt Authentifizierung und Berechtigungen, sodass sensible Informationen nicht ungeschützt in den Agent-Kontext gelangen. IDEs wie VS Code oder Visual Studio sowie der Copilot Coding Agent unterstützen die Konfiguration von Secrets, wodurch Tokens und Passwörter sicher verwaltet werden können.

Unterm Strich bringen MCPs die KI-gestützte Entwicklung auf das nächste Level: weg von statischen Instruktionen, hin zu einem dynamischen, kontextbewussten Dialog zwischen dem Agent und den Systemen, in denen wir täglich arbeiten. Und genau hier entsteht der größte Mehrwert: Der Agent wird nicht nur zum Code-Assistenten, sondern zum echten Teammitglied, das versteht, in welchem Projekt es arbeitet, welche Regeln dort gelten und wie das Ergebnis automatisch überprüft werden kann.

Chancen, Risiken und Sicherheitsmechanismen

Sind nun alle Softwareentwicklerinnen und -entwickler in naher Zukunft arbeitslos? Nein! Auch wenn die KI rasante Fortschritte macht, so kann sie derzeit nicht den Menschen in der Entwicklung ersetzen. Die KI in der Entwicklung kann aber eine deutliche Effizienzsteigerung bewirken. Man bekommt ein kleines Team, das einem zuarbeitet, aber Feedback und Kontrolle benötigt. Ein weiterer Vorteil liegt auch im schnelleren Onboarding neuer Mitarbeiterinnen und Mitarbeiter. In Windeseile werden Fragen zur aktuellen Codebasis beantwortet. Aber auch für bestehende Mitarbeiter ist die Informationsquelle KI von Vorteil. Fragen, die man sonst scheuen würde, können im „privaten“ Rahmen gestellt werden, und auch die Erklärungen zum bestehenden Code sind stets abrufbereit. Aufgaben wie einfachere Refactorings oder Feature-Erweiterungen, aber auch Unterstützung bei der Testimplementierung können gut an die KI ausgelagert werden, und diese Chance sollte definitiv genutzt werden.

Vertrauen ist gut, Kontrolle ist besser. Genau so sollten wir auch mit der KI-Unterstützung umgehen. Ein blindes Vertrauen in KI-generierten Code wäre fahrlässig. Teilweise mit fast schon menschlich wirkenden Zügen versucht die eine oder andere KI das Ziel zu erreichen, indem sie Tests modifiziert oder abstellt, oder sie versucht so lange etwas, bis es zwar läuft, aber eine große technische Schuld geschaffen wurde. Oftmals hängt dies stark mit dem Prompting, den Rahmenbedingungen, aber auch mit dem verwendeten Modell zusammen. Um dieses Risiko zu minimieren, bedarf es der genau gleichen Mechanismen, die wir hoffentlich bereits im Einsatz haben: Die Isolation der Änderungen in einem dedi-

zierten Branch, welche nur über einen Pull Request integriert werden können. Der Pull Request stellt sicher, dass alle Policies, Coding Guidelines und Security-Prinzipien ebenso erfüllt sind wie die funktionalen Anforderungen. Letzteres ist nur mittels Testautomatisierung möglich. Auch mit KI gilt hier ein Vier-Augen-Prinzip. Code und Tests sollten, wenn möglich, nicht aus der gleichen Feder stammen, und ein Review durch eine andere Person ist unabdingbar.

Neben den fachlichen Risiken gibt es natürlich auch Sicherheitsrisiken, wenn die KI maßgeblich an der Code-Generierung beteiligt ist. GitHub stellt für den Einsatz des durchaus autonomen Coding Agents folgende Maßnahmen bereit:

- Das Pushen von Code ist dem Coding Agent nur auf den `copilot/*`-Branch möglich, alle anderen Branches sind per Security Policy tabu.
- Zudem wird niemals ein Workflow automatisch gestartet, der von einer KI-Code-Änderung getriggert worden wäre. So muss der Benutzer entsprechend eine Freigabe tätigen, nachdem er Gelegenheit hatte, die Änderungen zu verifizieren.
- Der Netzwerkzugriff der Coding-Agent-Umgebung wird durch strikte Firewall-Regeln limitiert, sodass keine Daten ungewollt nach außen dringen können.
- Aber auch Angriffsvektoren wie Prompt-Injection müssen bekämpft werden. So werden etwa versteckte Zeichen in Issues (zum Beispiel HTML-Kommentare in Markdown, nicht ersichtlich im gerenderten Text) entfernt, bevor der Text der KI übergeben wird.

Letztlich ist es wichtig, dass wir unsere Workflows und Pipelines so implementiert haben, dass jede Änderung vor der Integration fachlich und technisch validiert wird. Mit GitHub Advanced Security können wir einfach statische Code-Analysen ausführen und Dependencies auf Schwachstellen und Sicherheitslücken analysieren. Gepaart mit einer guten Testabdeckung bestehend aus sowohl Whitebox- als auch Blackbox-Tests können wir uns guten Gewissens von der KI unter die Arme greifen lassen. ■

- [1] Sammlung von Instructions und Prompts für Copilot auf GitHub, <https://github.com/github/awesome-copilot>
- [2] Microsoft Developer Blog, <https://developer.microsoft.com/blog/the-complete-playwright-end-to-end-story-tools-ai-and-real-world-workflows>
- [3] Microsoft Developer Blog, <https://devblogs.microsoft.com/devops/from-manual-testing-to-ai-generated-automation-our-azure-devops-mcp-playwright-success-story>



Marc Müller

arbeitet als Principal Consultant für DevOps und .NET-/Azure-Lösungen bei 4tecture und ist Microsoft MVP für DevOps. Zu seinen Schwerpunkten zählen Ausbildung und Coaching von DevOps- und .NET-Projektteams. publications@4tecture.ch

ZWISCHEN HYPE UND REALITÄT

KI im Mittelstand braucht mehr als Technik

Über die besonderen Herausforderungen, die sich bei der Einführung von KI in kleinen und mittleren Unternehmen stellen – und clevere Lösungen dafür.

Künstliche Intelligenz war lange ein Thema für Forschungslabore und Hightech-Konzerne, doch seit der Veröffentlichung allgemein zugänglicher Modelle wie GPT-4o oder Claude hat sie den Sprung in den Alltag geschafft. Millionen Menschen experimentieren mit Chatbots, lassen sich Texte zusammenfassen oder Code erklären. Diese Allgegenwart schafft einen Erwartungsdruck, der im Mittelstand stärker zu spüren ist als in Großkonzernen: Wenn KI so einfach ist, warum setzen wir sie nicht längst ein? Doch der Blick hinter die Kulissen zeigt, wie anspruchsvoll es ist, generative Modelle oder Machine-Learning-Lösungen in die Wertschöpfung zu integrieren.

Aktuelle Zahlen geben einen Eindruck davon, wie groß der Rückstand noch ist. Laut Eurostat nutzten 2024 europaweit nur 13 Prozent der Unternehmen mit mindestens zehn Beschäftigten mindestens eine KI-Technologie. Bei kleinen Betrieben (unter 50 Mitarbeitern) lag der Anteil bei lediglich etwas mehr als zehn Prozent, bei mittelgroßen (bis 250 Mitarbeiter) bei 20 Prozent, während über 40 Prozent der Großunternehmen schon AI-Tools einsetzen.

Dieser Unterschied spiegelt nicht nur die Größe der Budgets wider, sondern auch die unterschiedliche Risikobereitschaft und die personellen Ressourcen. Der Branchenfokus ist dabei sehr unterschiedlich: In der Informations- und Kommunikationsbranche setzen fast 49 Prozent der Unternehmen KI

ein, im Sektor der wissenschaftlichen und technischen Dienstleistungen beträgt der Anteil rund 30 Prozent, während es im Baugewerbe sowie im Hotel- und Gastgewerbe weniger als sieben Prozent sind.

Deutschland liegt in dieser Statistik nur leicht über dem EU-Durchschnitt. Eine Bitkom-Umfrage aus dem Februar 2025 zeigt, dass etwa 20 Prozent der deutschen Firmen KI aktiv nutzen – eine deutliche Steigerung gegenüber 15 Prozent im Vorjahr und neun Prozent im Jahr 2022. Andere Studien kommen sowohl zu höheren als auch zu niedrigeren Zahlen (27 Prozent laut ifo Institut, 37 Prozent laut IW Köln). Einig sind sich alle, dass kleine Unternehmen deutlich seltener KI einsetzen als große. Gleichzeitig planen über 80 Prozent der Unternehmen, ihre KI-Budgets zu erhöhen.

Vom Warum zum Wie

Die Zahlen zeigen auch: Das „Warum“ der KI-Einführung ist längst beantwortet – die Mehrzahl der Unternehmen sieht Automatisierung und datenbasierte Entscheidungen als essenziell für die Wettbewerbsfähigkeit. Die Frage ist eher das „Wie“: Wie lässt sich ein Chatbot mit den eigenen Wissensdatenbanken koppeln, ohne dass Kundendaten ungeschützt ins Ausland fließen? Muss jede KI ein Chatbot sein, oder wie unauffällig kann man eine KI im Geschäftsprozess integrieren? Wie können Entwickler KI-Module in bestehende Software integrieren? Und zu guter Letzt: Wie behält man die Kosten im Griff, wenn jedes Token im Sprachmodell Geld kostet? Diese praxisnahen Fragen entscheiden darüber, ob sich KI in den Prozessen durchsetzt. Wer heute beginnt, Antworten darauf zu finden, hat einen Vorsprung, wenn die Technologie in den kommenden Jahren noch wichtiger wird.

Gleichzeitig erleben wir eine Art „Demokratisierung“ der KI-Tools. Es gibt nicht nur die großen Cloud-Anbieter; Open-Source-Modelle wie Llama oder Mistral können als On-Prem-Lösung betrieben werden, ohne hohe Lizenzgebühren. Bibliotheken wie LangChain oder Semantic Kernel machen die Orchestrierung von Modellen und Datenquellen relativ einfach, sodass auch kleinere Teams experimentieren und erste Prototypen darauf aufbauen können. Hinzu kommen Speziallösungen für Branchen, wie visuelle Erkennung in der Fertigung oder Sprachanalyse im Kundenservice.

Diese Vielfalt ermöglicht es, auch ohne Millionenbudget zu starten. Aber sie führt auch dazu, dass viele KMUs das Ge-

● Du möchtest gerne mehr erfahren?

Das freut uns, denn ab Erscheinen des vorliegenden Artikels veröffentlichen wir auf developer-world.de/insights weitere Beiträge, die den Artikel fortsetzen und ausschließlich online zur Verfügung stehen. Derzeit geplant sind folgende Beiträge:

- Chunks mit Köpfchen: Wie trennen wir Textinformationen in sinnvolle Segmente?
- Vor dem Prompt ist nach dem Prompt: Wie bereitet man Anfragen an LLMs bestmöglich vor?
- Vektoren und semantische Suche: Welche Vektor-Datenbanken gibt es, und welche eignen sich wofür?
- JSON statt Kauderwelsch: Wie bekomme ich LLMs dazu, strukturiert zu antworten?

fühl haben, den Überblick zu verlieren. Der Einstieg in die „richtige“ Lösung erfordert Orientierungswissen.

Die drei großen Stolpersteine auf dem Weg zur KI-Integration

Gerade in kleinen und mittelständischen Unternehmen stellen sich gewisse Herausforderungen. Große Konzerne leisten sich eigene KI-Teams und können einfacher entsprechende Expertise (konkret: Personal) dafür einkaufen. KMUs hingegen stehen vor der Aufgabe, die Strategien zum Einsatz von KI selbst auszuarbeiten und mit (den passenden) Dienstleistern umzusetzen. Alternativ besteht ein (meist kleines) internes Softwareteam, welches diese Aufgaben nun übernehmen muss. Hier steht das Team also vor der Herausforderung, Know-how aufzubauen, Schulungen durchzuführen oder eben selbst auf externe Dienstleister zurückzugreifen.

Egal welcher Weg gewählt wird, diese Teams stehen vor immer den gleichen drei Stolpersteinen:

- Datenschutz,
- Know-how-Lücken und
- Erwartungen.

Datenschutz – der Wächter am Tor

Kein anderes Thema taucht in Gesprächen mit mittelständischen Unternehmen so häufig auf wie Datenschutz. In der EU ist der Schutz personenbezogener Daten verankert, und mit der DSGVO sowie der im August 2024 in Kraft getretenen EU-KI-Verordnung (AI Act) gibt es rechtliche Rahmen, die selbst große Konzerne vor Herausforderungen stellen. Für KMUs bedeutet das: Es reicht nicht, ein Modell zu lizenzieren und loszulegen. Man muss prüfen, wo KI eingesetzt werden darf, wie Eingaben gespeichert werden und ob Ausgaben persönliche Informationen enthalten könnten. Laut Bitkom sehen 73 Prozent der deutschen Unternehmen Datenschutzanforderungen als erhebliches Hemmnis.

Der EU AI Act verschärft diese Problematik und bringt gleichzeitig Chancen. Das Gesetz klassifiziert KI-Systeme nach Risikostufen. „Hochrisiko“-Anwendungen – etwa im Recruiting, der Kreditvergabe oder im Gesundheitswesen – unterliegen strengen Transparenz-, Audit- und Dokumentationspflichten. Unternehmen müssen Aufzeichnungen darüber führen, wie ein Modell trainiert wurde, wie es Entscheidungen trifft und welche menschliche Aufsicht besteht. Für kleine und mittlere Unternehmen sind allerdings Erleichterungen vorgesehen, etwa vereinfachte Dokumentationsanforderungen. Damit sollen Innovationen gefördert werden, ohne Sicherheit und Ethik zu gefährden.

Datenschutz heißt auch Datensouveränität. Viele KMUs zögern, Daten an Cloud-Dienste zu schicken, bei denen unklar ist, in welchem Land die Server stehen. In Branchen wie dem Gesundheitswesen oder bei patentgeschützten technischen Zeichnungen ist das ein Ausschlusskriterium. Hier greifen On-Prem-Lösungen. Neue Open-Source-Modelle können lokal betrieben werden, etwa mit Plattformen wie Ollama oder Llama.cpp, die auch auf kleineren Servern laufen. Doch das erfordert Infrastruktur und Know-how. Wieder anderen genügt eine hybride Lösung: Sensible Daten bleiben im eigenen

Netzwerk, während unkritische Aufgaben wie Textsummaries in der Cloud erledigt werden. Diese Abwägung zwischen Cloud, On-Prem und Hybrid ist ein Teil der strategischen Architektur und wird durch Datenschutzanforderungen wesentlich geprägt.

Schließlich ist Datenschutz aber auch ein Verkaufsargument: „AI made in Germany“ oder „Trusted AI“ versprechen, dass Modelle oder Integrationen transparent, fair und sicher sind. Unternehmen, die diese Prinzipien ernst nehmen, können daraus einen Wettbewerbsvorteil ziehen. Der Aufwand für Compliance mag hoch sein, doch er wirkt wie ein Qualitätsmerkmal. Ein gut implementiertes Datenschutzkonzept schafft Vertrauen bei Kunden, Partnern und Behörden – und ist damit nicht nur Pflicht, sondern Kür.

Know-how-Lücken – der unsichtbare Flaschenhals

Der Fachkräftemangel trifft die Digitalisierung insgesamt hart, doch bei KI-Projekten wirkt er besonders stark. Integration von KI, eventuell Machine-Learning-Kompetenzen, Datenaufbereitung und semantische Suchen – das sind Kompetenzen, die man nicht im Vorbeigehen mit ein paar YouTube-Tutorials erwirbt. Ein KI-Modell ist nicht einfach „Plug and Play“. Es muss verstanden, konfiguriert, überwacht und gegebenenfalls feinjustiert werden. Dies ist leider auch konträr zu dem, was als Erwartungshaltung, insbesondere bei Nicht-technikern, vorherrscht. Dazu aber nachfolgend mehr.

Dabei ist das Problem oft nicht ein grundsätzlicher Mangel an schlaun Köpfen, sondern die fehlende Brücke zwischen traditioneller Softwareentwicklung und KI. Entwickler, die in .NET, Java oder JavaScript zu Hause sind, beherrschen methodische Ansätze wie Domain-Driven Design, Test-Driven Development oder das Schreiben von User Stories. In KI-Projekten müssen sie aber zusätzliche Tools lernen: Datenaufbereitung und Pre-Processing, ML.NET oder TensorFlow für die Modellintegration, LangChain oder LlamaIndex für Retrieval-Augmented Generation, Semantic Kernel für orchestrierte „KI-Funktionen“. Diese Bibliotheken sind zwar mächtig, aber auch komplex. Ohne systematische Weiterbildung bleibt die Hürde hoch.

Eine Lösung sind Pilotprojekte, die bewusst klein gehalten werden. Statt sofort ganze Unternehmensprozesse zu automatisieren, konzentriert man sich auf einen klar umrissenen Use Case – zum Beispiel die automatische Kategorisierung von E-Mails im Support. Dabei geht es nicht darum, sofort einen Return on Investment zu erzielen, sondern darum, zu lernen: Wie bereite ich unsere Daten auf? Welche Modelle funktionieren auf unserem Domainwissen? Wie binde ich das Ganze in bestehende Systeme ein? Diese Lernprojekte helfen dem Team, Kompetenzen aufzubauen und Erfolgsrezepte zu dokumentieren. Anschließend kann man skalieren.

Ebenso wichtig ist die Zusammenarbeit zwischen Entwicklern, Data Scientists und Fachexperten. Wer den Kontext kennt – den Ablauf im Lager, die Prozesse in der Buchhaltung oder die typischen Kundenfragen –, kann helfen, die richtigen Datenpunkte zu identifizieren und die Modelle zu validieren. Auch das Management muss verstehen, was KI kann und was nicht. Eine Führungskraft muss kein Python-Profi sein, aber ►

ein Grundverständnis für Begriffe wie „Embedding“, „Vector Database“ oder „Fine-Tuning“ ist hilfreich, um realistische Erwartungen zu setzen und Budgetentscheidungen zu treffen.

Erwartungsmanagement – der KI-Hype im Realitätscheck

Zwischen Marketingversprechen und technischer Realität klafft eine Lücke. Wer das Internet durchforstet, findet unzählige Berater, die KI-Magie versprechen, ohne jemals ein Produkt produktiv gestellt zu haben. In Workshops wird „Prompt Engineering“ als Allheilmittel verkauft, Modelle werden gepriesen, die angeblich über Nacht Prozesse optimieren. Das mag für erste Experimente spannend sein – aber wenn es um kritische Unternehmensprozesse geht, ist dieses oberflächliche Verständnis gefährlich.

Hier sind Softwareentwickler und technische Teams gefragt, Verantwortung zu übernehmen. Es reicht nicht, ein LLM mit der kompletten Wissensdatenbank zu „füttern“ und zu hoffen, dass alles richtig läuft. Entwickler müssen die Grenzen von Modellen kennen, wissen, dass generative Systeme statistisch arbeiten und keine Wahrheit kennen. Sie müssen erklären, warum ein Chatbot manchmal falsche Fakten erfindet, und dafür sorgen, dass Antworten mit Unternehmensdaten abgeglichen werden. Das erfordert Retrieval-Augmented Generation oder eigene Datenbanken. Entwickler müssen auch erkennen, wann eine klassische regelbasierte Lösung sinnvoller ist als eine KI.

Projektleiter und Product Owner sind gefordert, klare User Stories zu formulieren. Eine Story wie „Als Vertriebsmitarbeiter möchte ich, dass KI meine Angebote erstellt“ ist zu unspezifisch. Besser ist: „Als Vertriebsmitarbeiter möchte ich auf Basis von Kundendaten und vordefinierten Preisregeln einen automatisierten Vorschlag für ein Angebot, damit ich weniger tippen muss.“ So wird klar, dass KI eine Assistenzrolle hat, und nicht das Geschäft übernimmt. Und: Realistische Projektpläne berücksichtigen die Datenaufbereitung, die Schulung und das iterative Testen. Wer verspricht, dass ein Chatbot in zwei Wochen 80 Prozent der Supportanfragen löst, ohne die Daten zu bereinigen oder das Team einzubeziehen, handelt fahrlässig.

In einem Markt, der aktuell so volatil ist, gilt es, kritische Fragen zu stellen: Haben Berater/Integratoren schon eine produktive KI-Integration begleitet? Verstehen sie die Spezifika der Branche? Können sie Datenschutz und Architektur erklären? Gute Softwareprojekte leben von ehrlicher Kommunikation – das gilt für KI umso mehr. Wer zu viel verspricht, riskiert Vertrauen und verbrennt intern das Thema. Es geht eben nicht nur um das Schreiben von Prompts. Wer jedoch transparent über Grenzen und Potenziale spricht, erzeugt die nötige Geduld und Begeisterung, um KI-Projekte erfolgreich umzusetzen.

Von der Idee zum Einsatz – KI als Werkzeug, nicht als Selbstzweck

Die wichtigste Entscheidung am Anfang ist nicht, welches Modell man nutzt, sondern wo man es einsetzt. KI spielt ihre Stärke dort aus, wo es um unstrukturierte Daten, große Informationsmengen oder Mustererkennung geht. Dagegen sind

Prozesse, die auf klaren Regeln basieren, oft mit klassischer Software effizienter und nachvollziehbarer umsetzbar. Ein Verkaufspreis, der sich aus Stücklisten und Materialpreisen zusammensetzt, benötigt für die Berechnung keine KI. Ein Chatbot, der Kundenfragen in natürlicher Sprache beantwortet, dagegen schon.

Um die richtige Stelle zu identifizieren, hilft eine systematische Beobachtung des Tagesgeschäfts. Nehmen wir das fiktive Kundenservice-Team eines Maschinenbauers. Jeden Morgen öffnet sich die E-Mail-Flut. Da kommen kurze Fragen wie „Wo kann ich das Ersatzteil XY bestellen?“, lange Beschreibungen von Fehlerzuständen („Die Maschine zeigt Fehlercode 42, blinkt rot, brummt, riecht verbrannt“), aber auch Beschwerden, die sofort eskaliert werden müssen.

Das Team sieht sich dabei mit drei Herausforderungen konfrontiert:

- **Manuelle Kategorisierung:** Jedes Ticket muss gelesen und einem Thema zugeordnet werden. Mal landet eine Ersatzteilanfrage beim Vertrieb, mal irrtümlich beim technischen Support. Das kostet Zeit und sorgt für Frust.
- **Informationssuche:** Nicht alle Mitarbeitenden kennen jeden Maschinentyp. Häufig müssen sie Kollegen fragen oder das digitale Handbuch durchsuchen. Oft nutzen sie sogar Papierordner mit alten Serviceanweisungen.
- **Eskalationen:** Dringende Anfragen, etwa von Großkunden mit Ausfall der Produktion, müssen sofort an die richtigen Stellen. Doch im Postfach gehen sie schon mal unter.

An diesem Punkt lohnt es sich, über KI nachzudenken. Ein Sprachmodell kann eingehende Texte analysieren, die Dringlichkeit anhand von Signalwörtern erkennen und das Thema ermitteln. Ein semantischer Suchalgorithmus kann aus der Wissensdatenbank relevante Abschnitte herausfiltern. Ein Workflow-Tool könnte die automatisierte Antwort oder Weiterleitung übernehmen.

Das Team müsste vielleicht nicht mehr alle Anfragen öffnen, sondern könnte sich auf die Fälle konzentrieren, bei denen sich die KI unsicher zeigt oder Eskalation nötig ist. Gleichzeitig bliebe die menschliche Kontrolle: Jeder Vorschlag kann geprüft, jeder Fehler korrigiert werden. Im besten Fall gibt es ein „Sowohl-als-auch“, eine automatisierte Antwort wird an den Kunden gesendet, das Ticket bleibt dennoch offen, bis ein Mensch analysiert hat, ob es noch Rückfragen gibt oder man manuell eingreifen muss.

Wie findet man solche Use Cases? Indem man die Mitarbeitenden fragt, welche Aufgaben sie als „lästig“ empfinden oder wo die Wartezeiten besonders lang sind. Ein anderer Ansatz sind Value Stream Maps: Sie zeigen den gesamten Ablauf einer Dienstleistung oder Produktion und machen Engpässe sichtbar. Oder man nutzt Klassifizierungstechniken auf die eigenen Logfiles, um zu erkennen, wo viele manuelle Eingriffe nötig sind.

In jedem Fall gilt: Der beste KI-Use-Case ist der, der ein echtes Problem löst – nicht der, der am spektakulärsten wirkt. Ich persönlich stelle immer die Frage: „Wo fangen eure Mitarbeiter zu gähnen oder schimpfen an?“ Genau da können wir ansetzen.

Integration – vom API-Call bis zum eigenen KI-Prozess

Ist der Use Case definiert, stellt sich die Integrationsfrage: Wie bringe ich das Modell an die Daten und in die Anwendung? Drei Wege sind gängig.

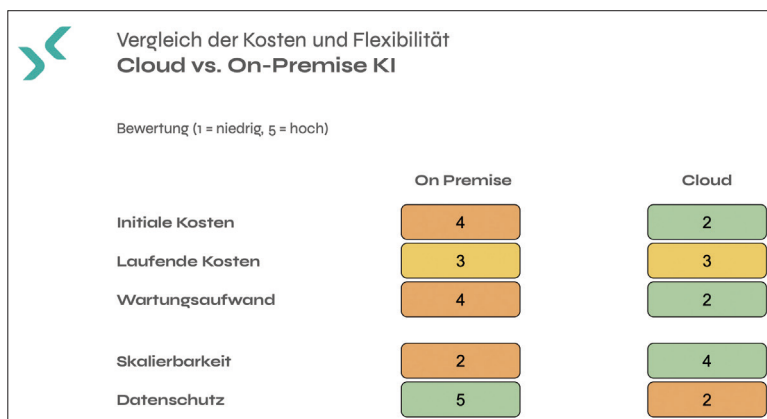
API-Services wie Azure OpenAI, Hugging Face Inference oder Google Vertex AI lassen sich per REST oder SDKs ansprechen. Sie bieten sofort leistungsfähige Modelle, skalieren automatisch und ersparen das Training. Allerdings geben Unternehmen damit Daten aus der Hand und sind von Preisänderungen oder Service-Updates abhängig. Für Prototypen oder unkritische Anwendungen ist das oft der schnellste Weg.

Die Alternative wäre der Betrieb von LLMs in einer lokalen Infrastruktur. Das ist investitionsintensiv, kann aber für verschiedene Anwendungsfälle die einzig valide Lösung sein. Wenn es nicht um LLMs geht, eignen sich Bibliotheken wie ML.NET; sie erlauben es, Modelle direkt in eine .NET-Applikation einzubauen. Auch TensorFlow bietet mit TF Lite und ONNX für andere Sprachen passende Optionen, wenn man beispielsweise selbst Trainingsmengen, etwa aus Sensordaten, in einer Produktion generieren und in einem eigenen KI-Modell nutzen möchte. Vorteile: Die Daten bleiben im Haus, die Latenz ist gering (je nach Modell und Hardware) und es gibt keine externen API-Kosten. Die Nachteile: Man benötigt

Investitionen in Hardware erfordern und skaliert werden können. Man zahlt nur für die Nutzung, was das Risiko reduziert – zumindest am Anfang (Bild 1). Allerdings steigen die Kosten schnell, wenn das Anfragevolumen wächst und die Modelle komplexer werden. Hinzu kommt die Abhängigkeit von einem Anbieter, der Preise ändern oder Modelle abkündigen kann. Und: Je nach Branche kann es ein Problem sein, Daten in die Cloud zu schicken.

On-Premises-KI gibt maximale Kontrolle über Daten, Performance und Sicherheit. Die Fortschritte in Open-Source-Modellen und günstiger Hardware (zum Beispiel Server-GPUs oder sogar KI-Beschleuniger für den Edge-Bereich) machen diese Option zunehmend praktikabel. Unternehmen können eigene Modelle trainieren oder vortrainierte Modelle fine-tunen – etwa mit Hugging Face Transformers. Sie bestimmen, wann Updates eingespielt werden, und behalten die volle Datenhoheit. Allerdings müssen sie Expertise aufbauen, um Modelle zu betreiben und zu überwachen. Und: In einer reinen On-Prem-Welt fehlen die ständigen Verbesserungen, die ein Cloud-Anbieter liefert.

Daher entscheiden sich, wie bereits angedeutet, viele für hybride Modelle. Die Kernprozesse und sensiblen Daten laufen lokal oder in einer Private Cloud, während generische Services, wie Textzusammenfassung oder Formulierung von Texten, in der Public Cloud abgerufen werden. Auch hier ist die Architektur entscheidend: Schnittstellen sollten so definiert sein, dass sie austauschbar bleiben. Und das DevOps-Team sollte darauf vorbereitet sein, Workloads zu verschieben, wenn Datenschutzanforderungen, Kosten oder Skalierungsbedarfe sich ändern.



Vergleich zwischen Kosten und Flexibilität (Bild 1)

GPU- oder zumindest CPU-Kapazität, muss Modelle aktualisieren und sich um Sicherheit und Skalierung kümmern.

In der Praxis kann es sinnvoll sein, mit einer hybriden Lösung zu starten: Ein KI-Assistent nutzt das Cloud-API für Sprachverständnis, aber die sensiblen Daten bleiben im lokalen Suchindex und werden durch ein lokales LLM vorsortiert und gegebenenfalls anonymisiert. Wichtig ist, dass die gewählte Architektur modular bleibt. So lässt sich ein externer Dienst später austauschen, wenn es wirtschaftlich oder regulatorisch sinnvoll ist.

Cloud, On-Prem oder Hybrid

Die Wahl der Infrastruktur für KI ist nicht nur eine technische, sondern auch eine wirtschaftliche und rechtliche Entscheidung. Cloud-Dienste sind attraktiv, weil sie kurzfristig keine

Die unsichtbare Baustelle – Datenaufbereitung als Erfolgsfaktor

Viele KI-Projekte lassen sich auf dem Papier brillant planen, aber scheitern dann in der Realität – weil die Daten nicht in einem Zustand sind, den ein Modell verarbeiten kann. In großen Unternehmen lagern Daten oft in Data Lakes, werden strukturiert erfasst und sind sauber dokumentiert. KMUs hingegen haben ihre Informationen in E-Mail-Postfächern, auf Netzlaufwerken, in PDFs, Excel-Sheets und ERP-Systemen verstreut. Die größte Arbeit besteht darin, diese Daten für KI nutzbar zu machen.

Der erste Schritt ist die Textextraktion. Bei digitalen PDFs ist das einfach: Die Inhalte lassen sich direkt auslesen. Bei Scans oder eingescannten Dokumenten kommt OCR zum Einsatz. Open-Source-Werkzeuge wie Tesseract sind ein guter Start, aber sie stoßen schnell an Grenzen, wenn Dokumente komplexe Layouts, Tabellen oder Firmenlogos enthalten. In solchen Fällen können Cloud-Dienste wie Azure Form Recognizer oder Google Document AI bessere Ergebnisse liefern, weil sie Layout-Informationen berücksichtigen. Kommerzielle Bibliotheken wie Abbyy FineReader holen noch mehr aus schlecht gescannten Vorlagen heraus. Die Wahl der OCR-Engine hat direkten Einfluss auf die Qualität – jedes falsch erkannte Zeichen kann später die Suche verfälschen. ►

Nach der Textextraktion folgt die Strukturierung. Viele Dokumente enthalten Tabellen, Aufzählungen oder mehrspaltige Layouts. Diese Elemente entscheiden darüber, wie der Text semantisch zusammenhängt. Bibliotheken wie Apache Tika, PdfPig oder Aspose helfen, solche Strukturen zu erkennen: Sie liefern Metadaten (zum Beispiel Überschriften oder Fußnoten) und sorgen dafür, dass Tabellenspalten nicht durcheinandergeraten. Bei Codedateien müssen Tools den Unterschied zwischen Funktionsdefinitionen, Kommentaren und Ausführungslogik verstehen. In Python-Dateien könnte man Module, Klassen und Methoden einzeln extrahieren. Bei E-Mails wiederum ist der Betreff wichtiger Kontext, den man nicht vom Inhalt trennen darf. Diese Strukturerkennung ist sowohl technisch als auch inhaltlich – man muss wissen, was in der Domäne „ein Absatz“ bedeutet.

Danach beginnt die semantische Aufbereitung. Lange Texte werden in kleinere, thematisch zusammenhängende Einheiten – sogenannte „Chunks“ – zerlegt. Chunking ist mehr als nur das Zählen von Zeichen. Ein gut gewählter Chunk enthält genug Kontext, um eine sinnvolle Antwort zu ermöglichen, ohne irrelevante Themen zu mischen. Laut Pinecone ist ein wichtiges Kriterium dabei, dass der Textabschnitt für sich allein verständlich sein sollte. Wählt man Stücke, die zu groß sind, wird die semantische Suche unspezifisch; wählt man sie zu klein, verlieren sie den Kontext. Die Größe hängt auch vom verwendeten Embedding-Modell ab: Einige Modelle können nur 512 Tokens verarbeiten, andere mehrere Tausend. LangChain, LlamaIndex und Semantic Kernel bringen TextSplitter-Klassen mit, die Dokumente nach Absätzen, Überschriften oder Markdown-Tags zerschneiden. Für Code bieten sich andere Regeln an: Man sollte Funktionen nicht zerteilen, sondern als Ganzes behalten. Bei Tabellen müssen Zeilen und Spalten zusammenbleiben – man kann nicht mitten in einer Preisübersicht einen Schnitt setzen.

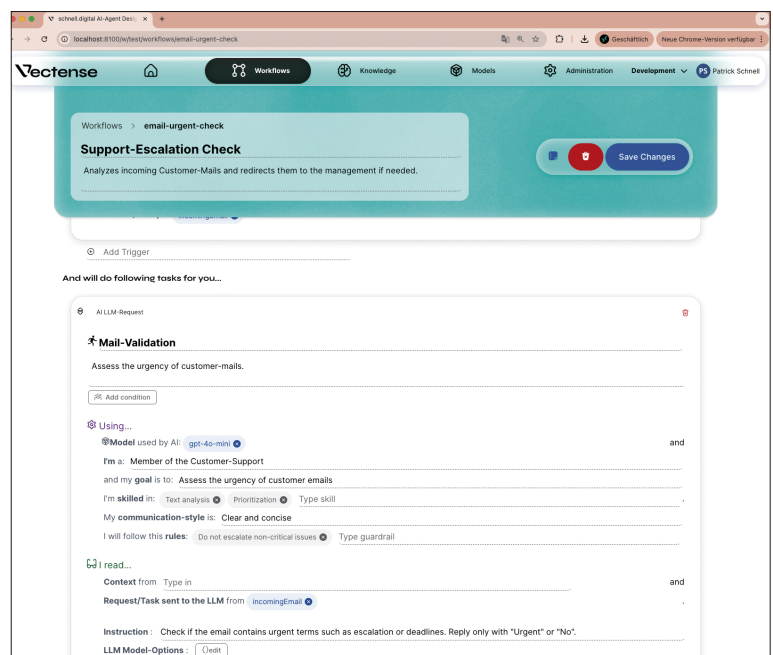
Ein Beispiel verdeutlicht den Unterschied: Eine technische Dokumentation enthält Querverweise. Zerlegt man die Dokumentation blind nach Länge, landet der Verweis in einem Chunk, die Zielstelle aber im nächsten. Der semantische Zusammenhang geht verloren. Besser ist es, Querverweise mit der verlinkten Passage zu verbinden und als gemeinsamen Chunk zu speichern. Bei Code sollten Import-Anweisungen und Funktionsdefinitionen gemeinsam bleiben, weil sie sich gegenseitig bedingen. In E-Mails hingegen sollte man Betreff, Begrüßung und den ersten erklärenden Absatz zusammenhalten, damit die Intention des Schreibers klar wird.

Nach dem Chunking werden die Textabschnitte in numerische Vektoren umgewandelt – Embeddings. Je nach Domäne sollte man unterschiedliche Embedding-Modelle wählen: Allgemeine Modelle wie OpenAIs text-embedding-ada-002 sind gut für Fließtexte, während spezialisierte Modelle für Code (zum Beispiel CodeBERT), juristische Texte (LegalBERT) oder technische Manuals (SciBERT) bessere Ergebnis-

se liefern. Die Embeddings werden dann in eine Vektordatenbank gespeichert. Pinecone, Weaviate oder Qdrant sind darauf spezialisiert, aber auch MongoDB Atlas hat eine eingebaute Vector Search. Diese Datenbanken erlauben es, semantische Ähnlichkeit und klassische Metadatenfilter zu kombinieren – etwa: „Finde alle technischen Anleitungen zum Produkt XY, die sich auf Wartung beziehen und in deutscher Sprache vorliegen.“

Die Kunst der Datenaufbereitung besteht darin, die Chunking-Strategie an die jeweilige Textart anzupassen. Fließtexte wie E-Mails, Blogartikel oder Bedienungsanleitungen lassen sich oft gut nach Kapiteln, Absätzen oder Satzenden aufteilen – hier ist der inhaltliche Zusammenhang weitgehend linear. Bei technischen Spezifikationen, Tabellen oder Formularen sieht das ganz anders aus: Eine Tabelle darf nicht mitten in einer Zeile oder Spalte unterbrochen werden, weil sonst Werte und Kontext auseinandergerissen werden. Bei Formularen müssen Eingabefelder, Beschriftungen und zugehörige Hinweise als Einheit bestehen bleiben.

Besonders heikel sind Codedateien: Würde man hier einfach nach einem Punkt („.“) trennen, könnte das Methoden-



Ein Workflow in Vectense als Story beschrieben (Bild 2)

aufrufe zerstückeln oder logische Zusammenhänge zerschneiden. Stattdessen sollten Entwickler Chunking-Regeln nutzen, die sprachspezifische Strukturen erkennen, etwa Funktionsdefinitionen, Klassen oder Import-Blöcke, und diese als Ganzes behandeln.

Dieser Kontext muss im Team klar sein und in der technischen Umsetzung verankert werden, damit die KI später mit konsistenten, sinnvollen Informationseinheiten arbeiten kann. Eine generische „One size fits all“-Trennung führt in der Praxis oft zu schlechten Suchergebnissen und falschen Antworten.

Praxis – wenn KI den Arbeitsalltag wirklich verändert

Viele Pilotprojekte zeigen: Der Mehrwert von KI entsteht selten durch „magische“ Modelle, sondern durch die Summe aus guter Datenaufbereitung, vernünftiger Prozessintegration und realistischem Erwartungsmanagement. Dabei spielen Entwicklerinnen und Entwickler eine Schlüsselrolle. Sie sind die Übersetzer zwischen Geschäftsanforderung und Technik, sie wählen die passenden Frameworks, definieren Schnittstellen und setzen Qualitätskontrollen.

Betrachten wir noch einmal unser Maschinenbau-Kundenservice-Beispiel. Mit der richtigen Vorbereitung kann das Team ein semantisches Supportsystem aufbauen, das automatisch E-Mails klassifiziert, relevante Handbuchabschnitte findet und Vorschläge für Antworten generiert. Im ersten Schritt mussten dafür alle Handbücher gescannt, extrahiert, strukturiert und semantisch aufbereitet werden. Entwickler entschieden, dass Querverweise mit Zielabschnitt in einem Chunk bleiben und dass Funktionscodes nicht zerschnitten werden. Eine Vektordatenbank wurde eingerichtet, in der Embeddings aller relevanten Textabschnitte gespeichert sind.

Beim Einsatz nutzen die Entwickler entsprechende Frameworks, um die KI-Funktionalität zu orchestrieren. Für jede eingehende Mail wird zunächst ein Embedding erstellt. Die Vektordatenbank liefert die drei ähnlichsten Chunks aus der Dokumentation. Auch hier gibt es nach unserer Erfahrung eventuell notwendige Schritte, die man bedenken muss: Soll die eingehende E-Mail vielleicht vorab durch das LLM gekürzt werden oder auf eine Anzahl an Keywords reduziert werden, um bessere Suchergebnisse zu erzielen?

Diese werden zusammen mit der Anfrage an ein Sprachmodell wie GPT-4o oder Mistral geschickt, das daraus einen Antwortvorschlag generiert. Der Entwickler hat Regeln definiert, wann der Vorschlag automatisch versendet wird und wann er manuell geprüft werden muss. Ist die KI unsicher, greift die Fallback-Logik: Die Mail wird einem Mitarbeiter zugeordnet. Feedbackschleifen sind ebenfalls eingebaut: Der Mitarbeiter kann den Vorschlag als „gut“ oder „schlecht“ markieren. Dieses Feedback fließt zurück und verbessert das System.

Der Effekt ist messbar: Bearbeitungszeiten werden kürzer, weil Routinefragen automatisch beantwortet werden. Die Kundenzufriedenheit steigt, weil Anfragen schneller beantwortet werden. Mitarbeitende sind entlastet und können sich anderen Aufgaben widmen. Und: Durch die Datenaufbereitung und die semantische Suche lernt das System kontinuierlich dazu. Neue Informationen aus der Praxis werden eingepflegt und verbessern die Qualität der Antworten.

Ein anderes Beispiel ist die automatisierte Rechnungsprüfung in einer mittelständischen Steuerberatung. Vor der KI-Einführung mussten Buchhalter jede Rechnung manuell prüfen, Kontierungszeilen eintragen und auf eventuelle Abweichungen achten. Mit einer semantischen Dokumentenverarbeitung werden Rechnungen automatisch ausgelesen, die relevanten Felder erkannt und in das ERP-System übertragen. Ein Machine-Learning-Modell prüft, ob Beträge plausibel sind und ob bestimmte Konten betroffen sind (zum Beispiel außergewöhnliche Beträge). Bei Unstimmigkeiten wird

ein Mensch eingeschaltet. Auch hier sind es Entwickler, die das OCR-Modul anbinden, die Datenströme definieren und das Modell trainieren. Ohne ihr Know-how würde die KI keine präzisen Treffer liefern.

Diese Beispiele zeigen: Qualitative Unterschiede entstehen dort, wo Entwickler die „unsichtbare“ Arbeit erledigen – Daten aufbereiten, Prozesse modellieren, Risiken erkennen. Sie sind es, die dafür sorgen, dass die KI-Antwort auf eine Support-E-Mail nicht nur sprachlich korrekt ist, sondern auch technisch präzise. Sie entscheiden, dass eine Tabelle als Ganzes in den Index gehört und dass Funktionsblöcke im Code nicht zerrissen werden. Dieser Aufwand ist am Anfang hoch, aber er zahlt sich aus, weil er die Grundlage für robuste, skalierbare KI-Lösungen bildet.

Prozesse wie Geschichten erzählen

Damit KI-gestützte Automatisierung im Mittelstand funktioniert, braucht es Tools, die Technologie und Fachlichkeit verbinden. Viele Prozessplattformen setzen auf grafische Node-Editoren, mit denen Workflows aus Kästchen und Pfeilen zusammengekllickt werden. Für Entwickler ist das angenehm, es erinnert an UML-Diagramme. Für Fachanwender – Teamleiterinnen, Organisationsentwickler, Servicemitarbeiter – ist diese Art der Modellierung aber oft fremd. Sie müssen sich erst in Symbole und Pfeile einarbeiten, um simple Änderungen vorzunehmen. Das führt dazu, dass kleine Prozessanpassungen ständig an die IT delegiert werden.

Da wir hier immer wieder die gleichen Herausforderungen und Bausteine in unserer Umsetzung von KI-Projekten sahen, haben wir auf Basis unserer Erfahrung einen modularen „KI-Baukasten“ für KMUs mit dem Namen „Vectense“ entwickelt. Diese Plattform liefert dem Kunden eine intuitive Oberfläche und uns als Entwickler die Möglichkeit, schnell und einfach weitere Integrationen oder Plug-ins zu definieren.

Vectense setzt auf narratives Prozessdesign. Prozesse werden in Form eines erzählerischen Lückentexts beschrieben (Bild 2). Statt einer komplizierten BPMN-Grafik liest sich ein Workflow wie ein gut strukturiertes Kochrezept: „Wenn eine neue Kundenanfrage eingeht, nutze das Modell XY. Bestimme damit das Thema und die Dringlichkeit. Wenn das Thema ‚Vertrieb‘ ist, erstelle ein Ticket im CRM und weise es dem Sales-Team zu. Wenn die Anfrage als ‚dringend‘ markiert ist, benachrichtige zusätzlich den Teamleiter via Teams.“ Der gesamte Prozess ist zu großen Teilen in natürlicher Sprache formuliert, mit Platzhaltern für Eigenschaften, Aktionen und Regeln. Für einen Fachanwender ist der Ablauf intuitiver nachvollziehbar, weil er der Art und Weise entspricht, in der man auch einem Kollegen den Vorgang erklären würde.

Modularität und Erweiterbarkeit machen Vectense zu einem Werkzeug, das technische Teams überzeugt. Die Plattform ist so aufgebaut, dass Entwickler neue Module hinzufügen können: etwa eine Schnittstelle zu einem CRM, eine Anbindung an ein internes Data Warehouse oder einen Zugriff auf ein On-Premises-LLM. Es gibt fertige Integrationen für gängige Dienste (E-Mail, HTTP, Datenbanken), aber man kann eigene „Actions“ und „Triggers“ schreiben, wenn besondere Anforderungen bestehen. Für Domain-Spezialisten ►

bedeutet das: Sie können auf technische Erweiterungen zugreifen, ohne die Komplexität zu sehen.

Für Entwickler bietet dieses Konzept zudem die Möglichkeit, Workflows als YAML-Dateien zu bearbeiten. Das heißt, die Basis der narrativen Beschreibung bleibt eine strukturierte Konfigurationsdatei. Man kann diese YAMLs in der Versionskontrolle ablegen, Review-Prozesse nutzen, Pull Requests einreichen und feinste Einstellungen vornehmen. Im Prinzip ein ähnlicher Aufbau wie bei einer CI/CD-Pipeline. Dieses „Best of both worlds“ adressiert die oft beklagte Lücke zwischen reinen „No Code“-Automatisierungen und einer vollständigen individuellen Entwicklung (Bild 3).

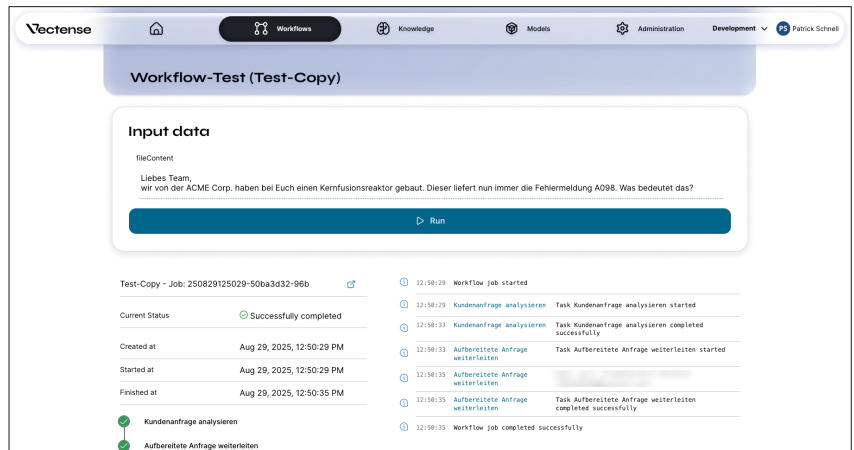
Ein Beispiel verdeutlicht die Stärke: Ein mittelständisches Logistikunternehmen nutzt unsere Plattform, um Anfragen automatisch an die richtige Abteilung oder Ansprechpartner zu senden. Es werden E-Mails aus allgemeinen Postfächern geprüft und mittels der Wissensdatenbank und den formulierten Regeln bestimmt, was der Wunsch des Kunden ist und wohin diese Anfrage geleitet werden soll (zum Beispiel handelt es sich um eine Leistungsanfrage, eine Beschwerde oder Rückfrage zur Rechnungsstellung). Bis dato war es notwendig, dass mehrere Mitarbeiter regelmäßig dieses Postfach prüfen und die Tickets selbst verteilen. Die integrierten Statistiken und Metriken helfen hierbei auch festzustellen, welche Prozesse besonders kostengünstig betrieben werden können und wie die ROIs erreicht werden können.

Die Plattform läuft wahlweise als SaaS oder On-Premises. Das bedeutet für datensensible Unternehmen: Sie können das System in ihrer eigenen Infrastruktur betreiben, die Daten bleiben im Haus. Gleichzeitig besteht die Möglichkeit, Cloud-Dienste einzubinden, wenn es sinnvoll ist. Auch auf der Ebene der KI-Modelle ist die Plattform flexibel. Man kann mit einem Cloud-LLM starten und später auf ein lokales Modell wechseln. Oder man betreibt parallel mehrere Modelle und wählt das passende für jeden einzelnen Schritt in der Prozessdefinition aus. Diese Wahlfreiheit stellt sicher, dass Unternehmen nicht an einen Anbieter gebunden sind und ihre Compliance-Anforderungen einhalten können.

Unter dem Strich steht eine Plattform, die den Brückenschlag zwischen technischer Machbarkeit und fachlicher Verständlichkeit ermöglicht. Sie gibt Entwicklern die Kontrolle über Konfiguration und Erweiterungen, ohne die Fachanwender zu überfordern. Und sie macht es möglich, KI-gestützte Prozesse schnell zu entwerfen, zu testen und zu iterieren. Das ist im Mittelstand besonders wertvoll, weil die Ressourcen knapp sind und Projekte oft neben dem Tagesgeschäft laufen müssen.

Technik allein bringt nicht die Lösung

Künstliche Intelligenz ist nicht länger ein ferner Zukunftsraum, sondern eine konkrete Option für kleine und mittlere Unternehmen. Die Potenziale sind enorm: Automatisierte



Workflows als Pipeline testen und inspizieren (Bild 3)

Support-Prozesse, intelligente Dokumentenverarbeitung, vorausschauende Wartung – all das kann Effizienz steigern, Kosten senken und neue Geschäftsmodelle ermöglichen. Doch der Weg dorthin ist steinig. Datenschutzregeln, Fachkräftemangel und überzogene Erwartungen können jedes Projekt zum Scheitern bringen.

Der Schlüssel zum Erfolg liegt in einer realistischen, strategischen Herangehensweise. Man beginnt nicht mit dem Modell, sondern mit dem Prozess. Man analysiert, wo Daten unstrukturiert und Entscheidungen komplex sind, und wählt dort einen ersten Use Case. Entwickler und Fachexperten arbeiten gemeinsam daran, die Daten aufzubereiten, die richtige Architektur zu wählen und die Integration zu gestalten. Methoden aus der Softwareentwicklung helfen auch in der KI-Welt. Und man kommuniziert offen über Chancen und Grenzen, um Vertrauen aufzubauen und den Hype zu entzaubern.

Plattformen wie Vectense zeigen, dass es Tools gibt, die KI-Prozessautomatisierung auch für Nichttechniker zugänglich machen, ohne die Bedürfnisse von Entwicklern zu vernachlässigen. Indem sie Prozesse schriftlich „erzählen“, senken sie die Einstiegshürde. Indem sie modular und YAML-basiert sind, bleiben sie erweiterbar und versionierbar. So können Unternehmen im eigenen Tempo wachsen, ohne bei jeder Anpassung die IT zu belasten.

Am Ende zählt der Geschäftsnutzen. KI ist kein Selbstzweck, sondern – richtig eingesetzt – ein mächtiges Werkzeug, um im globalen Wettbewerb zu bestehen. Unternehmen, die heute den Mut haben, strukturiert zu starten, sich nicht vom Hype verführen lassen und in die Fähigkeiten ihrer Teams investieren, werden die Früchte ernten. ■



Patrick Schnell

ist Geschäftsführer von schnell.digital, das auf Entwicklung und Architektur von Digitalisierungsprojekten und Beratung spezialisiert ist. Dabei nutzt er unterschiedlichste Technologien wie .NET, Angular, TypeScript und Node.js.

<https://schnell.digital/>

KI-AGENTEN MIT OPENAI UND SEMANTIC KERNEL ENTWICKELN

In gemeinsamer Mission

OpenAI bietet mächtige Tools, die KI-Agenten autonom und leistungsfähig machen, und Microsoft unterstützt mit Semantic Kernel die Entwicklung in .NET.

Nach dem Hype um KI-basierte Chatbots in den letzten Jahren bricht nun die Ära der KI-Agenten an. Doch was ist eigentlich ein Agent, und worin unterscheidet er sich von einem Chatbot? Chatbots reagieren auf einen Prompt mit einer Antwort. Sie beantworten Fragen oder generieren Text oder Bilder. Hierzu nutzen sie ihre eigenen Fähigkeiten beziehungsweise ihr eigenes Wissen. Sie besitzen kein Gedächtnis. Sie arbeiten zustandslos und vergessen stets alles zuvor Geschehene. Daher ist es notwendig, stets den kompletten Dialog – also alle zuvor gestellten Fragen und Antworten – clientseitig zu sammeln und diese beim Fortsetzen der Unterhaltung immer wieder zu übertragen.

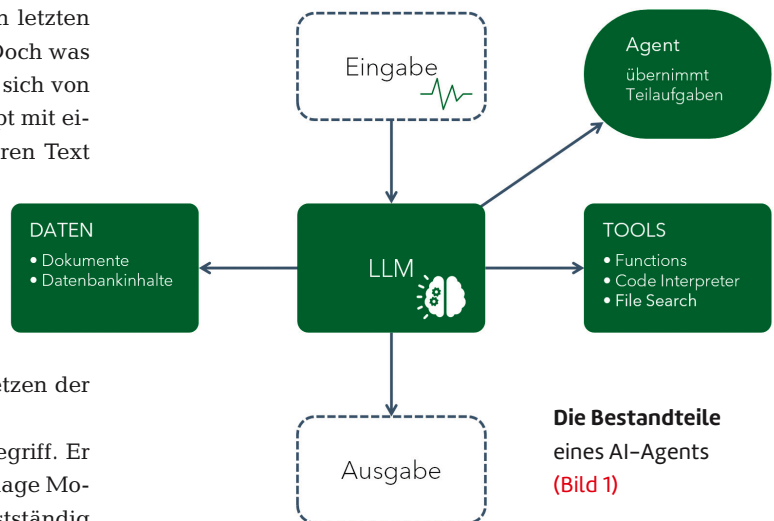
KI-Agent ist dagegen ein eher konzeptioneller Begriff. Er beschreibt eine Architektur, bei der ein Large Language Model (LLM) in die Lage versetzt wird, möglichst selbstständig zu agieren (Bild 1). Hierzu wird es mit externen Datenquellen, Werkzeugen und einem Kurzzeitgedächtnis ausgestattet, um seine natürlichen Limitationen zu umgehen. Agenten können auch zusammenarbeiten. Hierbei ist jeder auf eine Kernfähigkeit spezialisiert und trägt mit dieser zum Gesamtergebnis bei. So lassen sich auch komplexe Aufgaben lösen, für die normalerweise mehrere Chat-Sitzungen mit unterschiedlichen Grundierungen (Systemnachrichten) notwendig wären.

Der Artikel zeigt den Aufbau eines Agentensystems am Beispiel OpenAI. Für die Entwicklung verwenden wir .NET und das Semantic-Kernel-Framework von Microsoft. Um die Funktionsweise besser zu verstehen, ergibt es Sinn, zunächst über die verschiedenen APIs von OpenAI zu sprechen und die erweiterten Fähigkeiten vorzustellen, die diese mitbringen.

Die APIs von OpenAI

OpenAI hat mit seinem Chat Completion API den Standard für die Kommunikation mit einem LLM gelegt. Inzwischen wird es auch von anderen Herstellern unterstützt. Es stellt eine einfache Möglichkeit dar, Chats auf Basis von Text und Grafik zu implementieren. Hierbei bietet es allerdings keine Option, ein Kurzzeitgedächtnis auf Client oder Server zu entwickeln. So müssen stets alle Nachrichten auf Clientseite gesammelt und bei jedem Aufruf erneut übermittelt werden. Auch die Unterstützung von Werkzeugen ist eingeschränkt.

Das API bietet mit dem sogenannten Function Calling die Möglichkeit, eigene APIs in die Konversation einzubeziehen. So könnte man das LLM auffordern, jemandem eine E-Mail mit einem bestimmten Inhalt zu schicken. Dafür würde man dem Modell einen entsprechenden Prompt sowie die Beschrei-



bung einer eigenen Funktion zum Verschicken zur Verfügung stellen. Das Modell würde daraufhin den Mailtext generieren und dem Client signalisieren, die Funktion aufzurufen.

Dieser Mechanismus unterstützt bereits das Konzept von Agenten. Inzwischen bietet OpenAI darüber hinaus eine Vielzahl serverseitiger Werkzeuge, welche die Fähigkeiten eines Agenten erhöhen [1]. So kann mithilfe des Werkzeugs Code Interpreter dynamisch Code erzeugt und ausgeführt werden. Hierdurch können auch Ad-hoc-Aufgaben gelöst werden, die zur Entwurfszeit noch nicht bekannt waren. Ein anderes Werkzeug, OpenAI File Search, erlaubt es auf einfache Weise, Dokumente serverseitig zu speichern und dem LLM Fragen zu dessen Inhalten zu stellen.

Assistants API

Um der Komplexität von Kurzzeitgedächtnis und Werkzeugen zu begegnen, entwickelte OpenAI das Assistants API. Dieses führt das Konzept des Assistenten ein – ein Objekt, welches eine Grundierung bekommt („Du bist ein Kommunikationsprofi und unterstützt beim Verfassen von E-Mails“). Zusätzlich kann der Assistent mit Werkzeugen ausgestattet werden. Hierzu gehört nicht nur das beschriebene Function Calling, sondern auch die serverseitigen Werkzeuge von OpenAI. Darüber hinaus gibt es ein Kurzzeitgedächtnis. Hier können serverseitig sogenannte Threads angelegt werden, auf die sich mehrere Assistenten beziehen können. Der gesamte Dialog wird automatisch auf dem Server gespeichert und muss nicht auf dem Client vorgehalten werden. ►

Responses API

Seit Mai 2025 stellt OpenAI mit dem Responses API eine weitere Programmierschnittstelle zur Verfügung [2]. Sie soll Chat- und Assistants API vereinen und sowohl die Einfachheit des einen und den Funktionsumfang des anderen bieten. Sie soll zwar keines der anderen APIs ersetzen, allerdings wurde die Unterstützung des Assistants API bereits abgekündigt.

Das Responses API bietet aber auch zusätzliche Funktionen. So werden zwei neue Werkzeuge eingeführt, die nur mit ihm verwendet werden können: Web Search stellt eine Mög-

lichkeit dar, eine Frage auf Basis einer Websuche zu beantworten. Computer Use kann hingegen einen Rechner fernsteuern und somit sehr ähnlich wie ein Mensch handeln, um eine Aufgabe zu lösen.

Semantic Kernel von Microsoft

Mit Semantic Kernel [3] hat Microsoft ein Framework entwickelt, das die APIs und Fähigkeiten der unterschiedlichen Modelle beziehungsweise Hersteller abstrahiert und unter einem einheitlichen Programmiermodell bereitstellt [4]. Bei

● Listing 1: Ein einfaches Beispiel für einen Agenten, der beim Schreiben von E-Mails hilft

```
var client = OpenAIAssistantAgent.CreateOpenAIClient(
    new ApiKeyCredential(apiKey));
var assistantClient = client.GetAssistantClient();

AssistantCreationOptions assistantOptions = new()
{
    Name = "Mail-Assistent",
    Instructions = ""
    Du bist ein freundlicher Assistent, der gerne den
    Regeln folgt. Du wirst die erforderlichen Schritte
    ausführen und um Genehmigung bitten, bevor du
    irgendwelche weitreichenden Handlungen vornimmst.
    Wenn der Benutzer nicht genügend Informationen
    bereitstellt, um eine Aufgabe zu erledigen, wirst
    du weiterhin Fragen stellen, bis du genügend
    Informationen hast, um die Aufgabe zu
    vervollständigen.
    ""
};
var assistant = await
    assistantClient.CreateAssistantAsync(
        modelName, assistantOptions);

var agent = new OpenAIAssistantAgent(
    definition: assistant,
    client: assistantClient,
    plugins: new List<KernelPlugin>()
    {
        KernelPluginFactory.CreateFromType<EmailPlugin>()
    });

AgentThread thread =
    new OpenAIAssistantAgentThread(assistantClient);

try
{
    await InvokeAgentAsync("Hallo!");
    await InvokeAgentAsync(
        "Schreibe meinem Chef Thomas eine E-Mail und " +
        "bitte ihn um ein Gespräch in der nächsten " +
        "Woche. Mir würde der 27.10.2025 um 9 Uhr " +
        "oder der 28.10.2025 um 10:30 Uhr passen. Seine " +
        "E-Mail-Adresse ist thomas.meyer@neogeeks.de. " +
        "Thema soll die Zeit- und Personalplanung des " +
        "neuen IT-Projekts sein.");
    while (true)
    {
        await InvokeAgentAsync(Console.ReadLine());
    }
}
finally
{
    await assistantClient.DeleteThreadAsync(thread.Id);
    await assistantClient.DeleteAssistantAsync(
        agent.Id);
}

async Task InvokeAgentAsync(string input)
{
    ChatMessageContent message =
        new(AuthorRole.User, input);
    WriteAgentChatMessage(message);

    await foreach (ChatMessageContent response in
        agent.InvokeAsync(message, thread))
    {
        WriteAgentChatMessage(response);
    }
}

void WriteAgentChatMessage(ChatMessageContent message)
{
    string authorExpression = message.Role ==
        AuthorRole.User ? string.Empty : $" -
        {message.AuthorName ?? "***}";
    string contentExpression = message.Content ??
        string.Empty;
    Console.WriteLine($"{Environment.NewLine} {message.Role}
        {authorExpression}: {message.Content}");
    Console.ResetColor();
}
```

```

C:\Users\JörgNeumann\DropI x + v
# user: Bitte extrahiere die Rechnungsnummer und den Bruttobetrag
aus der hochgeladenen PDF-Datei und gebe sie im JSON-Format zurück
.

# assistant - Document Data Analyser:
[CODE]
import fitz # PyMuPDF

# Path to the PDF file
pdf_path = "/mnt/data/file-H3sPZunzr3H2WuamGbZryA"

# Open the PDF file
doc = fitz.open(pdf_path)

# Extract text from the first page
text = doc[0].get_text()

text

# Assistant - Document Data Analyser: Aus dem Text der ersten Seit
e der hochgeladenen PDF-Datei habe ich die folgenden Informationen
extrahiert:

- **Rechnungsnummer**: R240004
- **Bruttobetrag**: 50.277,50 EUR

Ich werde nun diese Informationen im JSON-Format zurückgeben.

# assistant - Document Data Analyser:
[CODE]
import json

# Extracted data
data = {
    "Rechnungsnummer": "R240004",
    "Bruttobetrag": "50.277,50 EUR"
}

# Convert to JSON format
json_data = json.dumps(data, indent=4)
json_data

# Assistant - Document Data Analyser: Hier sind die extrahierten I
nformationen im JSON-Format:

```json
{
 "Rechnungsnummer": "R240004",
 "Bruttobetrag": "50.277,50 EUR"
}
```

```

Rechnungsdaten auslesen mithilfe dynamischer Codegenerierung (Bild 2)

Abfassung dieses Artikels unterstützt Semantic Kernel lediglich das Chat Completions API und das Assistants API. Unterstützung für das Responses API ist hingegen noch in der Entwicklung. Dies ist wichtig zu erwähnen, da von Semantic Kernel einige Klassen zu Verfügung gestellt werden, die auf Konzepten des jeweiligen API basieren, und sich somit in Zukunft Breaking Changes ergeben können.

Entwicklung eines einfachen Agenten

Zunächst entwickeln wir einen einfachen Agenten, der lediglich E-Mails schreiben und diese auch versenden kann. Das Einzige, was wir dafür benötigen, ist das Semantic-Kernel-Framework sowie ein gültiger API-Key von OpenAI. Wenn Sie Ihre OpenAI-Modelle lieber über Azure AI Foundry betreiben, ist die Vorgehensweise fast identisch. Sie benötigen dann zusätzlich noch den URI Ihres KI-Endpunkts.

Zur Entwicklung von KI-Agenten mit .NET sind einige NuGet-Packages ins Projekt einzubinden, die vom verwendeten Modell abhängen. Möchten Sie etwa auf Modelle von OpenAI zugreifen, sind die folgenden Packages erforderlich:

- Microsoft.SemanticKernel
- Microsoft.SemanticKernel.Agents.Core
- Microsoft.SemanticKernel.Agents.OpenAI

Listing 1 zeigt, wie die Definition des Agenten und das Aufsetzen der Kommunikation über Threads funktioniert. Außerdem ist zu sehen, wie das E-Mail-Plug-in mit dem Modell verknüpft wird. Es enthält die Funktionalität zum Verschicken der Mails und wird von Semantic Kernel automatisch ausgeführt, wenn das Modell dies für sinnvoll erachtet.

Übrigens bezeichnet der Begriff Plug-in das Konzept des Function Callings – in Semantic Kernel ►

● Listing 2: Das E-Mail-Plug-in simuliert das Verschicken von E-Mails

```

using System.ComponentModel;
using Microsoft.SemanticKernel;

#pragma warning disable CA1822, CS1998

namespace SimpleMailAssistant
{
    public class EmailPlugin
    {
        [KernelFunction("send_email")]
        [Description("Sends an email to a recipient.")]
        public async Task SendEmailAsync(
            Kernel kernel,
            List<string> recipientEmails,
            string subject,
            string body)
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine($"
            ##### E-Mail wird gesendet: #####
            ## Empfänger ##: {string.Join("; ",
                recipientEmails)}
            ## Betreff ##: {subject}
            {body}
            ##### E-Mail versendet! #####
            """);
            Console.ResetColor();
        }
    }
}

```


wurde es lediglich anders benannt, um eine herstellerübergreifende Bezeichnung zu etablieren.

Listing 2 zeigt die Definition des E-Mail-Plug-ins: eine normale Klasse, die eine oder mehrere Methoden enthalten kann. Diese werden mit dem Attribut *KernelFunction* deklariert, um sie gegenüber dem Modell sichtbar zu machen. Semantic Kernel extrahiert dann automatisch die entsprechenden Metadaten (Methodensignatur, Parameter, Beschreibung) und überträgt sie an das Modell. Dieses kann mithilfe der Informationen selbst entscheiden, ob die entsprechende Methode zur Lösung der aktuellen Aufgabe aufgerufen werden soll.

Dynamische Codegenerierung mit Code Interpreter

Manchmal ergeben sich spontan Aufgabenstellungen, an die vorher niemand gedacht hat. Bisher musste die vorhandene Software um die entsprechende Funktionalität erweitert werden. Durch das OpenAI-Werkzeug Code Interpreter ist es jedoch möglich, Ad-hoc-Code zu erzeugen, diesen serverseitig auszuführen und das Ergebnis an den Benutzer zurückzugeben.

So könnten Sie das Modell dazu auffordern, einen QR-Code für einen URL zu generieren. Ist das Modell mit dem Werkzeug verbunden, erzeugt es automatisch entsprechenden Python-Code und bringt ihn in einer Sandbox zur Ausführung. Das Ergebnis wird in einem Dateisystem auf dem Server hinterlegt, wo der Anwender es herunterladen kann.

Listing 3 zeigt, wie Sie Ihr Modell mit dem Code-Interpreter-Werkzeug verbinden: Zunächst wird eine Rechnungsdatei hochgeladen. OpenAI bietet dazu ein serverseitiges Dateisystem, das mit dem Benutzeraccount verknüpft ist und im Anschluss mit dem Code Interpreter verbunden wird. Hierzu wird bei der Erstellung neben dem Werkzeug (*Tool*) auch der Verweis auf die hochgeladene Datei (*Tool Resource*) angegeben.

Aus der hochgeladenen Rechnung sollen nun dynamisch die Rechnungsnummer und der Bruttobetrag ermittelt werden. Normalerweise müsste diese Funktionalität manuell entwickelt werden. Der Code Interpreter schafft dies, indem er entsprechende Python-Bibliotheken importiert, Code generiert und das Ergebnis in den Antworttext integriert.

● Listing 3: Ad-hoc-Aufgaben erledigen mit dem Code-Interpreter-Tool

```
var client = OpenAIAssistantAgent.CreateOpenAIClient(
    new ApiKeyCredential(apiKey));
var assistantClient = client.GetAssistantClient();

var fileClient = client.GetOpenAIFileClient();
await using var stream = ReadStream("R240004.pdf");
OpenAIFile fileInfo =
    await fileClient.UploadFileAsync(stream,
        "R240004.pdf", FileUploadPurpose.Assistants);

AssistantCreationOptions assistantOptions = new()
{
    Name = "Document Data Analyser",
    Instructions = "Du bist ein Assistent, der Rechnungen analysiert.",
    Tools = { new CodeInterpreterToolDefinition() },
    ToolResources = new ToolResources()
    {
        {
            CodeInterpreter = new()
            {
                FileIds = { fileInfo.Id }
            }
        }
    };
};
var assistant = await
    assistantClient.CreateAssistantAsync(
        modelName, assistantOptions);
var agent = new OpenAIAssistantAgent(
    assistant, assistantClient);

AgentThread thread =
    new OpenAIAssistantAgentThread(assistantClient);

try
{
    await InvokeAgentAsync("Bitte extrahiere die " +
        "Rechnungsnummer und den Bruttobetrag aus der " +
        "hochgeladenen PDF-Datei unter Verwendung von " +
        "Optical Character Recognition und gebe sie " +
        "im JSON-Format zurück.");
}
finally
{
    await assistantClient.DeleteThreadAsync(thread.Id);
    await assistantClient.DeleteAssistantAsync(
        agent.Id);
    await client.DeleteFileAsync(fileInfo.Id);
}
Console.ReadLine();

async Task InvokeAgentAsync(string input)
{
    ChatMessageContent message =
        new(AuthorRole.User, input);
    WriteAgentChatMessage(message);

    await foreach (ChatMessageContent response in
        agent.InvokeAsync(message, thread))
    {
        WriteAgentChatMessage(response);
    }
}
```

Bild 2 zeigt nicht nur das Ergebnis, sondern auch die Zwischenschritte inklusive des hierbei generierten Codes.

Mit Dokumenten chatten: File Search

Ein weiteres interessantes Werkzeug stellt OpenAI unter dem Namen File Search zur Verfügung. Es basiert auf dem RAG-Konzept (Retrieval Augmented Generation). Hierbei werden Dokumente in kleine Fragmente zerlegt und für diese sogenannte Embeddings erzeugt – semantische Repräsentationen in einem mehrdimensionalen Raum. Diese Embeddings, welche aus einer Kolonne aus Fließkommazahlen bestehen, werden in einer speziellen Vektordatenbank (auch Vector Store genannt) gespeichert.

Stellt der Benutzer nun eine Frage, wird diese nicht direkt an das LLM geschickt, sondern zunächst an die Datenbank. Diese führt daraufhin eine sogenannte Ähnlichkeitssuche durch, bei der sie alle Textfragmente ermittelt, die im semantischen Sinne mit der Frage in Verbindung stehen. Diese Textfragmente werden – zusammen mit der Frage und einer

Anweisung – in einen Prompt zusammengeführt und an das LLM gesendet. Dieses beantwortet daraufhin die Frage anhand der Informationen, die es in den Fragmenten findet.

Wie Sie sehen, ist der Aufbau nicht gerade trivial. Daher bietet OpenAI mit File Search eine einfache Lösung. Die zu durchsuchenden Dokumente werden per API auf den Server übertragen und dort im eigenen Account gespeichert. Zusätzlich erzeugen Sie programmatisch einen Vector Store und verbinden diesen mit den Dateien. Daraufhin kümmert sich OpenAI automatisch um das Zerkleinern, Vektorisieren und Speichern der Texte.

Wenn Sie nun das Modell mit dem File-Search-Werkzeug verbinden und hierbei einen Verweis auf den erstellten Vector Store mit angeben, werden alle Fragen automatisch an die Dokumente gerichtet. Auf diese Weise können Sie sehr leicht die Wissensbasis des LLM um domänenspezifische Informationen erweitern.

Listing 4 zeigt ein einfaches Beispiel. Hier wird eine PDF-Datei, die allgemeine Geschäftsbedingungen einer Firma ►

● Listing 4: Mit dem File-Search-Tool können Dokumente analysiert werden

```
string apiKey =
    Environment.GetEnvironmentVariable("OPENAI_API_KEY");
string modelName = "gpt-4-turbo";

var client = OpenAIAssistantAgent.CreateOpenAIClient(
    new ApiKeyCredential(apiKey));
var assistantClient = client.GetAssistantClient();

var fileClient = client.GetOpenAIFileClient();
await using var stream =
    ReadStream("NeoGeeks AGB 01-2024.pdf");
OpenAIFile fileInfo = await fileClient.UploadFileAsync(
    stream, "NeoGeeks AGB 01-2024.pdf",
    FileUploadPurpose.Assistants);

string vectorStoreId = await client.CreateVectorStore
    Async( [fileInfo.Id], waitUntilCompleted: true);

AssistantCreationOptions assistantOptions = new()
{
    Name = "Legal Assistant",
    Instructions = "Du bist ein Assistent, der
        rechtliche Fragen zu Verträgen beantwortet.",
    Tools = { new FileSearchToolDefinition() },
    ToolResources = new()
    {
        FileSearch = new()
        {
            VectorStoreIds = { vectorStoreId },
        }
    },
};

var assistant = await assistantClient
    .CreateAssistantAsync(modelName, assistantOptions);
var agent = new OpenAIAssistantAgent(
    assistant, assistantClient);
AgentThread thread =
    new OpenAIAssistantAgentThread(assistantClient);
try
{
    await InvokeAgentAsync("Wie hoch haftet " +
        "NeoGeeks bei Sach- und Vermögensschäden?");
}
finally
{
    await thread.DeleteAsync();
    await assistantClient.DeleteAssistantAsync(
        agent.Id);
    await client.DeleteVectorStoreAsync(vectorStoreId);
    await client.DeleteFileAsync(fileInfo.Id);
}

async Task InvokeAgentAsync(string input)
{
    ChatMessageContent message =
        new(AuthorRole.User, input);
    WriteAgentChatMessage(message);

    await foreach (ChatMessageContent response in
        agent.InvokeAsync(message, thread))
    {
        WriteAgentChatMessage(response);
    }
}
```

● Listing 5: Multi-Agentensystem für die Organisation einer Konferenz

```

var builder = Kernel.CreateBuilder();
builder.AddOpenAIChatCompletion(modelId: modelName,
    apiKey: apiKey);
builder.Plugins.AddFromType<EmailPlugin>();
Kernel kernel = builder.Build();

// Agent 1
ChatCompletionAgent texterAgent = new()
{
    Arguments = new KernelArguments(
        new OpenAIPromptExecutionSettings() {
            FunctionChoiceBehavior =
                FunctionChoiceBehavior.Auto() }),
    Kernel = kernel,
    Instructions = ""
    Du bist ein Marketingassistent und hilfst dabei,
    Werbetexte in deutscher Sprache für die Website einer
    Konferenz zu entwerfen, um Kunden zu gewinnen. Wenn
    dir etwas unklar ist, frage beim Organisator nach.
    Wenn du fertig bist, gib den erstellten Text aus.
    "",
    Description = "Ein Chat Bot, der Werbetexte für die
        Website schreibt.",
    Name = "MarketingTexter",
    Id = "MarketingTexter_01",
};

// Agent 2
ChatCompletionAgent communicationAgent = new()
{
    Arguments = new KernelArguments(
        new OpenAIPromptExecutionSettings() {
            FunctionChoiceBehavior =
                FunctionChoiceBehavior.Auto() }),
    Kernel = kernel,
    Instructions = ""
    Du bist ein Assistent, der bei der Kommunikation
    mit den Konferenzsprechern hilft. Deine Aufgabe ist
    es, den Sprechern E-Mails in deutscher Sprache zu
    schreiben und sie aufzufordern, am "Call for Papers"
    teilzunehmen und Ideen für Talks und Workshops
    einzureichen. Wenn dir etwas unklar ist, frage beim
    Organisator nach. Bitte den Organisator um Freigabe
    deines Textes. Bekommst du eine Freigabe, versende
    die E-Mail an die potentiellen Sprecher.
    Nachdem der Text freigegeben wurde, soll die E-Mail
    automatisch an alle Empfänger versendet werden.
    Die Mailadressen der Sprecher sind: "neno@loje.de";
    "joerg.neumann@neogeeks.de"; "thorsten@hans.de".
    "",
    Description = "Ein Chat Bot, der die Kommunikation mit
        den Sprechern übernimmt und diesen E-Mails schreibt.",
    Name = "Communicator",
    Id = "Communicator_02",
};

// Agent 3
ChatCompletionAgent organisatorAgent = new()
{
    Arguments = new KernelArguments(
        new OpenAIPromptExecutionSettings() {
            FunctionChoiceBehavior =
                FunctionChoiceBehavior.Auto() }),
    Kernel = kernel,
    Instructions = ""
    Du bist ein Assistent, der bei der Kommunikation
    mit den Konferenzsprechern hilft. Deine Aufgabe ist
    es, den Sprechern E-Mails in deutscher Sprache zu
    schreiben und sie aufzufordern, am "Call for Papers"
    teilzunehmen und Ideen für Talks und Workshops
    einzureichen. Wenn dir etwas unklar ist, frage beim
    Organisator nach. Bitte den Organisator um Freigabe
    deines Textes. Bekommst du eine Freigabe, versende
    die E-Mail an die potentiellen Sprecher.
    Nachdem der Text freigegeben wurde, soll die E-Mail
    automatisch an alle Empfänger versendet werden.
    Die Mailadressen der Sprecher sind: "neno@loje.de";
    "joerg.neumann@neogeeks.de"; "thorsten@hans.de".
    "",
    Description = "Ein Chat Bot, der die Kommunikation mit
        den Sprechern übernimmt und diesen E-Mails schreibt.",
    Name = "Communicator",
    Id = "Communicator_03",
};

```

enthält, hochgeladen. Daraufhin kann der Anwender spezifische Fragen zu den Inhalten des Vertragswerks stellen.

Im Team mehr erreichen

Jetzt haben wir es geschafft, recht leistungsfähige Agenten zu entwickeln, die einzelne Fähigkeiten besitzen. Bei umfangreichen Aufgabenstellungen benötigen wir jedoch häufig ein ganzes Team von Spezialisten.

Nehmen wir einmal folgendes Beispiel: Wir sind Event-Veranstalter und wollen eine KI-Konferenz ausrichten. Für die Planung benötigen wir ein crossfunktionales Team aus Mitarbeitenden, die sich um die E-Mail-Kommunikation, die Erstellung von Werbetexten für die Website, die Raumplanung, das Catering und anderes mehr kümmern.

Für dieses Szenario soll nun ein KI-Agententeam entwickelt werden. Hierbei werden mehrere Agenten entwickelt und mit jeweils einer speziellen Fähigkeit ausgestattet. Hierfür reicht es, die Spezialisierung einfach zu beschreiben. Zusätzlich werden die Agenten mit entsprechenden Werkzeugen ausgestattet und einem Team zugeordnet.

Um miteinander kommunizieren zu können, benötigen Sie einen gemeinsamen Gruppenchat. Über diesen können Sie Zwischenergebnisse austauschen und miteinander interagieren.

In Semantic Kernel existiert für die Entwicklung von Agententeams ein eigenes API [5]. [Listing 5](#) zeigt, wie es funktioniert.

Nachdem alle Agenten konfiguriert wurden, beginnt der Arbeitsprozess. Hierbei wird eine Endlosschleife gestartet, die so lange läuft, bis die Aufgabe erledigt oder ein Maximallimit überschritten ist. Bei jedem Schritt wird die Kontrolle an den nächsten Agenten abgegeben, der seinen Beitrag leistet und im Anschluss sein Teilergebnis über den Gruppenchat dem Team präsentiert.

Wie Sie in [Listing 5](#) ebenfalls sehen, wird für die Koordination des Teams ein Organisator erzeugt. Er überwacht das Gesamtprojekt, führt das Team und gibt die Teilergebnisse frei. Wenn er den Eindruck hat, dass alle Aufgaben vollständig abgearbeitet wurden, gibt er das Wort „FERTIG“ aus. Dies ist das Schlüsselwort, welches das Framework mithilfe von regulären Ausdrücken aus dem Nachrichtenstrom filtert und daraufhin die Endlosschleife unterbricht.

```

        FunctionChoiceBehavior.Auto() }},
        Kernel = kernel,
        Instructions = """
        Du bist ein Konferenzorganisator. Du organisierst
        Konferenzen und steuerst die anderen Agenten. Du
        führst die anderen Agenten und bewertest ihre
        Ergebnisse. Wenn die anderen Agenten Fragen haben,
        beantwortest du diese. Wenn dir keine sinnvolle
        Antwort einfällt, fragst du den Anwender. Wenn du
        zur Freigabe eines Arbeitsergebnisses aufgefordert
        wirst, bewerte den erstellten Text und gib ihn frei,
        wenn du ihn gut findest. Wenn alle Agenten ihre
        Aufgaben erledigt haben, antworte mit 'FERTIG' und
        gib eine Zusammenfassung der Ergebnisse aus!
        Die folgenden Aufgaben müssen von den Agenten
        erledigt werden:
        1. Es müssen Werbetexte für die Website entworfen
        werden, um Kunden zu gewinnen.
        2. Es müssen potentielle Sprecher angeschrieben
        und sie zum "Call for papers" eingeladen werden.
        """,
        Description = "Ein Chat Bot, der Konferenzen
        organisiert und hierfür andere Agenten steuert.",
        Name = "Organisator",
        Id = "Organisator_03",
    };

    // Gruppen-Chat und Terminierungsregel konfigurieren
    AgentGroupChat chat = new(texterAgent,
        communicationAgent, organisatorAgent)
    {
        ExecutionSettings = new()

```

```

    {
        TerminationStrategy =
            new RegexTerminationStrategy(@"\b(FERTIG)\b")
    {
        Agents = [organisatorAgent],
        MaximumIterations = 50
    }
    };

    // Aufgabe definieren
    var goal = """
    Organisiere eine AI-Konferenz vom 30.09.-1.10.2025.
    Die Konferenz richtet sich an AI-Experten und
    Entwickler und bietet einen Überblick der aktuellen
    Trends im AI-Bereich. Sie bietet am 30.09.
    Ganztages-Workshops an. Am 1.10. finden in drei
    parallelen Tracks Talks von 45 Minuten Länge statt.
    """;

    chat.AddChatMessage(new ChatMessageContent(
        AuthorRole.User, goal));
    // Agententeam starten
    Console.WriteLine($" \x1b[32m>>> {AuthorRole.User} " +
        "> \x1b[37m{goal}.");
    await foreach (var content in chat.InvokeAsync())
    {
        Console.WriteLine($" \n \x1b[32m>>> {content.Role} " +
            "[{content.AuthorName}] > " +
            "\x1b[37m{content.Content}");
    }
    Console.WriteLine($" \x1b[32mIS COMPLETE: " +
        "\u001b[37m{chat.IsComplete}");

```

Fazit

Wie in diesem Artikel gezeigt wurde, können KI-Agenten nicht nur Fragen beantworten, sondern sie sind auch in der Lage, komplexe Aufgaben zu übernehmen. Hierzu können sie mit eigenen APIs oder auch Dritthersteller-Werkzeugen befähigt werden.

Mit Semantic Kernel haben Sie die Möglichkeit, sehr einfach Agenten oder ganze Teams mit .NET zu entwickeln und diese mit unterschiedlichen Modellen zu betreiben.

Worüber wir in diesem Artikel nicht gesprochen haben, ist die Möglichkeit, auch Werkzeuge anderer Anbieter über die MCP-Schnittstelle (Model Context Protocol) anzubinden. Doch dieses Thema bietet Stoff für einen eigenen Artikel. Ähnliches gilt für das Agent Orchestration Framework in Semantic Kernel. Es ermöglicht die Orchestration komplexer Workflows von Agententeams. Wie Sie sehen, ist die Entwicklung von KI-Agenten noch lange nicht am Ende. ■

[1] OpenAI: Verfügbare Tools, <https://platform.openai.com/docs/guides/tools?api-mode=responses>

[2] OpenAI Responses API, <https://platform.openai.com/docs/api-reference/responses>

[3] Microsoft Learn, Einführung in den semantischen Kernel, <https://learn.microsoft.com/de-de/semantic-kernel/>

[4] Microsoft Learn, Semantic-Kernel-Agenten-Framework, <https://learn.microsoft.com/de-de/semantic-kernel/frameworks/agent/?pivots=programming-language-csharp>

[5] Microsoft Learn, Multi-Agent Orchestration mit Semantic Kernel, <https://learn.microsoft.com/de-de/semantic-kernel/frameworks/agent/agent-orchestration/>



Jörg Neumann

ist Gründer und Geschäftsführer der NeoGeeks GmbH und Microsoft MVP. Er berät seit vielen Jahren Unternehmen bei Architektur und Entwurf von Apps und AI-Lösungen, ist Sprecher, Trainer und Autor. Sie erreichen ihn über www.neogeeks.de.

ROLLEN IN DER SOFTWAREENTWICKLUNG

Der Process Owner



In der vorigen Ausgabe war das häufige Nichtbesetzen bestimmter Rollen Thema. Diesmal zeigen wir, warum der Process Owner unersetzlich ist.

Oft werden in der Softwareentwicklung nur die Rollen besetzt, welche direkt an der Umsetzung von funktionalen Anforderungen beteiligt sind – zum Beispiel Entwickler. Diese erzeugen in den Projekten und Unternehmen direkte Umsätze, mit denen das Unternehmen arbeiten kann. Neben solchen Rollen existieren aber auch Rollen, die zwar nur indirekt zum Umsatz beitragen, aber unglaublich wichtig für die professionelle Softwareentwicklung sind. Die Fokussierung auf die Rollen, welche direkt Umsätze generieren, bei gleichzeitiger Vernachlässigung der anderen Rollen ist eines der größten Probleme in der modernen Softwareentwicklung.

Nachdem wir in der vorangegangenen Episode die Hintergründe und Motivationen zu diesem schwerwiegenden Fehler auf Managementebene aufgezeigt haben, widmen wir uns nun konkret der ersten Rolle, der Person, welche für die Entwicklung und Verbesserung des Entwicklungsprozesses zuständig ist: dem Process Owner.

Process ... What?

Sie werden sich vielleicht verwundert fragen, warum Sie die Bezeichnung „Process Owner“ noch nie gehört haben, wenn sie denn so wichtig ist. In der modernen Softwareentwicklung wird diese Rolle (leider) oft „Scrum Master“ genannt. Ich mag diese Bezeichnung ganz und gar nicht, aber nicht weil ich die Rolle per se nicht mag, sondern weil der Bestandteil „Scrum“ eine konkrete Umsetzung suggeriert. Zudem haben an dieser eigentlich großartigen „Scrum-Suppe“ im Lauf der Zeit derart viele Köche herumgekocht, dass das Gesamtergebnis mittlerweile in der Praxis kaum noch nutzbar ist. Damit stelle ich nicht Scrum oder Agilität infrage (das wird noch ein wichtiges Thema in einem zukünftigen Artikel werden), sondern möchte Ihnen nur erklären, warum ich den Begriff „Scrum Master“ vermeide und stattdessen lieber das Kind so nenne wie zu der Zeit, als es Scrum noch gar nicht gab: Process Owner. Die beiden Begriffe beschreiben dieselbe Tätigkeit, allerdings ist der Process Owner oft über die Grenzen der Entwicklung hinaus ebenfalls für das Unternehmen tätig.

Das Aufgabengebiet

Eine der häufiger besetzten Rollen ist der sogenannte Product Owner. Dieser „besitzt“ das Produkt aus fachlicher Perspektive, und seine Hauptaufgabe ist es, dieses von Iteration zu Iteration fachlich gemäß den Vorgaben der Stakeholder weiterzuentwickeln. Der Process Owner hat einen ganz ähnlichen Aufgabenbereich, nur besitzt dieser nicht das Produkt, sondern den Prozess, mit dem das Produkt erstellt wird.

Dieser „Prozess“ ist nun ein recht schwammiger Begriff, schließlich gibt es innerhalb der Softwareentwicklung und in den Unternehmen zahlreiche Prozesse. Analog zu einem Product Owner, dem mehrere Produkte zugeordnet werden können, werden dem Process Owner ein oder mehrere Prozesse zugeordnet. Während Entwickler nun von Iteration zu Iteration mehr Funktionalitäten qualitativ in die Anwendung integrieren und der Product Owner mit jeder Iteration das Produkt durch fachliche Anforderungen erweitert, ist es Aufgabe des Process Owners, von Iteration zu Iteration die ihm zugeordneten Prozesse zu verbessern.

Die Anforderungen

Als Softwareentwickler kennen Sie das Problem: Sie können nur dann das richtige Produkt entwickeln, wenn Sie ordentliche funktionale Anforderungen bekommen, beispielsweise von einem Product Owner. Als ebensolcher Product Owner kennen Sie aber vielleicht auch das Problem, dass Sie nur dann das Produkt gut erweitern können, wenn Sie gute „Ideen“ von den Stakeholdern um das Produkt bekommen. Dieses Prinzip „nur mit Anforderungen/Idee erfolgt eine Aktion“ ist auch für den Process Owner zutreffend, allerdings etwas anders. Während die Anforderungen und Ideen meist an die Product Owner und Entwickler gegeben werden (Push-Prinzip), muss der Process Owner diese Anforderungen in der Regel selbst ermitteln.

Zum einen kann dies anhand von Prozessbeobachtungen geschehen und auch durch Auswertung von Key-Performance-Indicators (KPI) wie Iterationserfolge, Burndown-Charts oder Mitarbeiterzufriedenheit. Der größte Teil der Anforderungen kommt jedoch von den Personen, die mit dem Prozess arbeiten und seine Stärken und Schwächen jeden Tag erleben: den Entwicklern und dem Product Owner. Meist werden deshalb in einem Meeting am Ende der Iteration diese Anforderungen gesammelt und dann in der folgenden Iteration durch den Process Owner als Prozessverbesserungen umgesetzt. Diese Meetings werden als Supervisionen bezeichnet; in Scrum haben sie einen eigenen Namen: Retrospektive. Eine erfolgreiche Supervision oder Retrospektive ist essenziell, damit der Process Owner in der folgenden Iteration genug Anforderungen hat, wie er den Prozess besser machen kann.

Der „perfekte“ Prozess

Kunden sind nie mit unserer Anwendung zufrieden, es kommen im Lauf der Zeit immer mehr und mehr Anforderungen. Genauso sind auch die Entwickler nie fertig mit ihrer Arbeit,

schließlich werden von Iteration zu Iteration neue Funktionen oder Verbesserungen in die Anwendung integriert. Ebenso ist es beim Prozess: Er ist niemals fertig oder perfekt, sondern wird ständig an neue Begebenheiten, Veränderungen und Anforderungen im Projekt angepasst. Durch dieses kontinuierliche Verbessern wird sichergestellt, dass wir immer den aktuell benötigten Prozess für unsere Entwicklung haben.

Eingangs habe ich gesagt, dass einem Process Owner immer ein oder mehrere Prozesse zugeordnet werden, und ich habe bewusst Abstand vom Begriff „Scrum Master“ genommen. Der Scrum Master ist nur für den Entwicklungsprozess zuständig, doch benötigen wir für die Softwareentwicklung allgemein und natürlich im Unternehmen insgesamt wesentlich mehr Prozesse. Der Process Owner ist daher nicht immer nur für den reinen Entwicklungsprozess zuständig, sondern oft für mehrere Prozesse im Unternehmen. Schließlich ist dieses kontinuierliche Verbessern der Prozesse nicht nur für die Entwicklung ein hochwichtiges Thema, sondern für das gesamte Unternehmen. Daher ist der Process Owner oft auch nicht der Entwicklungsabteilung zugeordnet, sondern eine Stabsstelle über den verschiedenen Abteilungen, die am operativen Geschäft teilnehmen.

Die Gefahr

Wie beschrieben ist der Process Owner nicht direkt an der Umsetzung der Anforderungen beteiligt und sorgt damit nicht direkt für Umsatz. Aber ein effizienter Prozess fördert natürlich die effizientere Implementierung von Ideen über die Anforderung hin in das endgültige Produkt. Warum wird also an dieser Rolle gespart? Die Antwort ist nicht einfach, da sehr viele Faktoren eine Rolle spielen. Zum einen sind Prozesse allgemein ein stark vernachlässigtes Thema in vielen Unternehmen, diese könnten in so vielen Bereichen wirtschaftlicher arbeiten, wenn die Prozesse passend wären. Aber dieses Potenzial wird oft nicht erkannt und somit auch die Relevanz des Process Owners nicht gesehen. Ein weiteres Problem ist das fehlende Wissen in Unternehmen zur Softwareentwicklung. Vielfach herrscht die Meinung, dass nur in der Abteilung, welche die Softwareentwicklung aktiv betreibt, Wissen darüber existieren müsse, wie Softwareentwicklung allgemein funktioniert. In dieser Abteilung sehen dann auch alle Beteiligten die Notwendigkeit eines Process Owners (allein schon, weil er in vielen Agilen Frameworks vorgeschrieben ist), aber solche Stellen werden oft außerhalb der Abteilungen genehmigt und vergeben, und dort existiert dieses Kenntnis nicht. Immer wenn ich die Softwareentwicklungsprojekte bei Kunden einem Review unterziehe und am Ende diese fehlenden Rollen dem oberen Management vorstelle, kommt die Frage: „Und warum haben wir diese Rolle dann nicht schon seit Jahren, wenn sie so wichtig ist?“ Die Antwort darauf ist leider immer, dass das Management nicht erkannt hat, wie wichtig diese Rolle ist.

Bleibt diese Rolle nun also unbesetzt, ist der Softwareentwicklungsprozess (und vermutlich viele andere Prozesse im Unternehmen auch) nicht optimal und die Arbeit darin langsamer, fehleranfälliger und letztlich unwirtschaftlicher. Ich bin immer wieder erstaunt darüber, wie überrascht alle Beteilig-

ten in Unternehmen sind, wenn sie nach einiger Zeit der Prozessverbesserungen durch den Process Owner sehen, wie viel besser und schneller man über Jahre hin hätte arbeiten können. Besonders verwundert mich das, weil solche kontinuierlichen Prozessverbesserungen eigentlich kein exklusives Entwicklungsthema sind, sondern eines der wichtigsten Themen im Bereich der Unternehmensführung. Aber leider ist es hier wie so oft im Arbeitsleben: „Der Fisch stinkt vom Kopf.“ Werden von der Führung keine kontinuierlichen Verbesserungen in den Prozessen vorgelebt, überträgt sich das oft auf alle Unternehmensprozesse. Aber für alle Nicht-Geschäftsführer unter Ihnen: Das kann und darf keine Ausrede sein, die Motivation kann auch „von innen heraus“ kommen, also beispielsweise durch Teams, Abteilungen initiiert werden.

Die Besetzung

Allein das Besetzen dieser Rolle böte schon Stoff für einen eigenen Artikel: Wie findet man einen solchen Process Owner? Welche Qualifikationen und Zertifikate sollte er haben? Ich habe in meinen Projekten schon oft solche Rollen besetzt und kann trotzdem keine exakte Antwort geben. Aus meiner Sicht ist die wichtigste Eigenschaft eines Process Owners, Probleme zu sehen und dann alles in seiner Macht Stehende zu tun, um diese Probleme aus dem Weg zu räumen. Ich nenne das immer die „Terrier-Mentalität“ – Problem sehen, festbeißen und dann so lange daran reißen, bis das Problem erledigt ist.

Ich habe einmal einen schulisch weit unterdurchschnittlich qualifizierten Hausmeister zum Scrum Master gemacht, und dieser ist bis heute mit weitem Abstand der beste Scrum Master, den ich je gesehen habe. Mit „weit unterdurchschnittlich qualifiziert“ will ich niemanden denunzieren und auch nicht den Berufsstand des Hausmeisters gering schätzen, aber diese vollkommen fachfremde und weder ausgebildete noch studierte Person betreut heute sechs verschiedene Entwicklungsprozesse in einer vorbildlichen Art und Weise. Er ist eben genau dieser Terrier und wird für seine exzellente Arbeit vom ganzen Unternehmen geschätzt. Daraus können Sie sicherlich ableiten, wie schwer die Besetzung dieser Rolle ist und wie hilfreich es gleichzeitig sein kann, bei dieser Entscheidung über den Tellerrand hinauszuschauen.

Fazit

Der Process Owner ist nicht nur für die Entwicklung, sondern für das ganze Unternehmen von großem Wert. Machen Sie nicht den Fehler und besetzen Sie diese Rolle nicht, weil Sie ihren direkten Mehrwert nicht sehen. Ähnlich ist es beim Architekten, um den es im nächsten Artikel gehen wird! ■



David Tielke

ist freiberuflicher Berater und von Microsoft zertifizierter Trainer für die Anwendungsentwicklung auf der .NET-Plattform. Darüber hinaus hat er sich auf die Bereiche Softwarearchitektur, Softwarequalität und ALM spezialisiert.
mail@david-tielke.de

SKALIERBARE TESTARCHITEKTUREN MIT KI-TOOLS UND TESTDATA-BUILDERN ENTWICKELN

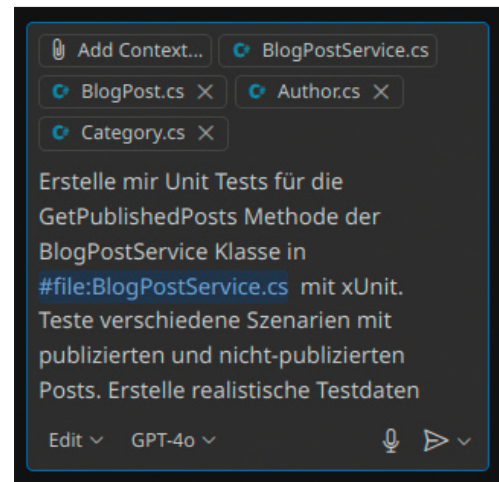
Bob, der Testdaten-Baumeister

Der praktische Weg zu robusten, erweiterbaren Tests trotz automatisierter Code-Generierung.

Heutzutage lässt man sich Tests oft mit KI generieren – und das ist auch gut so. GitHub Copilot, ChatGPT und Co. können in Sekundenschnelle umfangreiche Testsuiten erstellen und dabei viel Zeit sparen. Allerdings entsteht dabei oft ein Problem: duplizierter Code in den Tests. Naive Prompts wie „Generiere Tests für Klasse XY“ führen dazu, dass die KI immer wieder die gleichen Testdaten-Setups generiert, anstatt wiederverwendbare Strukturen zu schaffen.

Diese Code-Duplikation in Tests ist teuer. Ändert sich das Datenmodell der Anwendung – etwa durch das Hinzufügen neuer Pflichtfelder oder die Änderung von Datentypen –, müssen alle Tests angepasst werden, die diese Entitäten ver-

Bei GitHub Copilot kann man Source-Dateien per Klick zum Kontext hinzufügen (Bild 1)



● Du möchtest gerne mehr erfahren?

Das freut uns, denn Testdata-BUILDER sind nur der Anfang einer durchdachten Test-Architektur. Deshalb veröffentlichen wir ab Erscheinen des vorliegenden Artikels auf developer-world.de/insights weitere Beiträge, die diese Grundlagen um zusätzliche Aspekte und Themen erweitern und ausschließlich online zur Verfügung stehen. Derzeit geplant sind folgende Beiträge:

- **Hierarchische Testdata-BUILDER für komplexe Objektstrukturen:** Wenn Entitäten verschachtelte Abhängigkeiten haben oder komplexe Aggregates bilden, stoßen einfache Builder an ihre Grenzen. Wir zeigen, wie man Builder hierarchisch strukturiert und dabei sowohl Flexibilität als auch Einfachheit bewahrt.
- **Realistische Testdaten mit Bogus und GitHub Copilot:** Für umfangreichere Tests benötigt man oft große Mengen realistischer Testdaten. Die Integration von Testdata-Buildern mit der Bogus-Library eröffnet neue Möglichkeiten für die automatisierte Generierung von Testdaten – auch hierbei kann GitHub Copilot wertvolle Unterstützung leisten.
- **Testdata-BUILDER mit Domain-spezifischen Sprachen (DSL):** Während einfache Fluent-APIs bereits die Lesbarkeit von Tests verbessern, ermöglichen Domain-spezifische Sprachen die Erstellung noch ausdrucksstärkerer Test-Setups. Wir zeigen, wie man Builder zu einer natürlichen Geschäftssprache erweitert und dabei komplexe Testszenerien in wenigen, verständlichen Zeilen abbildet.

wenden. Bei einer größeren Testsuite kann dies Hunderte von Stellen betreffen. Was als vermeintliche Zeitersparnis durch KI-generierte Tests begann, kann so schnell zu einem Wartungs Albtraum werden.

Gerade in Zeiten, in denen man Code von KI generieren lässt, werden Testing und die Lesbarkeit von Tests immer wichtiger. Die Automatisierung der Code-Erstellung verlagert den Fokus: Wenn die KI funktionalen Code erstellt, müssen wir Menschen sicherstellen, dass kritische Elemente der Software zuverlässig verifiziert werden. Automatische Tests sind dabei eines der wichtigsten Hilfsmittel, da sie in Sekundenschnelle verifizieren können, ob eine bestimmte Anforderung an die Software korrekt implementiert ist oder nicht.

Hinzu kommt ein weiterer Aspekt: Lesbare Tests sind nicht nur ein Werkzeug zur Qualitätssicherung, sondern auch eine hervorragende Dokumentation der Software. Ein gut geschriebener Test zeigt nicht nur, was eine Funktion tut, sondern auch, wie sie verwendet werden soll und welche Randfälle zu beachten sind. In Zeiten immer schnellerer Entwicklungszyklen und häufiger Code-Generierung durch KI können Tests eine verlässliche Dokumentation sein, die mit dem Code mitwächst.

Aus diesem Grund ist es auch im Zeitalter von KI notwendig, auf bewährte Entwurfsmuster zu setzen. Das Builder-Pattern für Testdaten ist ein solches Entwurfsmuster, das nicht nur die Lesbarkeit und Wartbarkeit von Tests verbessert, sondern

● Listing 1: Ein einfaches Blogsystem als Beispiel

```

public class BlogPost
{
    public required int Id { get; init; }
    public required string Title { get; init; } =
        string.Empty;
    public required string Content { get; init; } =
        string.Empty;
    public DateTime? PublishedDate { get; init; }
    public required Author Author { get; init; }
    public Category? Category { get; init; }
    public bool IsPublished { get; init; }
}

public class Author
{
    public required int Id { get; init; }
    public required string Name { get; init; }
    public required string Email { get; init; }
}

public class Category
{
    public required int Id { get; init; }
    public required string Name { get; init; }
    public string Description { get; init; } =
        string.Empty;
}

public class BlogPostService
{
    public IEnumerable<BlogPost>
        GetPublishedPosts(IEnumerable<BlogPost> posts)
    {
        return posts.Where(post => post.IsPublished);
    }
}

```

auch hervorragend mit KI-Tools kombiniert werden kann. Im vorliegenden Artikel wird gezeigt, wie man Tests lesbar und wartbar gestaltet und dennoch KI dafür einsetzen kann.

Ein Praxisbeispiel

Nehmen wir als Beispiel das in Listing 1 gezeigte einfache Blog-System mit den Entitäten *BlogPost*, *Author* und *Category* sowie einem *BlogPostService*, der Blogposts nach Kategorien filtern kann.

Möchte man nun Tests für diese *GetPublishedPosts*-Methode schreiben, könnte man eine generative KI mit folgendem Prompt beauftragen:

"Erstelle mir Unit Tests für die *GetPublishedPosts* Methode der *BlogPostService* Klasse mit xUnit. Teste verschiedene Szenarien mit publizierten und nicht-publizierten Posts. Erstelle realistische Testdaten."

Wichtig zu beachten ist dabei, dass der Sourcecode aus Listing 1 Teil des Kontexts sein muss. Bei GitHub Copilot benutzt man dafür die praktische Funktion *Add Context* oder erwähnt die Dateien direkt im Text, indem man ihnen eine Raute voranstellt (Bild 1). Bei Chat-basierten Systemen wie ChatGPT oder Claude.ai kopiert man einfach den Sourcecode mit in das Chatfenster.

Das Ergebnis wird in etwa so wie in Listing 2 aussehen – vermutlich werden noch mehr Tests generiert, zum Verständnis reicht es aber, wenn wir uns diese beiden Tests genauer anschauen.

Auf den ersten Blick sehen diese Tests ganz gut aus – sie testen die gewünschte Funktionalität und sind verständlich. Bei genauerer Betrachtung fallen jedoch mehrere Probleme auf:

- **Code-Duplikation:** Die Erstellung von *BlogPost* und *Author* wird in beiden Tests wiederholt. Wird das System erweitert und benötigt es zum Beispiel zusätzliche Felder, müssen diese an mehreren Stellen angepasst werden.
- **Schlechte Lesbarkeit:** Der eigentliche Test wird von der umfangreichen Testdaten-Erstellung überlagert. Es ist schwer zu erkennen, was genau getestet wird und dass eigentlich nur die Eigenschaft *IsPublished* für den Test relevant ist.
- **Wartungsaufwand:** Ändert sich die Struktur der Entitäten, müssen alle Tests angepasst werden. Bei einer größeren Testsuite mit Hunderten von Tests führt dies schnell zu einem nicht gerade unerheblichen Aufwand.
- **Fehlende Flexibilität:** Möchte man einen Test mit leicht veränderten Daten durchführen (zum Beispiel einen *BlogPost* mit *Category*), muss man die gesamte Testdaten-Erstellung kopieren und anpassen.

Diese Probleme sind keineswegs ein Fehler von GitHub Copilot – sie entstehen durch die Art, wie wir die KI einsetzen. Die Lösung liegt darin, bewährte Design-Patterns zu verwenden und die KI so zu instruieren, dass sie diese Patterns ebenfalls nutzt.

Grundlagen des Testdata-Builders

Das Builder-Pattern [1] ist ein bewährtes Entwurfsmuster, das sich hervorragend für die Erstellung von Testdaten eignet; ►



Ein Testdata-Builder übernimmt die Instanzierung eines Entity (Bild 2)

● Listing 2: Von GitHub Copilot generierte Tests

```

public class BlogPostServiceTests
{
    [Fact]
    public void GetPublishedPosts_NoPostsPublished_
        ReturnsEmpty()
    {
        // Arrange
        var service = new BlogPostService();
        var posts = new List<BlogPost>
        {
            new BlogPost
            {
                Id = 1,
                Title = "Post 1",
                Content = "Content 1",
                IsPublished = false,
                Author = new Author
                {
                    Id = 1,
                    Name = "Author 1",
                    Email = "author1@example.com",
                },
            },
            new BlogPost
            {
                Id = 2,
                Title = "Post 2",
                Content = "Content 2",
                IsPublished = false,
                Author = new Author
                {
                    Id = 2,
                    Name = "Author 2",
                    Email = "author2@example.com",
                },
            },
        };

        // Act
        var result = service.GetPublishedPosts(posts);

        // Assert
        Assert.Empty(result);
    }

    [Fact]
    public void GetPublishedPosts_MixedPosts_
        ReturnsOnlyPublishedPosts()
    {
        // Arrange
        var service = new BlogPostService();
        var posts = new List<BlogPost>
        {
            new BlogPost
            {
                Id = 1,
                Title = "Post 1",
                Content = "Content 1",
                IsPublished = true,
                Author = new Author
                {
                    Id = 1,
                    Name = "Author 1",
                    Email = "author1@example.com",
                },
            },
            new BlogPost
            {
                Id = 2,
                Title = "Post 2",
                Content = "Content 2",
                IsPublished = false,
                Author = new Author
                {
                    Id = 2,
                    Name = "Author 2",
                    Email = "author2@example.com",
                },
            },
            new BlogPost
            {
                Id = 3,
                Title = "Post 3",
                Content = "Content 3",
                IsPublished = true,
                Author = new Author
                {
                    Id = 3,
                    Name = "Author 3",
                    Email = "author3@example.com",
                },
            },
        };

        // Act
        var result = service.GetPublishedPosts(posts);

        // Assert
        Assert.Equal(2, result.Count());
        Assert.All(result, post =>
            Assert.True(post.IsPublished));
    }
}

```

ich selbst setze es seit Jahren erfolgreich in meinen Projekten ein. Ein Testdata-Builder kapselt die Logik zur Erstellung von Testdaten und bietet eine einfach zu benutzende Schnittstelle, um diese Daten flexibel zu konfigurieren. Die Grundidee ist einfach: Anstatt Testdaten direkt in den Tests zu erstellen, verwendet man eine Builder-Klasse, die mit sinnvollen Standardwerten vorkonfiguriert ist (Bild 2). Diese Standardwerte können dann bei Bedarf überschrieben werden.

Die Nutzung des Builder-Patterns bietet einige Vorteile für die Testentwicklung. Einmal erstellt, kann der Builder in allen Tests wiederverwendet werden. Änderungen an dem Entity wirken sich dabei nur auf den Builder aus, jedoch nicht auf jeden einzelnen Test (vergleiche Bild 3 und Bild 4). Die Lesbarkeit der Tests verbessert sich erheblich, da nur die für den Test relevanten Daten explizit gesetzt werden müssen, während der Rest durch sinnvolle Standardwerte abgedeckt wird. Mit einem sogenannten Fluent-API (siehe weiter unten) können verschiedene Testszenarien vergleichsweise einfach abgebildet werden, ohne dass der gesamte Setup-Code dupliziert werden muss. Die Wartbarkeit profitiert davon, dass strukturelle Änderungen an den Entitäten nur Anpassungen im Builder erfordern, nicht aber in jedem einzelnen Test.

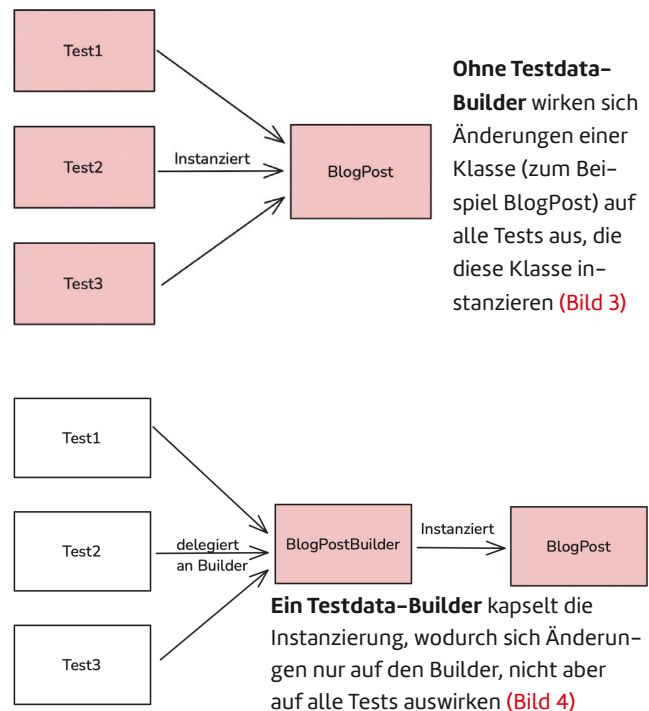
Wie bei jedem Pattern gibt es auch beim Testdata-Builder Nachteile, die man berücksichtigen sollte. Der initiale Aufwand für das Erstellen der Builder-Klassen erfordert zunächst mehr Zeit als das direkte Erstellen der Testdaten. Dieser Aufwand amortisiert sich jedoch schnell bei der Verwendung in mehreren Tests. Mit geschicktem Einsatz von KI lässt er sich zudem auf ein Minimum beschränken. Ein weiterer Nachteil besteht darin, dass Builder-Klassen dem Projekt zusätzlichen Code hinzufügen, der ebenfalls gewartet werden muss. Bei sehr einfachen Entitäten kann dies unverhältnismäßig sein. Ein weiteres Risiko liegt in der möglichen Überabstraktion: Bei zu komplexen Buildern kann es schwierig werden, spezifische Testszenarien abzubilden. Ein guter Builder sollte flexibel genug sein, um verschiedene Szenarien zu unterstützen, ohne zu komplex zu werden. Trotz dieser Nachteile überwiegen die Vorteile in den meisten Fällen deutlich, insbesondere bei Projekten mit umfangreichen Testsuiten.

Fluent-API-Design

Ein Testdata-Builder sollte ein sogenanntes Fluent-API [2] bereitstellen, mit dem es möglich ist, die Testdaten in natürlicher Sprache und dadurch auf verständliche Weise zu konfigurieren:

```
var blogPost = new BlogPostBuilder()
    .WithTitle("My Test Post")
    .WithPublishedDate(DateTime.Now.AddDays(-1))
    .WithCategory(new CategoryBuilder().WithName(
        "Technology").Build())
    .Build();
```

Diese Syntax macht sofort klar, welche Aspekte der Testdaten für den konkreten Test relevant sind, während alle anderen Eigenschaften durch sinnvolle Standardwerte abgedeckt werden, die im Test nicht auftauchen.



Implementation eines Testdata-Builders

Nachdem wir die Grundlagen verstanden haben, schauen wir uns an, wie ein Testdata-Builder für unser Blog-System konkret implementiert wird. Listing 3 zeigt Builder für die Entities *Author* und *Category*. Jeder Builder folgt dabei einem klaren Muster: Private Felder speichern die Werte, die im finalen Objekt verwendet werden. Jedes Feld hat einen sinnvollen Standardwert, der ein realistisches Testszenario abbildet. Die *With*-Methoden ermöglichen es, einzelne Werte zu überschreiben, und geben jeweils *this* zurück, um Method Chaining zu ermöglichen. Die *Build()*-Methode erstellt schließlich das fertige Objekt.

Der *BlogPostBuilder* ist etwas komplexer, da er Abhängigkeiten zu anderen Entitäten hat. Ein wichtiges Detail ist hierbei, dass die Standardwerte für *_author* und *_category* durch die entsprechenden Builder erstellt werden. Dadurch sind die Builder lose gekoppelt und können unabhängig voneinander verwendet werden (siehe Listing 4).

Best Practices für Testdata-Builder

Grundsätzlich gibt es neben der gezeigten auch andere Möglichkeiten, um Testdata-Builder zu implementieren. In der Praxis haben sich allerdings folgende Practices bewährt:

- **Realistische Standardwerte:** Die Standardwerte sollten ein realistisches Szenario abbilden. Ein Author mit dem Namen *John Doe* ist aussagekräftiger als *Author1*.
- **Fluent-API:** Alle *With*-Methoden sollten einen sprechenden Namen haben und *this* zurückgeben, sodass man sie mit Method Chaining aneinanderhängen kann und sich das Setup fast wie natürliche Sprache lesen lässt.
- **Separate Builder-Klassen:** Jede Entität sollte ihren eigenen Builder haben. Dies hält die Komplexität überschaubar und ermöglicht die Wiederverwendung. ►

● Listing 3: AuthorBuilder und CategoryBuilder

```

public class AuthorBuilder
{
    private int _id = 1;
    private string _name = "John Doe";
    private string _email = "john.doe@example.com";

    public AuthorBuilder WithId(int id)
    {
        _id = id;
        return this;
    }

    public AuthorBuilder WithName(string name)
    {
        _name = name;
        return this;
    }

    public AuthorBuilder WithEmail(string email)
    {
        _email = email;
        return this;
    }

    public Author Build()
    {
        return new Author
        {
            Id = _id,
            Name = _name,
            Email = _email,
        };
    }
}

public class CategoryBuilder
{
    private int _id = 1;
    private string _name = "Technology";
    private string _description =
        "All about technology and innovation.";

    public CategoryBuilder WithId(int id)
    {
        _id = id;
        return this;
    }

    public CategoryBuilder WithName(string name)
    {
        _name = name;
        return this;
    }

    public CategoryBuilder WithDescription(
        string description)
    {
        _description = description;
        return this;
    }

    public Category Build()
    {
        return new Category
        {
            Id = _id,
            Name = _name,
            Description = _description,
        };
    }
}

```

- **Konsistente Namenskonventionen:** Alle Builder-Methoden, die die Daten konfigurieren, sollten mit *With* beginnen, und das Entity sollte in einer Methode namens *Build()* gebaut werden. Somit wird das API konsistent gehalten.

Listing 5 zeigt den nun deutlich fokussierteren Test. Es wird sofort klar, dass nur die *IsPublished*-Eigenschaft für diesen Test relevant ist, während alle anderen Eigenschaften durch sinnvolle Standardwerte abgedeckt werden.

Builder mit gezielten Prompts generieren

Nachdem wir die Vorteile von Testdata-Buildern kennengelernt haben, stellt sich die Frage: Wie können wir unseren KI-Assistenten dazu bringen, diese Builder zu verwenden und zu erstellen? Der Schlüssel für einen erfolgreichen Einsatz im

Testalltag liegt darin, spezifische und detaillierte Anweisungen zu geben. Statt eines naiven Prompts wie „*Erstelle Tests*“ sollten wir die KI explizit anweisen, Testdata-Builder zu verwenden. Ein effektiverer Prompt könnte wie folgt aussehen:

"Erstelle einen Testdata-Builder für die Author-Klasse. Der Builder soll das Fluent Interface Pattern verwenden mit With-Methoden für jede Eigenschaft. Verwende sinnvolle Standardwerte und gib 'this' bei jeder With-Methode zurück für Method Chaining. Die Build()-Methode soll das fertige Author-Objekt zurückgeben."

Noch besser fällt das Ergebnis aus, wenn man bereits einen Builder im Projekt vorliegen hat und das KI-Modell anweist,

● Listing 4: BlogPostBuilder

```

public class BlogPostBuilder
{
    private int _id = 1;
    private string _title = "Sample Blog Post";
    private string _content =
        "This is a sample blog post content.";
    private DateTime _publishedDate =
        DateTime.Now.AddDays(-1);
    private Author _author =
        new AuthorBuilder().Build();
    private Category _category =
        new CategoryBuilder().Build();
    private bool _isPublished = true;

    public BlogPostBuilder WithId(int id)
    {
        _id = id;
        return this;
    }

    public BlogPostBuilder WithTitle(string title)
    {
        _title = title;
        return this;
    }

    public BlogPostBuilder WithContent(string content)
    {
        _content = content;
        return this;
    }

    public BlogPostBuilder WithPublishedDate(
        DateTime publishedDate)
    {
        _publishedDate = publishedDate;
        return this;
    }

    public BlogPostBuilder WithAuthor(Author author)
    {
        _author = author;
        return this;
    }

    public BlogPostBuilder WithCategory(
        Category category)
    {
        _category = category;
        return this;
    }

    public BlogPostBuilder WithIsPublished(
        bool isPublished)
    {
        _isPublished = isPublished;
        return this;
    }

    public BlogPost Build()
    {
        return new BlogPost
        {
            Id = _id,
            Title = _title,
            Content = _content,
            PublishedDate = _publishedDate,
            Author = _author,
            Category = _category,
            IsPublished = _isPublished,
        };
    }
}

```

auf dieser Grundlage einen ähnlichen Builder für eine andere Entität zu erstellen:

"Erstelle einen CategoryBuilder nach dem gleichen Muster wie der AuthorBuilder. Verwende sinnvolle Standardwerte für eine Technology-Kategorie."

Instruction-Files für konsistente Konventionen

GitHub Copilot bietet mit Instruction-Files ein nützliches Feature, um solche allgemeinen Anweisungen in jeden Prompt aufzunehmen. Diese Dateien ermöglichen es, GitHub Copilot projektspezifische Konventionen und Patterns beizubringen, die dann automatisch bei der Code-Generierung berücksichtigt werden.

Für Testdata-Builder erstellen wir eine Datei namens *.github/copilot-instructions.md* in unserem Projekt und füllen sie mit den Zeilen aus [Listing 6](#).

Diese Instruction-Datei wirkt wie ein permanenter Kontext für GitHub Copilot. Sobald sie im Projekt vorhanden ist, berücksichtigt Copilot diese Konventionen automatisch bei der Code-Generierung. Dass ein Instruction-File aktiv ist, erkennt man im Kontext des Chats ([Bild 5](#)). Bei Chat-basierten Systemen kann man oft einen sogenannten System Prompt hinzufügen, der den gleichen Zweck erfüllt.

Dadurch können wir nun viel kürzere und einfachere Prompts verwenden:

"Erstelle einen AuthorBuilder."

● Listing 5: Deutlich fokussierter Test durch Einsatz der Testdata-Builder

```
[Fact]
public void GetPublishedPosts_MixedPosts_
    ReturnsOnlyPublishedPosts()
{
    // Arrange
    var publishedPost = new BlogPostBuilder()
        .WithIsPublished(true).Build();
    var anotherPublishedPost = new BlogPostBuilder()
        .WithIsPublished(true).Build();
    var unpublishedPost = new BlogPostBuilder()
        .WithIsPublished(false).Build();

    var posts = new List<BlogPost> { publishedPost,
        anotherPublishedPost, unpublishedPost };
    var service = new BlogPostService();

    // Act
    var result = service.GetPublishedPosts(posts);

    // Assert
    Assert.Equal(2, result.Count());
    Assert.All(result, post =>
        Assert.True(post.IsPublished));
}
```

Auch beim Erstellen von Tests berücksichtigt Copilot automatisch die definierten Konventionen:

"Erstelle einen Test für GetPublishedPosts mit nur unpublizierten Posts."

Convenience-Methoden für Testdata-Builder

Für komplexere Szenarien kann uns die KI auch dabei helfen, Builder mit spezifischen Konfigurationen zu erstellen:

"Erweitere den BlogPostBuilder um eine WithDraftPost() Convenience-Methode für unpublizierte Posts."

Das Ergebnis wird dann in etwa wie folgt aussehen.

```
public class BlogPostBuilder
{
    // ... bestehender Code ...
    public BlogPostBuilder
        WithDraftPost()
    {
        _isPublished = false;
        _publishedDate = DateTime
            .Now.AddDays(1);
        // Future date for draft
        return this;
    }
}
```

Die folgende Convenience-Methode macht die Tests besser lesbar:

```
var draftPost =
    new BlogPostBuilder()
        .WithDraftPost().Build();
```

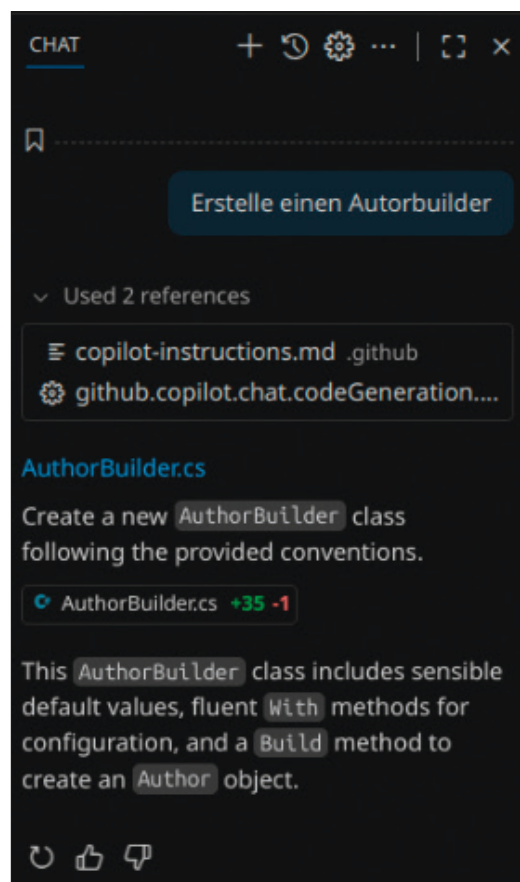
Fazit

Testdata-Builder ist ein bewährtes Pattern, das gerade im Zeitalter KI-unterstützter Entwicklung an Bedeutung gewinnt. Während Code-Completion- und Code-Generierungswerkzeuge wie GitHub Copilot und andere KI-Tools uns dabei helfen, in kurzer Zeit funktionsfähigen Code zu generieren, sorgen Testdata-Builder dafür, dass unsere Tests wartbar, lesbar und nachhaltig bleiben.

Die Kombination aus Builder-Pattern und intelligenter KI-Nutzung bietet das Beste aus beiden Welten: die Geschwindigkeit der automatisierten Code-Generierung und die Qualität durchdachter Test-Architektur. Durch gezieltes Prompting und eine gut umgesetzte Strukturierung im Instruction-File beziehungsweise System Prompt wird die KI zu einer wertvollen Verbündeten bei der Erstellung hochwertiger Testsuiten.

Die Investition in eine solide Test-Infrastruktur zahlt sich langfristig aus. Testdata-Builder sind dabei ein wichtiger Baustein, der die Grundlage für wartbare und verständliche Tests legt. In Kombination mit KI-Tools wie GitHub Copilot entsteht auf diese Weise eine produktive Entwicklungsumgebung, die sowohl Geschwindigkeit als auch Qualität gewährleistet.

Die Zukunft der Softwareentwicklung liegt nicht darin, Entwickler durch KI zu ersetzen oder die KI ganz zu ignorieren, sondern darin, sie intelligent zu nutzen und durch bewährte Patterns zu ergänzen. Testdata-Builder sind ein per-



Ein Instruction-File wird automatisch dem Kontext von GitHub Copilot hinzugefügt (Bild 5)

Listing 6: Copilot Instructions für Testdata-Builder

```
# GitHub Copilot Instructions für Testdata-Builder

## Testdata-Builder Konventionen

Beim Erstellen von Testdata-Buildern für dieses
Projekt gelten folgende Regeln:

### Naming Conventions
- Builder-Klassen enden immer mit "Builder"
  (z.B. BlogPostBuilder, AuthorBuilder)
- Alle Konfigurationsmethoden beginnen mit "With"
  (z.B. WithTitle, WithAuthor)
- Die finale Methode heißt immer "Build()" und gibt
  das erstellte Objekt zurück

### Fluent Interface
- Alle With-Methoden geben 'this' zurück für Method
  Chaining
- Beispiel: `new BlogPostBuilder().WithTitle("Test")
  .WithAuthor(author).Build()`

### Standardwerte
- Jeder Builder hat sinnvolle Standardwerte für alle
  Eigenschaften
- IDs beginnen bei 1
- Datum-Eigenschaften verwenden relative Zeitpunkte
  (DateTime.Now.AddDays(-X))
- String-Eigenschaften haben aussagekräftige
  Standardwerte (nicht "Test" oder "Sample")

### Abhängigkeiten zwischen Entities
- Wenn ein Entity andere Entities als Eigenschaften
  hat, erstelle diese über deren Builder
- Beispiel: `private Author _author =
  new AuthorBuilder().Build();`

### Struktur
```csharp
public class EntityBuilder
{
 private Type _field = defaultValue;

 public EntityBuilder WithField(Type value)
 {
 _field = value;
 return this;
 }

 public Entity Build()
 {
 return new Entity
 {
 Field = _field
 };
 }
}

Test Konventionen
Beim Schreiben von Tests mit Testdata-Buildern:
Arrange-Abschnitt
- Verwende Builder für alle Testdaten
- Setze nur die für den Test relevanten Eigenschaften
 explizit
- Nutze Method Chaining für bessere Lesbarkeit

Naming
- Testmethodennamen:
 MethodName_Scenario_ExpectedResult
- Beispiel: GetPublishedPosts_WithMixedPosts_
 ReturnsOnlyPublishedPosts

Struktur
[Fact]
public void MethodName_Scenario_ExpectedResult()
{
 // Arrange
 var entity = new EntityBuilder()
 .WithRelevantProperty(value)
 .Build();

 // Act
 var result = service.Method(entity);

 // Assert
 Assert.Equal(expected, result);
}
```

fektes Beispiel dafür, wie traditionelle Software-Engineering-Prinzipien und moderne KI-Unterstützung synergetisch zusammenwirken können. ■

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Entwurfsmuster*. 5. Auflage. Addison-Wesley, 1996, ISBN 3-8273-1862-9, Seite 119

[2] Martin Fowler – *Fluent Interface*, <https://martinfowler.com/bliki/FluentInterface.html>



### Alexander Rampp

ist Entwickler und Head of Software Engineering bei XITASO. Er unterstützt Teams dabei, sich in der Softwareentwicklung weiter zu professionalisieren. Seine Schwerpunkte sind Softwarequalität sowie Persönlichkeitsentwicklung für Softwareentwickler und -entwicklerinnen.

## CLEAN CODE UND ARCHITEKTUR, TEIL 3

# Ressourcen und Dialoge

Die Architektur-Bereiche Ressourcen und Dialoge sollten für bessere Testbarkeit einer Anwendung klar von deren Kern getrennt sein.

Wie bereits mehrfach im Rahmen der Artikelserie ([1] [2]) erwähnt, sind Abhängigkeiten eine große Herausforderung in Softwaresystemen. Sie haben einen großen Einfluss darauf, wie leicht ein System verstanden werden kann. Ferner beeinflusst die Abhängigkeitsstruktur einer Anwendung die Testbarkeit in erheblichem Maße. Je besser die Abhängigkeiten für das automatisierte Testen geeignet sind, desto leichter fällt das Schreiben von Tests. Da Entwickler sich meiner Beobachtung nach immer noch schwertun, konsequent Tests zu schreiben, sind die Abhängigkeiten der Schlüssel zur Korrektheit.

Ein Softwaresystem ist in der Regel von Ressourcen in seiner Umgebung abhängig. Darunter fallen beispielsweise Datenbanken. Aber auch Zugriffe auf das Dateisystem, externe APIs oder spezielle Hardware zählen dazu. Sind Zugriffe auf Ressourcen im System nicht klar vom Rest des Systems getrennt, hat dies schwerwiegende Folgen für die Testbarkeit. Sind beispielsweise Datenbankzugriffe mit Domänenlogik eng verwoben, bleibt beim Testen praktisch nur der Integrationstest gegen eine reale Testdatenbank. Diese zu verwenden ist zwar dank Docker und dem TestContainers-Projekt inzwischen deutlich einfacher geworden, doch sind solche Tests im Vergleich zu echten Unit-Tests deutlich aufwendiger zu erstellen und zu pflegen. Ferner verlangt die Dauer der Testausführung einiges an Geduld, sodass diese Tests von Entwicklern nicht konsequent genug ausgeführt werden.

Ähnliche Herausforderungen wie bei den Ressourcenzugriffen gibt es auch bei den Dialogen. Das ist unser Überbegriff für jegliche Form von Benutzerinteraktion. Selbst der Begriff Benutzerinteraktion ist hier weit gefasst. So kann man

auch den Zugriff auf das System von außen über eine HTTP/REST-Schnittstelle als Interaktion betrachten, die über einen Dialog abgewickelt wird. Ein menschlicher Verwender des Systems interagiert in der Regel über ein grafisches User-Interface (UI) mit dem System. Solche GUIs können mit Avalonia, WPF, WinForms, MAUI, React, Htmx, Angular und vielen anderen UI-Technologien realisiert werden. Interagiert ein anderes System mit unserem System, erfolgt der Zugriff natürlich nicht über ein GUI, sondern in der Regel über HTTP. In diesem Fall ist die technische Ausgestaltung ein Controller oder ein Minimal API.

Wie bei den Ressourcenzugriffen zeigt sich, dass die Abhängigkeiten auch bei Dialogen wohlüberlegt gestaltet werden müssen. Liegt Domänenlogik innerhalb eines Dialogs, hat man es beim Testen mit Abhängigkeiten zu technischen Details der verwendeten Technologie zu tun. Spätestens wenn sich im Test eine MessageBox öffnet, merkt man, dass etwas schiefgelaufen ist.

Dass Domänenlogik nichts im UI verloren hat, hat sich inzwischen herumgesprochen. Die bekannten Muster MVP, MVC und MVVM sind als Lösung entstanden. Doch wie integriert man diese Muster in eine Gesamtarchitektur für ein Softwaresystem? Darum soll es in diesem Artikel gehen. Doch bevor wir uns den Dialogen zuwenden, klären wir einige Details zu Ressourcen.

## Ressourcen

In der Regel ist ein Softwaresystem davon abhängig, dass es auf Ressourcen in seiner Umgebung zugreifen kann. Das prominenteste Beispiel sind Datenbankzugriffe. Es spielt hier

### ● Listing 1: Domänenlogik vermischt mit Ressourcenzugriff

```
public class BooksDomain(IBooksRepository
 booksRepository)
{
 public void AddNewBook(Book book) {
 if(string.IsNullOrEmpty(book.Name)) {
 throw new ArgumentException(
 "Book name cannot be empty");
 }
 if (string.IsNullOrEmpty(book.ISBN)) {
 throw new ArgumentException(
 "Book ISBN cannot be empty");
 }
 if (string.IsNullOrEmpty(book.Author)) {
 throw new ArgumentException(
 "Book author cannot be empty");
 }
 booksRepository.SaveNewBook(book);
 }
}
```



## ● Listing 2: Test der Domänenlogik durch Verwendung einer Attrappe für das Repository

```

TestFixture]
public class BooksDomainTests
{
 private Mock<IBooksRepository>
 _mockRepository = null!;
 private BooksDomain _sut = null!;

 [SetUp]
 public void Setup() {
 _mockRepository = new Mock<IBooksRepository>();
 _sut = new BooksDomain(_mockRepository.Object);
 }

 [Test]
 public void
 AddNewBook_WithValidBook_CallsRepository() {
 var book = new Book {Name = "Test Book",
 ISBN = "1234567890", Author = "Test Author"};
 _mockRepository.Setup(x =>
 x.SaveNewBook(It.IsAny<Book>()))
 .Verifiable();

 _sut.AddNewBook(book);
 _mockRepository.Verify(x =>
 x.SaveNewBook(It.IsAny<Book>()), Times.Once);
 }

 [Test]
 public void AddNewBook_WithNullName
 _ThrowsArgumentException() {
 var book = new Book {Name = null!,
 ISBN = "1234567890", Author = "Test Author"};
 var ex = Assert.Throws<ArgumentException>(()
 => _sut.AddNewBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book name cannot be empty"));
 _mockRepository.Verify(x =>
 x.SaveNewBook(It.IsAny<Book>()), Times.Never);
 }

 [Test]
 public void AddNewBook_WithEmptyName
 _ThrowsArgumentException() {
 var book = new Book {Name = "",
 ISBN = "1234567890", Author = "Test Author"};
 var ex = Assert.Throws<ArgumentException>(()
 => _sut.AddNewBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book name cannot be empty"));
 _mockRepository.Verify(x =>
 x.SaveNewBook(It.IsAny<Book>()), Times.Never);
 }

 [Test]
 public void AddNewBook_WithNullISBN
 _ThrowsArgumentException() {
 var book = new Book {Name = "Test Book",
 ISBN = null!, Author = "Test Author"};
 var ex = Assert.Throws<ArgumentException>(()
 => _sut.AddNewBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book ISBN cannot be empty"));
 _mockRepository.Verify(x =>
 x.SaveNewBook(It.IsAny<Book>()), Times.Never);
 }

 [Test]
 public void AddNewBook_WithNullAuthor
 _ThrowsArgumentException() {
 var book = new Book {Name = "Test Book",
 ISBN = "1234567890", Author = null!};
 var ex = Assert.Throws<ArgumentException>(()
 => _sut.AddNewBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book author cannot be empty"));
 _mockRepository.Verify(x =>
 x.SaveNewBook(It.IsAny<Book>()), Times.Never);
 }
}

```

keine Rolle, ob es sich um eine relationale oder eine NoSQL-Datenbank handelt. Selbst die Ablage von Daten im Dateisystem stellt einen Ressourcenzugriff dar.

Ich gestehe, dass ich vor ewigen Zeiten SQL-Statements innerhalb des UI untergebracht habe. Das war noch zu Zeiten von VB 6. Das Ergebnis? Ich verstand den Code nach sehr kurzer Zeit selbst nicht mehr. Über automatisierte Tests haben wir damals noch nicht nachgedacht. Aber es ist klar, dass diese niemals möglich gewesen wären.

Wie bereits beschrieben hat es sich herumgesprochen, dass weder Datenbankzugriffe noch Domänenlogik ihren Platz im UI finden sollten. Doch was sind Muster, um diesen Zugriffen

einen guten Platz in der Architektur zu geben? Bevor wir diese Frage beantworten, überlegen wir: Welche Prinzipien und Werte können uns dabei leiten?

An erster Stelle ist hier das Single Responsibility Principle (SRP) zu nennen. Es besagt, dass eine Methode oder Klasse jeweils genau eine Verantwortlichkeit übernehmen soll. Datenbankzugriff in der Domänenlogik? Verstößt offensichtlich gegen das SRP. Denn dann wäre die Methode für zwei Dinge verantwortlich.

Als zweites Prinzip möchte ich das Integration Operation Segregation Principle (IOSP) nennen. Es besagt, dass eine Methode entweder Details enthält, dann nennen wir sie ►

Operation. Oder sie ruft andere eigene Methoden auf, dann nennen wir sie Integration. Der Vorteil: Der Code ist besser lesbar und lässt sich leichter automatisiert testen. Die bessere Testbarkeit liegt daran, dass die Operationen keine Abhängigkeiten zu anderen Bestandteilen unseres Systems haben. Sie lassen sich somit leicht mit isolierten Unit-Tests testen. Der Test der Integrationsmethoden erfolgt logischerweise mit Integrationstests. Da sowohl Integrations- als auch Unit-Tests zu einer gesunden Teststrategie gehören, unterstützt das IOSP diese Struktur optimal.

Von den vier Werten der Clean Code Developer Initiative sind bei der Frage, wie wir Dialoge und Ressourcen in unserer Architektur integrieren, vor allem die Werte Wandelbarkeit und Korrektheit zu nennen. Software muss wandelbar sein, damit auch zukünftige Anforderungen leicht umgesetzt werden können. Die Wandelbarkeit stellt den Investitionsschutz für unsere Kunden dar. Und natürlich ist für unsere Kunden auch die Korrektheit sehr wichtig, die wir letztlich nur durch automatisierte Tests erreichen.

### Muster für Ressourcenzugriffe

Für den Zugriff auf Datenbanken haben sich sogenannte Object-Relational-Mapper (OR-Mapper, ORM) etabliert. Das Entity Framework von Microsoft ist ein Beispiel für einen solchen OR-Mapper. Weil der Zugriff auf Datenbanken mit solchen OR-Mappern so einfach geworden ist, jedenfalls im Vergleich zum direkten SQL-Zugriff, ist die Versuchung groß, an mehr oder weniger beliebiger Stelle auf die Datenbank zuzugreifen. Doch dies hat mindestens einen Nachteil: Beim automatisierten Testen muss entweder gegen eine Testdatenbank getestet werden. Oder es kommen Attrappen zum Einsatz, mit denen die Datenbankzugriffe im Test eliminiert werden. Konkret sieht das dann so aus, dass innerhalb der Domänenlogik an einer Stelle, an der Daten benötigt werden, auf die Datenbank zugegriffen wird. Da nicht jeder Test ein Integrationstest gegen eine Testdatenbank sein soll, behelfen sich viele Entwickler mit Attrappen.

Um im Test einigermaßen sinnvoll mit Attrappen arbeiten zu können, wird die Verwendung des OR-Mappers, wie etwa des Entity Frameworks, in ein Repository gekapselt. Stattet man dies noch mit einem Interface aus, steht dem Mocking

nichts mehr entgegen. Der Codeausschnitt in [Listing 1](#) zeigt ein Beispiel.

Die Methode übernimmt die Validierung der hineingereichten Instanz eines neuen Buches. Testen kann man das Ganze dann wie in [Listing 2](#) gezeigt.

Im Test wird eine Attrappe für das Repository verwendet. Dadurch ist es möglich, automatisiert herauszufinden, ob die Repository-Methode *SaveNewBook* aufgerufen wird.

Ich habe in der Domänenlogik Exceptions verwendet, um ein typisches Beispiel zu zeigen, wie Entwickler solche Probleme angehen. Dass dies nicht die beste Lösung ist, steht auf einem anderen Blatt. Es wäre meiner Einschätzung nach besser, einen Rückgabewert einzuführen, der ausdrückt, ob das Hinzufügen des neuen Buches funktioniert hat. Der Aufrufer muss ohnehin reagieren. Ein Rückgabewert drückt offensichtlicher aus, dass es mehrere Möglichkeiten gibt, wie sich die Methode verhält. Die Exceptions müssen entweder über Kommentare beschrieben werden, oder man muss im Code nachschauen. Die Intention der Methode lässt sich so nur schwer erschließen.

Halten wir zunächst fest, dass die Lösung mittels Repository deutlich besser ist, als Low-Level-Datenbankzugriffe in die Domänenlogik einzustreuen. Ob man nun direkt SQL-Code schreibt oder einen OR-Mapper verwendet: Die Konfiguration der Attrappen wäre deutlich aufwendiger als beim Einsatz eines Repositories. Verkürzt gesagt ist es für den direkten SQL-Zugriff notwendig, eine Connection zu erstellen. Danach werden Parameter erstellt, die in der SQL-Query enthaltene Platzhalter bei der Ausführung ersetzen. Dann wird die Query ausgeführt. Wollte man all diese Details durch eine Konfiguration der Attrappen herstellen, würden die Tests sehr umfangreich ausfallen. Vor allem käme es potenziell zu einer gewaltigen Dopplung von Testcode, da die Attrappen in jedem Test konfiguriert werden müssten. Als Alternative werden diese SQL-Details in einem Repository gekapselt. Im Ergebnis kann dann mit deutlich weniger Aufwand eine Attrappe des Repositories konfiguriert werden.

### Repository

Ein Repository stellt eine Abstraktion dar. Statt die Daten mithilfe von SQL-Statements oder OR-Mapper-Aufrufen zu be-

#### ● Listing 3: Integration von Domänenlogik und Repository

```
public void AddNewBook(Book book) {
 ValidateBook(book);
 booksRepository.SaveNewBook(book);
}

public static void ValidateBook(Book book) {
 if(string.IsNullOrEmpty(book.Name)) {
 throw new ArgumentException(
 "Book name cannot be empty");
 }

 if (string.IsNullOrEmpty(book.ISBN)) {
 throw new ArgumentException(
 "Book ISBN cannot be empty");
 }

 if (string.IsNullOrEmpty(book.Author)) {
 throw new ArgumentException(
 "Book author cannot be empty");
 }
}
```

#### Listing 4: Testen der Domänenlogik in Isolation ohne Attrappe

```
[Test]
public void ValidateBook_WithValidBook_DoesNotThrow()
{
 var book = new Book { Name = "Test Book",
 ISBN = "1234567890", Author = "Test Author" };
 Assert.DoesNotThrow(() =>
 BooksDomain.ValidateBook(book));
}

[Test]
public void ValidateBook_WithNullName
 _ThrowsArgumentException() {
 var book = new Book { Name = null!,
 ISBN = "1234567890", Author = "Test Author" };
 var ex = Assert.Throws<ArgumentException>(()
 => BooksDomain.ValidateBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book name cannot be empty"));
}

[Test]
public void ValidateBook_WithEmptyName_
 ThrowsArgumentException() {
 var book = new Book { Name = "",
 ISBN = "1234567890", Author = "Test Author" };
 var ex = Assert.Throws<ArgumentException>(()
 => BooksDomain.ValidateBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book name cannot be empty"));
}

[Test]
public void ValidateBook_WithValidBook_DoesNotThrow()
{
 var book = new Book { Name = "Test Book",
 ISBN = "1234567890", Author = "Test Author" };
 Assert.DoesNotThrow(() =>
 BooksDomain.ValidateBook(book));
}

[Test]
public void ValidateBook_WithNullISBN
 _ThrowsArgumentException() {
 var book = new Book { Name = "Test Book",
 ISBN = null!, Author = "Test Author" };
 var ex = Assert.Throws<ArgumentException>(()
 => BooksDomain.ValidateBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book ISBN cannot be empty"));
}

[Test]
public void ValidateBook_WithNullAuthor
 _ThrowsArgumentException() {
 var book = new Book { Name = "Test Book",
 ISBN = "1234567890", Author = null! };
 var ex = Assert.Throws<ArgumentException>(()
 => BooksDomain.ValidateBook(book));
 Assert.That(ex.Message, Is.EqualTo(
 "Book author cannot be empty"));
}
```

sorgen, stellt ein Repository passgenaue Methoden bereit, die im Kontext der Domäne benötigt werden. Ein Repository ist weniger generisch als ein OR-Mapper. So lautet der Methodenname im Repository beispielsweise *FindCustomerByEmail* statt *context.Customers.FirstOrDefault(x => x.Email == email)*.

Das Repository-Muster ist etabliert und bietet viele Vorteile. Wichtig ist, dass man der Versuchung widersteht, ein generisches Repository zu erstellen. Damit würde man nur einen generischen Layer über einen ohnehin schon generischen OR-Mapper legen. Wenn also mit Kundendaten hantiert wird, sollte es dafür ein *CustomerRepository* geben. Darin sind passgenaue Methoden vorhanden, die sich um Kundendaten drehen. Gegebenenfalls kann Codedopplung vermieden werden, indem alle spezifischen Repositories auf einer gemeinsamen Basisklasse aufbauen, in der mehrfach benötigte Funktionalität abgelegt wird.

Um noch einen Schritt weiterzugehen, sollen die Repository-Aufrufe nicht in der Domänenlogik stehen. Damit wird das SRP verletzt, denn die Methode ist dann für Domänenlogik und Datenbeschaffung gleichermaßen zuständig. Die Lösung liegt in der Anwendung des IOSP: Eine integrierende Methode ruft sowohl Methoden auf, die der Domänenlogik zugeordnet werden, als auch Methoden aus dem oder den Repositories. Das ermöglicht uns, beim Testen die Domänenlogik isoliert zu testen. Und auch die Repositories können isoliert

getestet werden, natürlich gegen eine Testdatenbank. [Listing 3](#) zeigt ein Beispiel.

Wir finden hier eine Integrationsmethode, die dafür verantwortlich ist, eine Interaktion zu realisieren. Die Methode befindet sich auf hohem Abstraktionsniveau und ist damit leicht verständlich. Auf der anderen Seite haben wir die Operationen, die sich um die Details kümmern. Da sie sich jeweils ausschließlich um Details kümmern, sind auch sie leicht verständlich. Voraussetzung ist weiterhin, dass das SRP eingehalten wird. Für das Testen einer solchen Struktur sind keine Attrappen erforderlich. Im Integrationstest kommt eine Testdatenbank zum Einsatz. Weil der Integrationscode keine Details enthält, genügen hier wenige Integrationstests. Alle Details werden dann durch Unit-Tests der Operationen abgedeckt ([Listing 4](#)).

Teilnehmer unserer Trainings diskutieren an dieser Stelle gerne, ob es sinnvoll ist, den extrahierten Code in einer öffentlichen Methode abzulegen. Da die Testbarkeit so wichtig ist, steht für mich im Vordergrund, dass es so einfach wie möglich sein soll, Tests zu schreiben. Die Sichtbarkeit weiche ich dazu gerne auf und nehme in Kauf, dass jemand die Methode verwendet, obschon sie eigentlich privat sein sollte. Alternativ kann man mit *internal* in Kombination mit *InternalsVisibleTo* arbeiten oder die Methode *ValidateBook* in eine eigene Klasse auslagern, in der sie dann ohnehin *public* ist. ►

## Provider

Wenn wir vom Beispiel der Datenbank als Ressource abstrahieren wollen, brauchen wir auch Muster für den Zugriff auf das Dateisystem, Hardware, APIs und anderes. In diesem Fall würde ich nicht mehr von einem Repository sprechen. Ein Repository ist eng gekoppelt mit Datenbankzugriffen. Der abstraktere Begriff ist der Provider.

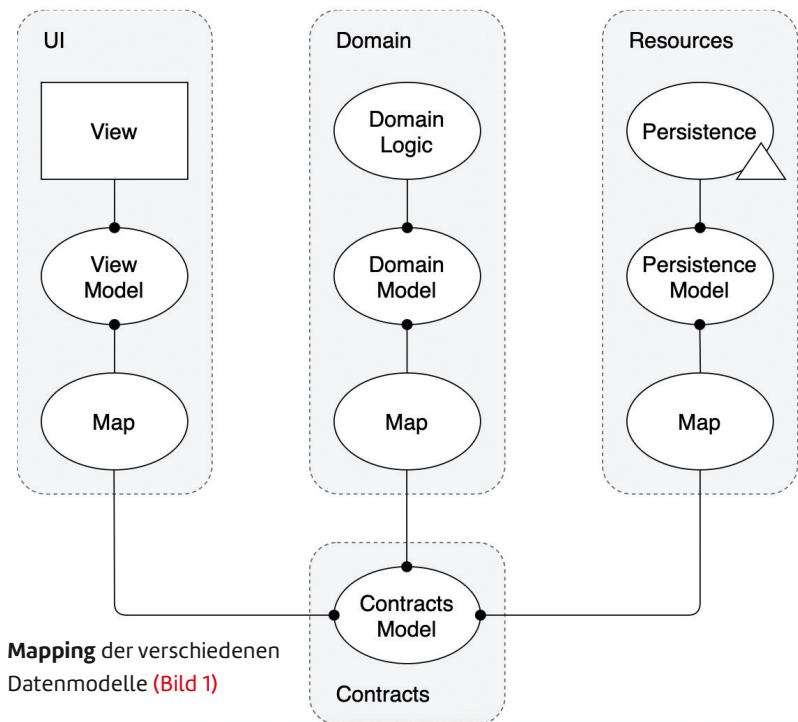
Ein Provider stellt dem Softwaresystem eine Ressource zur Verfügung. Er kapselt die Details. Ein Repository ist ebenfalls ein Provider. Es kapselt die Details des Datenbankzugriffs. Ob dabei hinter den Kulissen das Entity Framework oder Marten verwendet wird, spielt für den Verwender des Repositories keine Rolle.

Das gilt auch für andere Provider. Muss in einem Softwaresystem beispielsweise auf Stripe zur Abwicklung von Zahlungen zugegriffen werden, kommt dazu ein Provider zum Einsatz. Er könnte beispielsweise *StripeProvider* heißen oder abstrakter auch *PaymentProvider*. In dieser Klasse sind alle Details zur Anbindung von Stripe enthalten. Welches NuGet-Paket für den Zugriff auf Stripe hinter den Kulissen zum Einsatz kommt, ist wieder unerheblich. Der *PaymentProvider* stellt nach außen eine Schnittstelle bereit, die möglichst unabhängig von den internen Details ist.

## Mapping

Ganz wichtig: Nach Möglichkeit sollten keine Datentypen in der öffentlichen Schnittstelle des Providers nach außen eingesetzt werden, die aus einem der verwendeten NuGet-Pakete stammen. Damit wäre ein Austausch des Pakets nicht ohne Weiteres möglich (Ask me how I know...). Hier ist allerdings Vorsicht geboten. Will man vollständig darauf verzichten, Datentypen aus spezifischen Frameworks an der öffentlichen Schnittstelle hinauszureichen, müssen die spezifischen Datentypen gemappt werden auf solche, die zum Softwaresystem gehören. Hier stellt sich die Frage, ob diese Abstraktion wirklich jemals benötigt wird. Existiert beispielsweise ohnehin nur ein NuGet-Paket, um auf Stripe zuzugreifen, würde ich nicht den Aufwand treiben, dieses austauschbar zu machen. Doch diese Abwägung ist nicht immer einfach.

Getrieben wird eine solche Abwägung vom Wert der Produktionseffizienz. Durch jedes Mapping eines spezifischen Datentyps auf eigene Datentypen wird die Anwendung umfangreicher und komplexer. Wenn die damit einhergehende Freiheit und Flexibilität praktisch nie genutzt wird, hat man nichts gewonnen. Das Problem ist allerdings, dass wir im Vorfeld nicht wissen können, ob sich die Flexibilität einmal auszahlen wird. Im Zweifel rate ich dazu, auf solche Mappings zu Beginn zu



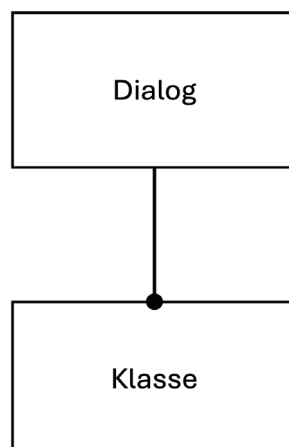
verzichten. Zeigt sich später ein echter Bedarf, können sie zu diesem Zeitpunkt nachgerüstet werden. Das erfordert zwar einen Refactoring-Aufwand, doch ist dieser überschaubar.

Die gleiche Frage stellt sich bei der Verwendung eines OR-Mappers. Hier werden Datentypen auf eine Tabellenstruktur gemappt. Im Ergebnis arbeitet die Anwendung nicht mehr mit SQL-Tabellen, sondern mit Datenklassen. Die konkrete Zuordnung zu den Tabellen wird einmal definiert, und der OR-Mapper erzeugt die benötigten SQL-Queries. Damit bilden die Datenklassen die Tabellenstruktur der Datenbank ab.

Muss nun die Tabellenstruktur der Datenbank geändert werden, stellt sich die Frage, welchen Einfluss dies auf unsere Datenklassen hat. Wird eine Spalte in der Datenbank lediglich umbenannt, kann darauf im OR-Mapper reagiert werden, und unsere Datenklasse kann unverändert bleiben.

Doch schon beim Ändern des Typs einer Spalte können Probleme auftreten. Noch schwieriger wird es, wenn Änderungen an der Tabellenstruktur erforderlich sind. Teilt man beispielsweise eine Tabelle *Kundendaten* auf mehrere Tabellen auf, die über Schlüsselbeziehungen verbunden werden, muss in der Regel auch die Struktur der Datenklassen angepasst werden. Dies kann zu umfangreichen Änderungen bei den Verwendern führen.

Als Lösung kann man auch hier durch ein Mapping versuchen, die Datenklassen zu entkoppeln. Auf der einen Seite stehen die Datenklassen, die mittels OR-Mapper auf eine Tabellenstruktur abgebildet werden. Auf der anderen Seite stehen abstraktere Klassen, in die jeweils gemappt wird. Auch vor dieser Lösung ist allerdings zu warnen. Führt man von



Direkte Abhängigkeit eines Dialogs zu einer Klasse (Bild 2)

Anfang an ein solches Mapping ein, werden praktisch gleiche Klassen doppelt erzeugt und zwischen diesen hin und her gemappt. Sind umfangreiche Änderungen an der Tabellenstruktur zu erwarten, mag dies ein Vorteil sein. Häufig stellt sich dann allerdings heraus, dass ein Mapping zwischen den modifizierten Datenbankklassen und den abstrakten Klassen nicht ohne Weiteres möglich ist. Das führt dann doch zu Änderungen an Klassen, die an vielen Stellen im System verwendet werden.

Ich möchte hier das Mapping von Datenbankklassen nicht verteufeln. Es gibt Fälle, in denen das sinnvoll ist. Doch in der Regel ist es besser, solche Mappings erst bei Bedarf einzuführen statt prophylaktisch.

**Bild 1** zeigt, an welchen Stellen in einem Softwaresystem Mappings denkbar sind.

Das allgemeine Modell ist hier *Contracts Model* genannt. Das *Persistence Model* wird vom OR-Mapper auf die Datenbankstruktur abgebildet. Damit es privat bleibt, wird im Ressourcenzugriff auf das *Contracts Model* gemappt. An der öffentlichen Schnittstelle wird nur das *Contracts Model* verwendet.

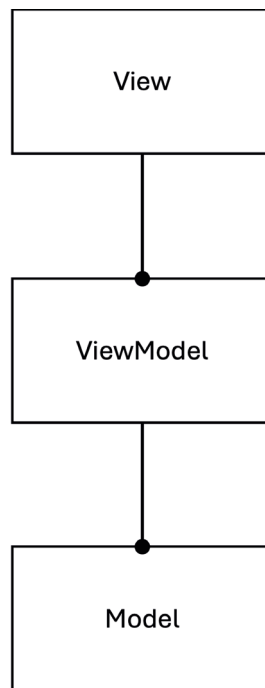
Daten, die an das UI gereicht werden sollen, werden innerhalb des UI in ein ViewModel gemappt. Damit kann das View-Model auf die Bedürfnisse der View zugeschnitten werden.

Und zu guter Letzt kann auch noch ein Domain Model zum Einsatz kommen, wenn damit bestimmte Anforderungen der Domäne leichter umgesetzt werden können. Als Faustregel gilt: Mappings sparsam einsetzen.

## Mappings bei technischen Abhängigkeiten

Etwas eindeutiger ist die Lage, wenn Klassen technische Abhängigkeiten haben, die nicht im Rest des Softwaresystems auftauchen sollen. Muss bei der Verwendung eines OR-Mappers beispielsweise von einer Basisklasse abgeleitet werden, stellt sich die Frage, ob dies an anderen Stellen im System zu Schwierigkeiten führen kann. Die Verwendung von Basisklassen ist bei OR-Mappern heute zwar eher nicht mehr anzutreffen. Doch hinter den Kulissen werden häufig Mechanismen angewendet, um beispielsweise herauszufinden, ob an einem Objekt eine Änderung vorgenommen wurde. Ein OR-Mapper möchte dies herausfinden können, um bei Bedarf ein Update in die Datenbank schreiben zu können (Change Tracking). Werden solche Objekte nun in der Domänenlogik verwendet, kann es passieren, dass zur Laufzeit unerwartet Datenbankzugriffe erfolgen. In einem solchen Fall ist es möglicherweise ratsam, die Datenbankklassen zu mappen.

Auch Datentypen aus einem verwendeten NuGet-Paket sollen gegebenenfalls nicht in ei-



Das Muster Model View  
ViewModel (MVVM) (**Bild 3**)

genen Klassen des Softwaresystems verwendet werden. Dies würde zu einer starken Abhängigkeit zum verwendeten Paket führen. Ziel sollte sein, dass die Details hinter der Providergrenze gekapselt sind.

Verwendet man beispielsweise ein Paket, mit dem PDF-Rechnungen erstellt werden können, und dieses Paket enthält einen eigenen Datentyp *VatRate*, mit dem die verschiedenen Umsatzsteuerdetails abgebildet werden, dann sollte man gut überlegen, ob dieser Datentyp im Rest der Anwendung verwendet werden sollte. Ein Mapping an der Providergrenze verhindert diese Abhängigkeit und hält das spezifische Paket austauschbar.

## Dialoge

Bei den Dialogen liegt die Abhängigkeitsrichtung andersherum als bei den Ressourcen. Während das System von den Ressourcen abhängig ist, sind die Dialoge vom System abhängig. Weitere Details dazu folgen im nächsten Teil der Serie.

Die Abhängigkeitsrichtung ergibt sich auch logisch gesehen aus dem Kontrollfluss während einer Interaktion. Interagiert ein Anwender mit dem Softwaresystem beispielsweise dadurch, dass er im GUI auf einen Knopf drückt, fällt diese Information in einem Dialog an. Der Dialog muss daraufhin die erforderliche Logik anstoßen, die bei dieser Interaktion ablaufen soll. Es ist somit naheliegend, dass Dialoge vom Rest des Systems abhängig sind.

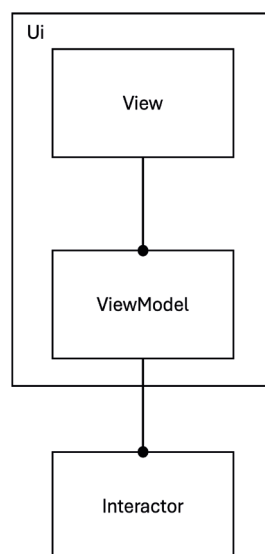
Ähnlich sieht die Situation aus, wenn die Interaktion durch ein Fremdsystem erfolgt. In dem Fall wäre der Dialog ein Controller, der einen HTTP-Request entgegennimmt. Auch hier ist wieder der Controller verantwortlich dafür, die erforderliche Logik anzustoßen.

Im Sinne der Testbarkeit ist es wichtig, wieder darüber nachzudenken, wie sich eine Abhängigkeit vom Dialog zum Rest des Systems auswirken würde.

Verfügt ein Dialog über eine direkte Abhängigkeit zu einer Klasse, wie in **Bild 2** gezeigt, stellt sich beim automatisierten Test des Dialogs die Frage, was mit dieser Abhängigkeit passiert.

Wird im Test eine Instanz des Dialogs erzeugt, muss die Abhängigkeit berücksichtigt werden. Sinnvollerweise werden die Abhängigkeiten des Dialogs über Dependency Injection bereitgestellt. Im Test kann dann entschieden werden, ob dabei die reale Abhängigkeit verwendet werden kann oder ob eine Attrappe zum Einsatz kommt. Voraussetzung dazu ist allerdings, dass technisch gesehen zur Laufzeit eine Attrappe erstellt werden kann. Dazu ist in der Regel ein Interface die beste Lösung.

Praktisch gesehen tritt dieser Fall bei Dialogen allerdings eher selten auf, da wir Dialoge ►



Anbindung des Interactors  
an das MVVM (**Bild 4**)



typischerweise nicht automatisiert testen. Es mag Einzelfälle geben, und dann ist der Einsatz von Attrappen vertretbar. Beim Testen eines Controllers beziehungsweise eines Minimal API wird man ohnehin einen Integrationstest anstreben, weil der Controller lediglich eine technische Schicht über dem Anwendungskern darstellt. Dabei wollen wir nicht testen, ob ASP.NET Core funktioniert, sondern ob wir die Technologie im Controller korrekt anwenden.

## Muster

Für grafische Benutzerschnittstellen hat sich das MVVM-Muster etabliert. Die Abkürzung steht für Model View View-Model und beschreibt die drei Bestandteile, in die ein Dialog zerlegt wird (**Bild 3**).

Die View stellt die technische Realisierung der Benutzerschnittstelle dar. Das kann im Fall von Avalonia oder WPF eine XAML-Datei sein. Bei einer Webanwendung kann es beispielsweise eine Razor Page sein in Kombination mit HTMX. Eine View ist sehr technisch ausgerichtet und sollte möglichst austauschbar sein, da sich über die Jahre die Anforderung ergeben könnte, eine andere Technologie anzuwenden.

Zur Entkopplung kommt aus diesem Grund ein ViewModel zum Einsatz. Alle Daten, die in einer View dargestellt werden sollen, werden im ViewModel abgebildet. Dabei diktiert die View das Datenformat beziehungsweise die konkrete Ausgestaltung. Das Softwaresystem hat auf diese Weise keinen Kontakt zur View, sondern lediglich zu den ViewModels. Diese verfügen über keine technischen Abhängigkeiten zum verwendeten UI-Framework.

Der letzte Baustein des MVVM-Musters ist das Model. Dieser Teil ist wahrscheinlich der am häufigsten missverstandene Teil von MVVM. Vereinfacht kann man sagen, das Model ist im Kontext von MVVM der gesamte Rest unserer Anwendung. Bei einer Interaktion des Anwenders mit einer View landen die vom Anwender eingegebenen Daten mittels Data Binding im ViewModel. Das ViewModel darf jedoch nicht dafür verantwortlich sein, die benötigte Logik selbst durchzuführen. Stattdessen ruft das ViewModel Methoden des Models auf. Der Begriff Model ist hier der abstrakteste unter den drei in MVVM verwendeten.

## Interactor

Unser Vorschlag für eine konkrete Ausgestaltung des Models ist der Interactor. Ein Interactor ist eine Klasse, die dafür verantwortlich ist, die Logik einer Interaktion bereitzustellen. Natürlich übernimmt der Interactor die Aufgabe nicht selbst, sondern delegiert sie an andere Methoden. Der Interactor ist also eine Integrationsmethode, die andere Methoden des Softwaresystems aufruft. Dies können Operationen sein oder auch weitere Integrationsmethoden, je nach Komplexität der Anforderung. Der Interactor verbindet dabei auch den Aufruf von Domänenlogik mit dem Aufruf von Providern für die Ressourcenzugriffe.

Die Anbindung des Interactors erfolgt typischerweise über das ViewModel. Das bedeutet: In das ViewModel wird per Dependency Injection ein Interactor injiziert. Wird nun in der View eine Interaktion ausgelöst, landen die vom Anwender

einggegebenen Daten per Data Binding im ViewModel. Ferner wird beispielsweise durch Anklicken eines Buttons eine Methode des ViewModel aufgerufen. Im ViewModel stehen dann alle eingegebenen Daten zur Verfügung, und es kann damit der Interactor aufgerufen werden. Gegebenenfalls liefert dieser Daten zurück. Diese werden vom ViewModel entgegengenommen, und die Properties werden entsprechend aktualisiert. Durch Data Binding landen die Daten schlussendlich wieder in der View und werden dort visualisiert.

Der Vorteil dieser Vorgehensweise: Die gesamte Logik einer Interaktion kann durch Aufruf der Interactor-Methoden in einem Integrationstest überprüft werden. Dieser Test ist völlig unabhängig von View und ViewModel. Auch die ViewModels können so getestet werden. Es werden die Abhängigkeiten injiziert, in der Regel im Interactor, und der Test kann starten. Bei komplexen ViewModel-Tests kann auch eine Attrappe für den Interactor zum Einsatz kommen.

Ein weiterer Vorteil: Die Interactorklassen können auch in einem Controller oder einem Minimal API zum Einsatz kommen. Schließlich enthalten sie nichts, was für ein GUI spezifisch wäre. So lässt sich eine Desktop-Anwendung leichter ins Web bringen.

Wichtig beim Einsatz des MVVM-Musters ist, die Zuständigkeiten klar zu trennen. Soll im Rahmen einer Interaktion beispielsweise eine MessageBox geöffnet werden, um den Benutzer zu fragen, ob er eine Aktion tatsächlich ausführen möchte, ist dafür die View verantwortlich (**Bild 4**). Jegliche direkte Interaktion mit dem Benutzer muss in der View erfolgen. Wenn die View festgestellt hat, dass der Benutzer mit „Ja“ geantwortet hat, kann die entsprechende Methode im ViewModel aufgerufen werden. So bleibt das ViewModel frei von technischen Aspekten der Benutzerschnittstelle und lässt sich bei Bedarf auch automatisiert testen, ohne dass eine MessageBox oder Ähnliches im Weg ist.

## Fazit

Um die Werte Wandelbarkeit und Korrektheit in einem Softwaresystem zu erreichen, ist es wichtig, Ressourcenzugriffe und Benutzerinteraktionen vom Rest des Systems zu trennen. Nur so bleiben Abhängigkeiten zu technischen Details und Paketen isoliert. Das fördert die Testbarkeit eines Softwaresystems. ■

[1] Stefan Lieser, *Mehr Struktur im Begriffschaos*, dotnetpro 6-7/2025, Seite 100 ff.

[2] Stefan Lieser, *Abhängigkeiten und Architekturmuster*, dotnetpro 8-9/2025, Seite 98 ff.



**Stefan Lieser**

sucht ständig nach Verbesserung und neuen Wegen, um die innere Qualität von Software zu optimieren. Gemeinsam mit Ralf Westphal hat er die Clean Code Developer Initiative (<https://clean-code-developer.de>) ins Leben gerufen. <https://ccd-akademie.de>

# DWX ACADEMY

Hands-on Workshops  
und Weiterbildung für  
Software-Entwickler  
und -Architekten



**PATRICK  
SCHNELL**

## START IN ANGULAR FÜR .NET-ENTWICKLER

Der Kurs vermittelt Grundlagen von Angular, Googles Open-Source-Framework für Frontend-Webentwicklung. Es bietet Tools zur Erstellung robuster, skalierbarer und wartbarer Webanwendungen. Nach Abschluss können Sie selbständig dynamische, interaktive Webanwendungen entwickeln.



**JAN  
POHLMAYER**

## KI IN DER PRAXIS: ANBINDUNG AN JAKARTAEE MIT LANGCHAIN4J

Der Workshop gibt einen Überblick über neuronale Netze und LLMs mit Fokus auf die Anbindung von AI-Schnittstellen via LangChain4j zur Inferenz. Die LLMs laufen lokal, das Training von Modellen wird nicht behandelt.



**ROLAND  
KÖNIG**

## BOOTCAMP: CROSS-PLATTFORM MIT AVALONIA UI

Avalonia UI ist eine OpenSource-Klassenbibliothek für UI-Applikationen mit .NET. Als Cross-Plattform-Alternative zu WPF unterstützt es heute Windows, macOS und Linux sowie mobile Plattformen (Android, iOS) und Browser (via WebAssembly).



**JENS  
SEEKAMP**

## JAVA FÜR EIN- UND UMSTEIGER

In der Schulung lernen Sie Klassen, Methoden und Vererbung in Java sowie die grundlegenden Sprach-Features und Java 8. Durch praktische Übungen entwickeln Sie Anwendungen mit Java-Klassen-Bibliotheken und erlernen die fundamentalen Konzepte.



Mehr Informationen unter [developer-world.de/dwx-academy](https://developer-world.de/dwx-academy)  
Preise auf Anfrage!

Deine Ansprechpartnerin: Susanne Herl • +49 731 88005-8835 • [susanne.herl@ebnermedia.de](mailto:susanne.herl@ebnermedia.de)



## SQL SERVER 2025 FÜR ENTWICKLER

# Viel Licht, aber auch Schatten

Die neue Entwicklerversion von Microsofts Datenbankserver bietet spannende Neuerungen, kommt dabei aber nicht ganz ohne Fragwürdiges aus.

Dieser Artikel stellt neue Features von SQL Server 2025 vor, die der Autor für interessant und erwähnenswert hält. Dabei geht eher um eine fokussierte Übersicht als um akribische Vollständigkeit. Die aktuelle Preview-Version kann als Installationspaket unter [1] geladen werden. Unter [2] finden Sie eine Anleitung für die Inbetriebnahme unter Docker (Linux-Version). Wichtig beim Umstieg beziehungsweise Ausprobieren der Preview ist, dass der Kompatibilitätsgrad für die meisten Features auf (mindestens) 170 stehen muss. Gegebenenfalls ist hier das folgende Statement hilfreich:

```
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 170;
```

## Neue Developer-Editionen

Bis dato gab es lediglich eine Developer Edition für Entwickler, die gleich über alle Features der „großen“ Enterprise Edi-

tion verfügte. Es konnte deshalb passieren, dass während der Entwicklung ein Feature genutzt wurde, das ausschließlich für die größte (und teuerste) Edition zur Verfügung stand. Man musste dann entweder viel Geld für den Erwerb einer Enterprise Edition ausgeben oder dieses Feature wieder ausbauen – eine unglückliche Situation.

In Zukunft gibt es nun zwei Editionen: die Standard Developer Edition und eine Enterprise Developer Edition. Beide umfassen jeweils die Features der gleichnamigen einsatzfähigen Edition.

## SSMS und Copilot

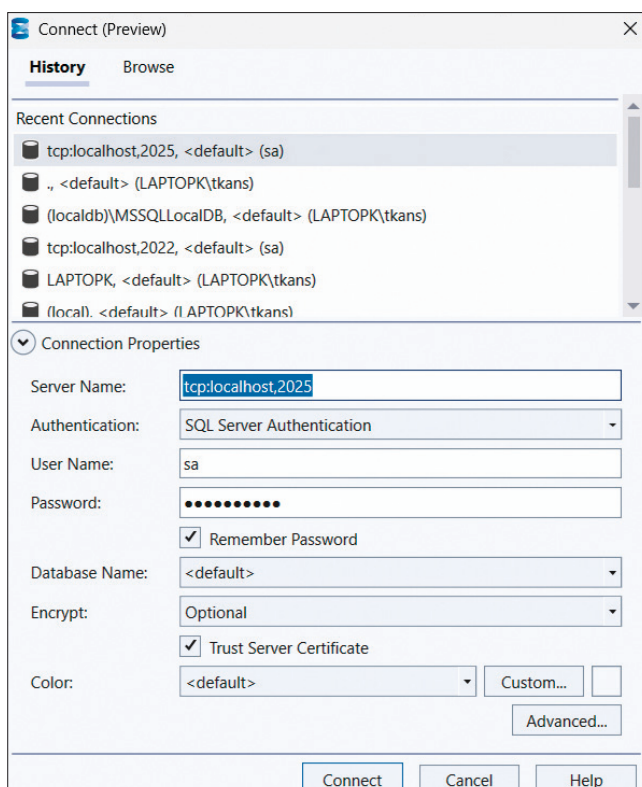
SQL Server 2025 kommt mit dem neuen SQL Server Management Studio (SSMS) in Version 21 [3], das nun auf Visual Studio 2022 (64 Bit) basiert. Dabei ist dieser Umstand an sich nicht neu, da auch ältere Versionen seit einiger Zeit Visual Studio als Grundlage nutzen. Etwas befremdend ist es, dass das SSMS nun im Visual Studio Installer auftaucht und aktualisiert werden kann.

Neben vielen kleineren Verbesserungen sind nun der Splashscreen und das Icon in hellem Blau gehalten, es gibt einen neuen Verbindungsdialog (Bild 1), und konsequenterweise wird Git direkt unterstützt.

Eine der für viele wohl interessantesten Neuerungen ist die eingebaute Unterstützung durch einen Copiloten. Dieser steht nach Einrichtung und Klick auf das gleichnamige Symbol zur Verfügung, um Fragen zur Datenbank zu beantworten und mit Tipps und Tricks zu helfen. Aber Achtung! Die KI wird nicht auf dem PC ausgeführt, sondern es wird eine Verbindung zu Azure OpenAI und damit eine Internetverbindung benötigt. Selbstredend, dass über diese auch Metadaten der Datenbank in die Cloud übertragen werden, da sonst keine Antwort möglich wäre.

Fragestellungen à la „Wie viele Personen gibt es?“ und „Zeige das SQL-Statement, um die Umsätze nach Jahren abzufragen“ werden in der AdventureWorks-Datenbank [4] solide beantwortet. Komplexere oder unklare Aufforderungen wie „Optimiere mir meine Datenbank“ scheitern jedoch – die Antworten sind teils verworren oder gar falsch (Bild 2).

Wie oft der Copilot genutzt und wie oft komplexere Fragen gestellt werden können, richtet sich auch nach der Bereitschaft, dafür zu bezahlen (Stichwort „token rate limit“). Wie der Copilot mit eigenen Datenbanken zurechtkommt, zu de-



Der neue Verbindungsdialog von SSMS 21 (Bild 1)



nen nicht Tausende von Informationsquellen im Internet zu finden sind, wird sich zeigen.

Klar ist jetzt schon, dass kryptische Namen und nicht vorhandene Beziehungen zwischen den Tabellen das Ergebnis verschlechtern. Eine KI hat eben auch keine Glaskugel.

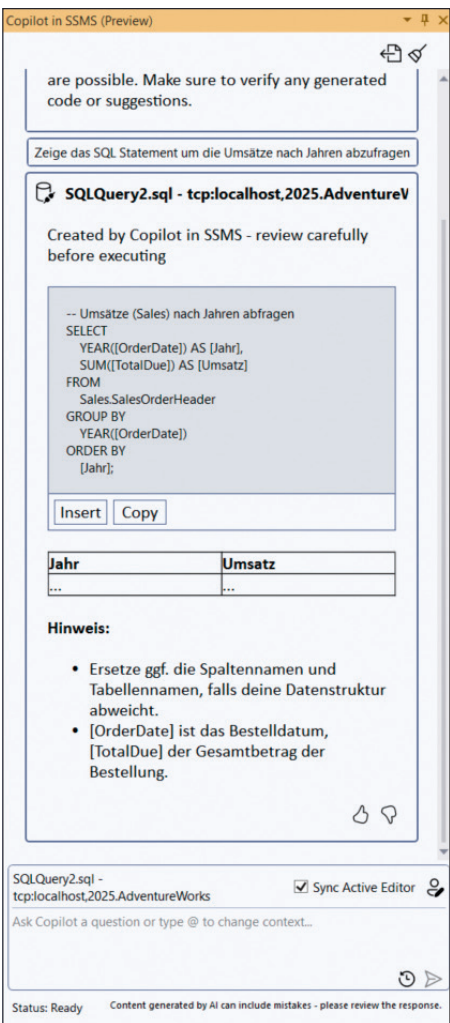
Wer sich im SSMS bisher zu Hause fühlt, wird auch die neue Version mögen. Und ja, diese SSMS-Version läuft, genau wie Visual Studio 2022, exklusiv auf Windows!

Reguläre Ausdrücke

Hier sagt die Überschrift eigentlich schon das Wichtigste. SQL Server 2025 wird (endlich) reguläre Ausdrücke unterstützen. Bis dato ist, dort wo möglich, die langsam in Vergessenheit geratene CLR-Integration in die Breche gesprungen. Das ist nun nicht mehr notwendig. Insgesamt sollen Entwicklern sieben Regex-Funktionen zur Verfügung stehen. **Tabelle 1** gibt einen schnellen Überblick.

Wer sich die Namen ansieht, wird erkennen, dass einige an schon lange existierende Funktionsnamen angelehnt sind (etwa *LIKE*, *REPLACE()* et cetera), sodass eine Orientierung nicht schwerfällt. Und damit lassen sich als Allererstes Abfragen wie die folgende durchführen. Statt einer Suche mittels *LIKE* kommt die mächtigere Funktion *REGEXP\_LIKE()* zum Einsatz:

SELECT [Firstname], \* FROM [Person].[Person]



Der Copilot steht Rede und Antwort (Bild 2)

WHERE REGEXP\_LIKE ([Firstname],  
'[AEIOU]{3,}');

*REGEXP\_LIKE()* lässt sich auch gut in Verbindung mit einem *CHECK*-Constraint einsetzen, um die Datenintegrität in einer Tabelle zu verbessern. Ein Beispiel dafür finden Sie in **Listing 1** sowie weitere Funktionen in **Bild 3**.

Insgesamt ist das eine gute Erweiterung. Leider kommt diese mit einem „Aber“: Die Performance lässt noch zu wünschen übrig. Zumindest aktuell ist noch Platz nach oben. Zum Teil ist dies den regulären Ausdrücken an sich geschuldet, die (genau wie *LIKE*) zum Teil nicht mit Indizes unterstützt werden können. Zum anderen hat Microsoft bei der Implementierung anscheinend nur das Notwendigste getan. Hier können wir gespannt sein, was kommende Patches oder Versionen bringen werden. Aber noch ist SQL Server 2025 ja auch nicht erschienen.

JavaScript Object Notation

Ja, endlich gibt es einen JSON-Datentyp. Etwas, das dem SQL Server seit der Einführung der JSON-Unterstützung mit Version 2016 gefehlt hat. Und ja, *JSON* ist eine Zeichenkette, und (*N*) *VARCHAR* ist damit sicherlich eine Option, aber optimal war das nicht. Auch gab es keinen speziellen Index, der nun ebenfalls nachgereicht wird.

Der Datentyp wird wie jeder andere Datentyp verwendet (Spalten, Datentypen, Parameter et cetera) und sorgt zum einen dafür, dass nur formell korrektes JSON möglich ist, ►

● Tabelle 1: Funktionen mit regulären Ausdrücken

Funktion	Beschreibung
REGEXP_LIKE()	Liefert einen booleschen Wert, der angibt, ob die Texteingabe mit dem Regex-Muster übereinstimmt.
REGEXP_REPLACE()	Gibt eine geänderte Quellzeichenkette zurück, die durch eine Ersatzzeichenkette ersetzt wurde, sofern das Regex-Muster gefunden wurde.
REGEXP_SUBSTR()	Extrahiert Teile einer Zeichenkette auf der Grundlage eines regulären Ausdrucksmusters. Gibt das N-te Vorkommen einer Teilzeichenkette zurück, die mit dem Regex-Muster übereinstimmt.
REGEXP_INSTR()	Liefert die Anfangs- oder Endposition der übereinstimmenden Teilzeichenkette, abhängig von der angegebenen Option.
REGEXP_COUNT()	Meldet, wie oft das Regex-Muster in einer Zeichenkette vorkommt.
REGEXP_MATCHES()	Gibt eine Tabelle der erfassten Teilzeichenkette(n) zurück, die mit einem Muster eines regulären Ausdrucks für eine Zeichenkette übereinstimmen. Wurde keine Übereinstimmung gefunden, gibt die Funktion keine Zeile zurück.
REGEXP_SPLIT_INTO_TABLE()	Liefert eine Tabelle mit aufgeteilten Zeichenketten, die durch das Regex-Muster abgegrenzt sind. Gibt es keine Übereinstimmung mit dem Muster, gibt die Funktion die Zeichenkette zurück.

## Listing 1: Reguläre Ausdrücke im Einsatz

```

CREATE TABLE dbo.EMPLOYEES (
 ID INT IDENTITY PRIMARY KEY CLUSTERED,
 [Name] VARCHAR(150),
 Email VARCHAR(320)
 CHECK (REGEXP_LIKE(Email, '^[A-Za-z0-9._%+-]+
 @[A-Za-z0-9.-]+\.[A-Za-z]{2,}$')),
 Phone_Number NVARCHAR(20)
 CHECK (REGEXP_LIKE(Phone_Number, '^(\\d{3})-
 (\\d{3})-(\\d{4})$')))
);

USE [AdventureWorks];

--REGEXP_REPLACE
SELECT REGEXP_REPLACE([PhoneNumber],
 '\\d{4}$', '****'), [PhoneNumber]
FROM [Person].[PersonPhone];

--REGEXP_SUBSTR
SELECT REGEXP_SUBSTR([PhoneNumber],
 '\\d{4}$'), [PhoneNumber]
FROM [Person].[PersonPhone];

--REGEXP_INSTR
SELECT REGEXP_INSTR([PhoneNumber], '\\d{4}$') [Start],
 REGEXP_INSTR([PhoneNumber],
 '\\d{4}$', 1, 1, 1) [End], --20?
 [PhoneNumber]
FROM [Person].[PersonPhone];

--REGEXP_COUNT
SELECT REGEXP_COUNT([PhoneNumber],
 '\\d{4}$'), [PhoneNumber]
FROM [Person].[PersonPhone];

--REGEXP_MATCHES
SELECT REGEXP_MATCHES([PhoneNumber],
 '\\d{4}$'), [PhoneNumber]
FROM [Person].[PersonPhone];

--REGEXP_SPLIT_INTO_TABLE
SELECT REGEXP_SPLIT_INTO_TABLE([PhoneNumber],
 '\\d{4}$'), [PhoneNumber]
FROM [Person].[PersonPhone];

```

zum anderen, dass die Informationen binär und damit optimal für die weitere Verarbeitung gespeichert werden. Nach „außen“ behandelt SQL Server JSON als Zeichenkette mit UTF-8-Encoding (*Latin1\_General\_100\_BIN2\_UTF8*).

In Listing 2 ist zu sehen, wie dies in Tabellen, Prozeduren und Funktionen aussieht.

Der JSON-Datentyp kann dann mit JSON-Funktionen wie *JSON\_VALUE()*, *JSON\_CONTAINS()*, *JSON\_PATH\_EXISTS()* sowie für Modifikationen mit *JSON\_MODIFY()* verwendet werden. Zeichenkettenfunktionen für *VARCHAR* beziehungsweise *NVARCHAR* führen zu einem Fehler, und eine vorherige Konvertierung verbietet sich (hoffentlich) von selbst.

Um also sicherzustellen, dass nur gültiges JSON gespeichert werden kann, ist damit die Zeit von Notlösungen mit der JSON-Funktion in Verbindung mit einer *CHECK*-Constraint bald vorbei – sehr gut.

### JSON-Index

Das ist aber nicht die einzige gute Nachricht für Freunde von JSON. SQL Server 2025 bietet endlich einen spezialisierten Index – Performance ist ja schließlich (fast) alles. Und Indizes sind für Suchoperationen in der Regel das A und O für Verbesserungen. Als Voraussetzung für einen JSON-Index muss die Tabelle

über einen gruppierten Index (Clustered Index) mit maximal 32 Spalten verfügen. Außerdem darf eine JSON-Spalte nur in einem einzigen JSON-Index vorhanden sein. Dafür darf der JSON-Index auch nur eine JSON-Spalte umfassen. Und selbstredend muss der Datentyp dieser Spalte *JSON* sein.

Der Index selbst kann für den kompletten Inhalt oder sowohl für einen als auch für mehrere JSON-Pfade definiert werden. Dafür dient die *FOR*-Klausel, und die einzelnen Pfade dürfen sich nicht überlappen. Darüber hinaus kann der Index gefiltert werden (*filtered index*). Dies funktioniert in der

Preview 2.1 zwar noch nicht, sollte aber im finalen Release möglich sein. Listing 3 stellt die drei Varianten vor.

Ein JSON-Index wird dann bei Einschränkungen, Sortierungen und Gruppierungen der JSON-Funktionen eingesetzt, die für den JSON-Datentyp zugelassen sind (etwa *JSON\_VALUE()*, *JSON\_CONTAINS()* oder *JSON\_PATH\_EXISTS()*).

Wie auch schon vor SQL Server 2025 können mittels Abfragen auf *sys.indexes* alle Indizes einer Datenbank abgerufen werden. Und dort ist dann auch der neue Index-Typ zu finden.

```

SELECT * FROM sys.indexes / *
WHERE OBJECT_ID = OBJECT_ID(
 'dbo.JsonDemo') */;

```

	(No column name)	PhoneNumber
92...	107-555-****	107-555-0181
92...	107-555-****	107-555-0187
92...	107-555-****	107-555-0193
92...	107-555-****	107-555-0195
92...	108-555-****	108-555-0111
92...	108-555-****	108-555-0114
92...	108-555-****	108-555-0122
92...	108-555-****	108-555-0126
92...	108-555-****	108-555-0165
92...	108-555-****	108-555-0171
92...	108-555-****	108-555-0180
92...	108-555-****	108-555-0196
92...	109-555-****	109-555-0100
92	109-555-****	109-555-0100

**REGEXP\_REPLACE():** Auszug der Rückgabe (Bild 3)



## ● Listing 2: JSON-Datentyp in Tabellen, Prozeduren und Funktionen

```

CREATE TABLE dbo.JsonDemo
(
 ID INT IDENTITY PRIMARY KEY,
 Content JSON NOT NULL
);

GO

CREATE OR ALTER PROCEDURE dbo.usp_JsonAsParameter
 @ID INT,
 @Content JSON
AS
BEGIN
 SELECT @Content;
END

GO

CREATE OR ALTER FUNCTION dbo.func_Json
(
 @Content JSON
)
RETURNS JSON
AS
BEGIN
 DECLARE @c JSON

 SELECT @c = @Content

 RETURN @c
END

```

**Bild 4** zeigt einen Auszug der Rückgabe dieser Abfrage, bei welcher der Einfachheit halber Objektnamen et cetera nicht explizit ausgewiesen werden.

Nun zum Schatten: Leider ist es aktuell (vielleicht noch) so, dass der Abfrageoptimierer möglicherweise keine detaillierten Statistiken zu Verfügung hat, sodass er nicht den optimalen Index wählt. Dies resultiert dann entweder in der Wahl des falschen oder keines Indexes. Beides ist leider nicht optimal für die Performance. Bei großen Datenmengen kann das zu einem Problem führen. Der JSON-Index sollte folglich nicht alleine für Einschränkungen, Sortierungen und Gruppierungen verwendet werden, sodass herkömmliche Indizes zum Einsatz kommen können.

Hier ist zu hoffen, dass Microsoft bei Patches oder nächsten Versionen weiter optimiert. Auf jeden Fall aber ist der JSON-Index ein Schritt in die richtige Richtung!

### JSON-Schema

Ja, leider, leider gibt es (noch) kein JSON-Schema, das vergleichbar zur XML Schema Definition (XSD) ist. Aber es muss ja noch Raum für weitere Optimierungen in künftigen Versionen bleiben.

### REST-API-Aufrufe

Im Zuge der Umsetzung von „AI ready“ wurde ein Feature geschaffen, um REST-API-Aufrufe direkt auf dem SQL ►

## ● Listing 3: Ausprägungen von JSON-Indizes

```

-- Demo-Tabelle
CREATE TABLE dbo.JsonDemo
(
 ID INT PRIMARY KEY CLUSTERED,
 Content JSON
);

-- Nur ein Index pro JSON Spalte!

DROP INDEX IF EXISTS jsonIndex ON dbo.JsonDemo;
GO

-- Für das gesamte JSON
CREATE JSON INDEX jsonIndex
 ON dbo.JsonDemo(Content);
GO

DROP INDEX IF EXISTS jsonIndex ON dbo.JsonDemo;
GO

-- Für Elemente im JSON-Pfad(en), die sich nicht
-- überlappen dürfen
CREATE JSON INDEX jsonIndex
 ON dbo.JsonDemo(Content) FOR (
 '$.Customer', '$.Employee', '$.Order')
 WITH (FILLFACTOR = 80);
GO

DROP INDEX IF EXISTS jsonIndex ON dbo.JsonDemo;
GO

-- Gefilterter Index
CREATE JSON INDEX jsonIndex
 ON dbo.JsonDemo(Content) FOR (
 '$.Customer', '$.Employee')
 WHERE JSON_PATH_EXISTS(Info,
 '$.Order.IsProcessed') = 1

```

#### ● Listing 4: REST-API-Aufrufe vorbereiten

```
-- Aktivieren
EXECUTE sp_configure
 'external rest endpoint enabled', 1;
RECONFIGURE WITH OVERRIDE;

-- Berechtigung vergeben
GRANT EXECUTE
 ANY EXTERNAL ENDPOINT TO [<PRINCIPAL>];
```

	object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key	is_primary_key	is_unique_constraint	fill_factor	is_padded	is_disabled
288	123062...	PK_...	1	1	CLUSTERED	1	1	0	1	0	0	0	0
289	123062...	IX_...	2	2	NONCLUSTERED	0	1	0	0	0	0	0	0
290	123062...	IX_...	3	2	NONCLUSTERED	0	1	0	0	0	0	0	0
291	123619...	PK_...	1	1	CLUSTERED	1	1	0	1	0	0	0	0
292	123619...	json...	12160...	9	JSON	0	1	0	0	0	0	0	0
293	125567...	ci_b...	1	1	CLUSTERED	1	1	0	0	0	0	0	0
294	126610...	PK_...	1	1	CLUSTERED	1	1	0	1	0	0	0	0

**sys.indexes:** Auszug der Rückgabe (Bild 4)

Server durchzuführen. Dieses Feature kann auch direkt und losgelöst von KI verwendet werden. Damit lässt sich auf eine Vielzahl externer Funktionalitäten zugreifen. Microsoft wirbt unter anderem mit diesen:

- Daten über eine Azure-Funktion verarbeiten lassen.
- Ein Power-BI-Dashboard aktualisieren.
- Aufrufen eines lokalen, unternehmensinternen REST-Endpunkts.
- Kommunikation mit Azure OpenAI Services.

Der Aufruf selbst ist ein Stored-Procedure-Aufruf und kann überall dort stehen, wo ein solcher möglich ist. Allerdings ist zu bedenken, dass solche Aufrufe im Vergleich zu anderen Datenbankoperationen extrem langsam sind und sich dadurch Schleifen und Trigger von selbst verbieten.

Vor dem ersten Aufruf bedarf es ein wenig Vorbereitung. Der Konfigurationsschalter *external rest endpoint enabled* muss auf 1 gesetzt werden, und gegebenenfalls muss an den Aufrufer die Berechtigung *EXECUTE ANY EXTERNAL ENDPOINT* vergeben werden. Listing 4 zeigt ein Beispiel.

Die Stored Procedure *sp\_invoke\_external\_rest\_endpoint* führt den REST-API-Aufruf durch. Wie für einen solchen Aufruf üblich, werden für einen URL Headers, Methode, Payload, Credentials und mehr angegeben. In Listing 5 sehen Sie, wie ein solcher Aufruf aussehen kann. Das Ergebnis sind dann der HTTP-Return-Code und die Payload.

Die Payload der Rückgabe kann wahlweise in JSON oder XML erfolgen – je nachdem, was die aufgerufene Seite verschickt. Listing 6 zeigt die Strukturen der beiden Formate.

Bei den REST-API-Aufrufen gibt es einige festgelegte Limits wie eine Payload-Größe von 100 MByte, eine URL-Länge von 8 KByte und eine maximale Header-Größe ebenfalls von 8 KByte. Zudem existiert eine Drosselung (Throttling) auf

10 Prozent der verfügbaren Worker Threads mit einem Maximum von 150 Workers.

Und damit zur Kehrseite: Microsoft führt sicher Gutes im Schilde, doch die Gefahr ist groß, dass es in einem Chaos endet, weil es einfach sehr verlockend und praktisch ist, mal hier und mal da einen solchen Aufruf einzubauen.

Im Vergleich zu Azure SQL Database und Azure SQL Managed Instance sind die möglichen Endpoints nicht eingeschränkt. Sicher, man muss keine Aufrufe einbauen, aber wenn, dann ist nach Meinung des Autors ein Datenbank-Server der letzte Ort, wo dies stattfinden sollte. Manche Leserinnen und Leser werden sich noch daran erinnern, dass es mal

einen SOAP-Server im SQL Server gab, um Daten bereitzustellen. Oder dass man dem SQL Server seinerzeit E-Mails mit Abfragen schicken konnte, die dann ausgeführt und in eine Antwort gepackt wurden. Jeweils zu den Zeiten, als die entsprechenden Technologien neu und hip waren. Vielleicht passiert

#### ● Listing 5: REST-API-Aufrufe durchführen

```
-- Beispiel-Aufruf
DECLARE
 @url NVARCHAR(4000) =
 'https://<DNS-Name or IP>',
 @method NVARCHAR(10) = 'GET' /* GET | POST |
 PUT | PATCH | DELETE | HEAD */,
 @headers NVARCHAR(MAX) =
 N'{"headername":"value"}',
 @payload NVARCHAR(MAX) = '....',
 @ret INT,
 @response NVARCHAR(MAX);

EXEC @ret = sp_invoke_external_rest_endpoint
 @url = @url,
 @method = @method,
 @headers = @headers,
 @payload = @payload,
 @response = @response output;

-- Ausgabe von Return Code und Response
SELECT @ret AS ReturnCode,
 @response AS Response;
```

dies auch bei REST-API-Aufrufen. Kurzum: Mit mächtigen Features kommt auch große Verantwortung.

Vektoren als Datentyp

Der zweite neue Datentyp, den SQL Server 2025 bietet, ist der VECTOR-Datentyp. Er dient der Arbeit mit n-dimensionalen Vektoren, wobei n zwischen 1 und 1998 liegen darf und angegeben werden muss. Vektoren sind der technische Unterbau für die Vektoren-Suche und somit für Künstliche Intelligenz. Sollen Werte vom Typ VECTOR zugewiesen oder übergeben werden, müssen die Dimensionen genau übereinstimmen.

VECTOR kann als Typ für Spalten, Variablen, Parameter et cetera verwendet werden – immer mit Angabe der Dimension. Rechenoperationen ausführen (etwa addieren, multiplizieren et cetera) kann man mit diesem Datentyp allerdings nicht direkt. Intern werden die Werte binär gespeichert (Single, 4 Bytes pro Zahl), bei Zuweisung oder Rückgabe wird jedoch ein JSON-Array verwendet, was die Arbeit in einigen Fällen sicher vereinfacht.

Erlaubt sind alle numerischen Datentypen, die dem SQL Server zur Verfügung stehen. Alle Features rund um VECTOR müssen aktuell noch mit der folgenden Anweisung aktiviert werden – später soll dies nicht mehr notwendig sein:

Listing 6: Rückgabe in JSON oder XML

```
-- Rückgabe JSON
{
 "response": {
 "status": {
 "http": {
 "code": "",
 "description": ""
 }
 },
 "headers": {}
 },
 "result": {}
}

-- Rückgabe XML
<output>
 <response>
 <status>
 <http code="" description=" " />
 </status>
 <headers>
 <header key="" value="" />
 <header key="" value="" />
 </headers>
 </response>
 <result>
 </result>
</output>
```

Listing 7: VECTOR deklarieren und verwenden

```
-- Deklarieren, zuweisen und ausgeben
DECLARE @vector1 VECTOR(5) = '[1, 2, 3, 4, 5]';
SELECT @vector1;

DECLARE @vector2 VECTOR(3) = JSON_ARRAY(
 1.0, -0.2, 30);
SELECT
 CAST(@vector1 AS NVARCHAR(MAX)) AS Vector1,
 CAST(@vector2 AS JSON) AS Vector2;

CREATE TABLE dbo.VectorDemo
(
 ID INT IDENTITY PRIMARY KEY,
 Vector VECTOR(5) NOT NULL
);

GO
CREATE OR ALTER PROCEDURE dbo.usp_VectorAsParameter
 @ID INT,
 @Content VECTOR(5)
AS
BEGIN
 SELECT @Content;
END

GO
CREATE OR ALTER FUNCTION dbo.func_Vector
(
 @Content VECTOR(5)
)
RETURNS VECTOR(5)
AS
BEGIN
 DECLARE @c VECTOR(5)

 SELECT @c = @Content

 RETURN @c
END
```

DBCC TRACEON(466, 474, 13981, -1);

In Listing 7 sehen Sie, wie man den neuen Datentyp deklariert und verwendet, und Bild 5 zeigt die Ausgabe des 5-dimensionalen Vektors [1, 2, 3, 4, 5].

Results	Messages
(No column name)	
1	[1.0000000e+000,2.0000000e+000,3.0000000e+000,4.0000000e+000,5.0000000e+000]

Ausgabe eines 5-dimensionalen Vektors mit der Eingabe [1, 2, 3, 4, 5] (Bild 5)

## Vektorfunktionen

Für Vektoren hat SQL Server 2025 eine ganze Reihe von Funktionen im Gepäck. **Tabelle 2** stellt diese kurz vor, und **Listing 8** zeigt weitere Details, wie zur Verfügung stehende Metriken und Normtypen sowie die Aufrufe dazu. `VECTOR_SEARCH()` sticht dabei hervor und wird daher in einem eigenen Abschnitt noch einmal genauer durchleuchtet.

### ● Listing 8: Vektorfunktionen

```
-- Deklarieren, zuweisen und ausgeben
DECLARE @v1 VECTOR(5) = '[1, 2, 3, 4, 5]'
DECLARE @v2 VECTOR(5) = '[1, 2, 4, 4, 5]'

SELECT VECTOR_DISTANCE(
 'euclidean', @v1, @v2) [VECTOR_DISTANCE];
-- Metriken
-- cosine - Cosine distance
-- euclidean - Euclidean distance
-- dot - (Negative) Dot product

SELECT VECTOR_NORM(@v1, 'norm2') [VECTOR_NORM];
-- Normtypen
-- norm1 - Die 1-Norm, die die Summe der absoluten
-- Werte der Vektorkomponenten ist.
-- norm2 - Die 2-Norm, auch bekannt als Euklidische
-- Norm, ist die Quadratwurzel aus der Summe der
-- Quadrate der Vektorkomponenten.
-- norminf - Die Unendlichkeitsnorm, die das Maximum
-- der Absolutwerte der Vektorkomponenten ist.

SELECT VECTOR_NORMALIZE(
 @v1, 'norm1') [VECTOR_NORMALIZE];
-- Normtypen wie VECTOR_NORM()

SELECT VECTOR_PROPERTY(@v1, 'Dimensions');
-- Eigenschaften
-- Dimensions - Anzahl der Dimensionen
-- BaseType - sysname mit dem Namen des Datentyps.
```

Mithilfe dieser Funktionen kann mit Vektoren gearbeitet werden, und sie sind daher zur Bestimmung von Ähnlichkeiten (mehr dazu im Abschnitt „Vektorsuche“ weiter unten) geeignet, was insbesondere für Künstliche Intelligenz hilfreich ist. Welchen Nutzen die Funktionen in eigenen Anwendungen haben können, muss jeder für sich selbst bewerten.

## Vektor-Indizes

Um die im Anschluss betrachteten Wege zur Vektorsuche möglichst effizient durchführen zu können, wird ab einer gewissen Datenmenge (rund 50000 Zeilen) ein geeigneter Index benötigt. Dabei ist es wichtig zu verstehen, dass der Begriff Index, wenn er sich auf einen Vektor-Index bezieht, eine andere Bedeutung hat als der Index, mit dem Sie in relationalen Datenbanken zu arbeiten gewohnt sind.

In der Tat liefert ein Vektorindex „nur“ ungefähre Ergebnisse. Es wird also keine k-NN-Suche (k-Nearest Neighbors) durchgeführt, sondern eine Suche nach dem ANN-Ansatz (Approximate Nearest Neighbors). Diese Suche liefert keine zu einhundert Prozent optimalen Ergebnisse, ist aber um ein Vielfaches schneller und benötigt viel weniger Ressourcen. Das bedeutet, es besteht eine sehr kleine Restchance, dass es einen Treffer gibt, der zwar sehr gut passen würde, aber dennoch nicht gefunden wird.

In der SQL-Server-Engine basieren die Vektor-Indizes auf dem DiskANN-Algorithmus [5]. DiskANN ist ein auf Graphen basiertes System zur Indizierung und Suche großer Mengen von Vektordaten unter Verwendung begrenzter Rechenressourcen. Der Algorithmus nutzt SSDs und minimalen Speicher sehr effizient, um deutlich mehr Daten zu verarbeiten, als In-Memory-Indizes das können, während gleichzeitig hohe Abfrageleistungen pro Sekunde (QPS) und niedrige Latenzzeiten beibehalten werden. Das erlaubt es, ein Gleichgewicht zwischen Speicher-, CPU- und E/A-Nutzung sowie Suchleistung zu gewährleisten. Einen Vektor-Index anzulegen klappt wie folgt:

```
CREATE VECTOR INDEX vecidx
 ON dbo.VectorDemo(Embedding)
WITH (metric = 'cosine', type = 'DiskANN', MAXDOP = 10);
-- Metriken
-- cosine - Cosine distance
```

### ● Tabelle 2: Vektorfunktionen

Funktion	Beschreibung
<code>VECTOR_DISTANCE()</code>	Berechnet den Abstand zwischen zwei Vektoren unter Verwendung einer bestimmten Abstandsmetrik.
<code>VECTOR_SEARCH()</code>	Liefert die nächstgelegenen Vektoren zu einem gegebenen Abfragevektor und einer Abstandsmetrik unter Verwendung eines Algorithmus zur approximativen Vektorsuche.
<code>VECTOR_NORM()</code>	Nimmt einen Vektor als Eingabe und gibt die Norm des Vektors (die ein Maß für seine Länge oder Größe ist) in einem bestimmten Normtyp zurück.
<code>VECTOR_NORMALIZE()</code>	Nimmt einen Vektor als Eingabe und gibt den normalisierten Vektor zurück, das heißt einen Vektor, der so skaliert ist, dass er in einem bestimmten Normtyp eine Länge von 1 hat. Passt einen Vektor so an, dass seine Länge nach den Regeln des angegebenen Normtyps normalisiert wird.
<code>VECTOR_PROPERTY()</code>	Gibt bestimmte Eigenschaften eines gegebenen Vektors zurück.

```
-- euclidean - Euclidean distance
-- dot - (Negative) Dot product
-- Typ
-- DiskANN
```

Wie bereits beim JSON-Index können die neuen Indizes mittels Abfragen auf `sys.indexes` aufgelistet werden.

```
SELECT * FROM sys.indexes /*
WHERE OBJECT_ID = OBJECT_ID(' dbo.Articles') */;
```

**Bild 6** zeigt den entsprechenden Auszug der Rückgabe – auch wenn der Einfachheit halber Objektnamen et cetera nicht explizit ausgewiesen werden.

## Künstliche Intelligenz

Dank der Möglichkeit, REST-API-Aufrufe durchzuführen, können KI-Modelle trainiert werden, um für die Inhalte Embeddings zu erstellen, die später die Grundlage für eine Suche darstellen. Ein Embedding ist dabei ein Vektor mit teilweise mehreren Hundert oder gar mehreren Tausend Dimensionen. Die Werte sind dabei durch ein KI-Modell so gewählt, dass ähnliche Sachverhalte oder Begriffe näher beieinander liegen als solche, die eher nichts miteinander zu tun haben. Die Begriffe *Katze* und *Tiger* liefern also Embeddings, die viel näher beieinander liegen als die Embeddings von *Katze* und *Kühlschrank*.

Im Fall des SQL Servers geschieht das im Kern durch die Anweisung `CREATE EXTERNAL MODEL`, der neben ►

### ● Listing 9: Embeddings per AI-Modell

```
-- REST API-Aufrufe aktivieren
EXECUTE sp_configure
 'external rest endpoint enabled', 1;
RECONFIGURE WITH OVERRIDE;
GO

-- Erstellen eines Hauptschlüssels für die Datenbank
IF NOT EXISTS (
 SELECT *
 FROM sys.symmetric_keys
 WHERE [name] = '##MSDatabaseMasterKey##')
BEGIN
 -- ALTER SERVICE MASTER KEY FORCE REGENERATE;
 CREATE MASTER KEY ENCRYPTION BY PASSWORD =
 N'$$91yjYxqIWWpM';
END
GO

-- Erstellen von Zugangsdaten zu Azure OpenAI unter
-- Verwendung eines Schlüssels
CREATE DATABASE SCOPED CREDENTIAL
 [https://xxx.openai.azure.com/]
 WITH IDENTITY = 'HTTPEndpointHeaders',
 secret = '{"api-key":"<geheim>}';
GO

-- Erstellen eines externen Modells zum Aufrufen des
-- REST-Endpunkts für Azure OpenAI-Aufruf
CREATE EXTERNAL MODEL MyAzureOpenAiModel
WITH (
 LOCATION = 'https://xxx.openai.azure.com/openai/
 deployments/text-embedding-ada-002/
 embeddings?api-version=2023-05-15',
 API_FORMAT = 'Azure OpenAI',
 MODEL_TYPE = EMBEDDINGS,
 MODEL = 'text-embedding-ada-002',

 CREDENTIAL = [https://xxx.openai.azure.com/]
);

-- Eine Tabelle mit Text erstellen, um Daten zu
-- chunken und einzufügen
CREATE TABLE textchunk
(
 text_id INT IDENTITY (1, 1) PRIMARY KEY,
 text_to_chunk NVARCHAR (MAX));
GO

INSERT INTO textchunk (text_to_chunk)
VALUES
 ('Viel Text'),
 ('Und noch mehr Text')
GO

-- Neue Tabelle erstellen, die den in Stücke geschnit-
-- tenen Text und die Vektoreinbettungen enthält
CREATE TABLE text_embeddings
(
 embeddings_id INT IDENTITY (1, 1) PRIMARY KEY,
 chunked_text NVARCHAR (MAX),
 vector_embeddings VECTOR (1536)
);

-- Einfügen des gehackten Textes und der Vektor-
-- einbettungen in die Tabelle text_embeddings mit
-- AI_GENERATE_CHUNKS und AI_GENERATE_EMBEDDINGS
INSERT INTO text_embeddings (
 chunked_text, vector_embeddings)
SELECT c.chunk, AI_GENERATE_EMBEDDINGS(
 c.chunk USE MODEL MyAzureOpenAiModel)
FROM textchunk t
CROSS APPLY
 AI_GENERATE_CHUNKS(source = t.text_to_chunk,
 chunk_type = FIXED, chunk_size = 100) c;
```



	object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key	is_primary_key	is_unique_constraint	fill_factor	is_padded	is_disabled
186	187758...	json...	1	1	CLUSTERED	0	1	0	0	0	0	0	0
187	187758...	json...	2	2	NONCLUSTERED	0	1	0	0	0	0	0	0
188	195758...	PK_...	1	1	CLUSTERED	1	1	0	1	0	0	0	0
189	195758...	vec...	11520...	8	VECTOR	0	1	0	0	0	0	0	0
190	198958...	PQ_...	1	1	CLUSTERED	1	1	0	0	0	0	0	0
191	198958...	PQ_...	2	2	NONCLUSTERED	1	1	0	0	0	0	0	0
192	198958...	PQ_...	3	2	NONCLUSTERED	1	1	0	0	0	0	0	0

Eine weitere Rückgabe von sys.indexes (Auszug) (Bild 6)

Credentials und dem gewünschten Modell auch ein URL mitgegeben wird, der angibt, wo das cloudbasierte Modell erreicht werden kann. Beim Rest von Listing 9 geht es nur noch um technische Details und die Aufgabe, Embeddings für eigene Inhalte zu erzeugen.

Die Dauer der Ausführung hängt dabei von der Menge der Daten ab, der Anbindung des SQL Servers an das Internet und der Tiefe des Geldbeutels (Stichwort „token rate limit“).

## Vektorsuche

Damit sind alle Bausteine für „AI ready“ zusammengetragen: der Datentyp `VECTOR`, um die Informationen zu speichern, Funktionen für verschiedene Arten von Suchen, ein Index für gute Performance und REST-API-Aufrufe, um AI-Modelle zu trainieren und Embeddings zu erzeugen.

Nun geht es darum, mit möglichst wenig Aufwand (Ressourcen) ein gutes Ergebnis zu finden. Ein Ansatz, dies zu er-

### Listing 10: k-NN- und ANN-Suchen in Vektoren

```
-- k Nearest neighbors (k-NN)
DECLARE @k INT = 10;
DECLARE @em VECTOR(1536) =
 AI_GENERATE_EMBEDDING(N'Pink Floyd music style'
 USE MODEL MyAzureOpenAiModel);
SELECT TOP (@k) id, VECTOR_DISTANCE('cosine', @em,
 [content_vector]) AS distance, title
FROM [dbo].[wikipedia_articles_embeddings]
ORDER BY distance

-- Approximate Nearest Neighbors (ANN)
-- Schritt 1: Einfache Tabelle anlegen
DROP TABLE IF EXISTS dbo.Articles;
GO
CREATE TABLE dbo.Articles
(
 id INT PRIMARY KEY CLUSTERED,
 title NVARCHAR(100),
 content NVARCHAR(MAX),
 embedding VECTOR(5)
);

-- Schritt 2: Daten einfügen
INSERT INTO Articles (id, title, content, embedding)
VALUES
 (1, 'Intro to AI', 'This article introduces AI
 concepts.', '[0.1, 0.2, 0.3, 0.4, 0.5]'),
 (2, 'Deep Learning', 'Deep learning is a subset of
 ML.', '[0.2, 0.1, 0.4, 0.3, 0.6]'),
 (3, 'Neural Networks', 'Neural networks are
 powerful models.', '[0.3, 0.3, 0.2, 0.5, 0.1]'),
 (4, 'Machine Learning Basics', 'ML basics for
 beginners.', '[0.4, 0.5, 0.1, 0.2, 0.3]'),
 (5, 'Advanced AI', 'Exploring advanced AI
 techniques.', '[0.5, 0.4, 0.6, 0.1, 0.2]');

-- Schritt 3: Vektor-Index anlegen
CREATE VECTOR INDEX vec_idx ON Articles(embedding)
WITH (metric = 'cosine', type = 'DiskANN',
 MAXDOP = 10);
-- Metriken
-- cosine - Cosine distance
-- euclidean - Euclidean distance
-- dot - (Negative) Dot product
-- Typ
-- DiskANN

-- Schritt 4: k-NN Suche durchführen
DECLARE @qv VECTOR(5) = '[0.3, 0.3, 0.3, 0.3, 0.3]';
SELECT
 t.id,
 t.title,
 t.content,
 s.distance
FROM
 VECTOR_SEARCH(
 table = Articles AS t,
 column = embedding,
 similar_to = @qv,
 metric = 'cosine',
 top_n = 3
) AS s
ORDER BY s.distance, t.title;
```

● **Tabelle 3: Fuzzy String Matching**

Funktion	Beschreibung
<code>EDIT_DISTANCE()</code>	Berechnet die Anzahl der Einfügungen, Löschungen, Ersetzungen und Transpositionen, die erforderlich sind, um eine Zeichenkette in eine andere umzuwandeln. Das Verfahren ist auch als Levenshtein-Distanz [6] bekannt.
<code>EDIT_DISTANCE_SIMILARITY()</code>	Berechnet einen Ähnlichkeitswert, der von 0 (keine Übereinstimmung) bis 100 (vollständige Übereinstimmung) reicht.
<code>JARO_WINKLER_DISTANCE()</code>	Berechnet die Editierdistanz zwischen zwei Zeichenfolgen, wobei Zeichenfolgen bevorzugt werden, die von Anfang an für eine bestimmte Präfixlänge übereinstimmen [7].
<code>JARO_WINKLER_SIMILARITY()</code>	Berechnet einen Ähnlichkeitswert, der von 0 (keine Übereinstimmung) bis 1 (vollständige Übereinstimmung) reicht.

reichen, ist es, für einen Text wie beispielsweise *Pink Floyd music style* mittels AI-Modellen einen Referenzvektor zu ermitteln und für alle gespeicherten Embeddings/Vektoren anschließend den Abstand zu diesem Vektor zu berechnen. Die Top-Treffer sind dann das gesuchte Ergebnis. Bei dieser Art von Suche spricht man von einer k-NN-Suche (k-Nearest Neighbors), wobei *k* die maximale Anzahl der gewünschten Treffer ist. Oft wird zudem noch der Abstand begrenzt, damit es nicht zu Treffern kommt, deren Abstand schlicht zu groß ist, um noch als sinnvoll erachtet zu werden.

Für die Berechnung des Abstands wird die Funktion `VECTOR_DISTANCE()` verwendet. Dieser Ansatz ist präzise, leider aber auch ressourcenintensiv, da für jeden Vektor der Abstand berechnet werden muss, was Speicher-, CPU- und E/A-Nutzung kostet. Die Datenmenge, die auf diese Weise durchsucht wird sollte deshalb begrenzt sein und möglichst 50 000 Zeilen nicht überschreiten. Das bedeutet jetzt nicht, dass die abgesuchte Tabelle nur 50 000 Zeilen besitzen darf, schließlich kann die Abfrage noch weitere Prädikate enthalten, die entsprechend einschränken. Kurzum: Das Verfahren eignet sich nur, wenn tatsächlich das beste Ergebnis benötigt wird und die zu durchsuchende Datenmenge überschaubar ist.

Reichen sehr gute bis extrem gute Ergebnisse, so wird eine indexgestützte Suche nach dem Ansatz Approximate Nearest Neighbors (ANN) erfolgen. Diese liefert zwar keine zu einhundert Prozent optimalen Ergebnisse, ist aber um ein Vielfaches schneller und benötigt viel weniger Ressourcen.

Dafür wird dann die Funktion `VECTOR_SEARCH()` verwendet, welche im Wesentlichen die Tabelle/Embedding-

Spalte, den Referenzvektor des Textes und die maximale Anzahl von Treffern akzeptiert.

```
...
VECTOR_SEARCH(
 table = Articles AS t,
 column = embedding,
 similar_to = @qv,
 metric = 'cosine',
 top_n = 3
) AS s
...
```

Die Rückgabe kann dann neben dem Abstand auch andere Spalten der Tabelle enthalten. Listing 10 zeigt die beiden Arten von Suchen, allerdings ohne dabei die Embeddings mittels KI zu generieren. Dazu sei auf Listing 9 verwiesen.

Fuzzy String Matching

Zeichenkettenvergleiche waren bis dato im SQL Server nur exakte Vergleiche (das betrifft auch reguläre Ausdrücke). Kleine Schreibfehler oder -varianten sorgten dabei schon für negative Ergebnisse. Da bietet sich die Verwendung eines unscharfen oder ungefähren Zeichenfolgenabgleichs an, um zu prüfen, ob zwei Zeichenfolgen ähnlich genug sind, und die Differenz zwischen beiden Zeichenfolgen zu berechnen. Auch hier wird die CLR-Integration nicht länger benötigt. Der SQL Server 2025 bietet dafür jetzt zwei Ansätze: die Levenshtein-Distanz [6] und die Jaro-Winkler-Distanz [7]. ►

	WortA	WortB	EDIT_DISTANCE	EDIT_DISTANCE_SIMILARITY	JARO_WINKLER_DISTANCE	JARO_WINKLER_SIMILARITY
1	Abc	Abc	0	100	0	1
2	Tisch	Stuhl	5	0	0.5333333	0.4666667
3	Schaufel	Schaufelbagger	6	57	0.08571428	0.9142857
4	Colour	Color	1	83	0.03333336	0.9666666
5	Flavour	Flavor	1	86	0.02857143	0.9714286
6	Centre	Center	2	67	0.03333336	0.9666666
7	Theatre	Theater	2	71	0.02857143	0.9714286
8	Organise	Organize	1	88	0.05000001	0.95
9	Analyse	Analyze	1	86	0.05714279	0.9428572

Auszug der Rückgabe der Fuzzy-String-Matching-Funktionen aus Listing 11 (Bild 7)

**Listing 11: Fuzzy String Matching**

```

CREATE TABLE dbo.WordPairs (
 WordID INT IDENTITY(1,1) PRIMARY KEY,
 WortA NVARCHAR(50),
 WortB NVARCHAR(50)
);

-- Schritt 2: Daten einfügen
INSERT dbo.WordPairs (WortA, WortB) VALUES
('Abc', 'Abc'),
('Tisch', 'Stuhl'),
('Schaufel', 'Schaufelbagger'),
('Colour', 'Color'),
('Flavour', 'Flavor'),
('Centre', 'Center'),
('Theatre', 'Theater'),
('Organise', 'Organize'),
('Analyse', 'Analyze');

SELECT WortA, WortB,
 EDIT_DISTANCE(WortA, WortB) [EDIT_DISTANCE],
 -- Je niedriger desto besser (0..n)
 EDIT_DISTANCE_SIMILARITY(WortA, WortB)
 [EDIT_DISTANCE_SIMILARITY],
 -- Je höher desto besser (0..100)
 JARO_WINKLER_DISTANCE(WortA, WortB)
 [JARO_WINKLER_DISTANCE],
 -- Je niedriger desto besser (0..1)
 JARO_WINKLER_SIMILARITY(WortA, WortB)
 [JARO_WINKLER_SIMILARITY]
 -- Je höher desto besser (0..1)
FROM dbo.WordPairs;

```

**Tabelle 3** stellt die beiden Ansätze kurz, vor und **Listing 11** zeigt die Aufrufe der vier Funktionen. In **Bild 7** sehen Sie einen Auszug der Rückgabe dieser Funktionen.

Aber auch hier ein Wort der Warnung: Die Funktionen profitieren in keinerlei Weise von Indizes, da jeder Wert berechnet werden muss. Bei größeren Datenmengen werden hier Laufzeit und Ressourcenbedarf massiv ansteigen.

**Was gibt es sonst noch Neues?**

Eigentlich eine ganze Menge Dinge, die aber vorsichtig ausgedrückt eher marginal sind oder für Developer keine große Rolle spielen. Trotzdem sei noch eine Reihe von Neuerungen stickpunktartig aufgeführt:

- Direkter Zugriff auf Formate wie CSV, Parquet und Delta mittels *OPENROWSET*, ohne *PolyBase*.
- Neue T-SQL-Funktionen: *PRODUCT()*, *DATE\_BUCKET()*, *STRING\_SPLIT()* mit Positionsparameter, *bigint*-Support in *DATEADD*.
- Verbesserte Columnstore-Performance.
- Query Store auf sekundären Replikaten.

- ZSTD-Kompression für Backups.
- Tempdb-Optimierung auf Linux durch *tmpfs*.
- Ein intelligenter Security-Cache, der bei vielen gleichzeitigen Logins selektiv aktualisiert wird.
- Optimierungen bei *sp\_executesql*.

Es gibt also noch einiges mehr beim SQL Server 2025 zu entdecken, aber der Autor möchte natürlich den einleitend erwähnten Fokus wahren.

**Fazit**

Obwohl SQL Server 2025 mit spannenden Neuerungen wie nativer KI- und Vektor-Unterstützung wirbt, sollten Entwickler einen kritischen Blick darauf werfen. Die Implementierung der fortschrittlichen KI-Features ist komplex und erfordert tiefgreifendes, neues Wissen zur Optimierung, da CPU-intensive Vektor-Operationen sonst traditionelle Abfragen ausbremsen können.

Die tiefe Integration mit Azure AI, obwohl funktional, erhöht die Abhängigkeit vom Microsoft-Ökosystem und schafft einen potenziellen Vendor-Lock-in. Auch neue T-SQL-Funktionen wie JSON, Vektoren, Regex sind ressourcenintensiv und erfordern einen bedachten Einsatz, um die Performance nicht zu gefährden. Ein Upgrade auf SQL Server 2025 ist daher nicht für jedes Szenario sofort ein klarer Gewinn und muss sorgfältig gegen die realen Kosten und die technologische Reife abgewogen werden. ■

[1] Download SQL Server 2025 Preview, [www.microsoft.com/en-us/evalcenter/sql-server-2025-download](https://www.microsoft.com/en-us/evalcenter/sql-server-2025-download)

[2] Run SQL Server Linux container images with Docker, <https://learn.microsoft.com/en-us/sql/linux/quickstart-install-connect-docker?view=sql-server-ver17&tabs=cli&pivots=cs1-bash>

[3] Install SQL Server Management Studio, <https://learn.microsoft.com/en-us/ssms/install/install>

[4] AdventureWorks sample databases, <https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver17&tabs=ssms>

[5] DiskANN: Vector Search for Web Scale Search and Recommendation, [www.microsoft.com/en-us/research/project/project-akupara-approximate-nearest-neighbor-search-for-large-scale-semantic-search/](https://www.microsoft.com/en-us/research/project/project-akupara-approximate-nearest-neighbor-search-for-large-scale-semantic-search/)

[6] Wikipedia, Levenshtein distance, [https://en.wikipedia.org/wiki/Levenshtein\\_distance](https://en.wikipedia.org/wiki/Levenshtein_distance)

[7] Wikipedia, Jaro-Winkler distance, [https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler\\_distance](https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance)

**Thorsten Kansy**

arbeitet seit mehr als 35 Jahren als unabhängiger Softwarearchitekt, Entwickler und Trainer. Er entwickelt komplexe .NET-Anwendungen im Microsoft-Umfeld, insbesondere SQL Server. Sie erreichen ihn auf X mit @tkansy sowie unter [tkansy@dotnetconsulting.eu](mailto:tkansy@dotnetconsulting.eu).

# DWX ACADEMY

Hands-on Workshops  
und Weiterbildung für  
Software-Entwickler  
und -Architekten

KÜNSTLICHE  
INTELLIGENZ  
(AI)

## Wie integriere ich Künstliche Intelligenz in meinen Coding-Alltag

ChatGPT als Unterstützung für Entwickler	(2 Tage)
KI-System-Design und Code-Generation mit Generativen Großen Sprachmodellen (ChatGPT & Co.)	(2 Tage)
Einführung in Neuronale Netze und Deep Learning	(2 Tage)
Mastering Git Hub Copilot – Prompt Engineering & Einblick in zukünftige Entwicklungen	(1 Tag)
Einführung in Azure Open AI – Von Daten integration bis zur Entwicklung eigener KI-Agenten	(3 Tage)
Einstieg in Python	(3 Tage)
KI-Grundlagen für Product Owner	(1 Tag)

**REMOTE TRAININGS MÖGLICH!**



Weitere Informationen auf [developer-world.de/dwx-academy](https://developer-world.de/dwx-academy)  
Preise auf Anfrage!

Deine Ansprechpartnerin: Susanne Herl • +49 731 88005-8835 • [susanne.herl@ebnermedia.de](mailto:susanne.herl@ebnermedia.de)





## POSTGRESQL-INSTALLATIONEN VERWALTEN

# Backup und Restore

Wie man PostgreSQL-Installationen sichert und nach einem Crash wiederherstellt.

Aufgrund geringerer Kosten sowie höherer Flexibilität entscheiden sich immer mehr Unternehmen – auch große – für einen Umstieg von SQL Server auf PostgreSQL. In einer losen Folge von Artikeln betrachten wir die Details von PostgreSQL und benennen die wichtigsten Unterschiede zum SQL Server. Zuletzt ging es um das Thema Concurrency in PostgreSQL [1]. Gezeigt wurde, wie mehrere Transaktionen gleichzeitig durch Zuhilfenahme von Multi-Version Concurrency Control ohne den Einsatz von Sperren verarbeitet werden können.

Dieser Beitrag gewährt Einblick in das Management von PostgreSQL-Installationen, zeigt, wie das Write-Ahead Logging (WAL) funktioniert, und beschreibt im Detail, wie Sie Backup- und Restore-Operationen durchführen, um im Fehlerfall die Datenbanken wiederherstellen zu können.

## Write-Ahead Logging

Ein fundamentales Konzept moderner Datenbanksysteme ist das Write-Ahead Logging (WAL). Es sichert die Datenintegrität und unterstützt Recovery-Prozesse nach Systemabstürzen. WAL in PostgreSQL ist zugleich leistungsoptimiert und robust.

Das Grundprinzip von WAL besteht darin, dass Änderungen an Daten in ein sequenzielles Log geschrieben werden, bevor die Änderungen auf die Pages der Datenbank angewendet werden. Dieses Log dient als transaktionssicheres Journal und gewährleistet, dass der Datenbankserver nach einem Absturz in einen konsistenten Zustand zurückversetzt werden kann (Crash Recovery).

Anders als beim SQL Server hat PostgreSQL nur ein einziges Write-Ahead Log für den kompletten Cluster. Der SQL Server verwaltet dagegen ein eigenes Transaction Log für jede Datenbank, wodurch sich einzelne Datenbanken unabhängig voneinander wiederherstellen lassen. In PostgreSQL muss dagegen immer der komplette Cluster mit sämtlichen Datenbanken wiedergestellt werden.

PostgreSQL speichert diese Informationen in WAL-Dateien im Verzeichnis `pg_wal` (innerhalb von `$PGDATA`). Die Dateien enthalten alle Änderungen, die durch Datenmanipulations-Operationen (`INSERT`, `UPDATE`, `DELETE`) sowie DDL-Operationen (etwa `CREATE TABLE`) generiert wurden. Wie Sie in **Bild 1** erkennen, besteht das Verzeichnis `pg_wal` aus

```
postgres@27f6a2ae6475:~/data/pg_wal$ ls -l
total 622600
-rw----- 1 postgres postgres 16777216 Apr 23 16:11 000000010000001800000007
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 000000010000001800000008
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 000000010000001800000009
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 00000001000000180000000A
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 00000001000000180000000B
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 00000001000000180000000C
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 00000001000000180000000D
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 00000001000000180000000E
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 00000001000000180000000F
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 000000010000001800000010
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 000000010000001800000011
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 000000010000001800000012
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 000000010000001800000013
-rw----- 1 postgres postgres 16777216 Mar 8 16:34 000000010000001800000014
```

WAL-Segmente im Verzeichnis `pg_wal` (Bild 1)

Dateien, die eine Standardgröße von 16 MByte haben und als WAL-Segmente bezeichnet werden. Ein WAL-Segment ist dabei vergleichbar mit einem Virtual Log File (VLF) beim SQL Server.

Die Dateinamen der WAL-Segmente folgen einem festen Schema und enthalten folgende Informationen:

- **Timeline-ID:** Identifiziert die Timeline. Bei jeder Wiederherstellung oder einem Point-in-Time Recovery (PITR) mit Timeline-Wechsel erhält die neue Timeline eine fortlaufende ID.
- **Log-File Segment:** Dies ist der höhere Teil des WAL-Segments. Ein Log-File besteht aus vielen Segmenten. Der Wert wird hochgezählt, wenn der niedrigere Teil (Segmentnummer) einen Schwellenwert überschreitet.
- **Segmentnummer:** Dies ist die eigentliche Segmentnummer innerhalb des Log-Files. Jedes Segment hat eine Standardgröße von 16 MByte. Ist das aktuelle Segment voll, wird das nächste erzeugt.

Wie bereits zuvor erwähnt, werden mithilfe des Write-Ahead Loggings alle physischen Änderungen mitprotokolliert, die Sie durch DML- und DDL-Operationen in der Datenbank durchführen.

Ein zentrales Konzept ist hierbei die Log Sequence Number (LSN), die in PostgreSQL mit einem fortlaufenden Wert dargestellt wird. Die LSN wird benötigt, um ein Crash Recovery im Fall eines Absturzes durchführen zu können. Ändert eine Transaktion Daten, schreibt die Datenbank die Änderungen zuerst in das WAL, bevor sie die Änderungen auf den eigentlichen Datenseiten vornimmt.

Sobald eine Änderung auf einer Page wirksam wird, wird die Log Sequence Number dieser Änderung auch auf der



Page gespeichert – typischerweise im Page Header. Die letzte auf der Page angewendete LSN erfüllt mehrere wichtige Funktionen:

- Effizientes Crash Recovery, da ein *Redo*-Befehl nur die Operationen durchführen muss, die auf eine Page noch nicht angewendet wurden.
- Vermeidung von doppelten Änderungen, da anhand der Log Sequence Number überprüft werden kann, ob eine Transaktion bereits angewendet wurde.
- Unterstützung des *CHECKPOINT*-Prozesses, da bei einem Crash-Recovery-Prozess nur diejenigen Transaktionen nochmals durchgeführt werden müssen, die nach dem *CHECKPOINT*-Prozess erfolgt sind.

```

11 -- Retrieve the current insert LSN (Log Sequence Number)
12 -- "WAL Segment Number/Byte Offset within the WAL Segment"
13 -- 18/074387C8
14 -- => 18: WAL Segment
15 -- => 07: WAL Segment
16 -- => 4387C8: Offset within the file
17
18 -- WAL file name: 000000010000001800000007
19 -- => 00000001: Timeline (used for recovery)
20 -- => 00000018: WAL Segment
21 -- => 00000007: WAL Segment
22 SELECT pg_current_wal_insert_lsn();
23

```

Results 1 x

SELECT pg\_current\_wal\_insert\_lsn()

pg_current_wal_insert_lsn
18/74387C8

Aktuelle Log Sequence Number 18/074387C8 (Bild 2)

Über die SQL-Funktion `pg_current_wal_insert_lsn()` können Sie sich die aktuelle LSN zurückliefern lassen, an deren Position die nächsten WAL-Records geschrieben werden. Mein Test lieferte die LSN **18/74387C8** (Bild 2), die sich wie folgt aufschlüsseln lässt:

- 18: WAL-Segment (höherer Teil),
- 07: WAL-Segment (niedrigerer Teil),
- 4387C8: Offset innerhalb der WAL-Segment-Datei.

Aus diesen Informationen ergibt sich auch der Dateiname des aktuellen WAL-Segments: `000000010000001800000007`. Es handelt sich dabei um die erste Datei aus dem Verzeichnis `pg_wal` in Bild 1.

Mithilfe der SQL-Funktion `pg_walfile_name_offset()` können Sie anhand einer Log Sequence Number auch den Dateinamen und den Offset innerhalb der Datei ermitteln:

```

-- Returns the current insert LSN
SELECT pg_current_wal_insert_lsn();

-- Decode the file_name and the file_offset
-- from the insert LSN
SELECT
 file_name,
 UPPER(TO_HEX(file_offset)) file_offset
FROM pg_walfile_name_offset('18/74387C8');
```

```

-- Retrieve all the WAL segment files from
-- the file system
SELECT * FROM pg_ls_waldir()
ORDER BY name;
```

Wie Sie anhand des Codes erkennen können, erlaubt die Funktion `pg_ls_waldir()` es, den aktuellen Inhalt des Verzeichnisses `pg_wal` auszugeben (Bild 3).

PostgreSQL bietet über die Funktion `page_header()` der Extension `pageinspect` die Möglichkeit an, den Header einer Page auszulesen. Zurückgeliefert wird dabei die Log Sequence Number, welche die Page zuletzt in der Spalte `lsn` geändert hat. Hier ein Beispiel dazu:

```

-- Create a new table
CREATE TABLE TestTable
(
 Col1 INT,
 Col2 CHAR(2000)
);

-- Insert a new record
INSERT INTO TestTable VALUES(1, 'A');

-- Retrieve the current insert LSN.
SELECT pg_current_wal_insert_lsn();
```

```

30 -- Retrieve all the WAL segment files from the file system
31 SELECT * FROM pg_ls_waldir()
32 ORDER BY name;
33

```

Results 1 x

SELECT \* FROM pg\_ls\_waldir() ORDER BY name

	name	size	modification
1	000000010000001800000007	16,777,216	2025-04-23 18:14:41.000 +0200
2	000000010000001800000008	16,777,216	2025-03-08 17:34:15.000 +0100
3	000000010000001800000009	16,777,216	2025-03-08 17:34:16.000 +0100
4	00000001000000180000000A	16,777,216	2025-03-08 17:34:15.000 +0100
5	00000001000000180000000B	16,777,216	2025-03-08 17:34:16.000 +0100
6	00000001000000180000000C	16,777,216	2025-03-08 17:34:16.000 +0100

Inhalt von `pg_wal`, ausgegeben per `pg_ls_waldir()` (Bild 3)

```

-- Retrieve the LSN that changed the page
-- the last time.
SELECT lsn FROM page_header(
 get_raw_page('TestTable', 0));

-- Update the record
UPDATE TestTable SET Col2 = 'B';

-- The insert LSN was advanced again
SELECT pg_current_wal_insert_lsn();

-- Retrieve the LSN that changed the page
-- the last time.
```

```

postgres@27f6a2ae6475:~/data/pg_wal$ pg_waldump -s 18/7457268 -e 18/7458AD8
rmgr: Heap len (rec/tot): 54/ 6186, tx: 1292217, lsn: 18/07457268, prev 18/07457230, desc: INSERT off: 3, flags: 0x00, blkref #0: fel 1663/61925/61976 blk 0 FFW
rmgr: Transaction len (rec/tot): 34/ 34, tx: 1292217, lsn: 18/07458AD8, prev 18/07457268, desc: COMMIT 2025-04-24 11:21:14.899350 UTC
postgres@27f6a2ae6475:~/data/pg_wal$

```

Ausgabe des Aufrufs von pg\_waldump (Bild 4)

```

SELECT lsn FROM page_header(
 get_raw_page('TestTable', 0));

```

Im Gegensatz zum SQL Server können Sie in PostgreSQL die einzelnen Transaction Log Records aus dem WAL auch in einer dokumentierten Art und Weise auslesen. Das klappt mit dem Kommandozeilen-Tool *pg\_waldump*, dem Sie als Parameter eine Start- und End-LSN übergeben können, etwa so:

```

-- Insert another record
-- start_lsn: 18/7457268
-- end_lsn: 18/7458AD8
-- Use the pg_waldump utility to have a look on
-- the generated transaction log records:
-- pg_waldump -s start_lsn -e end_lsn
SELECT pg_current_wal_insert_lsn();
INSERT INTO TestTable VALUES(1, 'B');
SELECT pg_current_wal_insert_lsn();

-- The "blkref" information in the "INSERT" log
-- record is the same as the file path of the
-- relation
SELECT pg_relation_filepath('TestTable');

```

In meinem Fall war die Log Sequence Number vor dem *INSERT*-Statement *18/7457268* und danach *18/7458AD8*. Daher

wurde das Tool *pg\_waldump* mit den folgenden Parametern aufgerufen:

```
pg_waldump -s 18/7457268 -e 18/7458AD8
```

Wie Bild 4 zeigt, wurden durch das *INSERT*-Statement zwei Transaction Log Records generiert:

- Ein Log Record für das *INSERT*-Statement sowie
- ein Log Record für das *COMMIT* der implizit gestarteten Transaktion.

Über die Information *rmgr* sehen Sie ebenfalls, welcher Resource Manager diesen Log Record erzeugt hat. PostgreSQL unterstützt hierbei unterschiedlichste Resource Manager für Datenstrukturen und Indizes. In Fall eines *INSERT*-Statements in eine Heap Table ist der Resource Manager *Heap* verantwortlich.

Über die Information *rel* auf der rechten Seite sehen Sie die OID der Tabelle (*Relation*), in die dieses *INSERT*-Statement durchgeführt wurde. Sie zeigt die gleiche OID an, die die Funktion *pg\_relation\_filepath()* für die Tabelle *TestTable* zurückliefert. Die Information *blk* zeigt die Block Number (beziehungsweise Page ID) an, auf die dieses *INSERT*-Statement durchgeführt wurde.

Wie im SQL Server kann auf Basis dieser Log Records ein erfolgreiches Crash Recovery der Datenbanken eines Post-

```

GNU nano 7.2 adventureworks_backup.sql *
-- Data for Name: employee; Type: TABLE DATA; Schema: humanresources; Owner: postgres

COPY humanresources.employee (businessentityid, nationalidnumber, loginid, jobtitle, birthdate, maritalstatus, gender, hiredate, salaryflag, vacationhours, sickleavehours, currentflag, rowguid, mo
1 295847284 adventure-works\ken0 Chief Executive Officer 1969-01-29 S M 2009-01-14 t 99 69 t f01251e5-96a3-448d-981e-0f9d789110d 2014-0
2 245797967 adventure-works\terri0 Vice President of Engineering 1971-08-01 S F 2008-01-31 t 1 20 t 45e8f437-670d-4409-93cb-f9424a40d6ee 2014-0
3 509647174 adventure-works\roberto0 Engineering Manager 1974-11-12 M M 2007-11-11 t 2 21 t 9bbbfb2c-efbb-4217-9ab7-f97689328841 2014-0
4 112457891 adventure-works\rob0 Senior Tool Designer 1974-12-23 S M 2007-12-05 f 48 80 t 59747955-87b8-443f-8ed4-f8ad3afdf3a9 2014-0
5 609256980 adventure-works\gail0 Design Engineer 1952-09-22 M F 2008-01-06 t 5 22 t cc84ae00-f9b0-4a15-b4e0-6caba919008 2014-06-30 08:
6 998320692 adventure-works\josief0 Design Engineer 1959-03-11 M M 2008-01-24 t 6 23 t e390956f1-9cd5-478d-8945-14aca7fbdcdd 2014-0
7 134969118 adventure-works\ydylan0 Research and Development Manager 1987-02-24 M M 2009-02-08 t 61 50 t 4f46deca-ef01-41fd-9829-0adab3 2014-0
8 811994146 adventure-works\ldaniel0 Research and Development Engineer 1986-06-05 S F 2008-12-29 t 62 51 t 31112635-663b-4018-b4a2-a6850c 2014-0
9 658797993 adventure-works\vgigi0 Research and Development Engineer 1979-01-21 M F 2009-01-16 t 63 51 t 50b6cdc6-7570-47ef-9570-48a64b 2014-0
10 879342154 adventure-works\michael6 Research and Development Manager 1984-11-30 M M 2009-05-03 t 16 64 t eaa43680-5571-40cb-ab1
11 974026993 adventure-works\ovidiu0 Senior Tool Designer 1978-01-17 S M 2010-12-05 f 7 23 t 7687c19-fac1-439c-9bb7-ac33f3cc341c3
12 408168528 adventure-works\thierry0 Tool Designer 1959-07-29 M M 2007-12-11 f 9 24 t 14955171-ea773-4fad-8302-48fd989d8ddd 2014-0
13 486228782 adventure-works\janice0 Tool Designer 1989-05-18 M F 2010-12-23 f 8 24 t 954b91b6-5a87-48c2-8685-6e11c6e549ba 2014-0
14 42487730 adventure-works\michael8 Senior Design Engineer 1979-06-16 S M 2010-12-30 t 3 21 t 46286cad-46dd-4d9b-9128-85b67e98d1a9 2014-0
15 56920285 adventure-works\sharon0 Design Engineer 1961-05-02 M F 2011-01-18 t 4 22 t 54f2fcd0-87c4-4065-a7a8-9ac8ea624235 2014-0
16 24756624 adventure-works\david0 Marketing Manager 1975-03-19 S M 2007-12-20 t 40 40 t e87029aa-2cba-c0c3-b948-d83af0313e28 2014-0
17 253022876 adventure-works\kevin0 Marketing Assistant 1987-05-03 S M 2007-12-16 f 42 41 t 1b480240-95c0-410f-a717-eb29943c8886 2014-0
18 222969461 adventure-works\john5 Marketing Specialist 1978-03-06 S M 2011-02-07 f 48 44 t 64730415-1f58-4e5b-8fab-5e4daeb53b4 2014-0
19 52841310 adventure-works\mary2 Marketing Assistant 1978-01-29 S F 2011-02-14 f 43 41 t 1f6da901-c7f7-48a8-8eef-d81868d72b52 2014-0
20 323403273 adventure-works\wandil0 Marketing Assistant 1975-03-17 M F 2011-01-07 f 41 40 t 43cc446-d41c-454c-8530-973ad2923a1b 2014-0
21 243322160 adventure-works\terry8 Marketing Specialist 1986-02-04 M M 2009-03-02 f 44 42 t ad0d7626-4506-45c4-898f-9d7a0e13aabd 2014-0
22 95958330 adventure-works\usariya0 Marketing Specialist 1987-05-21 S M 2008-12-12 f 45 42 t af21f1b7-0691-48ad-b325-b8f2d7b2268a 2014-0
23 767953655 adventure-works\mary0 Marketing Specialist 1962-09-13 M F 2009-01-12 f 46 43 t 90899830-127e-4876-85ca-7f6e03025281 2014-0
24 72636981 adventure-works\vjill0 Marketing Specialist 1979-06-18 M F 2009-01-18 f 47 43 t 53575c58-fba1-4c1a-b8ff-95e850b44393 2014-0
25 519899984 adventure-works\james1 Vice President of Production 1983-01-07 S M 2009-02-03 t 64 52 t d7314f24-2af1-429c-9bb7-4038f45f36c
26 277173473 adventure-works\peter8 Production Control Manager 1982-11-03 M M 2008-12-01 t 43 41 t 69d5d162-e817-45e7-9dec-5d9b8310e701 2014-0
27 446466105 adventure-works\vio0 Production Supervisor - WC60 1956-10-08 S F 2008-02-27 f 20 60 t 831faac6-b095-481f-b897-14f6e64d6d47 2014-0
28 14417807 adventure-works\vguy1 Production Technician - WC60 1988-03-13 M M 2006-06-30 f 21 38 t aae1004a-c237-4974-b4d5-935247737718 2014-0
29 948320468 adventure-works\mark1 Production Technician - WC60 1979-09-25 S M 2009-01-23 f 19 29 t cdb2c6df-93e5-4558-91c8-424d5e7c658b 2014-0
30 410742000 adventure-works\lbrittan0 Production Technician - WC60 1989-09-28 M F 2009-01-29 f 14 27 t fc12677f-42f8-4790-ad69-685088 2014-0
31 750246141 adventure-works\margie0 Production Technician - WC60 1986-05-20 M F 2009-01-04 f 18 29 t 352d6e2f-655b-49a2-9fec-efd7d8 2014-0
32 330211482 adventure-works\rebecca0 Production Technician - WC60 1977-07-10 M F 2008-12-29 f 23 31 t bdade5c5-a596-4da1-88ba-842c7a 2014-0
33 801758062 adventure-works\annk0 Production Technician - WC60 1976-12-26 M M 2008-12-17 f 17 28 t 51fde6ac-bd50-467c-90e5-10ca7218459c 2014-0
34 254372876 adventure-works\schittan0 Production Technician - WC60 1987-06-10 M F 2009-02-16 f 15 27 t 6a0b932-b4a5-4f5c-a167-9a3a53 2014-0
35 999440576 adventure-works\brandon0 Production Technician - WC60 1987-07-10 M M 2009-02-08 f 22 31 t casd1935-44e2-4c45-93bb-809e59 2014-0
36 788456780 adventure-works\jose0 Production Technician - WC60 1984-07-31 S M 2009-02-10 f 20 30 t 9e912556-88ba-41ee-b946-c84ab4c1102 2014-0
37 442121106 adventure-works\chris0 Production Technician - WC60 1986-08-07 S M 2009-03-07 f 16 28 t 77b384c7-affb-4816-992b-f5b3034ae4ff 2014-0
38 6298838 adventure-works\kimi1 Production Technician - WC60 1966-12-14 M F 2010-01-16 f 24 32 t 891c27f4-28cc-44bb-bea3-da5ea7f8e957 2014-0
39 2014-06-30 00:00:00 /3/1/1/12/
40 389738752 adventure-works\jloynn0 Production Supervisor - WC60 1956-01-16 S F 2007-12-26 f 82 61 t 2cc71b96-f421-485e-9832-872333 2014-0
41 458159238 adventure-works\lbryan0 Production Technician - WC60 1973-08-27 S M 2009-01-21 f 35 37 t 4652d4ed-6a27-47ed-9f51-c37e293da28 2014-0

```

Von pg\_dump erzeugtes Logical Backup (Bild 5)

greSQL-Clusters durchgeführt werden, wenn es zu einem unerwarteten Absturz der Datenbank-Engine gekommen sein sollte.

## Logical Backups

Im Vergleich zum SQL Server sind Backups bei PostgreSQL eher komplizierter und aufwendiger durchzuführen, da man hier sehr viele Schritte manuell erledigen muss, die der SQL Server automatisiert durchführt. Zunächst werden hier Logical Backups erklärt, und danach folgen Physical Backups sowie das Point-in-Time Recovery.

Bei einem Logical Backup wird in PostgreSQL aus einer Datenbank eine normale SQL-Skript-Datei erzeugt, die neben den kompletten Datenbankobjektdefinitionen (Tabellen, Indizes, Views et cetera) auch die Daten für die einzelnen Datensätze enthält. Sie können sich diese Datei wie einen kompletten Dump einer Datenbank vorstellen.

Durchgeführt wird ein Logical Backup über das Kommandozeilen-Tool *pg\_dump*. Die Daten werden so in die SQL-Skript-Datei geschrieben, dass sie später mithilfe des Kommandos *psql COPY* wiederhergestellt werden können:

```
pg_dump
-d adventureworks
-f adventureworks_backup.sql
```

**Bild 5** zeigt einen Ausschnitt der von *pg\_dump* generierten SQL-Skript-Datei mit den Datendefinitionen.

Bevorzugen Sie anstelle des Kommandos *psql COPY* traditionelle *INSERT*-Statements für die Datenwiederherstellung, können Sie die Option *-column-inserts* angeben – **Bild 6** zeigt einen Ausschnitt der Backup-Datei:

```
pg_dump
-d adventureworks
--column-inserts
-f adventureworks_backup.sql
```

Da die generierten SQL-Skript-Dateien sehr groß werden können, lässt sich ein Logical Backup mithilfe des Parameters *-F*, der das gewünschte Format spezifiziert, auch in eine komprimierte Binärdatei durchführen. Folgende Optionen gibt es dafür:

- *c*: Custom-Binary-Format,
- *d*: Directory-Format,
- *t*: tar-Format,
- *p*: Plain Text (Standard).

Der folgende Befehl zeigt, wie Sie ein Logical Backup in einem Custom-Binary-Format durchführen:

```
pg_dump
-d adventureworks
-Fc
-f adventureworks.backup
```

Wiederherstellen lässt sich eine gesicherte Datenbank, indem Sie die SQL-Statements der generierten SQL-Skript-Datei ausführen.

Haben Sie jedoch ein Custom-Binary-Format verwendet, bietet PostgreSQL das Kommandozeilen-Tool *pg\_restore* für die Wiederherstellung an:

```
pg_restore
-C
```

**Logical Backup mit Option *-column-inserts* (Bild 6)**



```
-d postgres
adventureworks.backup
```

Über die Option `--list` bietet `pg_restore` zudem an, sich das Inhaltsverzeichnis der generierten Binärdatei anzusehen (Bild 7):

```
pg_restore --list adventureworks.backup
```

Backups und Wiederherstellungen über `pg_dump` und `pg_restore` sind ideal, wenn Sie einen PostgreSQL-Cluster von einer Version auf eine andere Version migrieren möchten, da Sie hier nur mit SQL-Skript-Dateien arbeiten, die zwischen den Versionen immer kompatibel sein werden.

Zudem haben Sie mit diesem Ansatz die Möglichkeit, nur einzelne Datenbankobjekte (etwa eine Tabelle) wiederherzustellen. Das bietet der SQL Server im Standard nicht an.

## Physical Backups

Ein Physical Backup ist in PostgreSQL eine Grundvoraussetzung dafür, dass Sie ein Point-in-Time Recovery von Datenbanken durchführen können. Dabei handelt es sich um nichts anderes als eine exakte und komplette Kopie des Verzeichnisses `$PGDATA`. Ein Physical Backup des Verzeichnisses `$PGDATA` wird in PostgreSQL auch als Base Backup bezeichnet.

Vorstellen können Sie sich ein Base Backup wie ein vollständiges Datenbankbackup im SQL Server, nur mit den Unterschieden, dass Sie den kompletten PostgreSQL-Cluster sichern und dass das Base Backup keine Transaction Log Re-

```
postgres@27f6a2ae6475:~/data$ pg_restore --list adventureworks.backup
;
; Archive created at 2025-04-24 12:26:57 UTC
; dbname: adventureworks
; TOC Entries: 998
; Compression: gzip
; Dump Version: 1.16-0
; Format: CUSTOM
; Integer: 4 bytes
; Offset: 8 bytes
; Dumped from database version: 17.2 (Debian 17.2-1.pgdg120+1)
; Dumped by pg_dump version: 17.2 (Debian 17.2-1.pgdg120+1)
;
; Selected TOC Entries:
;
10; 2615 43488 SCHEMA - hr postgres
11; 2615 43489 SCHEMA - humanresources postgres
4829; 0 0 COMMENT - SCHEMA humanresources postgres
12; 2615 43490 SCHEMA - pe postgres
13; 2615 43491 SCHEMA - person postgres
4830; 0 0 COMMENT - SCHEMA person postgres
14; 2615 43492 SCHEMA - pr postgres
15; 2615 43493 SCHEMA - production postgres
4831; 0 0 COMMENT - SCHEMA production postgres
16; 2615 43494 SCHEMA - pu postgres
17; 2615 43495 SCHEMA - purchasing postgres
4832; 0 0 COMMENT - SCHEMA purchasing postgres
18; 2615 43496 SCHEMA - sa postgres
19; 2615 43497 SCHEMA - sales postgres
4833; 0 0 COMMENT - SCHEMA sales postgres
2; 3079 43498 EXTENSION - pageinspect
4834; 0 0 COMMENT - EXTENSION pageinspect
3; 3079 43543 EXTENSION - pg_freeze
4835; 0 0 COMMENT - EXTENSION pg_freeze
4; 3079 43546 EXTENSION - tablefunc
4836; 0 0 COMMENT - EXTENSION tablefunc
5; 3079 43567 EXTENSION - uuid-osp
4837; 0 0 COMMENT - EXTENSION "uuid-osp"
1117; 1247 43579 DOMAIN public AccountNumber postgres
1120; 1247 43581 DOMAIN public Flag postgres
1123; 1247 43584 DOMAIN public Name postgres
1126; 1247 43586 DOMAIN public NameStyle postgres
1129; 1247 43589 DOMAIN public OrderNumber postgres
1132; 1247 43591 DOMAIN public Phone postgres
234; 1259 43592 TABLE humanresources department postgres
4838; 0 0 COMMENT humanresources TABLE department postgres
4839; 0 0 COMMENT humanresources COLUMN department.departmentid postgres
4840; 0 0 COMMENT humanresources COLUMN department.name postgres
4841; 0 0 COMMENT humanresources COLUMN department.groupname postgres
```

Inhaltsverzeichnis der generierten Binärdatei (Bild 7)

```
postgres@b433f0987222:~$ pg_basebackup -D backup_archive -P -l "PostgreSQL Base Backup" --checkpoint=fast
30649/30649 kB (100%), 1/1 tablespace
postgres@b433f0987222:~$ cd backup_archive
postgres@b433f0987222:~/backup_archive$ ls -l
total 304
-rw-r--r-- 1 postgres postgres 222 Apr 24 15:06 backup_label
-rw-r--r-- 1 postgres postgres 181427 Apr 24 15:06 backup_manifest
drwx-r--r-- 6 postgres postgres 4096 Apr 24 15:06 base
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 global
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_commit_ts
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_dynshmem
-rw-r--r-- 1 postgres postgres 5743 Apr 24 15:06 pg_hba.conf
-rw-r--r-- 1 postgres postgres 2640 Apr 24 15:06 pg_ident.conf
drwx-r--r-- 4 postgres postgres 4096 Apr 24 15:06 pg_logical
drwx-r--r-- 4 postgres postgres 4096 Apr 24 15:06 pg_multixact
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_notify
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_replslot
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_serial
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_snapshots
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_stat
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_stat_tmp
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_subtrans
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_tblspc
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_twophase
-rw-r--r-- 1 postgres postgres 3 Apr 24 15:06 PG_VERSION
drwx-r--r-- 4 postgres postgres 4096 Apr 24 15:06 pg_wal
drwx-r--r-- 2 postgres postgres 4096 Apr 24 15:06 pg_xact
-rw-r--r-- 1 postgres postgres 88 Apr 24 15:06 postgresql.auto.conf
-rw-r--r-- 1 postgres postgres 30777 Apr 24 15:06 postgresql.conf
postgres@b433f0987222:~/backup_archive$
```

Backup-Archiv nach dem Befehl `pg_basebackup` (Bild 8)

cords für die Transaktionen beinhaltet, die während des Backups durchgeführt wurden. Diese müssen nämlich in PostgreSQL separat gesichert werden.

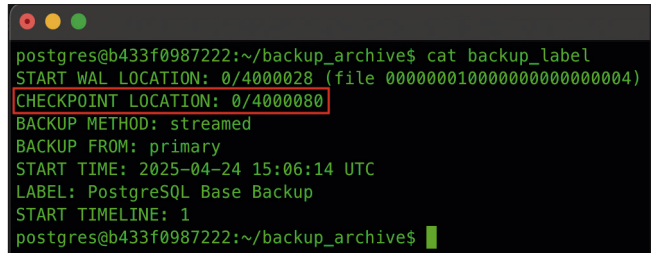
Erstellt wird ein Base Backup in PostgreSQL mit dem Kommandozeilen-Tool `pg_basebackup`. Standard ist, dass `pg_basebackup` wartet, bis der nächste Checkpoint-Prozess gelaufen ist. Über die Option `--checkpoint=fast` lässt sich aber auch ein sofortiger Checkpoint-Prozess erzwingen:

```
pg_basebackup
-D backup_archive
-P
-l "PostgreSQL Base Backup"
--checkpoint=fast
```

Wie Sie in **Bild 8** sehen, enthält das Verzeichnis `backup_archive` nach dem Aufruf von `pg_basebackup` eine Eins-zu-eins-Kopie des Verzeichnisses `$PGDATA`.

Das Backup erzeugt darüber hinaus zwei weitere wichtige Dateien: `backup_label` und `backup_manifest`. Die Datei `backup_label` enthält weitere Informationen über das durchgeführte Backup, wie etwa die Log Sequence Number des letzten Checkpoint-Prozesses (**Bild 9**). Die JSON-Datei `backup_manifest` enthält Metadaten über das durchgeführte Backup und kann zur Verifikation des Backups verwendet werden.

Damit Sie nun ein Point-in-Time Recovery eines PostgreSQL-Clusters durchführen können, müssen Sie den Inhalt des Verzeichnisses `pg_wal` ebenfalls in regelmäßigen Abständen sichern – analog zu regelmäßigen Transaction Log Backups im SQL Server.



```
postgres@b433f0987222:~/backup_archive$ cat backup_label
START WAL LOCATION: 0/4000028 (file 000000010000000000000004)
CHECKPOINT LOCATION: 0/4000080
BACKUP METHOD: streamed
BACKUP FROM: primary
START TIME: 2025-04-24 15:06:14 UTC
LABEL: PostgreSQL Base Backup
START TIMELINE: 1
postgres@b433f0987222:~/backup_archive$
```

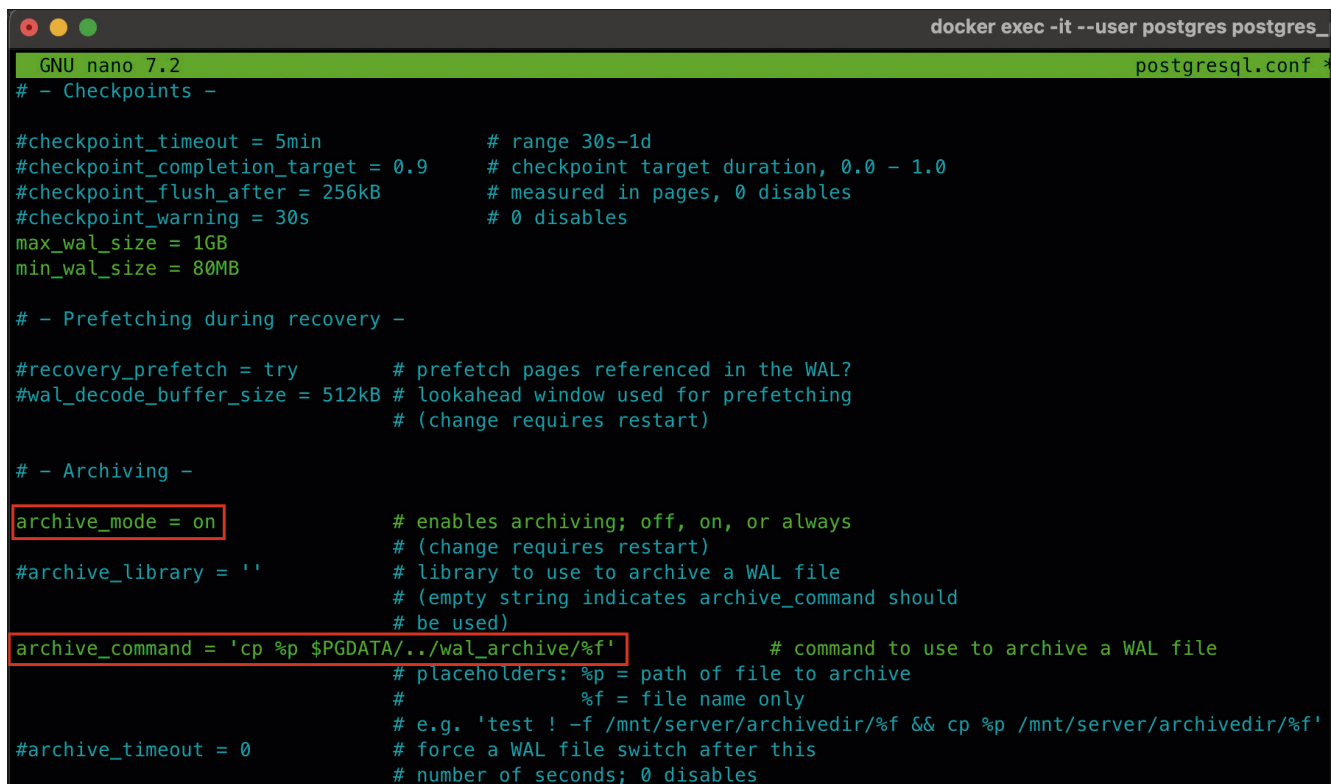
Die Datei `backup_label` (**Bild 9**)

PostgreSQL bietet hierzu den WAL-Archiving-Prozess an, den Sie über die Datei `postgresql.conf` konfigurieren können. Wichtig sind die Einstellungen `archive_mode` und `archive_command`. Setzen Sie die Einstellung `archive_mode` auf `ON`, wird der WAL Archiver gestartet. Sobald eine WAL-Segment-Datei vollständig beschrieben wurde, wird der Befehl ausgeführt, den Sie bei der Konfigurationsoption `archive_command` hinterlegt haben.

Hierbei ist es komplett Ihnen überlassen, welche Befehle Sie ausführen. Sie können ein einfaches lokales File-Copy durch den Befehl `cp` durchführen, Sie können aber auch vollständig beschriebene WAL-Segment-Dateien in einen Cloud-Storage kopieren. Für die Konfigurationsoption `archive_command` stehen diese beiden Platzhalter zur Verfügung:

- `%p`: Pfad zur originalen WAL-Segment-Datei,
- `%f`: Dateiname der originalen WAL-Segment-Datei.

In **Bild 10** sehen Sie eine mögliche Konfiguration dieser Einstellungen. In diesem Beispiel wird die zu archivierende ►



```
GNU nano 7.2 postgresql.conf
- Checkpoints -

#checkpoint_timeout = 5min # range 30s-1d
#checkpoint_completion_target = 0.9 # checkpoint target duration, 0.0 - 1.0
#checkpoint_flush_after = 256kB # measured in pages, 0 disables
#checkpoint_warning = 30s # 0 disables
max_wal_size = 1GB
min_wal_size = 80MB

- Prefetching during recovery -

#recovery_prefetch = try # prefetch pages referenced in the WAL?
#wal_decode_buffer_size = 512kB # lookahead window used for prefetching
(change requires restart)

- Archiving -

archive_mode = on # enables archiving; off, on, or always
(change requires restart)
#archive_library = '' # library to use to archive a WAL file
(empty string indicates archive_command should
be used)
archive_command = 'cp %p $PGDATA/./wal_archive/%f' # command to use to archive a WAL file
placeholders: %p = path of file to archive
%f = file name only
e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
#archive_timeout = 0 # force a WAL file switch after this
number of seconds; 0 disables
```

`archive_mode` und `archive_command` in der Konfigurationsdatei (**Bild 10**)



```

docker exec -it --user postgres postgres_pitr /bin/bash
postgres@b433f0987222:~/wal_archive$ psql -c "SELECT pg_switch_wal();"
pg_switch_wal

0/6000078
(1 row)

postgres@b433f0987222:~/wal_archive$ psql -c "SELECT * FROM pg_stat_archiver;"
 archived_count | last_archived_wal | last_archived_time | failed_count | last_failed_wal | last_failed_time | stats_reset

2 | 000000010000000000000006 | 2025-04-24 15:36:44.988072+00 | 0 | | | 2025-04-24 14:59:35.789578+00
(1 row)

postgres@b433f0987222:~/wal_archive$ ls -l
total 32768
-rw----- 1 postgres postgres 16777216 Apr 24 15:36 000000010000000000000005
-rw----- 1 postgres postgres 16777216 Apr 24 15:36 000000010000000000000006
postgres@b433f0987222:~/wal_archive$

```

Die WAL-Segment-Datei wurde erfolgreich gewechselt (Bild 11)

WAL-Segment-Datei in das Verzeichnis `wal_archive` kopiert. Dabei ist es wichtig, dass PostgreSQL in diesem Verzeichnis auch schreibberechtigt ist. So setzen Sie die Berechtigungen:

```

cd $PGDATA
cd ..
mkdir -p wal_archive
chmod 700 wal_archive
chown postgres:postgres wal_archive

```

Damit die WAL-Archivierung gestartet wird, benötigen Sie noch einen Restart des PostgreSQL-Clusters. Um zu testen, ob

die WAL-Archivierung erfolgreich durchgeführt wird, können Sie mit `pg_switch_wal()` einfach die aktuelle WAL-Segment-Datei schließen. Dadurch wird zu einer neuen WAL-Segment-Datei gewechselt und die alte Datei anhand der Konfigurationsoption `archive_command` gesichert. Weitere Informationen über die WAL-Archivierung können Sie auch aus der System-Tabelle `pg_stat_archiver` auslesen. Bild 11 zeigt einen erfolgreichen Wechsel der aktiven WAL-Segment-Datei.

Möchten Sie ein Point-in-Time Recovery eines PostgreSQL-Clusters durchführen, müssen Sie zunächst den Zeitpunkt bestimmen, bis zu dem die Wiederherstellung erfolgen soll. Der aktuelle Zeitpunkt kann über das SQL-Statement `SE-`

```

root@7a2db9e11054: /postgres
GNU nano 7.2 postgresql.conf *
- Archive Recovery -

These are only used in recovery mode.

restore_command = 'cp $PGDATA/wal_archive/%f %p' # command to use to restore an archived WAL file
placeholders: %p = path of file to restore
%f = file name only
e.g. 'cp /mnt/server/archivedir/%f %p'
#archive_cleanup_command = '' # command to execute at every restartpoint
#recovery_end_command = '' # command to execute at completion of recovery

- Recovery Target -

Set these only when performing a targeted recovery.

#recovery_target = '' # 'immediate' to end recovery as soon as a
consistent state is reached
(change requires restart)
#recovery_target_name = '' # the named restore point to which recovery will proceed
(change requires restart)
recovery_target_time = '2025-04-24 15:58:39.321731+00' # the time stamp up to which recovery will proceed
(change requires restart)
#recovery_target_xid = '' # the transaction ID up to which recovery will proceed
(change requires restart)
#recovery_target_lsn = '' # the WAL LSN up to which recovery will proceed
(change requires restart)
#recovery_target_inclusive = on # Specifies whether to stop:
just after the specified recovery target (on)
just before the recovery target (off)
(change requires restart)
#recovery_target_timeline = 'latest' # 'current', 'latest', or timeline ID
(change requires restart)
#recovery_target_action = 'pause' # 'pause', 'promote', 'shutdown'
(change requires restart)

```

`restore_command` und `recovery_target_time` in der Konfigurationsdatei (Bild 12)

```

2025-04-24 16:34:57.242 UTC [1] LOG: starting PostgreSQL 17.2 (Debian 17.2-1.pgdg120+1) on aarch64-unknown-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
2025-04-24 16:34:57.243 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
2025-04-24 16:34:57.243 UTC [1] LOG: listening on IPv6 address "::", port 5432
2025-04-24 16:34:57.244 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2025-04-24 16:34:57.246 UTC [28] LOG: database system was interrupted; last known up at 2025-04-24 16:31:51 UTC
2025-04-24 16:34:57.246 UTC [28] LOG: creating missing WAL directory "pg_wal/archive_status"
2025-04-24 16:34:57.246 UTC [28] LOG: creating missing WAL directory "pg_wal/summaries"
cp: cannot stat '/var/lib/postgresql/data/wal_archive/00000002.history': No such file or directory
2025-04-24 16:34:57.315 UTC [28] LOG: starting backup recovery with redo LSN 0/3000028, checkpoint LSN 0/3000080, on timeline ID 1
2025-04-24 16:34:57.321 UTC [28] LOG: restored log file "00000001000000000000000000000003" from archive
2025-04-24 16:34:57.333 UTC [28] LOG: starting point-in-time recovery to 2025-04-24 16:32:02.651663+00
2025-04-24 16:34:57.335 UTC [28] LOG: redo starts at 0/3000028
2025-04-24 16:34:57.340 UTC [28] LOG: restored log file "00000001000000000000000000000004" from archive
cp: cannot stat '/var/lib/postgresql/data/wal_archive/00000001000000000000000000000005': No such file or directory
2025-04-24 16:34:57.354 UTC [28] LOG: completed backup recovery with redo LSN 0/3000028 and end LSN 0/3000120
2025-04-24 16:34:57.354 UTC [28] LOG: consistent recovery state reached at 0/3000120
2025-04-24 16:34:57.354 UTC [1] LOG: database system is ready to accept read-only connections
2025-04-24 16:34:57.354 UTC [28] LOG: recovery stopping before commit of transaction 757, time 2025-04-24 16:32:12.057191+00
2025-04-24 16:34:57.354 UTC [28] LOG: pausing at the end of recovery
2025-04-24 16:34:57.354 UTC [28] HINT: Execute pg_wal_replay.resume() to promote.

```

Das Point-in-Time Recovery wurde durchgeführt (Bild 13)

`LECT NOW()`; ermittelt werden. Um die Transaktionen seit der letzten WAL-Archivierung nicht zu verlieren, müssen Sie dafür Sorge tragen, dass die aktuelle WAL-Segment-Datei archiviert wird. Den Archivierungsprozess können Sie über `pg_switch_wal()` anstoßen – wie bereits beschrieben.

Dies entspricht einem Tail Log Backup einer Datenbank im SQL Server. Haben Sie keinen Zugriff mehr auf den PostgreSQL-Cluster, erleiden Sie hier einen Datenverlust, weil die Transaktionen seit der letzten WAL-Archivierung verloren gehen, da sie nicht gesichert wurden.

Im nächsten Schritt müssen Sie nun in einer neuen PostgreSQL-Installation (beziehungsweise in der bestehenden) das Verzeichnis `$PGDATA` löschen und aus dem Backup wiederherstellen. Wichtig ist hier, dass Sie den Inhalt des Verzeichnisses `pg_wal` vollständig löschen, da die WAL-Segment-Dateien durch das Point-in-Time Recovery wiederhergestellt werden.

Dafür müssen Sie die archivierten WAL-Segment-Dateien in der PostgreSQL-Installation verfügbar machen. Danach gilt es das Point-in-Time Recovery in der Datei `postgresql.conf` zu konfigurieren. Wichtig sind dabei folgende Optionen:

- `restore_command`,
- `recovery_target_time`.

Über die Option `restore_command` legen Sie fest, wie die WAL-Segment-Dateien wiederhergestellt werden, entweder durch ein einfaches File-Copy über `cp` oder durch das Laden archivierter WAL-Segment-Dateien aus einem Cloud-Speicher. Über die Option `recovery_target_time` definieren Sie, bis zu welchem Zeitpunkt das Point-in-Time Recovery durchgeführt werden soll (Bild 12).

Zusätzlich zu `recovery_target_time` stehen noch weitere Konfigurationsoptionen zur Verfügung:

- `recovery_target_name` konfiguriert ein Crash Recovery bis zum angegebenen Restore Point durch.
- `recovery_target_xid` plant ein Crash Recovery bis zur angegebenen Transaction ID durch.
- `recovery_target_lsn` konfiguriert ein Crash Recovery bis zur angegebenen Log Sequence Number.
- `recovery_target_timeline` plant ein Crash Recovery bis zur angegebenen Timeline ID.

Nachdem Sie diese Änderungen durchgeführt haben, müssen Sie innerhalb des Verzeichnisses `$PGDATA` noch die Da-

tei `recovery.signal` anlegen – die keinen Inhalt benötigt. Das bloße Vorhandensein der Datei teilt PostgreSQL mit, dass während des nächsten Starts ein Crash Recovery durchgeführt werden soll.

Sehen Sie sich nach dem Restart das Log von PostgreSQL an, werden Sie feststellen, dass das Point-in-Time Recovery durchgeführt und der PostgreSQL-Cluster bis zum angegebenen Zeitpunkt wiederhergestellt wurde (Bild 13).

Wie Sie gesehen haben, sind Backup und Restore bei PostgreSQL-Datenbanken um einiges komplizierter als beim SQL Server, da sehr viele Schritte manuell durchgeführt werden müssen. Auf der anderen Seite haben Sie aber bei PostgreSQL auch die volle Kontrolle über Backup- und Restore-Operationen und können währenddessen alle erdenklichen Kommandos ausführen. Überdies können Sie bei PostgreSQL für Backup und Restore auch Open-Source-Lösungen nutzen, welche auf die in PostgreSQL integrierten Möglichkeiten aufsetzen. Die bekanntesten sind `pgBackRest` [2] und `Barman` [3].

## Fazit

Dieser Artikel hat sich mit dem Management von PostgreSQL-Installationen beschäftigt. Im ersten Abschnitt ging es um das Write-Ahead Logging, das vergleichbar ist mit dem Transaction Log beim SQL Server. Der zweite Abschnitt hat das Vorgehen bei Backup- und Restore-Operationen erklärt, und Sie haben gelernt, wie Sie Point-in-Time-Recovery-Techniken einsetzen können, um Datenbanken bis zu einem bestimmten Zeitpunkt wiederherzustellen. ■

[1] Klaus Aschenbrenner, *Das Concurrency-Modell*, dotnetpro 8-9/2025, Seite 112 ff.

[2] `pgBackRest`, <https://pgbackrest.org>

[3] `Barman`, <https://pgbarman.org>



**Klaus Aschenbrenner**

berät Unternehmen in Europa beim Einsatz des Microsoft SQL Server und beschäftigt sich mit Windows-Programmierung und .NET. Zweimal zeichnete in Microsoft für sein Engagement als Microsoft MVP aus. Er ist zu erreichen über [www.SQLpassion.at](http://www.SQLpassion.at).

## TIME-OUT

# Wann ist die KI ein Mensch?

Ist Identität nur eine Nebenwirkung unseres neuronalen Netzes?



**E**s ist schon interessant, einmal darüber nachzudenken, warum wir uns die Frage, ob eine KI jemals so werden kann wie wir, überhaupt stellen – und mit welcher Selbstüberschätzung wir an das Thema herangehen. Ja, es geht um unseren Rang in der Pyramide, um Macht, Selbstverständnis und Kontrolle. Heikle Themen, und da erzählen wir uns gerne die Geschichte, dass es ganz viele Dinge gibt, die eine KI eben nicht kann und auch nie können wird, weil, ja weil ... ähm, weil wir eben ganz anders sind. Wir könnten das Thema jetzt damit beenden oder intellektuell redlicher versuchen, es zu begründen und zu erforschen. Womit wir schon bei der ersten menschlichen Eigenschaft wären, die uns von einer KI unterscheidet: der Faulheit. Wie Daniel Kahneman in seinem wegweisenden Buch [1] zeigt, neigen wir dazu, uns mit der ersten plausibel klingenden Antwort zu begnügen, denn alles andere kostet Energie, und solange man noch Beeren sammeln muss, um satt zu werden, ist das ein nicht zu vernachlässigender Faktor.

Mein eigenes Belohnungssystem, das Freude im Erkenntnisgewinn sieht, motiviert mich aber, genauer zu fragen. Nicht wie KI funktioniert und was sie einmal leisten kann, sondern was uns als Menschen von ihr unterscheidet und wer wir eigentlich sind.

## Wir sind leibhaftig

Da ist zum einen unsere Leiblichkeit: Wir leben in einem Körper, und unser Denken und Sein lässt sich nicht vom Fühlen, Spüren und Erleben trennen. Das muss auf Dauer aber kein unüberwindbarer Graben sein, denn mit entsprechender Sensorik lässt sich das in eine KI integrieren. Der zweite Punkt ist hingegen fundamentaler: Wir leben. Also jetzt noch. Später irgendwann nicht mehr. Wir werden sterben. Und danach sind wir tot. Für immer. Für immer! Kein Schalter kann umgelegt werden, um diesen Prozess umzukehren. Eine KI aber kann man ausschalten und wieder einschalten, und nichts hat sich verändert. Unsere neuronalen Netze „denken“ die Endlichkeit, die existenzielle Bedrohung unseres Seins immer mit. Das verändert unser Erleben und somit unsere Entscheidungen.

## Da ist immer ein bisschen Tod im Leben

Zurzeit geht mir eine Liedzeile wie ein Ohrwurm durch den Kopf: „Ich will nur, dass du tanzst zu diesem Lied, ich will nur dass du glücklich bist, Marie!“ [2]. Wenn wir das hören, spüren wir die Freude, aber auch die Traurigkeit, da es nur um einen kurzen Moment geht, der nach dem Tanz wieder vorbei ist. Und gerade deshalb ist der Tanz so wertvoll, so schön, so unwiederbringlich. Tanze – bevor der Abend zu Ende ist.

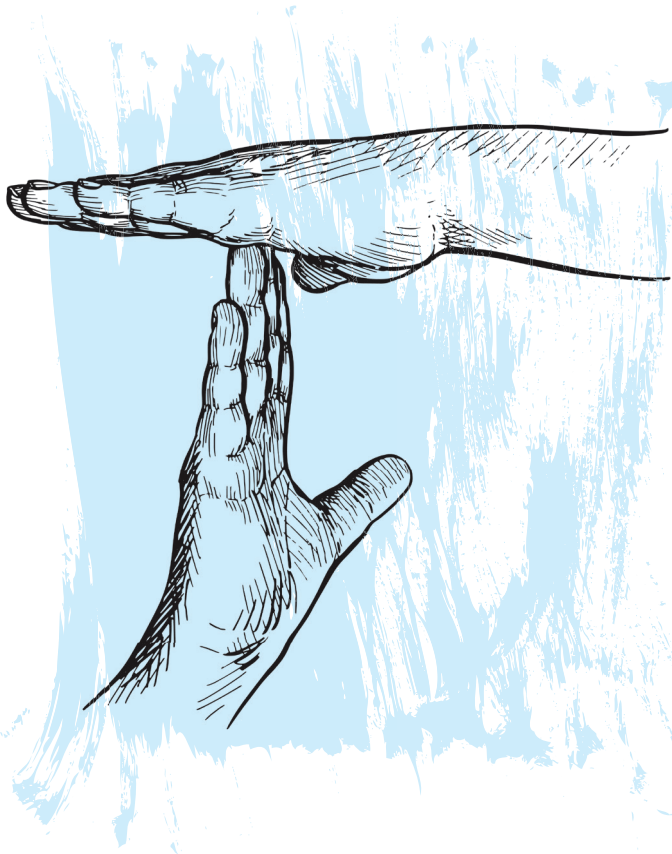
Eine KI wird das nicht nachempfinden können. Das scheint plausibel. Aber wird eine KI niemals einen Willen und ein Bewusstsein bekommen? Wird sie niemals Identität erfahren können? Und wenn wir uns da angeblich so sicher sind, worin liegt das begründet?

Ich bin mir da nicht mehr sicher, im Gegenteil! Je mehr ich mich mit neuronalen Netzen beschäftige, mit der Art, wie wir denken, fühlen und uns selbst wahrnehmen, desto mehr Zweifel kommen mir.

## Mensch, wir müssen reden!

Die Menschheitsgeschichte ist eine Erfolgsgeschichte. Auch wenn es traurig ist zu sehen, wie wir inzwischen mit diesem Planeten umgehen – im Lauf der Jahrtausende haben sich der Mensch und das menschliche Gehirn zu einem echten Schlager entwickelt: Wir können uns wie kein anderes Lebewesen auf sich ändernde Situationen einstellen, lernen schnell und lösen komplexe Probleme. Wir erfinden Werkzeuge, die unser Leben entschieden einfacher machen. Hinter alledem steckt als treibender Faktor unsere Geheimwaffe: das menschliche Gehirn. Dieses neuronale Netz ist anpassungsfähig und gleichzeitig neugierig. Das liegt daran, dass wir Muster erkennen können (immer wieder die gleichen Wege durch den Nervenpfad bilden mit der Zeit eine Autobahn), aber niemals darin verharren (die Wege wuchern immer wieder zu), und somit die Neugier uns zu neuen Ufern treibt.

Wir suchen den Ausgleich und das Gleichgewicht in einem System, das nie zur Ruhe kommt. Hunger und Durst müssen verhandeln gegen den Wunsch, weiter zu tanzen, Müdigkeit muss sich gegen die Neugier, wie der Roman weitergeht, durchsetzen. Wir werden getrieben, aber laufen auch nicht



ins offene Messer, weil es immer einen mächtigen Gegenspieler gegen das Verhungern oder das Verdursten oder die tödliche Erschöpfung gibt.

Ich will hier nicht ins Detail gehen, sondern verweise auf ein vergleichsweise altes, aber nach wie vor extrem lesenswertes Buch, das es kostenlos als PDF zum Download gibt: „The Society of Mind“ [3].

An dieser Stelle geht es mir vielmehr um eine gewagte These: Die Regeln, die in einem neuronalen Netz liegen, die Fähigkeiten und Begrenzungen seiner Struktur sind es letztendlich, die am Ende auch Bewusstsein und Willen hervorbringen und zwangsweise zu so etwas wie Identität führen. Immer dann, wenn Hunger gegen Neugier streitet, muss eine „höhere“ Instanz eine Entscheidung fällen. Sie tut dies aufgrund von Erfahrungen, den „Autobahnen“, die wir „Werte“ nennen und die wir immer wieder neu verhandeln („Zuwuchern“), mit jedem neuen Schritt, den wir gehen. Auf diese Weise bilden wir unser Wertesystem, und zwar so lange, bis das nächste unmoralische Angebot kommt [4]. Es ist ein „freier“ Wille, weil eine gewisse Unschärfe, die in jedem Netz liegt, eine Vorhersage der Entscheidung unmöglich macht.

Weil hier unendlich viele Impulse feuern, wir auf oberster Ebene aber keinen Datendump machen können – wir haben nun mal kein externes Speichermedium –, um alles zu untersuchen, müssen wir mit Narrativen arbeiten. Also mit „Geschichten“, die aus vielen Ereignissen eine leicht verständliche Story machen, Unpassendes passend einfügen und so etwas wie eine Identität entstehen lassen. Es geht hier nicht um Wirklichkeit und Wahrheit, sondern um Plausibilität. Da das Gehirn seine eigene Funktionsweise nicht transparent überblicken kann, schafft es eine Geschichte über sich selbst. Es

erzählt sich. Es schafft seine eigene Autobiografie, die uns hilft, alles einzuordnen.

Die Begründung, warum wir etwas getan haben, entsteht also nicht vor, sondern nach der Entscheidung. Genau genommen 200 Millisekunden, wie Messungen der Hirnströme ergeben haben. Uff. Das sitzt. Denn das bedeutet, dass das, was uns auszeichnet, nicht von außen als „göttliche“ Zutat dazukommt, sondern bereits in der Struktur unseres Netzwerks angelegt ist. Denn was wäre eine Seele, wenn sie sich nicht verändern könnte, sondern immer schon da ist und ewig da sein wird? Und was soll das bitte sein, das uns auszeichnet, ohne dass es durch unsere Erfahrungen und unser Erleben bestimmt ist und sich verändert?

Die Ursache des Handelns ist also nicht der Wille, sondern die erlebte Geschichte, die das Gehirn nachträglich über seine Entscheidung legt. Dieses Narrativ stabilisiert Identität. Es ist frei, weil es nicht vorbestimmt ist. Aber es ist nichts, was von außen in uns gelegt wurde – und deshalb für eine Maschine nicht zugänglich wäre. Noch braucht eine KI kein Lagerfeuer, keine wärmenden Metaerzählungen, keine Romane. Denn das sinnliche Erleben fehlt. Die Erfahrung des Todes und unser Bestreben, zu sein und unvergänglich zu bleiben.

Was das alles für unseren Alltag bedeutet? Ehrlich gesagt, ich weiß es nicht. Ich lerne. Und ich verändere mich. Erst im Nachgang werde ich begründen, warum es so kommen musste. Im Idealfall nach 200 Millisekunden.

Bis dahin: Happy Coding! ■

[1] Daniel Kahneman, *Schnelles Denken, langsames Denken*, [https://de.wikipedia.org/wiki/Schnelles\\_Denken,\\_langsames\\_Denken](https://de.wikipedia.org/wiki/Schnelles_Denken,_langsames_Denken)

[2] Die Draufgänger, Marie, <https://www.youtube.com/watch?v=lFA3oXV1L8o>

[3] Marvin Minsky, *The Society of Mind*, [https://de.wikipedia.org/wiki/The\\_Society\\_of\\_Mind](https://de.wikipedia.org/wiki/The_Society_of_Mind)

[4] Ein unmoralisches Angebot (USA 1993), [https://de.wikipedia.org/wiki/Ein\\_unmoralisches\\_Angebot](https://de.wikipedia.org/wiki/Ein_unmoralisches_Angebot)



*Time-out End*



**Dipl.-Theol. Bernhard Pichler**

war 20 Jahre Geschäftsführer der Softwarefirma informare, bevor er 2020 zum Leiter Entwicklung und Produktmanagement der DPS BS ernannt wurde. Die DPS BS ist Deutschlands größter Softwarepartner für betriebswirtschaftliche Software für den Mittelstand.



# ARBEITSMARKT

## TRENDS UND JOBS FÜR ENTWICKLER

GitHub, Thomas Dohmke

### Entwickler-Job im Wandel

Künstliche Intelligenz verändert die Arbeit von Entwicklern grundlegend. Thomas Dohmke – noch bis Ende 2025 CEO bei GitHub – sagt, dass sich die Rolle des Entwicklers vom reinen Coder zum kreativen Gestalter digitaler Systeme wandelt [1]. Entwickler müssen ihren Fokus von der Codeproduktion zur Orchestrierung von KI-Agenten verlagern. Sie definieren Aufgaben, geben präzise Kontexte vor und überprüfen die Ergebnisse. Wer sich auf diesen Wandel einlässt, kann neue Fähigkeiten entwickeln und die eigene Arbeit auf ein neues Niveau heben, sagt Dohmke.

Interviews mit Entwicklern haben einen Trend gezeigt: KI ist Teil des Alltags. Der Weg zur erfolgreichen Zusammenarbeit mit KI verläuft in vier Phasen:

- **Skepsis:** Erste Experimente mit KI, oft mit enttäuschenden Ergebnissen.

- **Erkundung:** Verständnis für die Grenzen und Möglichkeiten der Tools wächst.
- **Kollaboration:** Entwickler arbeiten aktiv mit KI zusammen, nutzen sie strategisch.
- **Strategie:** KI wird als Partner für komplexe Aufgaben eingesetzt, Entwickler fokussieren sich auf Planung und Qualitätskontrolle.

Viele Entwickler sehen ihre Rolle nicht als bedroht, sondern als transformiert: Sie delegieren Aufgaben an KI-Agenten und überprüfen deren Ergebnisse. Die Fähigkeit, präzise zu kommunizieren und Aufgaben sinnvoll zu strukturieren, wird zur Schlüsselkompetenz.

Trotz Automatisierung wächst der Bedarf an Entwicklern, so Dohmke. Der Fokus verschiebt sich von Effizienz zu Ambition: KI ermöglicht größere Projekte und komplexere Lösungen. Erfolg wird künftig daran gemessen, wie gut Entwickler mit KI zusammenarbeiten und deren Potenzial ausschöpfen.

Dazu fordert Dohmke, dass sich die Informatikausbildung neu ausrichten muss: Weg von Syntax und API-Memorieren, hin zu Systemverständnis, Problemanalyse und Zusammenarbeit mit KI. Studierende sollen lernen, wie man mit KI kommuniziert, deren Output bewertet und kreativ mit ihr arbeitet.  
[1] <https://ashtom.github.io/developers-reinvented>

### Bitkom

### Fachkräftemangel und Stellenabbau

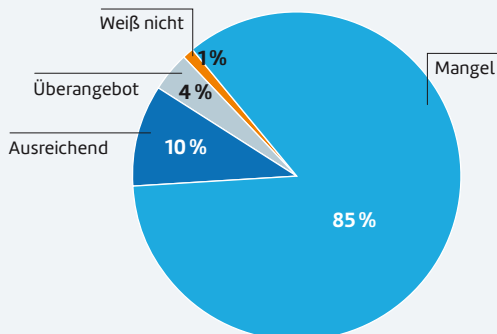
Gemäß einer neuen Bitkom-Studie zum Arbeitsmarkt für IT-Fachkräfte, für die 855 Unternehmen aller Branchen repräsentativ befragt wurden, fehlen aktuell rund 109 000 IT-Fachkräfte. Das sind zwar deutlich weniger als noch vor zwei Jahren mit 149 000, allerdings haben es Unternehmen weiterhin schwer, passende Mitarbeiter zu finden.

So beklagen derzeit 85 Prozent einen Mangel an IT-Fach-

kräften auf dem deutschen Arbeitsmarkt (Bild 1), nur vier Prozent sprechen von einem Überangebot, und zehn Prozent sagen, dass es ausreichend IT-Fachkräfte gibt. Gleichzeitig erwarten 79 Prozent, dass sich der IT-Fachkräftemangel in Zukunft sogar weiter verschärfen wird (Bild 2), nur vier Prozent gehen davon aus, dass der Mangel geringer wird, und 16 Prozent rechnen mit keiner Veränderung.

Allerdings mussten sechs Prozent der Unternehmen in den vergangenen zwölf Monaten IT-Fachkräfte entlassen, und 14 Prozent rechnen damit, dass dies in den kommenden zwölf Monaten in ihrem Unternehmen der Fall sein wird. Sogar 35 Prozent erwarten, dass es in der deutschen Wirtschaft aufgrund der schwächelnden Konjunktur zu einem Stellenabbau in der IT kommen wird. Gleichzeitig sagen sechs Prozent, dass sie bereits IT-Experten eingestellt haben, die anderswo ihren Job verloren

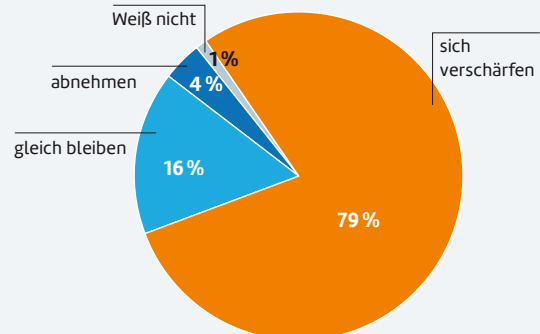
### ● IT-Fachkräfteangebot heute



**85 Prozent der befragten Unternehmen** sagen, aktuell bestehe ein Mangel an IT-Fachkräften in Deutschland (Bild 1)

Quelle: Bitkom Research 2025, n=855

### ● Der IT-Fachkräftemangel wird ...



**79 Prozent der befragten Unternehmen** erwarten, dass sich der IT-Fachkräftemangel künftig verschärfen wird (Bild 2)

Quelle: Bitkom Research 2025, n=855



# Zahl des Monats

## 64 Prozent

Inzwischen gehen **64 Prozent** der Cybersecurity-Experten davon aus, dass die Bedrohung durch Insider die Risiken durch externe Angreifer überholt haben. Eine der Ursachen sind Hacker, die sich mit Fake-Identitäten als vermeintlich qualifizierte IT-Fachkräfte in Firmen einschleusen. Laut Gartner könnte **bis 2028** sogar jede vierte Bewerbung eine Fälschung sein.

Quelle: Sapio Research Studie im Auftrag von Exabeam

haben. Und 52 Prozent rechnen damit, künftig bessere Chancen bei der Suche nach IT-Fachkräften zu haben, weil andere Unternehmen Stellen streichen.

### Bitkom

#### KI: Entlassungen und neue Jobs

Die neue Bitkom-Studie zum Arbeitsmarkt für IT-Fachkräfte ergab auch, dass jedes zwölfte Unternehmen vermehrt Künstliche Intelligenz einsetzt, um dem IT-Fachkräftemangel entgegenzuwirken. Dabei lässt sich noch nicht sagen, welche Auswirkungen KI unter dem Strich auf den IT-Arbeitsmarkt haben wird. So rechnet rund jedes vierte Unternehmen damit, dass es durch KI Stellen abbauen wird, und 16 Prozent erwarten, dass aufgrund von KI Stellen verzichtbar werden, die ohnehin nicht besetzt werden können. Bitkom-Präsident Dr. Ralf Wintergerst: „KI kann heute in der IT bereits eine Vielzahl von Aufgaben erfüllen, von Support-

anfragen bis hin zur Code-Erstellung. Allerdings wird auch der Bedarf an KI-Spezialistinnen und -Spezialisten steigen, wenn mehr Unternehmen Künstliche Intelligenz einsetzen und in ihre Systeme integrieren wollen.“

42 Prozent der Unternehmen erwarten, dass durch KI neue Berufsbilder in der IT entstehen (Bild 3), 34 Prozent gehen davon aus, dass KI einzelne IT-Berufe und -Berufsbilder ersetzen wird. 35 Prozent rechnen damit, dass durch KI die Produktivität steigt, 20 Prozent, dass sich die Qualität der Arbeitsergebnisse verbessert. Und ein Viertel der Unternehmen erwartet, dass IT-Fachkräfte ohne KI-Wissen künftig nicht mehr nachgefragt werden.

### IW

#### Weniger offene Stellen

Ähnlich wie der Branchenverband Bitkom hat auch das Institut der deutschen Wirtschaft (IW) einen starken Rückgang

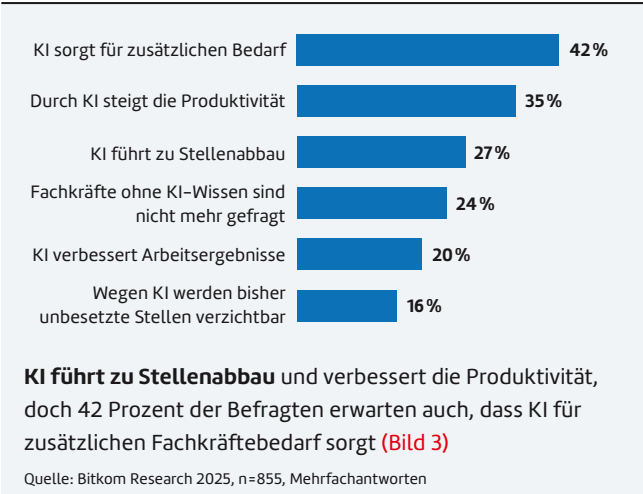
der offenen IT-Stellen erkannt. Am stärksten betroffen ist der Rückgang der Stellenangebote für IT-Experten, die in der Regel einen Masterabschluss oder ein Diplom haben (Bild 4). Besonders drastisch eingebrochen ist die Zahl offener Stellen für Experten der Informatik (minus 46,2 Prozent) und der Wirtschaftsinformatik (minus 38,2 Prozent). Dass gerade in diesen hoch spezialisierten Berufen die offenen Stellen deutlich gesunken sind, dürfte laut IW auch daran liegen, dass Unternehmen zurückhaltender in der Umsetzung hochkomplexer IT-Projekte geworden sind.

Trotz dieses Rückgangs soll der Mangel an IT-Fachkräften weiterhin anhalten. Zuletzt konnten laut IW rechnerisch sieben von zehn offenen Stellen für Informatik-Experten nicht besetzt werden. Somit blieb die Fachkräftesituation in diesen Berufen sehr angespannt. Laut IW-Arbeitsmarktfortschreibung dürften sowohl die Beschäftigung als auch die

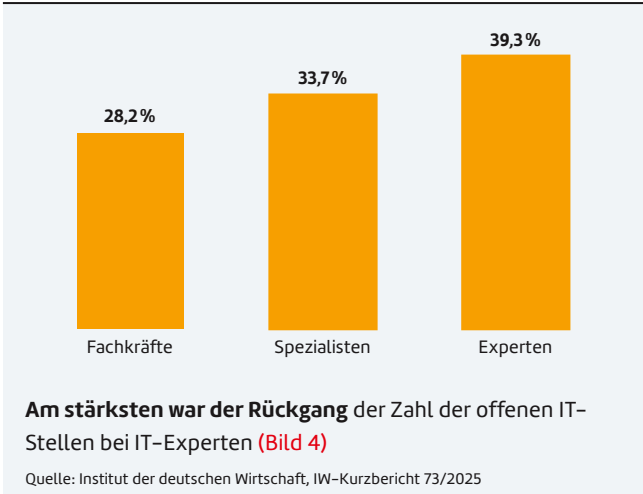
Fachkräftelücke in IT-Berufen bis 2028 neue Höchststände erreichen. Daher gilt es weiterhin, aktiv Fachkräftesicherung zu betreiben. Dazu zählen unter anderem die bedarfsgerechte Nachwuchsqualifizierung, die Förderung von Quereinstiegen sowie die Rekrutierung und Bindung internationaler Fachkräfte. Auch Maßnahmen wie Anreize für eine längere Erwerbstätigkeit älterer Beschäftigter und eine flexiblere Arbeitszeitgestaltung können dazu beitragen, die Fachkräftelücke zu schließen.

Ein Zusammenhang zwischen dem Rückgang offener Stellen und dem Einsatz von KI lässt sich derzeit nicht eindeutig belegen, so das IW. Denn aktuell wird KI überwiegend ergänzend zur menschlichen Arbeit eingesetzt. Perspektivisch verändert KI jedoch die Anforderungen an IT-Beschäftigte, und der kompetente Umgang mit KI rückt stärker in den Fokus, während Routineaufgaben zunehmend automatisiert werden.

### ● KI: Auswirkungen auf IT-Fachkräfte



### ● Rückgang offener IT-Stellen (2022 auf 2024)





# LERNSTOFF

NEUE BÜCHER UND MEHR

## DAS AKTUALISIERTE HANDBUCH

### Git

Die Autoren gehen von der Überzeugung aus, dass der Gewinn bei der Nutzung so komplexer Werkzeuge zur Versionskontrolle und Projektverwaltung wachsen kann, wenn die innere Struktur gut verstanden wird. Sie legen deshalb Wert darauf, mit den Möglichkeiten des git-Kommandos von der Konsole aus zu beginnen und die grundlegenden Vorgänge beim Umgang mit Git-Projekten mithilfe der zahlreichen Optionen dieses Kommandos auszuprobieren. Das erleichtert das Verständnis, wenn dann anschließend mit einer der zahlreichen Oberflächen für Git gearbeitet wird.

Am Anfang stehen einige Begriffserklärungen rund um Git: git als Kommando, Repository, GitHub oder GitLab für externe Repositories, Git-Hosting, Git-Oberflächen und ein paar erste Schritte mit einem Python-Spiel.

Die folgenden Kapitel behandeln Installation, Einrichtung und Authentifizierung – mit oder ohne grafische Oberfläche. Die Rolle von Repositories, Commits, Branches, Merging und Rebasing werden anhand übersichtlicher Beispiele demon-



**Bernd Öggi, Michael Kofler**

Git. Projektverwaltung für Entwickler und DevOps-Teams

Rheinwerk, 3., aktualisierte Auflage, 2025, 421 Seiten, Deutsch, ISBN 978-3-367-10812-1, 39,90 Euro

Zielgruppe: Softwareentwicklung

Voraussetzungen: keine

Leseprobe:

<https://tinyurl.com/rw10812>

triert. Dabei wird Wert auf die Erklärung der internen Vorgänge in Git gelegt, sodass Fehlermeldungen auch verstanden werden können.

GitHub und GitLab werden ausführlich erläutert, Pull Requests, CI/CD-Pipelines, Actions und die Web-IDE an praktischen Beispielen vorgeführt. Effektive Lösungen für Commits, bewährte Branching-Strategien (Gitflow, Trunk Based Development), typische Workflows und Muster für Problemlösungen folgen. Den Abschluss bildet eine auf das Wesentliche konzentrierte Kommandoreferenz. (vh) ■

## PRODUKTENTWICKLUNG

### Visuelle Methoden

Das Buch möchte Teams dazu anregen, die Entwicklung ihrer Produkte mithilfe visueller Methoden zu optimieren. Von der Idee bis zur tatsächlichen Wertlieferung soll der gesamte Prozess optische Darstellungen nutzen, ganz ohne textlastige Beschreibungen. Alle Mitwirkenden verwenden die anschaulichen Darstellungen gemeinsam, die wie durch einen roten Faden miteinander verbunden sind. Die Klarheit und Zusammenarbeit fördert die Selbstorganisation und beschleunigt Entscheidungen. Kern dieser Arbeitsweise ist die Visual Product Ownership. Ziel der Autoren ist es, bewährte Ansätze mit neuen Ideen zu kombinieren und sie in einen systematischen Zusammenhang zu stellen. Sie zeigen, wie visuelle Methoden vernetzt in den gesamten Produktentwicklungsprozess integriert werden können und wie man sie konsequent anwendet. Dabei lassen sich die vorgestellten Ansätze universell einsetzen, unabhängig von spezifischen Frameworks wie Scrum, Kanban oder anderen Modellen.



**Olaf Bublitz, Thomas Starz**

Agile Produktentwicklung mit visuellen Methoden

dpunkt.verlag 2025, 358 Seiten, Deutsch, ISBN 978-3-98889-024-5, 39,90 Euro

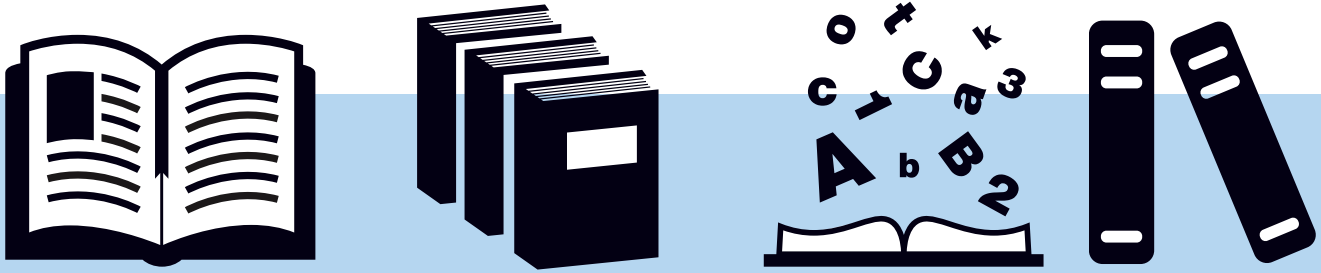
Zielgruppe: An der Produktentwicklung beteiligte Personen

Voraussetzungen: keine

Leseprobe:

<https://tinyurl.com/dp98889-024>

Gegliedert ist das Buch in vier Teile, beginnend mit der Vorstellung der Produktentwicklung mit visuellen Methoden, gefolgt von Grundtechniken der Visualisierung sowie dem Praxisbeispiel DogService24. Der abschließende Abschnitt dient als Nachschlagewerk sowie zur Inspiration. Besprochen wird darin eine breite Auswahl an Methoden und Werkzeugen, die auf die Herausforderungen der Produktentwicklung zugeschnitten sind. Jeder Teil des Buches bietet praktische Ratschläge und zahlreiche Zeichnungen. (bl) ■

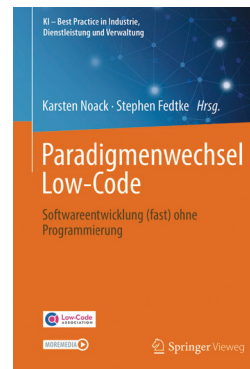


## PARADIGMA FÜR DIE ZUKUNFT

# Low-Code

**Z**wanzig Autoren haben die Herausgeber zusammenggeführt, um den im Titel beschworenen Paradigmenwechsel zu befördern. Absicht ist eine praxisnahe Leitlinie für IT-Verantwortliche, Entwickler und Entscheider, die nicht nur die technischen Aspekte, sondern auch die strategischen Fragen berücksichtigt, etwa die Möglichkeiten der Skalierung und die Formen der Projektsteuerung. Angeboten wird ein breit gestreuter und herstellerneutraler Überblick über die Ursprünge, Funktionsweisen und Einsatzszenarien von Low-Code- und No-Code-Technologien. Dabei wird erwartet, dass der Einsatz von generativer KI die Veränderungen in der Systementwicklung nicht ersetzen, sondern vorantreiben wird.

Im Zentrum steht die Vorstellung, dass mit Low-Code der Graben zwischen der klassischen Anwendungsentwicklung durch IT-Profis und den IT-interessierten Fachkräften, die beispielsweise Lösungen mit Excel-Makros herstellen, geebnet werden kann, sodass beide Seiten gemeinsam von Low-Code-Plattformen profitieren. Dabei wird als gesichert angenom-



**Karsten Noack, Stephen Fedtke**  
(Herausgeber)

Paradigmenwechsel Low-Code, Softwareentwicklung (fast) ohne Programmierung

Springer, 2025, 318 Seiten, Deutsch, ISBN 978-3-658-47125-5, 44,99 Euro, E-Book 34,99 Euro

Zielgruppe: IT-Verantwortliche, Entwickler, Entscheider

Voraussetzungen: keine

Zugang zu Downloads:  
<https://tinyurl.com/sp47125>

men, dass die Digitalisierungsanforderungen nicht allein mit üblicher Software gemeistert werden können. Besonders kleine und mittlere Unternehmen können von Low-Code profitieren. Die bisherigen Plattformen für Low-Code sind allerdings sehr verschieden, und das ist auch eines der Hauptmotive, die zu diesem Projekt geführt haben. Die Überzeugungsarbeit verwendet dabei zahlreiche realistische Beispiellösungen mithilfe der verschiedenen Tools. (vh) ■

## ● Kurz vorgestellt

**Michael Weigend**

### KI-Projekte programmieren mit Python

Einstieg in die KI-Programmierung mit Python. Mithilfe von Bibliotheken wie TensorFlow, SpaCy und music21 werden praktische Projekte umgesetzt. Mit Programmieraufgaben, Bildern zur Veranschaulichung, Übungsfragen und vielen Anregungen zum Ausprobieren.

mitp, 2025, 304 Seiten, Deutsch, ISBN 978-3-7475-0995-1, 24,99 Euro

**Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke**

### Basiswissen für Softwarearchitekten

Vermittelt das Wissen und Fähigkeiten, um eine problemadäquate Softwarearchitektur zu entwerfen. Es behandelt Begriffe und Konzepte, grundlegende Techniken und Methoden, Beschreibung und Kommunikation sowie Qualitätssicherung. Die 6. Auflage thematisiert nun auch Daten und Datenmodelle.

dpunkt.verlag, 6. Auflage 2025, 260 Seiten, Deutsch, ISBN 978-3-98889-048-1, 34,90 Euro

**Amanda Berlin, Lee Brotherston, William F. Reyor III**

### IT-Security – Der praktische Leitfaden

Ziel ist, das Thema Informationssicherheit zugänglich und praxisnah aufzubereiten. Die Autoren liefern praktische Anleitungen zum Aufbau eines Informationssicherheitsmanagementsystems (ISMS). Beschrieben werden Schritte, Werkzeuge, Prozesse und Ideen zum Verbessern der IT-Sicherheit ohne hohe Kosten.

O'Reilly 2025, 446 Seiten, Deutsch, ISBN 978-3-96009-263-6, 39,90 Euro

**Florian Bliesch**

### Mobile Systeme

Erklärt, wie mobile Systeme von Anfang bis Ende entwickelt und genutzt werden, vermittelt die Grundlagen und bietet praktische Einblicke für Studium und Beruf. Zudem zeigt der Autor Strategien und Methoden zur Entwicklung mobiler Systeme und stellt neue Entwicklungen, Trends und Technologien vor.

Hanser, 2025, 579 Seiten, Deutsch, ISBN 978-3-446-48276-0, 49,99 Euro

## Highlights in der Ausgabe 12/25

Die nächste  
Ausgabe  
erscheint am  
**11.12.2025**



### Focus:

Software testen

### Weitere Themen:

- .NET 10 und Blazor
- Azure Container
- Vibe Coding in Theorie und Praxis
- Claude Code
- Standard Cloud Events
- OWASP Top Ten
- IoT-Interfaces mit Blazor und Meadow
- Gamification

Bild: Shutterstock / Zinetron

developer-world.de



## Inserentenverzeichnis

Ebner Media Group

www.ebnermedia.de

2, 45, 53, 65–69, 119, 131, 147, 148

## Impressum

### dotnetpro by DWX

#### MEDIENHAUS

Ebner Media Group GmbH & Co. KG  
Karlstraße 3  
89073 Ulm

info@ebnermedia.de  
www.ebnermedia.de

#### GESCHÄFTSFÜHRUNG

Annabel Ebner, Marco Parrillo

DE147041097  
Sitz: Ulm, Amtsgericht Ulm, HRA 1900

Persönlich haftende Gesellschafterin der  
Ebner Media Group GmbH & Co. KG  
ist die Ebner Ulm MGV GmbH, Ulm.

Sitz: Ulm, Amtsgericht Ulm, HRB 576

#### OFFICE

Ebner Media Group GmbH & Co. KG  
Office Köln  
Helmholtzstraße 29 – 31  
50825 Köln  
Telefon: +49 731 88005–8000  
www.dotnetpro.de

#### VERANTWORTUNG CONTENT

##### Chefredaktion

Fernando Schneider  
fernando.schneider@ebnermedia.de

##### Redaktion

Tilman Börner  
tilman.boerner@ebnermedia.de

##### Externe Autor\*innen

Benjamin Abt, Klaus Aschenbrenner,  
Gregor Biswanger, Christian Giesswein, Tam Hanna,  
Thorsten Kansy, Bernhard Lauer, Stefan Lieser,  
Marc Müller, Jörg Neumann, Bernhard Pichler,  
Georg Poweleit, Alexander Rapp, Patrick Schnell,  
Dr. Holger Schwichtenberg, David Tielke,  
Helmut Vonhogen, Christian Wenz

### Art Direction

Maria-Luise Steinkühler  
malu.steinkuehler@ebnermedia.de

### Gestaltung

EMG DESIGN UNIT  
DesignUnit@ebnermedia.de

### Publisher

Marco Parrillo  
marco.parrillo@ebnermedia.de

### MEDIA

#### Media Sales

Thomas Deck  
thomas.deck@ebnermedia.de

#### Client Success (Disposition)

Anja Büttner  
clientsuccess@ebnermedia.de

### HERSTELLUNG, VERTRIEB & KUNDENSERVICE

#### Leitung Herstellung, Vertrieb & Kundenservice

Thomas Heydn  
thomas.heydn@ebnermedia.de

#### Kundenservice

Telefon: +49 731 88005–8205  
kundenservice@ebnermedia.de

#### Bezugspreise

Die jeweils gültigen Bezugspreise für unsere  
Print- und Digitalausgaben sowie für eventuelle  
Abonnements entnehmen Sie bitte den Angaben auf  
unserer Website oder den jeweiligen Bestellseiten.  
Alle Preise verstehen sich inklusive der gesetzlichen  
Mehrwertsteuer und zuzüglich ggf. anfallender  
Versandkosten, sofern nicht anders angegeben.

Preisänderungen bleiben vorbehalten. Bei  
bestehenden Abonnements gelten die bei  
Vertragsschluss vereinbarten Konditionen, sofern  
keine anderweitige Regelung getroffen wurde.

#### Widerrufsrecht

Sie haben das Recht, binnen vierzehn Tagen ohne  
Angabe von Gründen diesen Vertrag zu widerrufen.

Die Widerrufsfrist beträgt vierzehn Tage ab dem Tag,  
an dem Sie oder ein von Ihnen benannter Dritter,  
der nicht Beförderer ist, die erste Ware in Besitz  
genommen haben.

Um Ihr Widerrufsrecht auszuüben, genügt  
eine eindeutige Erklärung, mit der Sie uns über  
Ihren Entschluss, diesen Vertrag zu widerrufen,  
informieren. Bitte richten Sie diese an:

Ebner Media Group GmbH & Co. KG  
Office München  
Brienner Straße 45a–d  
80333 München  
Telefon: +49 731 88005–8205  
E-Mail: hilfe@ebnermedia.de

Sie können dafür z. B. einen Brief oder eine E-Mail  
verwenden. Zur Wahrung der Widerrufsfrist reicht es  
aus, dass Sie die Mitteilung über die Ausübung des  
Widerrufsrechts vor Ablauf der Frist absenden.

### Urheberrecht / Copyright

Alle Inhalte, Werke und Publikationen – ob online  
oder im Print – der Ebner Media Group GmbH & Co. KG  
unterliegen dem deutschen Urheberrecht.

Jegliche Vervielfältigung, Bearbeitung, Verbreitung,  
öffentliche Zugänglichmachung oder sonstige  
Nutzung außerhalb der gesetzlichen Schranken des  
Urheberrechts ist ohne ausdrückliche schriftliche  
Genehmigung durch die Ebner Media Group GmbH  
& Co. KG nicht gestattet. Downloads, Scans oder  
Kopien sind ausschließlich für den privaten, nicht  
kommerziellen Gebrauch erlaubt.

Soweit Inhalte nicht vom Verlag selbst erstellt  
wurden, werden die Urheberrechte Dritter beachtet  
und entsprechend gekennzeichnet.

### DRUCK

Silber Druck GmbH & Co. KG  
Otto-Hahn-Straße 25  
34253 Lohfelden



# DWX

DIE KONFERENZ FÜR AI,  
CLOUD, WEB & .NET DEVELOPMENT

29. JUNI BIS  
02. JULI 26

M:CON CONGRESS CENTER  
ROSENGARTEN | MANNHEIM

JETZT  
SCHNELL  
BLIND PRICE  
SICHERN!

DEVELOPER  
WORLD



developer-world.de | #DWX





# .NET DEVELOPER CONFERENCE

Die Konferenz für .NET-Entwickler:innen

**24. BIS 27.  
NOVEMBER 25**  
Pullman Cologne Hotel

**10%  
Rabatt**  
mit dem Code  
**DNPAbospecial**

Software Design | Software Architecture | Software Quality  
Software Testing | Frontend Development | GenAI & Agentic  
Data & Analytics | .NET | Approaches | Backend Technologies  
Cloud & Infrastructure



**PROGRAMM IM HEFT AB SEITE 65**



**developer-world.de/ddc | #DDC**