

Daten im Flow

Datenzugriff unter .NET: Vom Wechsel des
Datenbanksystems bis zum Backend-Dienst
in der Cloud für .NET-MAUI-Apps. Ab Seite 12



30. JUNI BIS 03. JULI 2025

MOVING TO MANNHEIM!

www.developer-week.de



Good people

Bei aller Euphorie in Sachen KI rückt in jüngster Zeit der Mensch wieder in den Mittelpunkt. Das hat seinen guten Grund. Und es fordert uns.

Seit geraumer Zeit beobachte ich eine Entwicklung, die mich noch vor Kurzem erstaunt hätte: Im Zusammenhang mit dem Einsatz von Künstlicher Intelligenz wird immer öfter über den Stellenwert des Menschen gesprochen. Ich halte solche Diskussionen für richtig und wichtig. Denn am Ende ist es der Mensch, der Systeme wie Gemini oder ChatGPT erschafft. Er steht im Mittelpunkt, ihm sollten diese Systeme zunutze sein. Wer glaubt, der Mensch sei durch KI ersetzbar, wie in diesen Diskussionen ab und an zu hören ist, dürfte eines Tages ein böses Erwachen erleben.

Statt also die KI einfach ungezügelt „von der Kette zu lassen“, sollten wir uns lieber fragen, wie wir sie zielgerichtet und zugleich verantwortungsvoll einsetzen können. Damit sie uns auch wirklich nutzt. Chancen und Risiken beim Einsatz von KI realistisch zu bewerten gehört ebenso dazu wie die Bereitschaft, ihre Ergebnisse kritisch zu hinterfragen. Das verlangt ein hohes Maß an Erfahrung und Wissen. Wissen, das noch nicht überall ausreichend vorhanden ist, selbst in unserer Branche, und das wir deshalb untereinander teilen müssen.

„Der zielgerichtete, verantwortungsvolle Einsatz von KI braucht Erfahrung und Wissen.“

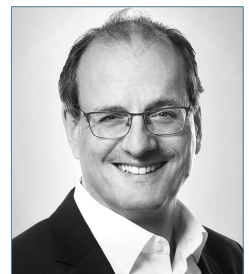
Hierbei möchten wir Sie mit der dotnetpro aktiv begleiten, und das nicht nur mit Blick auf die KI. Mit einer neuen Rubrikstruktur und neuen Themen im künftig zweimonatlich erscheinenden Magazin, neuen Formaten wie Reportagen, Interviews und Fallstudien sowie eigenständigen Online-Beiträgen, die wir mit den Heftinhalten abstimmen, wollen wir Sie ab der nächsten Ausgabe im April noch besser in Ihrer Arbeitsrealität abholen und zugleich für mehr Abwechslung sorgen. All das selbstverständlich bei gleichbleibender Qualität. Mehr erfahren Sie auf den folgenden Seiten.

Das alles geschieht nicht auf einmal, sondern ist ein Prozess und braucht Zeit. Daher meine Bitte: Geben Sie uns diese Zeit. Und begleiten Sie uns aktiv auf diesem Weg, indem Sie mir Ihre Anregungen und Wünsche für diesen Prozess schicken. Damit die dotnetpro noch mehr als bisher „Ihre“ dotnetpro wird!

Ich wünsche Ihnen eine schöne Zeit mit der neuen dotnetpro!

Herzlichst

Fernando Schneider
Chefredakteur dotnetpro
fernando.schneider@ebnermedia.de



Stefan Lieser

zeigt Ihnen, wie Sie ein System mit Bounded Contexts in unabhängige Bestandteile zerlegen (S. 38)



Fabian Deitelhoff

befasst sich im letzten Teil seiner KI-Serie mit der Umsetzung von RAG-Systemen in der Microsoft-Welt (S. 72)


```
using System;  
  
class Program  
{  
    static void Main(string[]  
args)  
    {  
        Console.WriteLine  
("HELLO, WORLD!");  
    }  
}
```

**GOOD CODE.
GOOD VIBES.
GOOD PEOPLE.**

Developer-Wissen, das du wirklich brauchst.



using System;

class Program

{

static void Main(string[]

{

Console.WriteLine("

}

WISSEN, DAS DU WIRKLICH BRAUCHST.

}

Was bewegt die Developer-Welt? Welche Tools helfen dir in der Praxis?

Welche Lösungsansätze und Cases wählen erfolgreiche Developer?

Seit 2002 haben sich Softwareentwickler in der **dotnetpro** verpflichtet, für andere Softwareentwickler relevantes und fundiertes Wissen weiterzugeben.

4 PRODUKTE MIT VORTEILEN:

01

Magazin

6x pro Jahr auf 148 Seiten geballtes Fachwissen mit Best Cases, Interviews, Lösungsansätzen und Testings – zum Entdecken, Inspirieren und Umsetzen

02

Newsletter

- 1x pro Woche kompakt mit den wichtigsten Themen in dein Postfach.
- Wöchentlich wechselnd 3 News aus der Softwareentwicklung und 3 ausgewählte Beiträge aus der dotnetpro.

Du willst erstmal testen? Dann ist der Newsletter das Richtige für dich: Die 3 Beiträge gibt es jeweils für eine Woche gratis.

INTERESSANT FÜR DICH?



args)

**KURATIERT. RECHERCHIERT.
REFLEKTIERT.");**

03

Exklusive Vorteile für Events

DNP-Abonnenten erhalten 10% Rabatt für exklusive Live-Events und Konferenzen wie

- DDC – .NET Developer Conference für Entwickler in Köln
- DWX – Developer Week in Mannheim

Bis zu 500 EUR sparen, Wissen und Skills boosten und spannende Menschen treffen!

JETZT CODE ANFORDERN UND VORTEIL SICHERN!

04

Digitale Wissenswelt

Von Entwicklern für Entwickler aggregiert die DNP-Plattform digitales Wissen, Code, Skills und News aus der Branche, die unsere Arbeit als Softwareentwickler bereichert.

Im digitalen Archiv lassen sich Code-Snippet-Schätze heben: einfach, schnell, digital und nicht-halluzinierend zum erprobten Quellcode.

WEITERE INFOS GIBT ES HIER:



using System;



class Program

{

static void Main(string[]
args)

{

Console.WriteLine
("COMING SOON");

}

}

RELEASE ANNOUNCEMENT 2025

Als Entwickler entwickeln wir uns weiter.
2025 launchen wir ein **neues digitales Kompaktformat**.

Ein Fokusthema als aggregierter Developer Deep Dive mit Wissen
und Best Cases aus der Praxis zum Anwenden in der Praxis.

STAY TUNED!



Sichere dir mit deinem Abo eine Hydro-Flask- Thermoflasche



6 x IM JAHR
ALS MAGAZIN
ODER E-PAPER



Jetzt bestellen

Per E-Mail

KUNDENSERVICE@EBNERMEDIA.DE

Per Telefon

+49 731 88005-8205

* Das Angebot ist nur in Deutschland erhältlich. Nur solange der Vorrat reicht.





INHALT

Update

Neuer Ort, weiterhin beste Fortbildung
Developer Week 2025

10

Schwerpunkt

In zwei Zungen sprechen

Mehrere Datenbankanbieter in einer Anwendung

12

Wolkenpower

Cloud-basierte Backends für .NET-MAUI-
Cross-Plattform-Apps

17

Offene Alternative

Open-Source-Datenbank PostgreSQL, Teil 1

26

Planung

Time-out: Exzellenz versus Glück

Was lohnt es sich aufzugeben für einen Platz an
der Weltspitze?

36

Sauber zerlegt

Bounded Contexts identifizieren und visualisieren

38

Dauids Deep Dive: Die Konsequenz von Schulden

Was aus Versäumnissen folgt

42

Frontend

Saluton Mondo!

Lokalisierung für ASP.NET Core und Blazor

44

Core

NETirol: Der geteilte String

Zeichenketten in .NET 9

50

Warum rücken wir Code ein?

Kontrollierte Experimente für die Programmierung

53

Doing the Time Warp

Zeitangaben unter .NET richtig handhaben

58

Luftschlange

Messaging mit Wolverine und .NET, Teil 9

62

Frameworks: Aber bitte typsicher

RegExtract: Daten extrahieren leicht gemacht

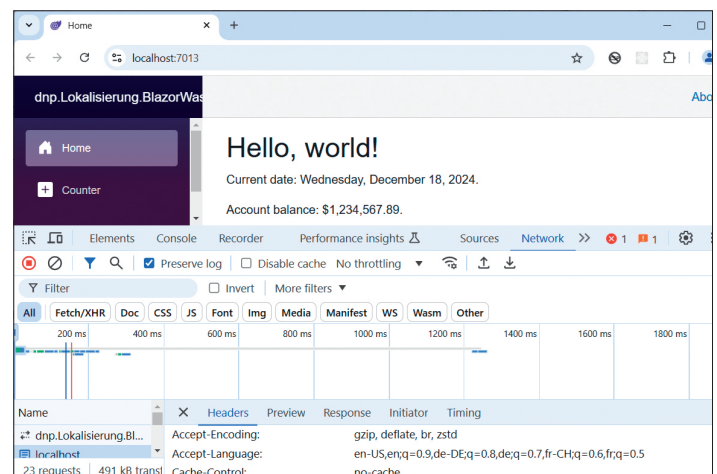
66



Bild: Midjourney

Daten im Flow: SQL Server und PostgreSQL im Verbund,
Backend-as-a-Service-Dienste in .NET-MAUI-Apps.

12



Schlusspunkt: Teil 3 der Serie zur Lokalisierung von
ASP.NET-Core-Anwendungen behandelt auch Blazor.

44

Experten in dieser Ausgabe



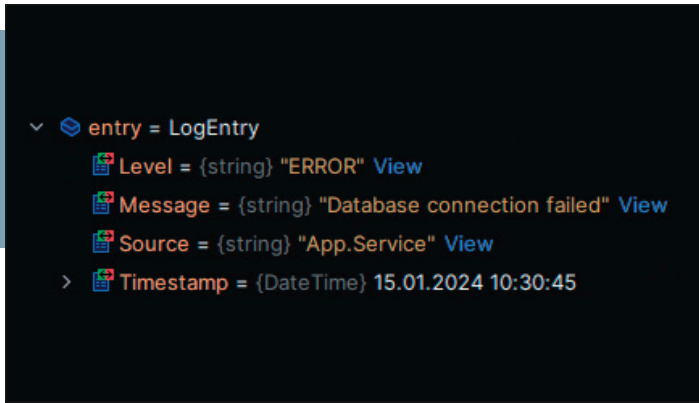
**Sebastian Klein-
schmager** fragt sich,
warum wir eigent-
lich Code einrücken
und wie groß der
Nutzen ist.

53



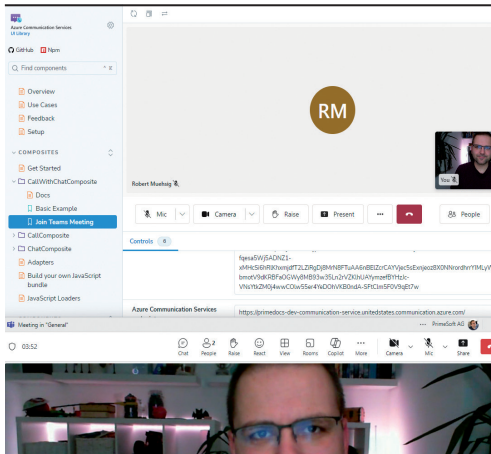
Christian Havel
zeigt den richtigen
Umgang mit den
.NET-Klassen für
Datums- und Zeit-
angaben.

58



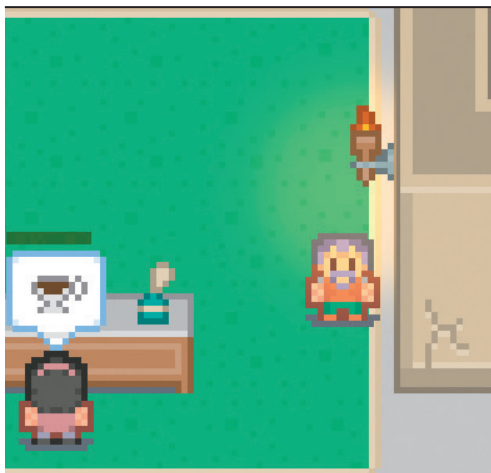
Typsicher: Mit RegExtract Daten aus Texten extrahieren, ohne auf die Typsicherheit von C# zu verzichten.

66



Eine Frage der Kommunikation: Wie Sie mittels Azure Communication Service Sprach-, Video- und Textchats in Ihre eigenen nativen Applikationen und Web-Apps einbinden

96



Game over? Der letzte Teil der MonoGame-Serie verpasst Ihrem Spiel den letzten Schliff und verrät, wie Sie es bei Bedarf weiter anpassen. Denn nach dem Spiel ist bekanntlich vor dem Spiel.

128

Quellcode gefällig?

Sie möchten den Quellcode eines Artikels ausprobieren? Oder den Artikel als PDF herunterladen?

Dann achten Sie auf den dnpCode im Autorenkasten am Ende des Artikels. Hängen Sie diesen Code an www.dotnetpro.de/ an. Dieser Link führt Sie zur passenden Seite im dotnetpro-Archiv.

KI

RAG-Time bei Microsoft

KI und Data, Teil 5

72

Übernimmt KI deinen Job?

Entwicklung der Stellen für Entwickler

78

Backend

Data Access: Nachgelegt

Weitere Neuerungen in System.Text.Json 9.0

80

Tools

Nachvollziehbarer Code statt Excel-Chaos

Polyglot Notebooks in VS Code und C#, Teil 2

90

Workout

Kommunikations-Baukasten

Azure Communication Service

96

Fast wie in echt

Simulatoren und Emulatoren für realitätsnahe Tests

102

Eine Frage der Echtzeit

Hybride Echtzeitsteuerung mit einer SPS und C#

106

Umstieg bei der Steuerung

Open Platform Communication – Unified Architecture

116

Zusatzverkauf

Shopify-Plug-ins mit .NET

121

Des Einstiegs letzter Teil

MonoGame für Einsteiger, Teil 8.2

128

Editorial

3

Lernstoff

88

Arbeitsmarkt

140

Impressum / Inserentenverzeichnis

144

Anbieterverzeichnis

145

Vorschau

146

DEVELOPER WEEK 2025

Neuer Ort, weiterhin beste Fortbildung

Die große Entwicklerkonferenz zieht nach Mannheim um.
Warum Sie unbedingt dabei sein sollten.

Hier, wo der Neckar in den Rhein fließt, wo die Straßennamen aus einzelnen Buchstaben und Zahlen bestehen und wo die Adler nicht durch die Lüfte zischen, sondern Eishockey spielen, hier beginnt für die große Entwicklerkonferenz Developer Week (DWX) [1] ein neues Kapitel: Sie wird ab 2025 in Mannheim stattfinden – oder „Monnem“, wie nicht nur Comedian Bülent Ceylan seine Stadt nennen würde. Nach elf Jahren im bayerischen Nürnberg zieht die DWX dieses Jahr nach Baden-Württemberg um.

Wen jetzt die Wehmut plagt, wer sich nach fränkischem Essen oder dem beeindruckenden NCC Ost der Messe Nürnberg sehnt, dem sei versichert: Dieser Wechsel bringt viele Vorteile mit sich. Gerade einmal zehn Minuten dauert der Weg vom Mannheimer Hauptbahnhof zur Developer Week 2025 – zu Fuß. Denn im Gegensatz zum Messegelände Nürnberg liegt das m:con Congress Center Rosengarten mitten in der Stadt. In unmittelbarer Nachbarschaft befinden sich Mannheims Wahrzeichen: der Wasserturm, die Fußgängerzone „die Planken“ und die Fressgasse.

„Das ist das Zeichen, das wir setzen wollen: Vorbei ist das Leben am Rand. Die Developer Week holt ihre Teilnehmerinnen und Teilnehmer in die Mitte der Stadt und der Gesellschaft“, kommentiert Charlotte Frieß, Director Events, den Umzug schmunzelnd.



Gregor Biswanger

„Die Developer Week ist für mich mehr als nur eine Konferenz – es ist ein Familientreffen der Entwickler-Community. Seit der ersten Stunde bin ich begeistert dabei und schätze den Austausch auf höchstem Niveau!“

Bei der Wahl der Location stand auch die ausgezeichnete Anbindung der Stadt im Vordergrund. „Aus München erreicht man Mannheim mit dem ICE in drei Stunden, aus Frankfurt und Stuttgart in einer Dreiviertelstunde und aus Hamburg und Berlin ist man in knapp fünf Stunden hier. Zentraler geht es nicht“, bestätigt Projektmanager Vlatko Stojanov.

Der Rosengarten ist eine der größten Veranstaltungslocations in Deutschland. Bis zu 9000 Menschen können hier Konzerten lauschen oder auch Veranstaltungen wie den Guitar Summit [2] (26. bis 28. September 2025) besuchen, den die Verlagskolleg:innen des Fachmagazins „Gitarre und Bass“ veranstalten. Mit über 550 Ausstellern aus aller Welt, weit über 100 Workshops auf sieben Bühnen und Konzerten mit den internationalen Topstars der Gitarrenszenen ist der Guitar Summit eine der größten Gitarrenshows weltweit und ein Muss für alle, die sich für Gitarre und Bass interessieren!

Das ist die Developer Week

Vom **30. Juni bis zum 3. Juli 2025** dreht sich im m:con Congress Center Rosengarten aber alles um die Softwareentwicklung. Die Developer Week bildet Softwareentwickler:innen, Architekt:innen und IT-Entscheider:innen mit Fachvorträgen fort. Rund 150 Sessions sowie halb- und ganztägige Workshops bieten Wissen in verschiedener Tiefe und Umfang.

Mit 29 Thementracks deckt die DWX weite Bereiche der Softwareentwicklung ab – **von rein technischen Themen wie KI, Webentwicklung, der gesamten .NET-Welt über Architektur und Softwarequalität bis hin zu personenorientierten Themen wie From Me to We oder Personal Growth.** Die Mischung aus Praxisrelevanz, hochkarätigen Sprecher:innen und intensiven Workshops macht die DWX zu einem Pflichttermin für Entwickler:innen.

Große Vielfalt durch diese 29 Tracks:

- Software Design Principles
- Software Architecture
- Software Quality
- Software Testing
- Software Security
- Doing Agile / Agile in Practice
- DevOps
- CI / CD
- Git, Github
- Web Hybrid
- Web Design
- Web Languages
- Cloud Development
- Container
- Data & Analytics
- Databases
- GenAI Fundamentals
- Robotics

- .NET Fundamentals
- .NET Backend
- .NET Frontend
- .NET Doing
- Web Frontend
- Web Backend
- GenAI Tools & Solutions
- Personal Growth
- From Me to We
- Sustainability
- IoT/Industry 4.0

Die Developer Week ist aber längst nicht mehr nur eine Konferenz, sie ist ein Erlebnis. Denn was wäre das Leben, bestünde es nur aus Wissensvermittlung? Auch der intensive Austausch mit Kolleginnen und Kollegen, coole Abendevents und exzellentes Essen gehören deshalb zur Developer Week – ein Community Event, das man erlebt haben muss. Ob Night Talks, Fachsimpeln bei Bier, Wein und Softdrinks oder die Casino Night. Deshalb: Nicht lang schnacken, Ticket einsacken.

Am Anfang steht das Programm

Das umfangreiche Vortragsangebot macht sich aber nicht von selbst – hierfür bedarf es fleißiger Menschen. Nachdem die Experten:innen ihre Vorschläge für Vorträge eingereicht haben, müssen die Track Chairs ran. Diese ehrenamtlichen Experten suchen aus den weit über 1000 Einsendungen die Vorschläge heraus, die sie für gut und wert halten, in ihren Tracks aufzutauchen.

Zum Schluss dürfen pro Track nur acht übrig bleiben. Die ersten fünf finden in dem jeweiligen Track statt. Fällt aber ein Sprecher aus, muss der Slot nachbesetzt werden. Hier kommen dann die drei zusätzlichen zum Zug. Glücklicherweise konnte das Team der Developer Week bekannte Größen der Softwareentwicklung als Track Chairs gewinnen.

Sind alle Tracks bestückt, prüfen die beiden Conference Chairs Fernando Schneider und Tilman Börner das gesamte Programm auf Überschneidungen, Dopplungen oder auch darauf, ob Sprecher:innen zu oft vertreten sind.

Erst wenn das Programm auch diese Hürde genommen hat, wird es veröffentlicht – auf der Website der Developer Week und in der dotnetpro.

Halb- und ganztägige Workshops für mehr Tiefe

Neben den Sessions, die ja nur eine Stunde lang und damit in Tiefe oder Umfang sehr begrenzt sind, werden noch halb- und eintägige Workshops angeboten. Sie finden am Montag, den 30. Juni 2025 statt. In den Workshops geht es richtig in die Tiefe. Auch der Faktor Mitmachen ist ein wichtiger Aspekt dieser Formate. So sind in den meisten Workshops die Teilnehmerinnen und Teilnehmer angehalten, ihren Laptop einsatzbereit dabei zu haben, um selbst mitzuprogrammieren. Der Lernerfolg durch Vertiefung und Verfestigung des Gelernten ist damit wesentlich stärker.

Die Tickets: Große Auswahl, sparen inklusive

Wenn Sie mit dabei sein wollen und die Premiere in Mannheim miterleben wollen, sollten Sie sich schnellstens ein Ticket besorgen. Hier stehen verschiedene zur Auswahl:

- Workshop-Ticket
- Ein-Tages-Ticket

- Zwei-Tages-Ticket
- Drei-Tages-Ticket
- XL-Ticket
- DWX Premium-Ticket
- Studententicket

Teilnehmer erhalten mit den Tickets zusätzlich Zugang zu allen Aufzeichnungen der Developer Week 2025 bis Jahresende. Beim Premium-Ticket gilt das sogar für ein Jahr.

Gehen mehrere Kollegen einer Firma zur Developer Week, sinkt ihr Ticketpreis. Community Managerin Susanne Herl freut sich, „dass wir jedes Jahr komplette Softwareteams auf der Konferenz haben. Immer mehr Teams bringen sogar ihre Auszubildenden mit, weil sie die zukünftigen Kolleg:innen gleichberechtigt weiterbilden möchten. Beides unterstützen wir gerne mit entsprechenden Rabatten.“

Für Partner: Ausstellen in der Fachmesse

Wertvolle Informationen an den Ständen der Partner, ungezwungener Austausch über die Branche oder das ein oder andere Gewinnspiel: All das findet sich im Bereich der Fachausstellung, die die Developer Week seit dem Jahr 2013 abrundet. „Unsere Partner haben die einzigartige Möglichkeit, an einem Ort mit der Zielgruppe der Softwareentwickler intensiv in Kontakt zu treten“, erklärt Thomas Deck, Sales Manager Developer Events und Media. „Attraktive Standpakete können mit Add-ons ergänzt werden. Das schafft maximale Flexibilität für Branding, Produkte, Leads und Networking.“

Jetzt sind Sie an der Reihe

Seien Sie dabei, wenn die Developer Week zum ersten Mal in Mannheim stattfindet. Neue Stadt, neue Location, weiterhin beste Fortbildung von den besten Experten. Holen Sie sich Ihr Ticket oder sprechen Sie mit Susanne Herl, wenn Ihre Kollegen dabei sein wollen. **Wir sehen uns in Monnem.** ■

[1] Developer Week, www.developer-week.de

[2] Guitar Summit, www.guitarsummit.de



Foto: Midjourney

MEHRERE DATENBANKANBIETER IN EINER ANWENDUNG

In zwei Zungen sprechen

PostgreSQL als Ergänzung neben MS SQL Server: Wie Sie mit minimalen Eingriffen an einer bestehenden Anwendung einen weiteren Datenbankanbieter hinzuzufügen.

Das Beherrschen mehrerer Sprachen kann einem viele Türen öffnen. Doch das gilt natürlich nicht nur für Globetrotter während ihrer Reisen, sondern auch in der Softwareentwicklung. Deshalb soll heute die Tür geöffnet werden, die es erlaubt, Ihre Anwendung flexibler zu machen, was die Abfrage und Abfrage von Daten angeht.

Dass es hier um PostgreSQL [1] und MS-SQL [2] geht, ist der Tatsache geschuldet, dass der Autor erst kürzlich in einem seiner Projekte etwas Derartiges erfolgreich implementiert hat. Übereinstimmungen mit persönlich existierenden Vorlieben sind rein zufällig.

Die Ausgangslage

Ihre Anwendung ist fertig, ausgeliefert, bereits erfolgreich bei zahlreichen Kunden im Einsatz. Der bei allen Abnehmern genutzte Datenbankserver ist der von Microsoft, mit dem Sie sowieso schon seit Jahren per Du sind und mit dem Sie schon

immer entwickelt haben. Zwei Tage nach Vertragsabschluss mit dem Neukunden XY, dem versprochen wurde, natürlich könne man auch mit der Z-Datenbank umgehen, wird in der Entwicklung nachgefragt: „Äh, das können wir doch, oder ...?“ Also gilt es, mal eben die Ärmel hochzukrempeln.

Struktur

Die Beispielanwendung ist in ihrer Struktur an einen sogenannten modularen Monolithen angelehnt [3]. Das reduziert den Overhead, der entsteht, wenn mehrere Dienste untereinander Daten austauschen müssen, ermöglicht aber trotzdem eine saubere Trennung der Belange, die einzelnen Dienste untereinander haben keine Abhängigkeiten zueinander.

Es gibt einen Anwendungshost, der den Fuck-up-Point enthält (hier werden alle Abhängigkeiten registriert oder deren Registrierung an die einzelnen Dienste delegiert), und eben einen oder mehrere Dienste. Jeder Dienst hat sein eigenes

● Listing 1: Datenbankkontext erzeugen

```
public static WebApplicationBuilder
    CreateHostBuilder(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);
    ConfigureDatabase(builder.Services, args);
    return builder;
}

private static void ConfigureDatabase(
    IServiceCollection services, string[] args)
{
    services.AddDbContext<WhoAmIDbContext>(options =>
    {
        options.UseSqlServer(
            @"Server=(localdb)\mssqllocaldb;Database=
            WhoAmIDB;Trusted_Connection=True;");
    });
}
```

Datenbankschema und seinen eigenen DbContext (der auch auf gar keinen Fall mit anderen Diensten geteilt wird).

Der Anwendungshost verweist auf jeden der einzelnen Dienste. Es wird zu jedem Dienst eine separate Migrations-Assembly geben, auf die der Host ebenfalls verweist. Die Migrations-Assembly verweist wiederum auf ihren jeweils dazugehörigen Dienst. Warum die Migrations-Assembly gebraucht wird, wird im weiteren Verlauf dieses Artikels erläutert. In **Bild 1** ist die prinzipielle Struktur dargestellt. Das Beispielprojekt für diesen Artikel ist einfach gehalten – da ist architektonisch durchaus Luft nach oben. Im Grunde folgt es aber ebenfalls diesem Aufbau.

Falls dieser Aufbau nicht möglich oder gegeben ist und sich durch die notwendigen Projektverweise zirkuläre Referenzen ergäben, finden Sie auf den Microsoft-Hilfeseiten eine Lösung [4].

Konventionen

MS SQL Server und PostgreSQL unterscheiden sich im Bereich der Konventionen im Wesentlichen durch Groß- und Kleinschreibung: einerseits CamelCase, andererseits snake_case. An dieser Stelle darf sicherlich diskutiert werden, ob man sich dem anschließt oder sich für CamelCase entscheidet, was vermutlich auch den Konventionen entspricht, denen Sie in der NET-Programmierung folgen. Zudem übernimmt Npgsql [5] in der Standardkonfiguration sowieso das Casing aus dem Code. Und auch der *nameof*-Operator kann ohne Weiteres benutzt werden, zum Beispiel in *DbContext.OnConfiguring*:

```
modelBuilder.ToTable(nameof(EntityName))
```

In wahrscheinlich eher seltenen Fällen werden Sie SQL von Hand schreiben (warum auch, in der Regel erledigt dies das

Entity Framework). Falls doch, wird es sich auch hier bewähren, die NET-Konventionen übernommen zu haben. Sofern es sich um Standard-SQL handelt, nehmen Sie als Basis das MS SQL und ersetzen für PostgreSQL einfach die eckigen Klammern ([und]) durch doppelte Anführungszeichen. Für Tabellen- und Spaltennamen kann dann ebenfalls der *nameof*-Operator erhalten, gegebenenfalls schreiben Sie eine Erweiterungsmethode *ToPgsSql()*:

```
public static string ToPgsSql(this string sql) {
    return sql.Replace('[', '').Replace(']', '');
}
```

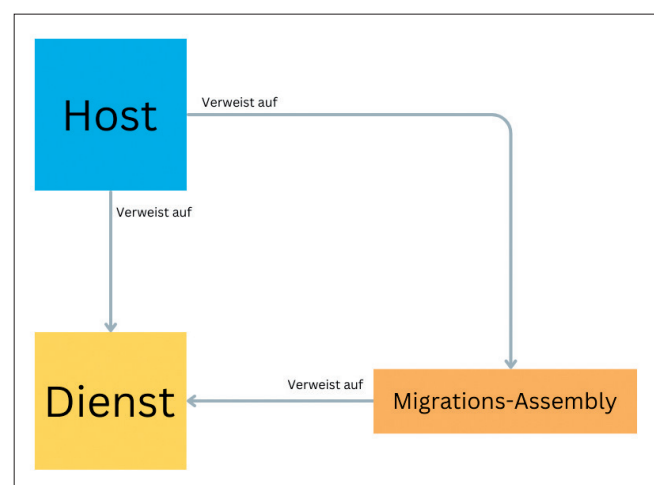
Das funktioniert so dann übrigens auch für Oracle; MySQL und MariaDb benutzen Backticks (`). Sie werden sicher bemerkt haben, dass Sie sich ersparen, herauszufinden, ob Sie nun das öffnende oder schließende „Tüddelchen“ brauchen, wenn Sie von dem SQL-Dialekt aus Redmond ausgehen.

Grundgerüst

Wenn das Grundgerüst der Anwendung so weit steht, gibt es einen Einsprungpunkt im Host-Projekt, Controller, Views, einen Datenbank-Kontext, eine initiale Datenbankmigration sowie etwas Code, um Daten abzufragen und an die View weiterzugeben. Bis jetzt ist aber immer noch alles einsprachig. Die Datenbankmigration wurde mit dem dotnet-ef-Tool erzeugt und ist standardmäßig im Ordner *Migrations* im Dienstprojekt – dort, wo auch der Datenbankkontext wohnt – angelegt worden. Die Projektvorlage ist die bekannte ASP.NET-Vorlage zum Erzeugen einer MVC-Anwendung. Eine initiale Tabelle ist mit einem Testdatensatz befüllt worden. Also so weit nichts Aufregendes. Trotzdem freut man sich kurz, wenn das erste Hochfahren der Anwendung unfallfrei klappt (siehe **Bild 2**).

Exkurs 1: dotnet-ef – Datenbankkontext erzeugen

Wenn wie hier der Datenbankkontext in der *Program.cs* per Dependency-Injection erstellt wird (siehe **Listing 1**), sucht ►



Grundsätzlicher Aufbau der Anwendung (Bild 1)

das dotnet-Tool nach einer Methode *CreateHostBuilder(string[] args)*, die für das Erzeugen des DbContext und damit das Erstellen der Datenbankmigrationen verwendet wird. Diese Methode muss also vorhanden sein, und diese Methode muss den Datenbankkontext instanzieren. Ein einfaches *dotnet ef migrations add* reicht nicht aus; mitgeteilt werden müssen der Host (Startprojekt) und das Projekt, in dem der Datenbankkontext liegt (Dienstprojekt):

```
dotnet ef migrations add Initial -s START-PROJEKT
-p DIENST_PROJEKT --context DB_CONTEXT
```

Wenn der Datenbankkontext wie in [Listing 1](#) gezeigt erzeugt wird, braucht es keine verschiedenen Datenbankkontexte; welchen Dialekt der Kontext spricht, ist hier nämlich in den meisten Fällen für die weitere Entwicklung unwichtig.

● Listing 2: Die Methode Database.IsNpgsql()

```
public class WhoAmIDbContext(DbContextOptions
    <WhoAmIDbContext> options) : DbContext(options)
{
    public DbSet<WhoAmIEntity> WhoAmICollection =>
        Set<WhoAmIEntity>();

    protected override void OnModelCreating(
        modelBuilder modelBuilder)
    {
        modelBuilder.HasDefaultSchema("example");

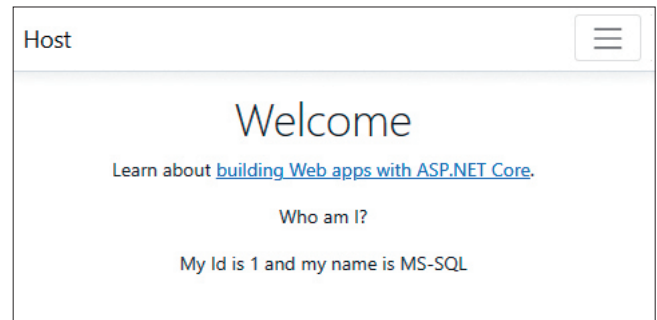
        modelBuilder.Entity<WhoAmIEntity>()
            .HasKey(e => e.Id);

        modelBuilder.Entity<WhoAmIEntity>()
            .Property(p => p.Id)
            .ValueGeneratedNever();

        modelBuilder.Entity<WhoAmIEntity>().
            HasData(Seed());

        modelBuilder.Entity<WhoAmIEntity>()
            .ToTable(nameof(WhoAmICollection));
    }

    private WhoAmIEntity Seed()
    {
        var name =
            Database.IsSqlServer() ? "MS-SQL" :
            Database.IsNpgsql() ? "PostgreSQL" :
            "unknown";
        return new WhoAmIEntity(1, name);
    }
}
```



Der erste Start der Anwendung (Bild 2)

Jetzt wird es ernst

Das Paket *Npgsql.EntityFrameworkCore.PostgreSQL* wird über den Paketmanager Ihres Vertrauens besorgt und installiert. Eine *UseNpgsql*-Methode sollte daraufhin zur Verfügung stehen, und die Datenbankkonfigurationsmethode aus [Listing 1](#) kann nun ergänzt werden:

```
var dbProvider = "MSSQL";
services.AddDbContext<WhoAmIDbContext>(options =>
{
    if (dbProvider == "MSSQL")
    {
        options.UseSqlServer(@"Server=(localdb)\mssqllocaldb;
            Database=WhoAmIDB;Trusted_Connection=True;");
    }
    else
    {
        options.UseNpgsql(@"Host=localhost;Database=WhoAmIDB;
            Username=postgres;Password=P@ssw0rd");
    }
});
```

Ob Sie den Schlüssel (*dbProvider*) und die Verbindungszeichenfolgen nun aus den *appSettings.json* oder den Umgebungsvariablen holen, ist relativ egal. Da aber Passwörter nicht im Klartext in irgendwelchen Einstellungsdateien gespeichert werden sollten, hat die Variante mit den Umgebungsvariablen sicherlich mehr Charme, und SonarCube, sofern Sie es verwenden, ist auch befriedet.

Für das Erzeugen des oben bereits erwähnten ersten Datensatzes steht jetzt eine *Database.IsNpgsql()*-Methode zur Verfügung, die zur inneren Selbstfindung des Datenbankkontexts benutzt wird. Beachten Sie dazu [Listing 2](#).

Exkurs 2: dotnet-ef – Migrations-Assembly

Das dotnet-Tool sucht immer in der Assembly nach Migrationen, wo es den Datenbankkontext findet. Es lässt sich nicht definieren, in welchem Ordner genau das Werkzeug nach Migrationen suchen soll. Wohl aber kann man eine Assembly angeben, in der die Migrationen liegen.

Daher braucht es ein weiteres Projekt für die PostgreSQL-Migrationen. Im Beispiel heißt es *WhoAmIService.PG*. Wenn dieses angelegt ist, wird mit dem bereits vorhandenen MS-SQL-Datenbankkontext eine leere Migration angelegt und in die Migrations-Assembly herüberkopiert. Die leere Migration ist erforderlich, damit das dotnet-Tool den Datenbank-

Listing 3: .NET- in PostgreSQL-Datentyp wandeln

```
public class WhoAmIDbContext(DbContextOptions
    <WhoAmIDbContext> options) : DbContext(options)
{
    ...

    protected override void ConfigureConventions(Model
        ConfigurationBuilder configurationBuilder)
    {
        if (Database.IsNpgsql())
        {
            configurationBuilder.Properties<string>().
                HaveColumnType("citext");
        }
    }
    ...
}
```

kontext findet, der ja in einer anderen Assembly definiert ist. Wenn Sie einen Blick in die erzeugte *Migration.Designer.cs* werfen, sehen Sie einen Verweis auf den Datenbankkontext. Jetzt dürfte auch klar sein, warum die oben beschriebenen Projektverweise benötigt werden.

Die Methode zur Konfiguration des Datenbankkontexts wird also ergänzt:

```
options.UseNpgsql(@"Host=localhost;Database=WhoAmIDB;
    Username=postgres;Password=P@ssw0rd", pg =>
{
    pg.MigrationsAssembly("WhoAmIService.PG");
});
```

Der hart codierte Konfigurationsparameter wird auf "PG" geändert:

```
var dbProvider = "PG";
services.AddDbContext<WhoAmIDbContext>(options => ...
```

Jetzt kann über die Developer Shell von Visual Studio eine erste wirkliche PostgreSQL-Migration erzeugt werden:

```
dotnet ef migrations add Initial -s Host
-p WhoAmIService.PG --context WhoAmIDbContext
```

Die leeren Migrationen werden nicht mehr benötigt und dürfen dem Daten-Nirwana übergeben werden.

Wenn der Autor keine Informationen unterschlagen hat, sollte sich das Projekt jetzt starten lassen. Statt der in **Bild 2** gezeigten Ansicht zeigt die Seite nun ihr anderes Gesicht und behauptet, sie sei PostgreSQL. Tut sie das wirklich? ►

DEV ACADEMY

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN

2 TAGE
REMOTE/INHOUSE

Entity Framework Core

- Grundlagen eines O/R-Mappers
- Unterschiede von EF Core und Entity Framework
- Plattformunabhängigkeit - Einsatz von EF Core und .NET Core
- Erstellen eines Datenbankmodells mit Code-First
- Konventionen beim Arbeiten mit EF Core
- Datenbankabfragen mit LINQ
- Performancetuning mit EF Core

TRAINING!

Christian Giesswein
Trainer

● Listing 4: Migration für PostgreSQL

```
protected override void Up(
    MigrationBuilder migrationBuilder)
{
    migrationBuilder.AlterDatabase()
        .Annotation(
            "Npgsql:PostgresExtension:citext", ".,.");

    migrationBuilder.AlterColumn<string>(
        name: "Name",
        schema: "example",
        table: "WhoAmICollection",
        type: "citext",
        nullable: false,
        oldClrType: typeof(string),
        oldType: "text");
}

protected override void Down(
    MigrationBuilder migrationBuilder)
{
    migrationBuilder.AlterDatabase()
        .OldAnnotation(
            "Npgsql:PostgresExtension:citext", ".,.");

    migrationBuilder.AlterColumn<string>(
        name: "Name",
        schema: "example",
        table: "WhoAmICollection",
        type: "text",
        nullable: false,
        oldClrType: typeof(string),
        oldType: "citext");
}
```

Nein, statt einer pfiffig und exzellent gestalteten Seite präsentiert der Browser eine Fehlermeldung, die sinngemäß lautet: „Ey, ich hab keine Daten!“ Frei übersetzt aus der Exception-Message einer `InvalidOperationException: Sequence contains no elements`.

Hoppla, ein Problem

So ziemlich jede Anwendung verfügt über eine Suche. Nehmen wir mal an, Sie haben eine Personen-Tabelle und wollen außer *Hans* vielleicht auch *Franz* oder *Fritz* oder so finden. Ihre Abfrage lautet:

```
var query = from p in dbContext.Persons
    where p.Name.StartsWith('fr')
    select p;
```

Wie gewohnt, werden Franz und Fritz gefunden, bis die Qualitätssicherung auf den Gedanken kommt, die Anwendung

mal mit der neuen PostgreSQL-Datenbank zu verbinden, und auf einmal findet die Suche nichts mehr!

Das liegt einfach daran, dass sich MS-SQL in der Regel nicht so sehr um Groß- und Kleinschreibung kümmert wie PostgreSQL. Wie bringt man PostgreSQL jetzt bei, ebenfalls Franz und Fritz zu finden, wenn der Suchparameter 'fr' lautet? Nun, die Lösung ist zum Glück recht einfach: Dem *Model-ConfigurationBuilder* muss lediglich mitgeteilt werden, dass der .NET-Datentyp *string* in den PostgreSQL-Datentyp *citext* zu übersetzen ist. Entnehmen Sie Listing 3 die Umsetzung. Vor dem nächsten Anwendungsstart muss wie oben beschrieben eine Migration für PostgreSQL hinzugefügt werden. Welchen Code das dotnet-Tool hier erzeugt, zeigt Listing 4.

Jetzt lässt sich die Anwendung fehlerfrei starten. Zum Abschluss hier noch die Abfrage, die durch die Änderung des Datentyps einer kleinen Wunderheilung unterzogen wurde:

```
var query = from q in context.WhoAmICollection
    where q.Name.StartsWith("m") || q.Name.StartsWith("p")
    select new WhoAmIDto(q.Id, q.Name);
return await query.FirstAsync();
```

Fazit

Bekanntlich führen viele Wege nach Rom, und der hier dargestellte ist nicht der einzig mögliche. Es war aber auf diesem Wege durchaus machbar, mit minimalen Eingriffen an einer bestehenden Anwendung einen weiteren Datenbankanbieter hinzuzufügen. Lediglich das Erstellen der benötigten Migrationen war eine kleine Fleißarbeit. Es bedurfte keiner weiteren Tests, wenn die Anwendung mit dem MS SQL Server verbunden war, denn da hatte sich nichts geändert. Der zusätzliche Testaufwand für PostgreSQL war zu vernachlässigen: Per Konvention testet die Qualitätssicherung an geraden Tagen auf MS SQL, an ungeraden auf PostgreSQL. Nur die Entwickler sind traurig, weil sie nun immer zwei Migrationen erstellen müssen, wenn sich das Datenbankschema ändert. Aber die sind ja Kummer gewohnt ... ■

[1] PostgreSQL, <https://www.postgresql.org>

[2] SQL Server 2022, www.dotnetpro.de/SL2503EFCore1

[3] André Kappes, *Modulare Monolithe statt Microservices*, www.dotnetpro.de/SL2503EFCore2

[4] Microsoft Learn, *Migrations with Multiple Providers*, www.dotnetpro.de/SL2503EFCore3

[5] Npgsql Entity Framework Core Provider, www.dotnetpro.de/SL2503EFCore4



Christian Sender

ist bei der sycat IMS GmbH in Hannover als Softwareentwickler im Frontend und Backend unterwegs. Sie erreichen ihn unter c.sender@sycat.com.

dnpCode

A2503EFCore

CLOUD-BASIERTE BACKENDS FÜR .NET-MAUI-CROSS-PLATTFORM-APPS

Wolkenpower

Welche Kriterien kennzeichnen ein gutes Backend für Apps mit .NET MAUI, und welche Möglichkeiten haben wir, wenn es um die Auswahl eines passenden Backends geht?

Die Entwicklung moderner Anwendungen bringt neue Anforderungen und Herausforderungen mit sich. In besonderem Maß gilt das, wenn es um die Erstellung plattformübergreifender Apps geht, die auf Mobilgeräten, Tablets und Desktops gleichermaßen funktionieren sollen. Eine größere Herausforderung ist dabei die Bereitstellung eines leistungsfähigen, skalierbaren und kosteneffizienten Backends, das die verschiedenen Plattformen einheitlich unterstützt.

In mobilen Anwendungen spielt das Backend eine wesentlich größere Rolle als der Zugriff auf klassische Datenbanken. Während in klassischen Desktop- oder Webanwendungen der direkte Datenbankzugriff ausreicht, benötigen mobile Apps oft ein Backend, das zusätzliche Funktionen und Flexibilität bietet. Die Gründe dafür lassen sich in mehrere zentrale Aspekte unterteilen:

- **Verteilte Datenhaltung und Synchronisation:** Mobile Anwendungen werden typischerweise auf Geräten genutzt, die nicht immer eine konstante Internetverbindung haben. Ein modernes Backend ermöglicht die Speicherung und Synchronisation von Daten, sodass Anwender auch offline arbeiten können. Bei einer erneut aufgebauten Verbindung werden die Daten automatisch mit dem Backend synchronisiert, was bei einem reinen Datenbankzugriff so nicht möglich ist.
- **Authentifizierung und Benutzerverwaltung:** In einer mobilen App ist die Benutzerverwaltung oft komplexer, da viele Apps eine individuelle Benutzererfahrung bieten und persönliche Daten verwalten. Ein Backend ermöglicht die Integration von Authentifizierungsdiensten wie OAuth, Google- oder Facebook-Login (Social-Media-Login) und bietet gleichzeitig eine zentrale Verwaltung der Benutzerdaten, während eine klassische Datenbank für diese Aufgaben häufig unzureichend ist.
- **Push-Benachrichtigungen und Real-Time-Updates:** Mobile Nutzer erwarten Benachrichtigungen und Updates in Echtzeit, sei es für Nachrichten, Angebote oder andere Ereignis-

se. Diese Funktionalität erfordert eine serverseitige Infrastruktur, die in klassischen Datenbanklösungen nicht verfügbar ist. Ein Backend ermöglicht hingegen den Versand von Push-Benachrichtigungen und stellt Echtzeit-APIs bereit, um Benutzer unmittelbar zu informieren.

- **Datenschutz und Sicherheit:** Daten in mobilen Anwendungen müssen häufig verschlüsselt, anonymisiert oder auf andere Weise vor unberechtigtem Zugriff geschützt werden, besonders wenn es um persönliche oder sensible Daten geht. Ein modernes Backend bietet hierfür die erforderlichen Sicherheitsmechanismen und Compliance-Unterstützung, die klassische Datenbanken in der Regel nicht direkt mitbringen.
- **Skalierbarkeit und Performance:** Mobile Anwendungen haben oft eine wachsende oder dynamische Benutzerbasis, die schnelle Reaktionszeiten und eine hohe Verfügbarkeit des Backends erfordert. Ein Backend as a Service (BaaS) oder eine moderne Backend-Architektur kann je nach ►

● **Tabelle 1: Kriterien für Auswahl eines BaaS-Dienstes für .NET MAUI**

Kriterium	Beschreibung
Kompatibilität mit .NET MAUI	Der BaaS-Anbieter sollte ein SDK oder ein API anbieten, das vollständig kompatibel mit .NET und .NET MAUI ist, um plattformübergreifend auf Android, iOS, Windows und macOS genutzt werden zu können.
Funktionsumfang	Der BaaS-Dienst sollte Kernfunktionen wie Datenbank, Authentifizierung, Push-Benachrichtigungen, Dateispeicherung und Offline-Synchronisation unterstützen, um den Anforderungen der App gerecht zu werden.
Skalierbarkeit und Performance	Der BaaS-Dienst sollte flexibel skalierbar sein, um Lastspitzen und wachsende Nutzerzahlen zu bewältigen, und eine hohe Performance auch bei hoher gleichzeitiger Nutzung gewährleisten.
Sicherheit und Compliance	Der Anbieter sollte Sicherheitsstandards wie Datenverschlüsselung und Zugriffskontrollen bieten und wichtige Datenschutzvorgaben (zum Beispiel die DSGVO) einhalten, um die App und die Benutzerdaten zu schützen.
Support und Wartung	Ein zuverlässiger technischer Support, Community-Ressourcen und regelmäßige Updates des Dienstes sind erforderlich, um eine stabile und unterstützte Nutzung des BaaS-Dienstes sicherzustellen.
Kosten und Preismodell	Flexible „Pay as you go“-Kostenmodelle und erschwingliche Einstiegsangebote für kleinere Anwendungen oder Entwicklerteams sollten verfügbar sein, um Budgetanforderungen zu erfüllen.
Performance und Verfügbarkeit	Für Anwendungen, die eine hohe Verfügbarkeit und Leistungsfähigkeit benötigen, sollte der Anbieter eine stabile Infrastruktur und global verfügbare Server anbieten, um Echtzeit-Funktionen zu unterstützen.



Nutzung von AWS über das AWS SDK for .NET (Bild 1)

Bedarf skaliert werden, um Lastspitzen abzufangen. Eine klassische Datenbank wäre hier oft ein Flaschenhals und würde diesen Anforderungen nicht gerecht.

Zusammengefasst: Ein modernes Backend sorgt in mobilen Anwendungen dafür, dass die App umfassendere Funktionen, bessere Benutzererlebnisse und erhöhte Sicherheit bietet – weit über die Möglichkeiten eines einfachen Datenbankzugriffs hinaus.

In diesem Artikel wird untersucht, wie BaaS-Dienste effektiv in .NET-MAUI-Projekte integriert werden können. Wir betrachten verschiedene BaaS-Anbieter, typische Einsatzmöglichkeiten und mögliche Herausforderungen und geben eine detaillierte Anleitung zur Implementierung.

BaaS in .NET MAUI

Eine der größten Stärken von .NET MAUI ist die Möglichkeit, Anwendungen für mehrere Plattformen – insbesondere Android, iOS, Windows und macOS – mit einer einzigen Codebasis zu entwickeln. Daher muss ein BaaS, das in .NET MAUI genutzt werden soll, ebenfalls plattformübergreifend arbeiten und Kompatibilität mit allen Zielsystemen sicherstellen. Für die Integration eines BaaS in .NET-MAUI-Anwendungen stehen zwei Optionen zur Verfügung: die Nutzung eines SDK oder der direkte Zugriff über HTTP/REST-APIs. Beide Methoden haben spezifische Vorteile, die je nach Projektanforderungen und Verfügbarkeit variieren.

Ein SDK (Software Development Kit) bietet eine besonders effiziente Möglichkeit, Backend-Funktionen in Ihre Anwendung einzubinden. Es enthält vorkonfigurierte Funktionen und Methoden, die die Nutzung des BaaS erleichtern und die Implementierung beschleunigen. Die wichtigsten Vorteile der SDK-Integration liegen in der Einfachheit und Effizienz: Entwickler können direkt auf häufig benötigte Funktionen wie Authentifizierung, Datenbankzugriff oder Push-Benachrichtigungen zugreifen, ohne eine eigene Schnittstelle für das Backend schreiben zu müssen. Ein SDK wird in der Regel fortlaufend vom BaaS-Anbieter aktualisiert und stellt sicher, dass Sicherheitsstandards und Compliance-Anforderungen automatisch berücksichtigt werden.

Darüber hinaus ist ein SDK häufig so konzipiert, dass es alle Plattformen unterstützt, die von .NET MAUI genutzt werden, und Entwicklern damit eine einheitliche Grundlage für die

Integration des BaaS in Android-, iOS-, Windows- und macOS-Anwendungen bietet. Allerdings kann ein SDK auch gewisse Einschränkungen mit sich bringen, da Entwickler auf die vorgefertigten Funktionen angewiesen sind und weniger Flexibilität haben. Zudem kann ein SDK die Größe der App erhöhen, was für mobile Anwendungen oft ein relevanter Faktor ist.

Zusammengefasst: Es ist von großem Vorteil, wenn es ein SDK gibt, das mit der aktuellen Version von .NET kompatibel ist und regelmäßig gepflegt wird. Es wird üblicherweise über ein NuGet-Package zur Verfügung gestellt.

Die direkte Integration eines BaaS über HTTP/REST-APIs bietet mehr Flexibilität und Kontrolle, da Sie jeden Aspekt des API-Zugriffs manuell steuern können. Über REST-APIs lässt sich das Backend durch HTTP-Anfragen erreichen, die Daten werden in der Regel über das JSON-Format ausgetauscht. Diese Methode ist sinnvoll, wenn der BaaS-Anbieter kein SDK für .NET MAUI bereitstellt oder wenn die Anwendung nur bestimmte Backend-Dienste nutzen soll.

Die Vorteile dieser Methode liegen in der Unabhängigkeit und der kleineren App-Größe, da keine zusätzlichen SDK-Bibliotheken eingebunden werden müssen. REST-APIs ermöglichen es Entwicklern, nur die benötigten Funktionen einzusetzen und spezifische Anpassungen vorzunehmen. Allerdings erfordert die Integration über REST-APIs einen höheren manuellen Aufwand, da man die Authentifizierung, die Fehlerbehandlung und das Handling von API-Antworten selbst implementieren muss. Ebenso muss man sicherstellen, dass die Implementierung bei Änderungen im API des BaaS-Anbieters stets aktuell bleibt, was zusätzlichen Wartungsaufwand verursachen kann.

Die Wahl zwischen einem SDK und der Nutzung von REST-APIs hängt von den spezifischen Anforderungen des Projekts ab. Ein SDK ist ideal, wenn eine schnelle und einfache Integration von Backend-Funktionen ohne tiefere Backend-Kenntnisse benötigt wird und der Anbieter eine plattformübergreifende Unterstützung für .NET MAUI bietet. Die REST-API-Integration hingegen bietet sich an, wenn Flexibilität, geringe App-Größe oder detaillierte Anpassungen erforderlich sind oder wenn kein passendes SDK vorhanden ist. Beide Methoden ermöglichen eine effektive Nutzung von BaaS in .NET MAUI und unterstützen die Entwicklung leistungsfähiger, plattformübergreifender Anwendungen.

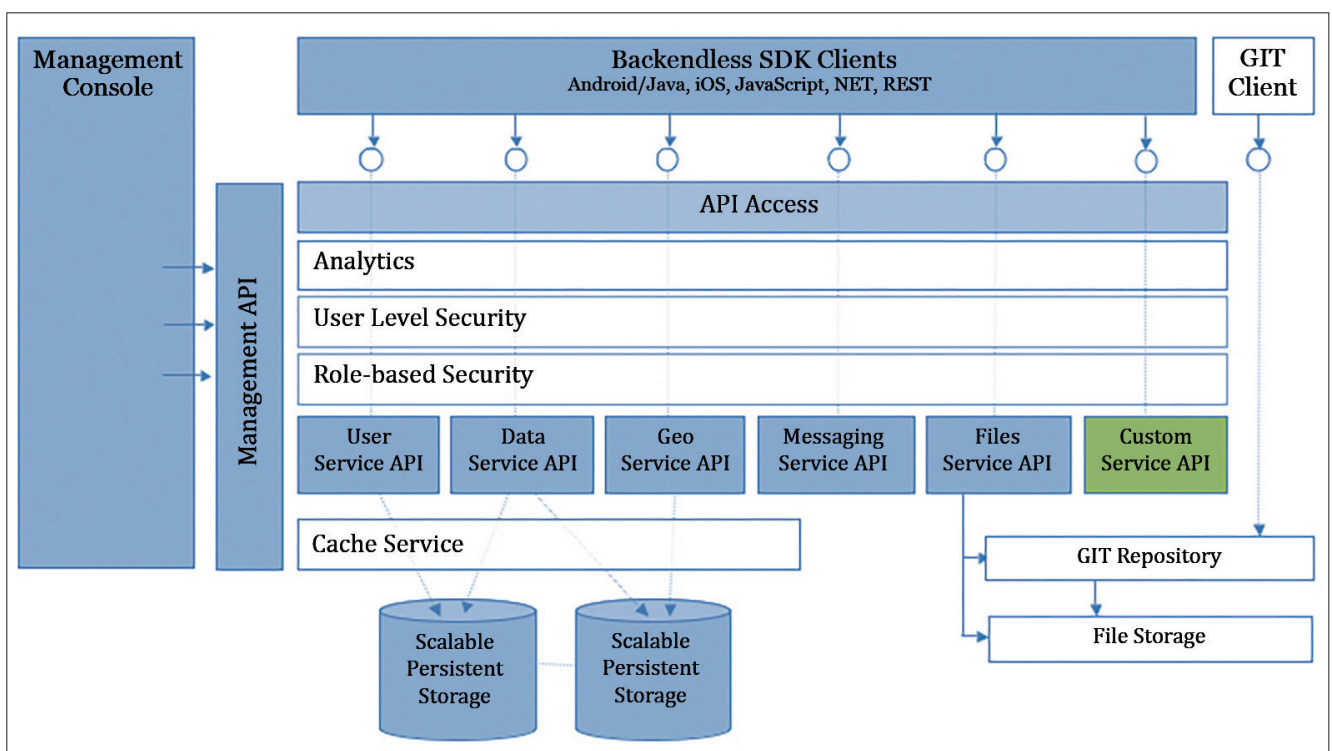
Einen passenden BaaS-Dienst auswählen

Die Wahl des passenden BaaS-Anbieters ist entscheidend. Diese Auswahl sollte auf einer sorgfältigen Analyse der angebotenen Funktionen, der Kompatibilität mit .NET MAUI sowie der langfristigen Unterstützung basieren. In **Tabelle 1** werden die wichtigsten Kriterien und Überlegungen zur Auswahl eines BaaS-Anbieters beschrieben. Für die folgenden konkreten BaaS-Anbieter, die für einen Einsatz im .NET-Universum geeignet erscheinen, sehen wir uns deren Cloud-Speicher und -Funktionen genauer an:

- **Firebase von Google [1]:** Firebase ist eine umfassende BaaS-Plattform von Google, die eine Vielzahl von Diensten für die App-Entwicklung bietet. Dazu gehören Echtzeit-Datenbanken, Authentifizierung, Cloud-Funktionen, Hosting und Analysen. Firebase unterstützt plattformübergreifende Entwicklungen und kann in .NET-MAUI-Anwendungen integriert werden. Die Hauptfunktionen sind: Echtzeit-Datenbank und Cloud Firestore, Benutzerauthentifizierung, Push-Benachrichtigungen und Analysetools. Firebase bietet ein kostenloses Einstiegspaket mit begrenzten Ressourcen. Für die erweiterte Nutzung gibt es ein Abrechnungsmodell, bei dem die Kosten je nach Verbrauch variieren. Preisinformationen sind unter [2] verfügbar.
- **AWS Amplify (Amazon):** AWS Amplify [3] ist eine Entwicklungsplattform von Amazon Web Services, die Tools und Dienste für die Erstellung von mobilen und Webanwendungen bietet. Unterstützt werden Funktionen wie Authentifizierung, APIs, Datenbanken, Storage und Hosting. Die Integration in .NET (auch .NET MAUI) erfolgt über das AWS SDK for .NET, vergleiche **Bild 1** [4]. Es werden umfassende Funktionen angeboten, darunter Dateispeicherung,

Datenbankverwaltung mit DynamoDB, Benutzerauthentifizierung, GraphQL- und REST-APIs, Hosting und CI/CD. AWS Amplify bietet ein kostenloses Nutzungskontingent für das erste Jahr. Anschließend werden die Kosten basierend auf der Nutzung berechnet [5].

- **Azure (Microsoft):** Bisher war das Visual Studio App Center der zentrale Anlaufpunkt für die Administration und Einrichtung von Backend-Services für mobile und Desktop-Apps. Voraussichtlich mit Wirkung zum 31. März 2025 wird die Unterstützung für das Visual Studio App Center vollständig eingestellt, einschließlich der darüber bereitgestellten Dienste. Entwickler, die bisher das App Center genutzt haben, müssen ihre Lösungen auf andere Azure-Dienste umstellen, um weiterhin ein robustes Backend bereitzustellen [6]. Das sind beispielsweise die Dienste Azure Cosmos DB für skalierbare Datenbanklösungen, Azure Blob Storage für die Speicherung von Dateien und Medien sowie Azure Functions für serverlose Logik. Mit allen diesen Diensten lassen sich moderne, plattformübergreifende Backends erstellen. Sie sind mit .NET MAUI kompatibel und bieten flexible Skalierung, hohe Verfügbarkeit und umfassende Sicherheitsfunktionen.
- **Backendless:** Backendless ist eine BaaS-Plattform, die eine Vielzahl von Backend-Diensten für mobile und Webanwendungen bietet [7]. Dazu gehören Datenbankverwaltung, Benutzerauthentifizierung, Push-Benachrichtigungen und Geodienste. Die Integration in .NET MAUI erfolgt über ein Client-API. Die Kernfunktionen dieses Dienstes sind: Datenbankverwaltung, Benutzerauthentifizierung, Push-Benachrichtigungen, Geodienste und Codeless-Logik. Backendless bietet ein kostenloses Paket mit begrenzten ►

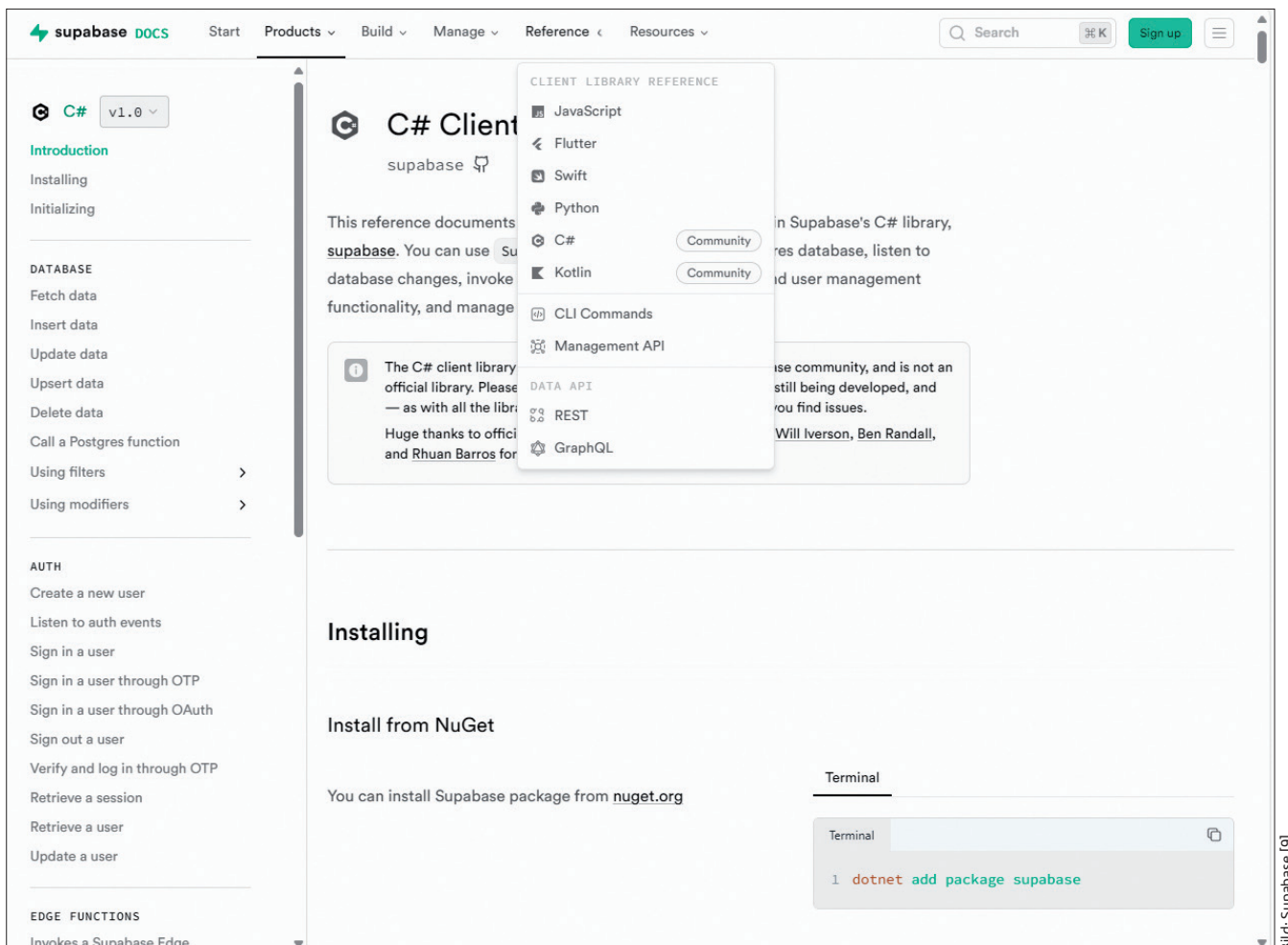


Architektur von Backendless (Bild 2)

Quelle: Backendless [7]

Ressourcen. Für erweiterte Nutzung gibt es verschiedene Preismodelle [8]. Die Architektur dieser umfassenden Backend-as-a-Service-Lösung (BaaS) wird deutlich, wenn man sich das Diagramm in **Bild 2** ansieht. Es zeigt die verschiedenen Ebenen und Komponenten, welche die Backendless Plattform strukturieren und deren Funktionalität definieren. Im oberen Bereich befindet sich die Management Console, die als zentrale Benutzeroberfläche dient, über die Entwickler Anwendungen verwalten, Dienste konfigurieren, Daten bearbeiten und Sicherheitsrichtlinien festlegen können. Direkt darunter ist der Bereich API Access angesiedelt, der den Zugriff auf die verschiedenen Dienste über SDK-Clients oder REST-APIs ermöglicht. Unterstützt werden zahlreiche Plattformen wie Android/Java, iOS, JavaScript, .NET und REST, sodass eine plattformübergreifende Integration gewährleistet ist. Eine zentrale Rolle spielt die Sicherheits- und Analytik-Ebene. Hier sorgt die User Level Security dafür, dass der Zugriff auf Daten und Dienste benutzerbasiert gesichert ist, während die Role-based Security eine rollenbasierte Zugriffssteuerung bietet. Darüber hinaus ermöglicht die Analytics-Komponente eine umfassende Überwachung und Analyse der Dienste, um deren Nutzung und Performance zu optimieren. Die nächste Ebene umfasst die Kerndienste, die über die jeweiligen APIs

bereitgestellt werden. Das User Service API verwaltet Benutzeranmeldungen, Authentifizierung und Benutzerprofile. Das Data Service API ermöglicht CRUD-Operationen (Create, Read, Update, Delete) auf der Datenbank, während das Geo Service API Funktionen für die Verwaltung und Abfrage von Standortdaten bietet. Für die Kommunikation sorgt das Messaging Service API, das Echtzeit-Nachrichtenübermittlung, Push-Benachrichtigungen und Pub/Sub-Kommunikation unterstützt. Darüber hinaus ermöglicht das Files Service API das Hochladen, Speichern und Verwalten von Dateien. Ergänzt wird diese Ebene durch das Custom Service API, das Entwicklern die Definition und Bereitstellung benutzerdefinierter Dienste erlaubt. Zur Unterstützung der Hauptdienste bietet die Plattform zusätzlich eine Cache-Service-Komponente, die häufig genutzte Daten zwischenspeichert, um die Leistung zu steigern. Die Scalable-Persistent-Storage-Lösung sorgt dafür, dass die Plattform mit wachsenden Anforderungen skaliert und eine zuverlässige Speicherung von Daten und Dateien gewährleistet. Im rechten Bereich des Diagramms zeigt die Plattform ihre Git-Integration. Über den Git-Client können Dateien und Konfigurationen versioniert und verwaltet werden, wobei der Zugriff auf ein Git-Repository und zusätzlichen File Storage möglich ist. Diese Integration ver-



Dokumentation von Supabase – aufgeräumt und übersichtlich (**Bild 3**)

einfacht die Zusammenarbeit und die Versionskontrolle im Entwicklungsprozess erheblich. Insgesamt verbindet das Management API alle Komponenten und bietet eine einheitliche Schnittstelle, um auf die Funktionen und Daten der Plattform zuzugreifen.

- **Supabase:** Supabase ist eine Open-Source-BaaS-Plattform, die auf PostgreSQL basiert und Echtzeit-Datenbanken, Authentifizierung, Storage und Edge-Funktionen bietet [9]. Die Integration in .NET MAUI erfolgt über eine Client-Library, die nicht offiziell durch den Anbieter bereitgestellt wird, sondern durch die Community erstellt wurde. Die Hauptfunktionen sind: Echtzeit-PostgreSQL-Datenbank, Benutzerauthentifizierung und Dateispeicherung. Supabase bietet ein kostenloses Paket mit begrenzten Ressourcen. Für erweiterte Nutzung gibt es verschiedene Preismodelle [10]. Ein Blick in die Dokumentation macht deutlich, dass es sich um einen Dienst handelt, der versucht, Komplexität zu vermeiden, und in dessen Fokus die Kernfunktionen eines Backends für eine (mobile) App stehen. Die Architektur des Dienstes wirkt sehr übersichtlich und aufgeräumt (Bild 3).

In den nachfolgenden Textabschnitten beschäftigen wir uns mit den Möglichkeiten und der Integration der Firebase- und der Azure-Cloud in .NET-MAUI-Apps.

Firestore von Google

Folgende Features sprechen für die Nutzung von Firestore:

- Integration in die umfassenden Cloud-Dienste von Google.
- Leistungsfähige Funktionen, welche die typischen Anforderungen vieler Use Cases abdecken.
- Vielfältige Verwendung und daher erprobt.
- Umfassende Dokumentation durch Google und die Community.
- Beständigkeit des Service und stetige Weiterentwicklung.
- Kostenfreie Nutzung eines umfassenden Leistungsumfangs – auch für eine eingeschränkte produktive Verwendung.

Wir wollen uns konkret den Service Firestore Realtime Database (Firestore-Echtzeitdatenbank) ansehen und beginnen dazu mit einem Überblick zu diesem Dienst. Die Firestore Realtime Database ist eine in der Cloud gehostete NoSQL-Datenbank [11], die durch diverse Client-Systeme genutzt werden kann. Dabei werden die Daten zwischen den angebundenen Clients hinweg synchronisiert (Echtzeitdatenbank) und im JSON-Format gespeichert. Vor der Verwendung muss man eine passende Struktur definieren. Die Daten werden – im Gegensatz zu einer relationalen Datenbank – nicht in einer Tabellenstruktur gespeichert, sondern als JSON-Objekte in einer Baumstruktur abgelegt. Werden Daten ergänzt, dann werden diese in die Baumstruktur eingeordnet. Dabei verfügt jedes Objekt über einen eindeutigen Schlüssel. Den Wert dieses Schlüssels kann man manuell bestimmen. Alternativ lässt sich der Schlüsselwert auch automatisch bereitstellen. Technisch gesehen kann die Datenbank mit bis zu 32 Ebenen ausgestattet werden. Dennoch ist es grundsätzlich keine gute Idee, eine derartig komplexe

```
// An index to track Ada's memberships
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      // Index Ada's groups in her profile
      "groups": {
        // the value here doesn't matter, just that the key exists
        "techpioneers": true,
        "womentechmakers": true
      }
    },
    ...
  },
  "groups": {
    "techpioneers": {
      "name": "Historical Tech Pioneers",
      "members": {
        "alovelace": true,
        "ghopper": true,
        "eclarke": true
      }
    },
    ...
  }
}
```

Bild: Firestore [11]

Eine sinnvolle Baumstruktur kann explizit auch Redundanz beinhalten (Bild 4)

Struktur zu verwenden. Neben der entstehenden Komplexität besteht ein weiterer Grund darin, dass man Lese- und Schreibberechtigungen für die einzelnen Knoten definiert. Gewährt man einem Nutzer einen Zugriff auf einen Knoten, dann hat dieser auch Zugriff auf alle darunterliegenden Ebenen. Das Ziel sollte daher sein, eine möglichst flache Struktur zu definieren.

Ebenso ist die Datenstruktur so festzulegen, dass die typischen Anfragen an die Datenbank möglichst schnell beantwortet werden können. Für eine performante Arbeit mit der Datenbank sollte es ein Maßstab sein, dass man möglichst wenige Daten aus der Baumstruktur abrufen muss, um die Anfrage zu beantworten. Dabei kann es sinnvoll sein, bewusst Redundanz in den Daten zuzulassen. Man spricht in diesem Zusammenhang auch von Denormalisierung. In der Dokumentation findet man dazu folgendes Beispiel [12]: Es geht um die gegenseitige Zugehörigkeit von Benutzern und Gruppen. Ein Nutzer kann zu keiner, einer oder mehreren Gruppen gehören. Gleichwohl umfasst eine Gruppe keine, einen oder mehrere Benutzer (N:M-Beziehung). Man legt dazu Knoten für Benutzer und für Gruppen an. Einerseits wird unterhalb eines Benutzer-Knotens die Zugehörigkeit zu den Gruppen und andererseits unterhalb eines Gruppen-Knotens die Mitgliedschaft der Benutzer angegeben (Bild 4).

Damit entsteht Redundanz bei der Datenspeicherung. Wenn ein Benutzer nicht mehr Mitglied in einer Gruppe ist, müssen sowohl der Eintrag unter dem User-Knoten als auch der Eintrag unter dem Gruppen-Knoten angepasst werden. Andererseits hat man aber den Vorteil, dass man schnell alle Mitglieder einer Gruppe ermitteln kann. Man muss nur den Knoten zur Gruppe auslesen und nicht alle – gegebenenfalls mehrere 10000 – Knoten zu den Usern nacheinander auf die Mitgliedschaft in einer Gruppe durchsuchen. Gleichwohl kann man auch jederzeit für einen bestimmten Benutzer ►

die Gruppenzugehörigkeiten feststellen und muss dazu auch nicht jede Gruppe auslesen. Daraus folgt: Die Datenstruktur muss sich nach den typischen Datenbankoperationen richten. Eine flache Hierarchie ist wichtiger als eine strenge Normalisierung.

Über Regeln kann bestimmt werden, welche Lese- und Schreibberechtigungen auf Ihre Datenbank bestehen. Diese Regeln haben wiederum Einfluss auf die Datenbankstruktur, denn die Berechtigungen werden für die einzelnen Knoten definiert. Die Regeln werden über die Firebase-Console definiert und ausschließlich serverseitig angewendet. Standardmäßig erlauben die Regeln keinen Zugriff auf die Datenbank. Es gibt folgende Regeltypen:

- **Read:** Hier wird festgelegt, wann welche Daten durch die Benutzer gelesen werden können.
- **Write:** Es wird definiert, welche Schreibberechtigungen für die Daten bestehen.
- **Validate:** Die Regel bestimmt, welches Format und Datentyp ein korrekter Datenwert hat.
- **IndexOn:** Es wird bestimmt, welches untergeordnete Element indiziert werden soll, um Sortier- und Abfrageoperationen zu unterstützen.

Die Firebase Realtime Database setzt die definierten Regeln automatisch durch, zum Beispiel mithilfe der Authentifizierungsmaßnahmen. Bei der Authentifizierung geht es darum, dass sich ein Benutzer gegenüber der Datenbank eindeutig identifiziert. Dazu verwendet man die Firebase-Authentifizierung. Diese bietet Dienste, SDKs und vorgefertigte UI-Bibliotheken an, um Benutzer zu authentifizieren. Einfache Authentifizierungen mit Passwort oder Telefonnummer werden von der Firebase-Authentifizierung ebenso unterstützt wie die Nutzung der Identitätsanbieter, zum Beispiel Google, Facebook und Twitter.

Es gibt zwei Integrationsstufen. Das Firebase Authentication SDK sorgt für eine clientseitige Integration, um die genannten

Verfahren der Authentication zu realisieren. Daneben gibt es das FirebaseUI. Dabei handelt es sich um vollständige User-Interface-Screens zur Einbindung in die eigene App. Sie bilden die typischen Use Cases der Benutzerverwaltung ab, zum Beispiel eine Anmeldung mit Benutzername und Passwort. Die Bibliotheken zu FirebaseUI findet man unter [13]. Für beide Varianten gibt es von Google kein offizielles SDK für die Nutzung mit .NET, was schade ist. Im Netz findet man jedoch Lösungsansätze; beispielsweise gibt es unter [14] einen Beitrag für die Nutzung der Firebase Authentication in .NET.

Sehen wir uns jetzt konkret an, wie wir Daten aus der .NET-App in die Firebase-Datenbank schreiben und von dort auch wieder auslesen können. Wir hatten schon erwähnt, dass es kein offizielles SDK von Google für Firebase gibt. Einen Überblick über .NET-Bibliotheken von Google findet man unter [15], beispielsweise für den Firestore-Service [16], jedoch nicht für die Firebase Realtime Database. Möchte man nicht über das generische REST-Interface mit Firebase die Daten austauschen, benötigt man eine alternative Client-Library. Eine solche findet man beispielsweise unter [17] mit der Bezeichnung *FirebaseDatabase.net*. Diese ist ein Wrapper um das REST-API der Firebase Realtime Database. Gemäß der Dokumentation der Library ist diese zu .NET Standard 2.0 kompatibel. Das letzte größere Update der Bibliothek liegt jedoch auch schon einige Jahre zurück. Dieses Problem haben wir immer dann, wenn wir kein offizielles SDK des Cloud-Anbieters nutzen können.

Um Daten aus einer App mit dem Firebase-Backend auszutauschen, durchlaufen wir die folgenden Schritte:

- Projekt in Firebase anlegen.
- Einbinden der Library in das betreffende .NET-Projekt.
- Initialisierung einer Firebase-Instanz im Client und Authentifizierung gegenüber dem Backend.
- Durchführen der CRUD-Operationen.

Starten wir mit dem Anlegen des Projekts in Firebase. Loggen Sie sich dazu in der Firebase-Console unter [18] ein und klicken Sie auf den Eintrag *Add Project*. Geben Sie einen Projektnamen an und legen Sie fest, ob Google Analytics für das Projekt aktiviert werden soll (Bild 5). Aktivieren Sie die Realtime Database. Wir haben die Möglichkeit, die Authentifikationsmethode – zum Beispiel über das Firebase Admin SDK – zu aktivieren. Es wird der Name der Datenbank und das Secret – ein geheimer Schlüssel – angezeigt. Dieses Secret brauchen wir später, um uns aus der mobilen App bei Firebase zu authentifizieren. Ebenso müssen wir noch eine Basisregel aufstellen, damit auf das Firebase-Backend zugegriffen werden kann. Klicken Sie dazu auf den Eintrag *Realtime Database*, dann auf den Menüpunkt *Regeln* und erfassen Sie die folgende Datenbankregel:

```
{
  "rules": {
    ".read": "auth!=null",
    ".write": "auth!=null"
  }
}
```

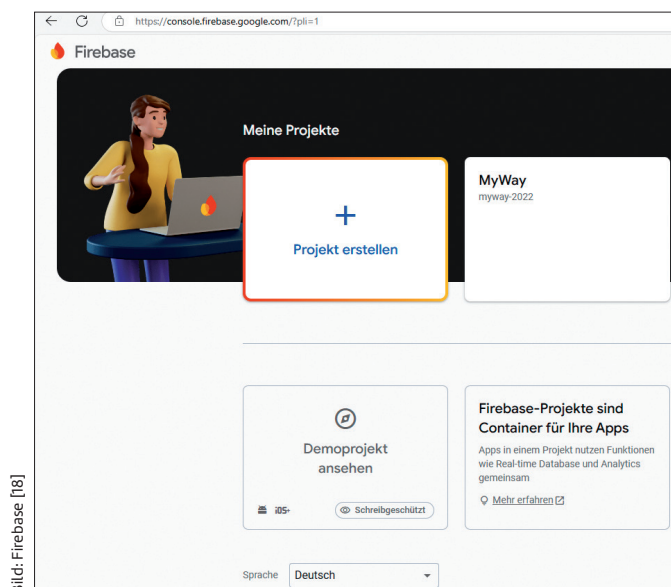


Bild: Firebase [18]

Neues Projekt in der Firebase-Console anlegen (Bild 5)

Diese Regel ist zunächst sehr pauschal und besagt, dass ein lesender und ein schreibender Zugriff über alle Objekte hinweg besteht. In diesem Fall ist ein Benutzer pauschal über die App mithilfe des Secrets identifiziert, das der Client an Firebase übermittelt. Bei dieser Art von Authentifikation werden wir als Datenbank-Administrator eine E-Mail erhalten, dass diese Zugriffsregel zu pauschal ist und keine ausreichende Sicherheit bietet. Dieser Hinweis soll uns daran erinnern, dass wir diesen Punkt in einer Produktions-App noch anpassen müssen. Den zweiten Schritt bildet die Einbindung der FirebaseDatabase.net-Bibliothek in die Client-App über das NuGet-Package des SDK. Danach folgt Schritt Nummer drei, die Initialisierung einer Instanz von Firebase im Client und die Authentifizierung mittels Secret. Für alle Aufgaben im Zusammenhang mit Firebase erstellen wir jeweils eine eigene Klasse und nennen diese beispielsweise *FirebaseHelper*. Im Konstruktor dieser Klasse initialisieren wir den Firebase-Client. Dazu brauchen wir den Datenbank-URL und das Datenbank-Secret. Der Datenbank-URL lautet *[Name der Datenbank].europe-west1.firebaseio.com*, und das *App-Secret* finden wir in der Firebase-Console. Der Konstruktor unserer Klasse sieht nun wie folgt aus:

```
public FirebaseHelper()
{
    var auth = "Secret";
    firebaseClient = new FirebaseClient(
        "https://[Name der Datenbank].europe-west1.firebaseio.com",
        new FirebaseOptions
        {
            AuthTokenAsyncFactory = () => Task.FromResult(auth)
        });
}
```

Das Datenbank-Secret wird als Parameter bei der Initialisierung eines Objekts der Klasse *FirebaseClient* übergeben. Mit diesem Objekt – hier *firebaseClient* – können nun die schreibenden und lesenden Zugriffe auf die Datenbank durchgeführt werden. Diese Datenbankoperationen betrachten wir im vierten Schritt. Starten wir mit einer Schreiboperation: Folgende Datentypen werden durch Firebase unterstützt: *String*, *Long*, *Double*, *Boolean*, *Map<String, Object>*, *List<Object>*, *Null* und *Object*. Neben den Basisdatentypen wie einer Zeichenkette oder einem Zahlenwert wird man häufig ein Objekt als Ganzes im Baum speichern. Das kann beispielsweise ein Objekt einer Klasse *Location* mit den Eigenschaften *Longitude (double)*, *Latitude (double)* und *DateTime (DateTime)* sein. Neben den geografischen Angaben wird auch die aktuelle Zeit in der Datenstruktur gespeichert. Es kann sich beispielsweise um Tracking-Informationen (Orts- und Zeitangaben) eines Benutzers handeln, die von der App an das Backend übermittelt werden. Gehen wir jetzt davon aus, dass wir ein Objekt der Klasse *Location* haben, dann können wir die Daten mit der Methode *WriteLocationPoint(...)* in die Firebase-Datenbank schreiben:



Datentupel in Firebase gespeichert (Bild 6)

```
public async void WriteLocationPoint(
    Location location, string username)
{
    await firebaseClient.Child(username)
        .PostAsync(location);
}
```

Die Methode gehört zur Klasse *FirebaseHelper* und hat zwei Argumente:

- *username*: Das ist die Angabe des Pfades in der Datenbank, in welchem die zu schreibenden Werte beziehungsweise Objekte abgelegt werden. Wir wollen in unserem Beispiel die Datenobjekte nach den Namen der Benutzer (*username*) ablegen.
- *location*: Hierbei handelt es sich um ein Objekt der Klasse *Location*, das im angegebenen Pfad der Datenbank abgelegt wird.

Das Ergebnis einer solchen Schreiboperation ist in Bild 6 zu sehen. Die Baumstruktur enthält die Pfade mit den Bezeichnungen der Benutzer (blaue Markierung). Unterhalb der Pfade werden die Objekte gespeichert. Die Objektstruktur ist grundsätzlich frei und kann sich auch von Objekt zu Objekt unterscheiden (Schemafreiheit). Das ist jedoch in der Regel nicht sinnvoll, und man könnte es über eine Validierungs-Regel einschränken. Jedes Objekt erhält eine automatische ID (Key), die das Objekt eindeutig kennzeichnet. Wenn man die ID nicht selbst generiert und verwaltet (Eindeutigkeit), wird diese von Firebase beim Schreibvorgang eigenständig erzeugt (rote Markierung). Ein Objekt besteht wiederum aus ►

den Einzelwerten für die Eigenschaften, in diesem Beispiel *Longitude*, *Latitude* und *DateTime* (grüne Markierung). Beachten Sie: Alle Transaktionen auf die Datenbank werden asynchron ausgeführt (*async*, *await*). Lesen- und schreibende Zugriffe auf einen Server – in diesem Fall auf das Firebase-Backend – sind lang laufende Transaktionen. Die Ausführung kann einen Augenblick in Anspruch nehmen. Damit das User-Interface stets reaktionsfähig bleibt, werden diese Transaktionen asynchron ausgeführt.

Kommen wir zum Lesen der Daten. Auch dazu ein einfaches Beispiel: Gehen wir davon aus, dass wir den Speicherbaum nach Benutzern (*username*) aufgebaut haben. Wir wollen das letzte Datenobjekt aus einem bestimmten Knoten abrufen. In der Praxis könnte es sich zum Beispiel um die letzte gespeicherte Position (Geodaten) handeln. Das sieht dann wie folgt aus:

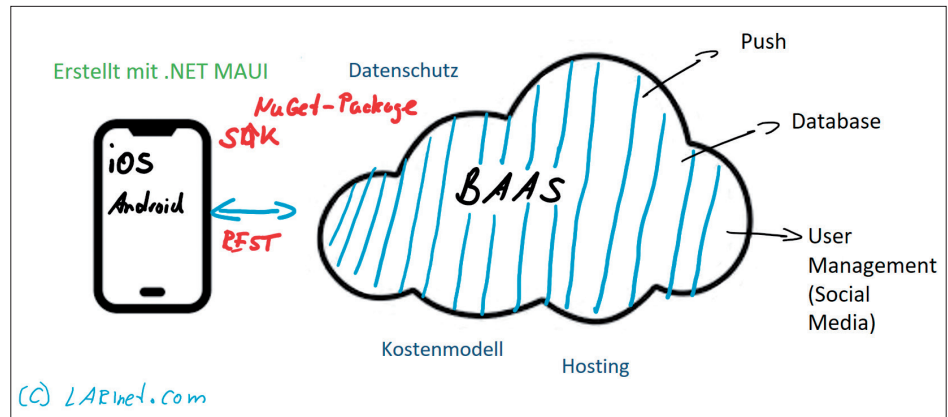
```
public async Task<Location> GetLastLocation(
    string username)
{
    return (await firebaseClient
        .Child(username)
        .OnceAsync<Location>()).Select(item => new Location
    {
        Latitude = item.Object.Latitude,
        Longitude = item.Object.Longitude,
        DateTime = item.Object.DateTime
    }).ToList().Last();
}
```

Zunächst wird der betreffende Knoten mittels *.Child(username)* adressiert. Dann erfolgt ein Abruf der Objekte mithilfe des Befehls *.OnceAsync<Location>().Select(...)*. Dabei werden die abgerufenen Daten den neu erstellten Objekten der Klasse *Location* zugewiesen.

Uns interessiert das zuletzt eingefügte Objekt der Liste, daher wenden wir den Befehl *Last()* an. Der Abruf der Daten kann hier sehr komfortabel mit LINQ, der SQL-ähnlichen Syntax in C#, erfolgen.

Azure als flexibles Backend

Die Erstellung eines Backends für .NET-MAUI-Anwendungen erfordert die Kombination verschiedener Azure-Dienste, die jeweils spezifische Aufgaben übernehmen. Azure bietet eine breite Palette von Diensten, die flexibel kombiniert werden können, um typische Backend-Funktionen wie Datenverwaltung, Benutzerauthentifizierung, Dateispeicherung und serverlose Verarbeitung abzudecken. Diese modulare Herangehensweise ermöglicht es, die Backend-Architektur an die spezifischen Anforderungen Ihrer App anzupassen,



Kernaspekte rund um die BaaS-Nutzung in .NET MAUI (Bild 7)

ohne auf eine vorgefertigte BaaS-Lösung angewiesen zu sein. Der Nachteil dieses Vorgehensweise ist ein höherer Aufwand bei der Auswahl und Einrichtung der Dienste.

Eine Kombination könnte wie folgt aussehen:

- **Azure Cosmos DB:** Wird als skalierbare NoSQL-Datenbank zur Speicherung von App-Daten verwendet.
- **Azure Active Directory B2C:** Übernimmt die Authentifizierung und Benutzerverwaltung mit Social-Logins und Sicherheitsfunktionen.
- **Azure Blob Storage:** Dient zur Speicherung von Dateien wie Bildern, Videos oder Dokumenten.
- **Azure Functions:** Ermöglicht serverlose Logik für API-Endpoints oder ereignisgesteuerte Aufgaben.

Diese Dienste kommunizieren über REST-APIs oder (einfacher) SDKs mit der .NET-MAUI-App und bieten skalierbare Lösungen für moderne plattformübergreifende Anwendungen. Im Folgenden werden die Aufgaben, Einrichtung und Integration dieser Dienste detaillierter beschrieben.

Azure Cosmos DB – Datenbankverwaltung

Die Aufgabe ist die Speicherung und Verwaltung von App-Daten wie Benutzerprofilen oder App-Inhalten. Richten Sie sich nach folgender Schrittfolge: Melden Sie sich im Azure-Portal an und erstellen Sie eine neue Azure-Cosmos-DB-Instanz. Wählen Sie das API, zum Beispiel *Core (SQL)*, basierend auf den Anforderungen Ihrer App. Konfigurieren Sie globale Replikationsoptionen, um eine hohe Verfügbarkeit zu gewährleisten. Zur Integration in .NET MAUI installieren Sie das NuGet-Paket *Microsoft.Azure.Cosmos*. Hier der Beispielcode für Datenzugriff:

```
using Microsoft.Azure.Cosmos;

var cosmosClient =
    new CosmosClient("<connection-string>");

var database = await
    cosmosClient.CreateDatabaseIfNotExistsAsync(
        "AppDatabase");

var container = await
    database.Database.CreateContainerIfNotExistsAsync(
        "Users", "/partitionKey");
```

```
var user = new { id = "1", name = "John Doe",
    partitionKey = "region1" };
await container.Container.CreateItemAsync(
    user, new PartitionKey("region1"));
```

Weiterführende Informationen sind unter [19] zu finden.

Azure Active Directory B2C – Benutzerverwaltung

Mithilfe dieses Dienstes kann die Verwaltung der Benutzer und die Authentifizierung mit Unterstützung für Social-Media-Logins (Google, Facebook und dergleichen) durchgeführt werden. Zur Einrichtung erstellen Sie eine Azure-Active-Directory-B2C-Instanz. Konfigurieren Sie Benutzerflows, beispielsweise die Anmeldung, Registrierung und Passwortzurücksetzung, und fügen Sie externe Identitätsanbieter wie Google oder Facebook hinzu.

Die Integration in eine .NET-MAUI-App erfolgt über das NuGet-Paket `Microsoft.Identity.Client`. In der Dokumentation unter [20] finden Sie dazu weitergehende Hinweise.

Azure Blob Storage – Dateispeicherung

Für die Speicherung und den Abruf von Dateien wie Bildern, Videos oder Dokumenten nutzen wir diesen Dienst aus dem Azure-Portal: Erstellen Sie ein Azure Storage-Konto und einen Blob-Container. Notieren Sie sich die Verbindungsdetails (Account-Name und Schlüssel) und installieren Sie das NuGet-Paket `Azure.Storage.Blobs`. Ein Dateiupload erfolgt dann beispielsweise nach dem folgenden Muster:

```
using Azure.Storage.Blobs;
var blobServiceClient =
    new BlobServiceClient("<connection-string>");
var containerClient =
    blobServiceClient.GetBlobContainerClient("images");
var blobClient =
    containerClient.GetBlobClient("sample.jpg");

using var fileStream =
    File.OpenRead("path/to/file.jpg");
await blobClient.UploadAsync(
    fileStream, overwrite: true);
```

Mehr Informationen zu diesem Dienst finden Sie unter [21].

Fazit und Ausblick

Viele (mobile) Apps benötigen ein Backend für die typischen Aufgaben wie Datenspeicherung oder Authentifizierung. Dieses hostet man – sofern es den Datenschutzbestimmungen nicht widerspricht – idealerweise in der Cloud. Entwickelt man die Apps für die Zielplattformen mit .NET MAUI, dann kommt die Anforderung hinzu, dass es möglichst eine .NET-kompatible Client-Library geben sollte, damit man sich den Aufwand über HTTPS/REST-Abfragen sparen kann.

Der Artikel hat gezeigt, dass wir eine Auswahl an Cloud-Anbietern haben, die dafür infrage kommen. Neben Firebase (Google) und Azure (Microsoft) kann es durchaus lohnend sein, die alternativen Optionen (beispielsweise Supabase) in

Betracht zu ziehen. Diese können ebenfalls punkten, unter anderem mit einer reduzierten Komplexität, was den Entwicklungsaufwand deutlich reduziert. In Bild 7 sind die wesentlichen Aspekte rund um die Nutzung eines BaaS in einer .NET-MAUI-App nochmals zusammengefasst. ■

- [1] Firebase, <https://firebase.google.com>
- [2] Firebase Pricing, <https://firebase.google.com/pricing>
- [3] Amazon AWS, <https://aws.amazon.com/de/amplify>
- [4] AWS SDK for .NET, <https://aws.amazon.com/de/sdk-for-net>
- [5] Preise für AWS Amplify, <https://aws.amazon.com/de/amplify/pricing>
- [6] Visual Studio App Center, www.dotnetpro.de/SL2503MAUIBackend1
- [7] Backendless, <https://backendless.com>
- [8] Backendless Pricing Plans, <https://backendless.com/pricing>
- [9] Supabase, <https://supabase.com>
- [10] Supabase Pricing Plans, <https://supabase.com/pricing>
- [11] Firebase Realtime Database, <https://firebase.google.com/docs/database>
- [12] Firebase-Datenbank strukturieren, www.dotnetpro.de/SL2503MAUIBackend2
- [13] FirebaseUI for Auth, www.dotnetpro.de/SL2503MAUIBackend3
- [14] Firebase Authentication C# Library, www.dotnetpro.de/SL2503MAUIBackend4
- [15] Google Cloud Client Libraries for .NET auf GitHub, www.dotnetpro.de/SL2503MAUIBackend5
- [16] Firebase Cloud Firestore, www.dotnetpro.de/SL2503MAUIBackend6
- [17] FirebaseDatabase.net auf GitHub, www.dotnetpro.de/SL2503MAUIBackend7
- [18] Firebase-Console, <https://console.firebase.google.com>
- [19] Azure Cosmos DB Documentation, www.dotnetpro.de/SL2503MAUIBackend8
- [20] Azure Active Directory B2C Documentation, www.dotnetpro.de/SL2503MAUIBackend9
- [21] Azure Blob Storage Documentation, www.dotnetpro.de/SL2503MAUIBackend10



Dr. Veikko Krypczyk
Elena Bochkor

Dr. Veikko Krypczyk ist seit Jahren begeisterter Softwareentwickler. Elena Bochkor beschäftigt sich mit dem Entwurf und Design mobiler Anwendungen, Webseiten und digitaler Produkte. Ihr technologischer Fokus liegt dabei auf C#, WPF, WinUI 3 und .NET MAUI. Beide bieten entsprechende Workshops für Entwickler an.

<https://larinet.com>

<https://www.tech-punkt.com>

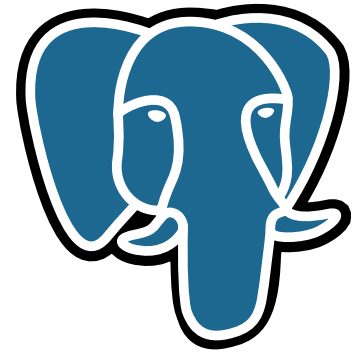


dnpCode

A2503MAUIBackend

OPEN-SOURCE-DATENBANK POSTGRESQL, TEIL 1

Offene Alternative



Installation der objektrelationalen Datenbank PostgreSQL und wichtige Unterschiede zum Microsoft SQL Server.

Auf der Microsoft-Plattform ist der SQL Server seit Jahrzehnten die meistgenutzte Datenbank. Doch in den zurückliegenden Jahren hat die leistungsstarke und kostenlose Open-Source-Datenbank PostgreSQL massiv an Bedeutung gewonnen. Immer mehr Unternehmen, darunter auch große Organisationen, entscheiden sich für den Umstieg von SQL Server auf PostgreSQL, vor allem aufgrund der geringeren Kosten und der hohen Flexibilität. Dieser Artikel bildet den Auftakt einer Artikelserie, die einen genauen Blick auf PostgreSQL wirft und die Unterschiede zum SQL Server zeigt.

Die Anfänge von PostgreSQL reichen bis in die 1980er-Jahre zurück. Das Datenbanksystem hat sich im Lauf der Jahrzehnte von einem akademischen Forschungsprojekt zu einer

Postgres führte einige (für die damalige Zeit) revolutionäre Funktionen ein, wie zum Beispiel das Multiversion Concurrency Control System (MVCC), das auch heute noch eine der Kernfunktionen von PostgreSQL ist. MVCC ist vergleichbar mit Optimistic Concurrency, das mit dem SQL Server 2005 eingeführt wurde. Es ermöglicht, lesende Transaktionen ohne Shared Locks auszuführen, indem mehrere Versionen von Datensätzen gleichzeitig verwaltet werden. Dies sorgt für eine hohe Effizienz in Mehrbenutzerumgebungen und gilt als einer der Hauptgründe für die Performance-Vorteile von PostgreSQL in komplexen Szenarien.

In den frühen 1990ern erhielt Postgres einige Erweiterungen, darunter die Implementierung der Structured Query Language (SQL) als Abfragesprache, wodurch der Grundstein für das heutige PostgreSQL gelegt wurde. 1996 wurde das Projekt offiziell in PostgreSQL umbenannt. Seitdem hat sich das System zu einer vollwertigen relationalen Datenbank mit umfassender SQL-Unterstützung und einer Vielzahl moderner Funktionen entwickelt.

Im Gegensatz zu kommerziellen Datenbanken wird PostgreSQL von einer globalen Entwickler-Community gepflegt und kontinuierlich weiterentwickelt. Dadurch hat PostgreSQL viele Funktionen erhalten, die oft erst Jahre später in kommerziellen Systemen auftauchten. Dazu gehören neben MVCC auch die Unterstützung für erweiterbare Datentypen, Indizes (wie GIN und GiST) und Replikationsmechanismen.

Heute gilt PostgreSQL als eine der leistungsfähigsten Open-Source-Datenbanken und wird von großen Unternehmen weltweit eingesetzt. Die kontinuierliche Weiterentwicklung durch eine engagierte Community, die robuste Architektur und die weitreichenden Anpassungsmöglichkeiten machen PostgreSQL zu einer ausgezeichneten Wahl für verschiedenste Anwendungsfälle – von kleinen Anwendungen bis hin zu großen, verteilten Systemen.

PostgreSQL versus Microsoft SQL Server

Werfen wir zunächst einen Blick auf die Unterschiede der Architekturen von PostgreSQL und SQL Server, da wir auf diese im weiteren Verlauf dieser Artikelserie immer wieder zu sprechen kommen werden.

```

~/Documents docker exec -it --user postgres postgres /bin/bash
postgres@5d71369c69ff:/$ psql
psql (17.0 (Debian 17.0-1.pgdg120+1))
Type "help" for help.

postgres=# select oid, datname from pg_database;
 oid | datname
-----+-----
   5 | postgres
   1 | template1
   4 | template0
27366 | adventureworks
29712 | testdatabase1
(5 rows)

postgres=# \q
postgres@5d71369c69ff:/$

```

Die laufende PostgreSQL-Installation, angezeigt vom Tool psql (Bild 1)

der weltweit leistungsfähigsten und meistgenutzten Open-Source-Datenbanken entwickelt.

Die Ursprünge von PostgreSQL liegen im Postgres-Projekt, das 1986 an der University of California, Berkeley vom Informatiker Michael Stonebraker ins Leben gerufen wurde. Er und sein Team entwickelten Postgres als Nachfolger der populären Ingres-Datenbank, die ebenfalls in Berkeley als Forschungsprojekt entstanden war. Der Name Postgres leitet sich von Post-Ingres ab, da das System viele der Konzepte von Ingres weiterentwickeln sollte. Stonebrakers Ziel war es, eine Datenbank zu schaffen, die mit objektorientierten Konzepten arbeitet und somit über die damals üblichen relationalen Datenbanken hinausgeht.

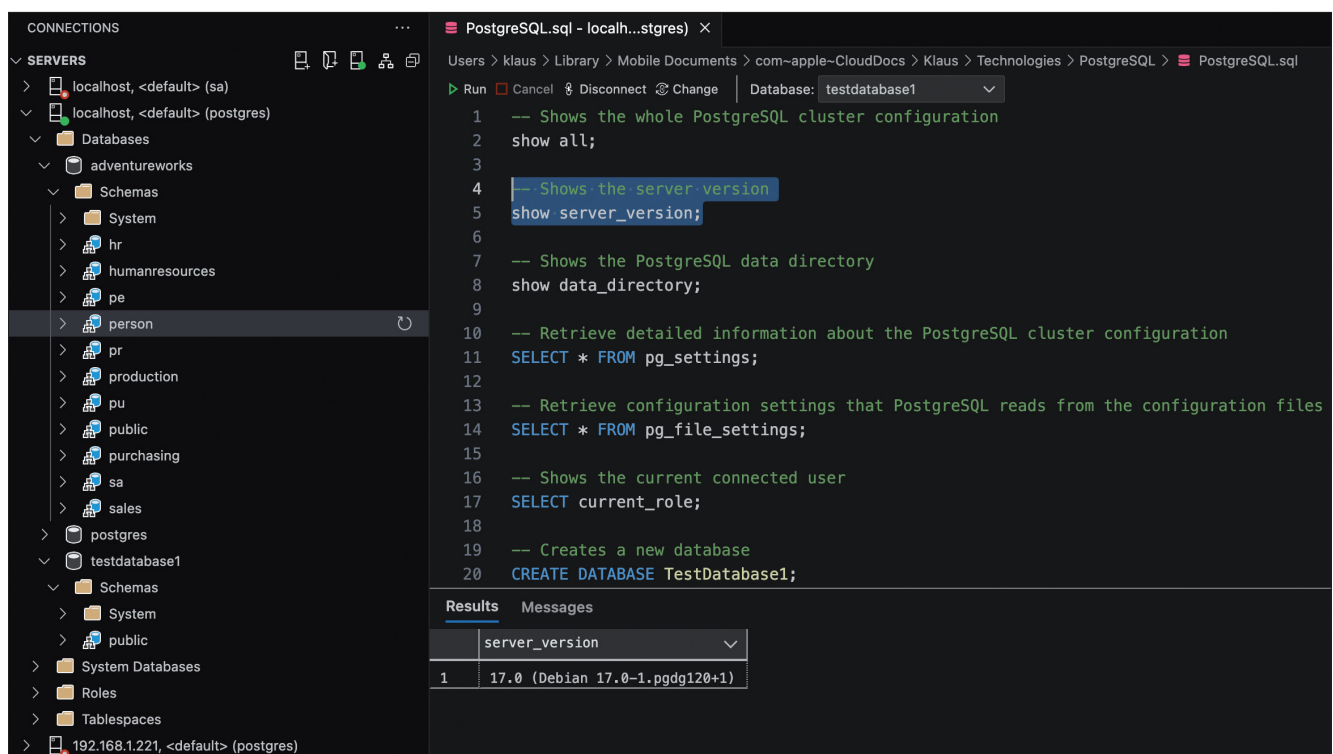
Ein fundamentaler Unterschied zwischen Microsofts SQL Server und PostgreSQL liegt darin, wie die Datenbanksysteme Anfragen verarbeiten. SQL Server setzt dabei auf eine Thread-basierte Architektur, während PostgreSQL prozessbasiert arbeitet.

Jede Client-Verbindung im SQL Server wird durch einen Thread innerhalb eines einzelnen Prozesses verwaltet. Dies führt zu einer effizienteren Speicherverwaltung, da Threads weniger Speicher erfordern als Prozesse. Threads können vom Betriebssystem auch um einiges schneller erzeugt werden als Prozesse. Diese Architektur ermöglicht es SQL Server, eine große Anzahl von Verbindungen und parallelen Anfragen zu verarbeiten, ohne zu viel Overhead zu generieren.

Im Gegensatz dazu startet PostgreSQL für jede eingehende Verbindung einen separaten Prozess. Diese laufen unabhängig voneinander, was zu einer stärkeren Isolation führt.

Im SQL Server hat jede Datenbank ihr eigenes Transaktions-Log. Dies bedeutet, dass alle Datenbanken einer SQL-Server-Instanz ihre Änderungen unabhängig voneinander protokollieren. Daraus ergibt sich der Vorteil, dass man eine einzelne Datenbank zurücksetzen oder wiederherstellen kann, ohne dass die anderen Datenbanken davon betroffen sind. Zudem erlaubt es ein Point-In-Time-Recovery (PITR) auf der Ebene einer einzelnen Datenbank.

PostgreSQL hingegen verwendet ein einheitliches WAL-Log für die komplette PostgreSQL-Installation. Das bedeutet, dass alle Datenbanken innerhalb der gleichen PostgreSQL-Installation in dasselbe WAL-Log schreiben. Der Vorteil dieses Ansatzes liegt in der vereinfachten Verwaltung – es gibt nur ein einziges Log, das überwacht und gesichert werden muss. Der Nachteil ist jedoch, dass ein Point-In-Time-Recovery immer die komplette PostgreSQL-Installation betrifft. Sie können



Zugriff auf eine PostgreSQL-Datenbank mit Azure Data Studio (Bild 2)

Sollte ein Prozess abstürzen, bleiben die anderen Verbindungen davon unberührt. Das führt zu einer höheren Stabilität. Der Nachteil liegt darin, dass Prozesse mehr Ressourcen erfordern, insbesondere mehr Speicher.

Ein weiterer großer Unterschied besteht in der Implementierung des Write Ahead Logging (WAL). Darunter versteht man einen Mechanismus, der sicherstellt, dass alle Datenbankänderungen erst in ein Protokoll geschrieben werden, bevor die Änderungen permanent in die Datenbank übernommen werden. Dies schützt vor Datenverlust bei Systemabstürzen und erlaubt es im Fehlerfall, die Daten wiederherzustellen. SQL Server und PostgreSQL implementieren diesen Mechanismus auf unterschiedliche Arten.

folglich eine einzelne Datenbank nicht unabhängig von den anderen Datenbanken wiederherstellen. Die vierte Folge dieser Serie wird sich näher mit dem WAL-Log und der Hochverfügbarkeit von PostgreSQL-Installationen beschäftigen.

Die Unterstützung von gleichzeitig aktiven Benutzern ist ein weiterer Bereich, in dem sich SQL Server und PostgreSQL deutlich unterscheiden. Dies beeinflusst, wie Transaktionen abgewickelt werden und wie sie sich aufeinander auswirken, insbesondere in Bezug auf Sperren und Blockierungen.

SQL Server verwendet als Standardeinstellung das Pessimistic-Concurrency-Control-Modell, bei dem Transaktionen Locks auf Datensätze setzen, um die Datenkonsistenz sicherzustellen. Diese Locks werden sowohl beim Lesen (Shared ►

Lock) als auch beim Schreiben (Exclusive Lock) von Datensätzen gesetzt. Dadurch kann es jedoch zu Blockaden kommen, wenn mehrere Transaktionen gleichzeitig auf dieselben Datensätze lesend und schreibend zugreifen möchten.

unterschiedlichen Versionen von Datensätzen arbeiten. Es gibt keine expliziten Locks für lesende Transaktionen, wodurch sich lesende und schreibende Transaktionen gegenseitig nicht blockieren können.

Jede Transaktion arbeitet auf einer eigenen Datensatzversion. Das verbessert die Leistung insbesondere bei vielen gleichzeitigen Lese- und Schreiboperationen. Ein Nachteil von MVCC ist, dass die Datenbank regelmäßig sogenannte VACUUM-Operationen durchführen muss, um alte, nicht mehr benötigte Datensatzversionen aufzuräumen. Mehr dazu erfahren Sie im dritten Teil der Artikelserie.

Die Art und Weise, wie Daten im SQL Server und in PostgreSQL gespeichert und indiziert werden, ist ebenfalls ein zentraler Unterschied zwischen den beiden Datenbanksystemen.

Im SQL Server können Tabellen entweder als Clustered Tables oder als Heap Tables gespeichert werden. Eine Clustered Table bedeutet, dass die Tabellendaten physisch in der Reihenfolge des Clustered Keys abgelegt werden. Jede Tabelle kann nur einen solchen Clustered Index haben, der definiert, wie die Daten physisch sortiert werden.

Alternativ können Heap Tables genutzt werden, welche die Datensätze in keiner spezifischen Reihenfolge speichern. In beiden Varianten lassen sich zusätzliche Non-Clustered Indizes verwenden, die auf die Datensätze in der eigentlichen Tabelle verweisen.

PostgreSQL speichert Tabellendaten immer in Heap Tables ab. Das bedeutet, dass die physische Speicherung der Daten nicht auf Basis eines Indexes erfolgt. Stattdessen

verwendet PostgreSQL sekundäre Indizes wie B-Tree, GIN, GiST und andere, um in Heap Tables gespeicherte Datensätze effizient abfragen zu können.

Es gibt zwar den Befehl *CLUSTER*, mit dem eine Tabelle basierend auf einem Index physisch neu sortiert werden kann, dies ist jedoch eine einmalige Operation, deren Ergebnis bei späteren Änderungen nicht automatisch aufrechterhalten wird. Der zweite Teil dieser Artikelserie wird sich näher mit den Themen Indizierung und Performance-Tuning von PostgreSQL-Datenbanken beschäftigen.

Schon dieser kurze Überblick zeigt, dass sich SQL Server und PostgreSQL deutlich voneinander unterscheiden. Beide

```
postgres@5d71369c69ff:/$ pg_ctl --help
pg_ctl is a utility to initialize, start, stop, or control a PostgreSQL server.

Usage:
  pg_ctl init[db]      [-D DATADIR] [-s] [-o OPTIONS]
  pg_ctl start         [-D DATADIR] [-l FILENAME] [-W] [-t SECS] [-s]
                        [-o OPTIONS] [-p PATH] [-c]
  pg_ctl stop          [-D DATADIR] [-m SHUTDOWN-MODE] [-W] [-t SECS] [-s]
  pg_ctl restart       [-D DATADIR] [-m SHUTDOWN-MODE] [-W] [-t SECS] [-s]
                        [-o OPTIONS] [-c]
  pg_ctl reload        [-D DATADIR] [-s]
  pg_ctl status        [-D DATADIR]
  pg_ctl promote       [-D DATADIR] [-W] [-t SECS] [-s]
  pg_ctl logrotate     [-D DATADIR] [-s]
  pg_ctl kill          SIGNALNAME PID

Common options:
  -D, --pgdata=DATADIR  location of the database storage area
  -s, --silent           only print errors, no informational messages
  -t, --timeout=SECS    seconds to wait when using -w option
  -V, --version         output version information, then exit
  -w, --wait            wait until operation completes (default)
  -W, --no-wait         do not wait until operation completes
  -?, --help            show this help, then exit
If the -D option is omitted, the environment variable PGDATA is used.

Options for start or restart:
  -c, --core-files      allow postgres to produce core files
  -l, --log=FILENAME    write (or append) server log to FILENAME
  -o, --options=OPTIONS command line options to pass to postgres
                        (PostgreSQL server executable) or initdb
  -p PATH-TO-POSTGRES  normally not necessary

Options for stop or restart:
  -m, --mode=MODE      MODE can be "smart", "fast", or "immediate"

Shutdown modes are:
  smart      quit after all clients have disconnected
  fast       quit directly, with proper shutdown (default)
  immediate  quit without complete shutdown; will lead to recovery on restart

Allowed signal names for kill:
  ABRT HUP INT KILL QUIT TERM USR1 USR2

Report bugs to <pgsql-bugs@lists.postgresql.org>.
PostgreSQL home page: <https://www.postgresql.org/>
postgres@5d71369c69ff:/$
```

Für diese Aufgaben taugt das Kommandozeilenwerkzeug `pg_ctl` (Bild 3)

Um das zu umgehen, bietet SQL Server seit 2005 das Optimistic-Concurrency-Modell an, das mit einer Datensatz-Versionierung arbeitet: Jede schreibende Transaktion kopiert die alte Datensatzversion in den sogenannten Version Store (der in der TempDb gehalten wird), damit lesende Transaktionen auf die Daten zugreifen können, ohne Shared Locks anfordern zu müssen. Damit können Blockaden zwischen lesenden und schreibenden Transaktionen vermieden werden.

PostgreSQL setzt hier Multiversion Concurrency Control (MVCC) ein, das die Möglichkeiten von Optimistic Concurrency ohne Zuhilfenahme von Shared Locks bei lesenden Zugriffen ermöglicht. MVCC erlaubt es, dass Transaktionen auf

Main	I/O										
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
59	postgres	20	0	8008	3908	2756	R	0.7	0.0	0:00.34	htop
1	postgres	20	0	401M	40216	37912	S	0.0	0.5	0:00.07	postgres: postmaster
26	postgres	20	0	401M	10248	7888	S	0.0	0.1	0:00.00	postgres: checkpointer
27	postgres	20	0	401M	8832	6472	S	0.0	0.1	0:00.04	postgres: background writer
29	postgres	20	0	401M	16072	13712	S	0.0	0.2	0:00.00	postgres: walwriter
30	postgres	20	0	403M	9360	6744	S	0.0	0.1	0:00.00	postgres: autovacuum launcher
31	postgres	20	0	403M	8656	6040	S	0.0	0.1	0:00.00	postgres: logical replication launcher
32	postgres	20	0	7068	3680	3168	S	0.0	0.0	0:00.02	/bin/bash

Das Tool htop zeigt die aktiven Prozesse (Bild 4)

sind zwar klassische relationale Datenbanken, implementieren jedoch die Grundkonzepte in einer anderen Art und Weise.

PostgreSQL installieren

PostgreSQL läuft auf allen modernen Betriebssystemen: Windows, Linux, macOS, BSD und Solaris. Darüber hinaus werden auch Hardware-Plattformen wie der ARM-basierende Raspberry Pi unterstützt. PostgreSQL lässt sich einerseits direkt über einen Installer einrichten oder auch direkt aus dem C-Quellcode kompilieren [1].

Die einfachste Möglichkeit, PostgreSQL auszuprobieren, ist die lokale Installation in einem Docker-Container, den Sie bei Bedarf einfach wegwerfen und neu aufbauen können.

Da die Beschreibung aller notwendigen Schritte einer Installation von PostgreSQL in einer virtuellen oder physischen Maschine diesen Artikel sprengen würde, habe ich diese in einen Blogpost ausgelagert, den Sie unter [2] finden.

Einfacher geht es mit Docker. Hier die erforderlichen Docker-Befehle, um PostgreSQL lokal in einem Container zu installieren:

```
docker pull postgres
```

```
docker run
```

```
--name postgres
```

```
-e POSTGRES_PASSWORD=passw0rd1!
```

-p 5432:5432

```
-d postgres
```

Im ersten Schritt wird das Docker-Image *postgres* geladen. Anschließend instanziiert der Befehl *docker run* daraus einen Docker-Container. Diesem Container wird der Name *postgres* zugewiesen, und über die Umgebungsvariable *POSTGRES_PASSWORD* wird das Passwort für den *postgres*-Benutzer gesetzt. Diesen Benutzer können Sie sich wie den *sa*-Benutzer beim SQL Server vorstellen.

Eine PostgreSQL-Installation ist unter dem Port 5432 erreichbar (Standardeinstellung). Daher wird über den Parameter `-p` auch dieser Port auf den lokalen Port 5432 gemappt, wodurch die PostgreSQL-Installation unter `localhost:5432` erreichbar ist.

Durch den Parameter *-d* wird der Docker-Container im Hintergrund im sogenannten Detached-Modus ausgeführt. Der sorgt dafür, dass der Docker-Container so lange aktiv bleibt, bis Sie ihn über den Befehl *docker stop postgres* stoppen. Der Befehl *docker start postgres* startet ihn erneut.

Nachdem Sie nun eine laufende Installation von PostgreSQL in Betrieb haben, stellt sich die Frage, wie Sie auf PostgreSQL zugreifen können. Hierzu gibt es mehrere Möglichkeiten:

- mit dem Kommandozeilen-Tool `psql`,
- per Azure Data Studio,

Main	I/O
PID USER PRI NI VIRT RES SHR S CPU%MEM TIME+ Command	
1 postgres 20 0 401M 40216 37912 S 0.0 0.5 0:00.08 postgres	
26 postgres 20 0 401M 10248 7888 S 0.0 0.1 0:00.00 postgres: checkpointer	
27 postgres 20 0 401M 8832 6472 S 0.0 0.1 0:00.04 postgres: background writer	
29 postgres 20 0 401M 16072 13712 S 0.0 0.2 0:00.00 postgres: walwriter	
30 postgres 20 0 403M 9360 6744 S 0.0 0.1 0:00.00 postgres: autovacuum launcher	
31 postgres 20 0 403M 8656 6040 S 0.0 0.1 0:00.00 postgres: logical replication launcher	
32 postgres 20 0 7068 3680 3168 S 0.0 0.0 0:00.02 /bin/bash	
72 postgres 20 0 403M 17808 14552 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(58767) idle	
73 postgres 20 0 403M 16804 13804 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(48356) idle	
75 postgres 20 0 403M 16808 13680 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(37034) idle	
76 postgres 20 0 403M 17572 14316 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(46272) idle	
77 postgres 20 0 403M 16872 13744 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(60420) idle	
78 postgres 20 0 403M 17504 14248 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(20654) idle	
79 postgres 20 0 403M 16808 13680 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(29743) idle	
80 postgres 20 0 403M 17572 14316 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(29703) idle	
81 postgres 20 0 403M 16872 13744 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(18062) idle	
82 postgres 20 0 403M 17504 14248 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(25624) idle	
83 postgres 20 0 403M 16808 13680 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(53411) idle	
84 postgres 20 0 403M 17576 14320 S 0.0 0.2 0:00.00 postgres: postgres testdatabase1 192.168.65.1(21041) idle	
85 postgres 20 0 8008 3908 2756 R 0.0 0.0 0:00.04 httpd	

Diverse Backend-Prozesse für aktive Datenbankverbindungen (Bild 5)

- über DBeaver,
- mit pgAdmin.

Die einfachste Möglichkeit ist der Zugriff über das Kommandozeilen-Tool `psql`, das Bestandteil jeder PostgreSQL-Installation ist (vergleichbar mit dem Kommandozeilen-Tool `sqlcmd` des SQL Servers). Führen Sie PostgreSQL in einem Docker-Container aus, müssen Sie dieses Tool manuell außerhalb des Docker-Containers installieren, oder Sie öffnen eine Shell-Session, in der Sie sich mit dem Docker-Container verbinden:

```
docker exec
-it --user postgres postgres /bin/bash
```

Innerhalb der Shell-Session können Sie anschließend das Werkzeug `psql` starten, das Sie mit der laufenden PostgreSQL-Installation verbindet.

Wie Sie in **Bild 1** sehen, läuft PostgreSQL im Docker-Container in einer auf Debian basierenden Linux-Distribution. Verwendet wird dabei die PostgreSQL-Version 17, die im Oktober 2024 erschienen ist.

Nach dem Aufruf von `psql` besteht eine aktive Verbindung zur PostgreSQL-Installation, die in einem eigenen Prozess läuft, und Sie können nun SQL-Statements ausführen. Wichtig ist dabei, dass jedes SQL-Statement mit einem Strichpunkt abgeschlossen werden muss. Fehlt der Strichpunkt, gibt es eine Fehlermeldung und das eingegebene Statement wird nicht ausgeführt. Um zu sehen, welche Datenbanken auf der PostgreSQL-Installation vorhanden sind, können Sie durch Abfrage von `pg_database` die Catalog View einschalten. Jeder Datensatz darin steht für eine Datenbank. `pg_database` ist vergleichbar mit `sys.databases` beim SQL Server.

Die PostgreSQL-Installation in **Bild 1** enthält fünf Datenbanken. Drei davon sind System-Datenbanken und in jeder Post-

● **Tabelle 1: Die wichtigsten Ordner eines PostgreSQL Clusters**

Verzeichnis	Zweck
<i>base</i>	Hier liegen die eigentlichen Datenbanken. Jede Datenbank wird in einem eigenen Unterverzeichnis gespeichert, das nach dem Object Identifier (OID) der Datenbank benannt ist.
<i>global</i>	Enthält globale Systemtabellen wie <code>pg_database</code> , die Informationen über alle Datenbanken im Cluster speichern.
<i>pg_wal</i>	Speichert die Dateien, die für das Write-Ahead-Logging benötigt werden.
<i>pg_stat</i>	Hier liegen statistische Informationen über die Datenbankaktivitäten, die von der Autovacuum-Funktion verwendet werden.
<i>pg_tblspc</i>	Speichert Verknüpfungen zu sogenannten Tablespaces (File Groups im SQL Server), die sich außerhalb des Verzeichnisses PGDATA befinden.
<i>pg_snapshots</i>	In diesem Ordner liegen persistente Snapshot-Dateien, die zur Wiederherstellung und zum Point-In-Time-Recovery verwendet werden können.
<i>pg_replslot</i>	Ordner für sogenannte Replication-Slots, die sicherstellen, dass WAL-Dateien für eine physische Replikation gespeichert bleiben, bis sie von den Secondary-Servern gelesen werden.
<i>pg_temp</i>	Ablageort für temporäre Daten, die während der Ausführung von Abfragen mit Sort- und Hash-Operatoren in das Dateisystem geschrieben werden müssen.

greSQL-Installation enthalten: `postgres`, `template1` und `template0`. Die Datenbanken `adventureworks` und `testdatabase1` wurden manuell angelegt. Über den Befehl `\q` wird das Kommandozeilen-Tool `psql` beendet, und es erscheint wieder die Kommandozeile des Docker-Containers.

Eine weitere Möglichkeit des Zugriffs auf PostgreSQL bietet Azure Data Studio von Microsoft, welches eine passende Extension für PostgreSQL anbietet [3], siehe **Bild 2**.

Auch das Tool DBeaver bietet Ihnen Zugriff auf PostgreSQL, und es unterstützt überdies weitere Datenbanken – auch den SQL Server. PostgreSQL bietet für diesen Zweck ein eigenes Tool namens `pgAdmin` [5] an, das Sie auch in einem Docker-Container laufen lassen können.

Ich persönlich setze primär auf Azure Data Studio und wechsle für Aufgaben, die Azure Data Studio nicht unterstützt, zu DBeaver oder `pgAdmin`.

Der PostgreSQL Cluster

Nun wird beleuchtet, wie eine PostgreSQL-Installation aufge-

Results Messages					
	backend_type	datid	datname	pid	leader_pid
1	client backend	29712	testdatabase1	72	NULL
2	client backend	29712	testdatabase1	75	NULL
3	client backend	29712	testdatabase1	77	NULL
4	client backend	29712	testdatabase1	78	NULL
5	client backend	29712	testdatabase1	79	NULL
6	client backend	29712	testdatabase1	80	NULL
7	client backend	29712	testdatabase1	81	NULL
8	client backend	29712	testdatabase1	82	NULL
9	client backend	29712	testdatabase1	83	NULL
10	client backend	29712	testdatabase1	84	NULL
11	autovacuum launcher	NULL	NULL	30	NULL
12	logical replication launcher	NULL	NULL	31	NULL
13	checkpointer	NULL	NULL	26	NULL
14	background writer	NULL	NULL	27	NULL
15	walwriter	NULL	NULL	29	NULL

Ausgabe zu `* FROM pg_stat_activity` (**Bild 6**)

baut ist und wie sie sich von SQL-Server-Installationen unterscheidet.

Es beginnt schon beim Namen: Eine komplette PostgreSQL-Installation wird als PostgreSQL Cluster bezeichnet. Dieser Cluster besteht aus den System- und Anwendungs-Datenbanken. Der Begriff Cluster hat damit in PostgreSQL eine andere Bedeutung als beim SQL Server. Ab jetzt werde ich nur mehr von einem PostgreSQL Cluster sprechen und nicht mehr von einer PostgreSQL-Installation.

Ein wichtiges Werkzeug zum Management eines PostgreSQL Clusters ist `pg_ctl`. Es erlaubt, den Cluster zu starten, zu stoppen sowie einen Restart durchzuführen. Außerdem können Sie über die Option `initdb` das Datenverzeichnis des Clusters initialisieren oder mithilfe der Option `promote` einen Failover von einem Primary Server auf einen Secondary Server durchführen. Bild 3 zeigt die Optionen.

Sobald Sie den Cluster starten, wird der Hauptprozess von PostgreSQL (der *postmaster*-Prozess) gestartet. Dieser hat innerhalb eines Docker-Containers immer die PID 1. Neben dem Hauptprozess laufen noch einige andere Prozesse, die regelmäßige Aktivitäten durchführen:

- *checkpointer*,
- *background writer*,
- *walwriter*,
- *autovacuum launcher*,
- *logical replication launcher*.

Eine Auflistung dieser Prozesse mithilfe des Tools `htop` sehen Sie in Bild 4.

```
postgres@5d71369c69ff:~/data$ ls -l
total 132
drwx----- 8 postgres postgres 4096 Oct 13 10:06 base
-rw-r--r-- 1 postgres dialout   39 Oct 15 12:11 custom.conf
drwx----- 2 postgres postgres 4096 Oct 17 15:40 global
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_commit_ts
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_dynshmem
-rw----- 1 postgres postgres 5743 Sep 30 17:24 pg_hba.conf
-rw----- 1 postgres postgres 2640 Sep 30 17:24 pg_ident.conf
drwx----- 4 postgres postgres 4096 Oct 17 15:44 pg_logical
drwx----- 4 postgres postgres 4096 Sep 30 17:24 pg_multixact
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_notify
drwx----- 2 postgres postgres 4096 Oct 14 16:38 pg_replslot
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_serial
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_snapshots
drwx----- 2 postgres postgres 4096 Oct 17 15:39 pg_stat
drwx----- 2 postgres postgres 4096 Oct 17 15:39 pg_stat_tmp
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_subtrans
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_tblspc
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_twophase
-rw----- 1 postgres postgres   3 Sep 30 17:24 PG_VERSION
drwx----- 4 postgres postgres 4096 Oct 14 17:00 pg_wal
drwx----- 2 postgres postgres 4096 Sep 30 17:24 pg_xact
-rw----- 1 postgres postgres  88 Sep 30 17:24 postgresql.auto.conf
-rw----- 1 postgres postgres 30827 Oct 16 12:27 postgresql.conf
-rw----- 1 postgres postgres  36 Oct 17 15:39 postmaster.opts
-rw----- 1 postgres postgres  94 Oct 17 15:39 postmaster.pid
postgres@5d71369c69ff:~/data$
```

Verzeichnis, das die Daten des PostgreSQL Clusters enthält (Bild 7)

Immer dann, wenn eine neue Datenbankverbindung zu PostgreSQL aufgebaut wird, erstellt der *postmaster*-Prozess einen neuen Backend-Prozess, der dann exklusiv für diese Datenbankverbindung zuständig ist. Bild 5 zeigt verschiedene Backend-Prozesse für aktive Datenbankverbindungen. Wie Sie in der Abbildung sehen, wurden Backend-Prozesse für zwölf aktive Datenbankverbindungen erzeugt. Ein großer Vorteil dieses Ansatzes ist, dass Datenbankverbindungen voneinander isoliert sind, was bedeutet, dass ein Fehler in einer

Datenbankverbindung keinen Einfluss auf andere Verbindungen hat. Die aktuellen Verbindungen können Sie sich auch über die Catalog View `pg_stat_activity` ausgeben lassen (Bild 6) – diese ist vergleichbar mit `sys.dm_exec_connections` im SQL Server.

```
SELECT backend_type,
       * FROM pg_stat_activity;
```

Wenn Sie eine neue Verbindung in PostgreSQL öffnen, ist es wichtig zu wissen, dass diese immer auf eine bestimmte Datenbank zielt.

Weil es in PostgreSQL keine Master-Datenbank gibt, müssen Sie für den Aufbau einer Verbindung immer auch eine Datenbank angeben. ►

```
GNU nano 7.2
#-----
# CONNECTIONS AND AUTHENTICATION
#-----
# - Connection Settings -

listen_addresses = '*'
# comma-separated list of addresses;
# defaults to 'localhost'; use '*' for all
# (change requires restart)
#port = 5432
# (change requires restart)
max_connections = 100
# (change requires restart)
#reserved_connections = 0
# (change requires restart)
#superuser_reserved_connections = 3
# (change requires restart)
#unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
# (change requires restart)
#unix_socket_group = ''
# (change requires restart)
#unix_socket_permissions = 0777
# begin with 0 to use octal notation
# (change requires restart)
#bonjour = off
# advertise server via Bonjour
# (change requires restart)
#bonjour_name = ''
# defaults to the computer name
# (change requires restart)
```

Eine Änderung des Eintrags `listen_addresses` wird erst nach einem Neustart des Clusters wirksam (Bild 8)

Der Pfad im Dateisystem, in dem sämtliche Daten eines PostgreSQL Clusters gespeichert werden, wird in einer Umgebungsvariablen namens *PGDATA* gespeichert. Diese zeigt in einem Docker-Container auf das Verzeichnis */var/lib/postgresql/data*, bei einer virtuellen beziehungsweise physischen Installation jedoch auf das Verzeichnis */var/lib/postgresql/17/main*, siehe **Bild 7**. Dieser Ordner enthält alle Daten, die vom PostgreSQL Cluster verwaltet werden, einschließlich der Konfigurationsdateien, des WAL-Logs, der Tabellen- und Indexdaten sowie anderer wichtigen Systeminformationen.

Bei der Installation eines PostgreSQL Clusters wird das Verzeichnis *PGDATA* mit dem Befehl *initdb* initialisiert; entsprechend muss das Verzeichnis beim Start von PostgreSQL als Parameter angegeben werden. Diese Schritte erfolgen beim Erzeugen des Docker-Containers automatisch, müssen aber bei einer virtuellen oder physischen Installation von Hand durchgeführt werden. **Tabelle 1** gibt einen Überblick über die wichtigsten Verzeichnisse und deren Zweck.

Die PostgreSQL-Konfiguration

Das Verzeichnis *PGDATA* enthält neben den Unterverzeichnissen auch die für den Betrieb von PostgreSQL erforderlichen Konfigurationsdateien:

- *postgresql.conf* und
- *pg_hba.conf*.

Über die Datei *postgresql.conf* wird die komplette Konfiguration des PostgreSQL Clusters vorgenommen. Im Rahmen dieser Artikelserie werden wir immer wieder auf diese Konfigu-

Results Messages			
	name	setting	description
1	allow_alter_system	on	Allows running the ALTER SYSTEM command.
2	allow_in_place_tablespace	off	Allows tablespaces directly inside pg_tblspc, for te...
3	allow_system_table_mods	off	Allows modifications of the structure of system tabl...
4	application_name		Sets the application name to be reported in statisti...
5	archive_cleanup_command		Sets the shell command that will be executed at ever...
6	archive_command	(disabled)	Sets the shell command that will be called to archiv...
7	archive_library		Sets the library that will be called to archive a WA...
8	archive_mode	off	Allows archiving of WAL files using "archive_command...
9	archive_timeout	0	Sets the amount of time to wait before forcing a swi...
10	array_nulls	on	Enable input of NULL elements in arrays.
	server_version		
1	17.0 (Debian 17.0-1.pgdg120+1)		
	listen_addresses		
1	*		
	data_directory		
1	/var/lib/postgresql/data		

Die Konfiguration mit show anzeigen (Bild 9)

rationsdatei zu sprechen kommen, da Sie hier zentrale Aspekte von PostgreSQL konfigurieren können.

Eine sehr wichtige Einstellung nach der Installation von PostgreSQL ist die Option *listen_addresses*. Diese ist zunächst auskommentiert, wodurch der PostgreSQL Cluster im Netzwerk nicht erreichbar ist – ein „Lockdown by Default“, genauso wie beim SQL Server.

Über die Einstellung *listen_addresses* definieren Sie die IP-Adresse(n), über die der PostgreSQL Cluster im Netzwerk erreichbar ist. Tragen Sie hier einen Stern (*) ein, ist der Cluster über alle IP-Adressen erreichbar, die der Maschine über Netzwerkkarten zugeordnet sind.

Nehmen Sie an dieser Einstellung eine Änderung vor, ist ein Neustart des PostgreSQL Clusters erforderlich, damit die Änderung wirksam wird, vergleiche **Bild 8**.

Die Konfiguration des PostgreSQL Clusters können Sie sich auch mit dem SQL-Befehl *show* anzeigen lassen. *show all* liefert alle Konfigurationseinstellungen, und *show <setting>* beschränkt die Ausgabe auf die angegebene Einstellung. Hier ein paar Beispiele, deren Ausgabe Sie in **Bild 9** sehen:

```
-- Shows the whole PostgreSQL cluster configuration
show all;
```

```
-- Shows the server version
show server_version;
```

```
-- Shows the IP addresses under which PostgreSQL
-- is accessible
show listen_addresses;
```

● Listing 1: Eine neue Datenbank anlegen

```
-- Creates a new database
CREATE DATABASE TestDatabase1;

-- Lists all the databases
SELECT * FROM pg_database
WHERE datistemplate = false;

-- Returns the size of a given database
SELECT
    pg_database_size('testdatabase1'),
    pg_sizepretty(
        pg_database_size('testdatabase1'));
```

● Listing 2: Eine neue Tabelle anlegen

```
-- Create a new table
CREATE TABLE TestTable1
(
    Col1 INT NOT NULL,
    Col2 VARCHAR(80),
    Col3 DATE,
    Col4 TIME
);

-- Insert a new record into the table
INSERT INTO TestTable1 (Col1, Col2, Col3, Col4)
VALUES (1, 'Klaus', '2024-01-16', '21:09');

-- Return the records from the table
SELECT * FROM TestTable1;

-- Return the records from the table
SELECT * FROM TestTable1;

-- Retrieve more information of the previous
-- created table
SELECT
    oid,
    relname,
    pg_size_pretty(pg_relation_size('testtable1')),
    reltuples,
    relpages,
    *
FROM pg_class WHERE relname = 'testtable1';

-- Retrieve the file path for the table
SELECT * FROM pg_relation_filepath('testtable1');
```

```
-- Shows the PostgreSQL data directory
show data_directory;
```

Damit sich ein Benutzer erfolgreich mit einem PostgreSQL Cluster verbinden kann, muss der Host-Based Access (HBA) in der Datei `pg_hba.conf` richtig konfiguriert sein. Standard ist in PostgreSQL die Peer Authentication für den Benutzer `postgres`, wodurch sich dieser nur lokal über das Unix-Konto `postgres` anmelden kann.

Um beispielsweise jedem Benutzer aus dem Subnetz `192.168.1.0/24` den Zugriff auf jede Datenbank zu erlauben, müssen Sie in der Datei `pg_hba.conf` den in **Bild 10** rot markierten Eintrag hinzufügen.

Eine weitere Besonderheit besteht darin, dass bei einer Installation, die nicht mit Docker durchgeführt wurde, der Datenbankbenutzer `postgres` kein Passwort hat. Für eine erfolgreiche Datenbankverbindung muss allerdings ein Passwort angegeben werden. Daher müssen Sie sich lokal mithilfe von

`psql` mit dem PostgreSQL Cluster verbinden und mit den folgenden Zeilen ein Passwort setzen:

```
sudo -u postgres psql
ALTER USER postgres PASSWORD 'passw0rd1!';
\q
```

Speicherverwaltung bei PostgreSQL und SQL Server

Die Speicherverwaltung ist ein zentraler Aspekt der Leistung jeder Datenbank und stellt sicher, dass Daten effizient im Arbeitsspeicher bereitgehalten werden, um langsame Zugriffe auf das Storage-Subsystem zu vermeiden.

PostgreSQL stellt dazu die Einstellung `shared_buffers` und SQL Server die Einstellung `max_server_memory` zur Verfügung. Diese beiden Einstellungen erfüllen ähnliche Zwecke, werden jedoch auf grundlegend unterschiedliche Weise gehandhabt. ►

#	TYPE	DATABASE	USER	ADDRESS	METHOD
#	"local" is for Unix domain socket connections only				
local	all		all		peer
#	IPv4 local connections:				
host	all		all	127.0.0.1/32	scram-sha-256
host	all		all	192.168.1.0/24	scram-sha-256
#	IPv6 local connections:				
host	all		all	:::1/128	scram-sha-256
#	Allow replication connections from localhost, by a user with the				
#	replication privilege.				
local	replication		all		peer
host	replication		all	127.0.0.1/32	scram-sha-256
host	replication		all	:::1/128	scram-sha-256

In der Datei `pg_hba.conf` den Zugriff über das Subnetz `192.168.1.0/24` freischalten (**Bild 10**)

```
GNU nano 7.2
#-----
# RESOURCE USAGE (except WAL)
#-----
# - Memory -
shared_buffers = 128MB          # min 128kB
                                # (change requires restart)
#huge_pages = try               # on, off, or try
                                # (change requires restart)
#huge_page_size = 0             # zero for system default
                                # (change requires restart)
#temp_buffers = 8MB             # min 800kB
#max_prepared_transactions = 0 # zero disables the feature
                                # (change requires restart)
```

Die Standardeinstellung für `shared_buffers`: 128 MB (Bild 11)

Die Einstellung `shared_buffers` in PostgreSQL bestimmt die Größe des Hauptspeichers, den PostgreSQL für den Buffer-Pool reserviert, in denen oft angeforderte Daten und Indexseiten zwischengespeichert werden. Dieser Hauptspeicherbereich wird zwischen den PostgreSQL-Prozessen über Shared Memory geteilt.

Der Speicherbereich dient dazu, Daten direkt aus dem Arbeitsspeicher zu lesen, indem die Daten vorübergehend im Speicher gehalten werden, bevor auf das Storage-Subsystem zugegriffen wird.

Im Gegensatz zum SQL Server verwendet PostgreSQL den Page Cache des Betriebssystems, der einen Großteil des Cachings der Daten übernimmt, die aus dem Storage-Subsystem gelesen werden.

Daher ist die allgemeine Empfehlung, die Einstellung `shared_buffers` auf etwa 25 Prozent des verfügbaren Arbeitsspeichers zu setzen, während der Rest des Speichers dem Betriebssystem überlassen wird, um die Datei- und Speicherverwaltung zu optimieren. Dieser Ansatz stellt sicher, dass das Betriebssystem über ausreichend Speicher verfügt, um Daten zwischenspeichern zu können, die nicht im Buffer Pool gehalten werden.

Im Vergleich mit PostgreSQL verfolgt SQL Server einen deutlich anderen Ansatz für die Speicherverwaltung. Die Einstellung `max_server_memory` legt den maximalen Speicher fest, den SQL Server für sich selbst reservieren darf. Dieser Speicher umfasst den Buffer Pool, der für die Zwischenspeicherung von Tabellen- und Indexdaten im Speicher verantwortlich ist. Microsoft SQL Server versucht, den Großteil des für Daten benötigten Speichers im eigenen Buffer Pool zu verwalten, anstatt sich auf das Betriebssystem zu verlassen.

Die Empfehlung beim Betrieb des SQL Servers ist es, 80 bis 90 Prozent des verfügbaren Arbeitsspeichers für den Buffer Pool zu reservieren. Dies liegt daran, dass SQL Server seine Speicherverwaltung selbst übernimmt und den Page Cache des Betriebssystems umgeht. Durch diese Arbeitsweise kann SQL Server seine Speicherressourcen besser kontrollieren und optimieren.

Nach einer Standard-Installation von PostgreSQL ist die Einstellung `shared_buffers` auf 128 MB gesetzt, siehe Bild 11.

Ihre erste PostgreSQL-Datenbank

Nachdem Sie die Grundkonzepte von PostgreSQL kennengelernt haben und auch die wichtigsten Unterschiede zum SQL Server bekannt sind, geht es nun darum, wie Sie Ihre erste Datenbank mit PostgreSQL erstellen.

Jeder neu installierte PostgreSQL Cluster besitzt die Datenbank `postgres`, mit der Sie sich verbinden können, um per SQL-Befehl `CREATE DATABASE` eine neue Datenbank anzulegen. Listing 1 zeigt ein Beispiel dazu.

Sobald die neue Datenbank angelegt ist, listet die Catalog View `pg_database` einen Eintrag dafür. Die aktuelle Größe der Datenbank lässt sich zum Beispiel durch Aufruf von `pg_database_size` ermitteln.

Jedes Objekt (Datenbank, Tabelle, Index, et cetera) wird in PostgreSQL über einen Object Identifier (OID) eindeutig identifiziert. Hierbei handelt es sich um einen fortlaufenden Integer-Wert.

Legen Sie eine neue Datenbank an, wird anhand der OID der Datenbank ein neues Unterverzeichnis im Ordner `$PG-DATA/base` angelegt. In unserem Beispiel hat die Datenbank `testdatabase1` die OID 29783, daher trägt auch das neu angelegte Unterverzeichnis diesen Namen, vergleiche Bild 12.

Innerhalb dieses Verzeichnisses werden nun sämtliche Daten (Tabellendaten, Indexdaten) dieser Tabelle in Form von Dateien abgespeichert. Das ist wieder ein großer Unterschied zum SQL Server, der sämtliche Daten in einem oder mehre-

```
postgres@5d71369c69ff:~/data/base$ ls -l
total 40
drwx----- 2 postgres postgres 4096 Oct 3 11:22 1
drwx----- 2 postgres postgres 20480 Oct 18 11:13 27366
drwx----- 2 postgres postgres 4096 Oct 18 12:03 29783
drwx----- 2 postgres postgres 4096 Oct 2 12:35 4
drwx----- 2 postgres postgres 4096 Oct 18 11:13 5
drwx----- 2 postgres postgres 4096 Oct 14 16:38 pgsql_tmp
postgres@5d71369c69ff:~/data/base$
```

Das Verzeichnis für die neu angelegte Datenbank (Bild 12)

```
postgres@5d71369c69ff:~/data/base/29783$ ls 29* -l
-rw----- 1 postgres postgres 8192 Oct 18 12:19 29784
-rw----- 1 postgres postgres 0 Oct 18 12:03 2995
-rw----- 1 postgres postgres 8192 Oct 18 12:06 2996
postgres@5d71369c69ff:~/data/base/29783$
```

Für die neue Tabelle hat PostgreSQL die neue Datei 29784 erstellt (Bild 13)

ren Daten-Files (.*mdf*, .*ndf*) abspeichert. Listing 2 zeigt, wie Sie mithilfe des Befehls *CREATE TABLE* eine neue Tabelle anlegen. Wichtig ist hierbei, dass Sie Ihre Datenbankverbindung zur neu angelegten Datenbank manuell ändern, weil PostgreSQL den Befehl *USE* nicht kennt, mit dem man beim SQL Server die aktive Datenbank wählt.

Wie Sie im Listing sehen, hat jede erstellte Tabelle einen Eintrag in der Catalog View *pg_class* und wird über eine OID eindeutig innerhalb der Datenbank identifiziert. Mithilfe der Funktion *pg_relation_filepath()* ermitteln Sie das Verzeichnis (inklusive Datei), in dem der Inhalt der Tabelle gespeichert wird. In meinem Fall hat die neue Tabelle *TestTable1* die OID 29784, daher gibt es auch im Ordner *\$PGDATA/base/29783* (OID der Datenbank) eine neue Datei. Wie Sie in Bild 13 sehen, hat diese Datei anfangs eine Größe von 8 Kilobyte. Dies entspricht der Standard-Page-Größe von PostgreSQL, die auf Wunsch ebenfalls auf einen anderen Wert gesetzt werden kann.

Jede Datei einer Tabelle darf maximal 1 GByte groß werden. Erfordern Tabellen beziehungsweise Indizes mehr Datenspeicher, wird einfach eine neue Datei mit der Dateinamenserweiterung *.1*, *.2*, ... angelegt. Listing 3 zeigt wie Sie eine größere Tabelle erstellen und mit Daten befüllen. Das Statement *INSERT INTO* befüllt die Tabelle mit 100 Millionen Datensätzen, wodurch sich eine Gesamtgröße von rund 3,5 GByte für die Tabelle ergibt. Im Dateisystem erfordert diese Tabelle insgesamt vier Dateien (mit der OID 29788), wie Bild 14 zeigt. Neben den Datendateien werden noch zwei weitere Dateien angelegt, nämlich die Free-Space-

Map-Datei (.*fsm*) und die Visibility-Map-Datei (.*vm*). Die Free-Space-Map-Datei nutzt PostgreSQL, um den freien Speicherplatz auf den Pages einer Tabelle zu verwalten. Die

```
postgres@5d71369c69ff:~/data/base/29783$ ls -l
-rw-r--r-- 1 postgres postgres 8.0K Oct 18 12:19 29784
-rw-r--r-- 1 postgres postgres 8.0K Oct 18 12:42 29787
-rw-r--r-- 1 postgres postgres 1.0G Oct 18 12:39 29788
-rw-r--r-- 1 postgres postgres 1.0G Oct 18 12:43 29788.1
-rw-r--r-- 1 postgres postgres 1.0G Oct 18 12:44 29788.2
-rw-r--r-- 1 postgres postgres 385M Oct 18 12:45 29788.3
-rw-r--r-- 1 postgres postgres 888K Oct 18 12:42 29788_fsm
-rw-r--r-- 1 postgres postgres 112K Oct 18 12:44 29788_vm
-rw-r--r-- 1 postgres postgres 1.0G Oct 18 12:42 29791
-rw-r--r-- 1 postgres postgres 1.0G Oct 18 12:45 29791.1
-rw-r--r-- 1 postgres postgres 95M Oct 18 12:47 29791.2
-rw-r--r-- 1 postgres postgres 0 Oct 18 12:03 2995
-rw-r--r-- 1 postgres postgres 8.0K Oct 18 12:06 2996
```

Die 3,5 GByte große Tabelle mit der OID 29788 im Dateisystem erfordert vier Datendateien (Bild 14)

se ist vergleichbar mit der PFS Page (Page Free Space) einer Heap-Tabelle im SQL Server.

Die Visibility-Map-Datei speichert, welche Pages einer Tabelle ausschließlich Datensätze enthalten, die für alle aktuellen und zukünftigen Transaktionen sichtbar sind. Diese Datei hilft PostgreSQL dabei, Operationen wie *VACUUM* und sogenannte Index Only Scans zu optimieren. Mehr dazu erfahren Sie in den nächsten beiden Folgen dieser Serie.

Fazit

Im Rahmen dieses Einführungsartikels haben Sie die Grundlagen von PostgreSQL kennengelernt und gesehen, in welchen Bereichen es sehr große Unterschiede zum SQL Server gibt. Die nächste Folge dieser Serie betrachtet die Indizierung von PostgreSQL-Datenbanken und wie man ein Performance-Tuning durchführen kann. ■

- [1] PostgreSQL Installation, www.postgresql.org/download
- [2] Installation von PostgreSQL in einer virtuellen Maschine, www.dotnetpro.de/SL2503PostgreSQL1
- [3] PostgreSQL-Extension für das Azure Data Studio, www.dotnetpro.de/SL2503PostgreSQL2
- [4] DBeaver, <https://dbeaver.io>
- [5] pgAdmin, <https://www.pgadmin.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.

dnpCode

A2503PostgreSQL

● Listing 3: Befüllen einer größeren Tabelle

```
-- Create a larger table
CREATE TABLE LargerTable
(
    ID INT GENERATED ALWAYS AS
        IDENTITY PRIMARY KEY NOT NULL,
    Number INT
);

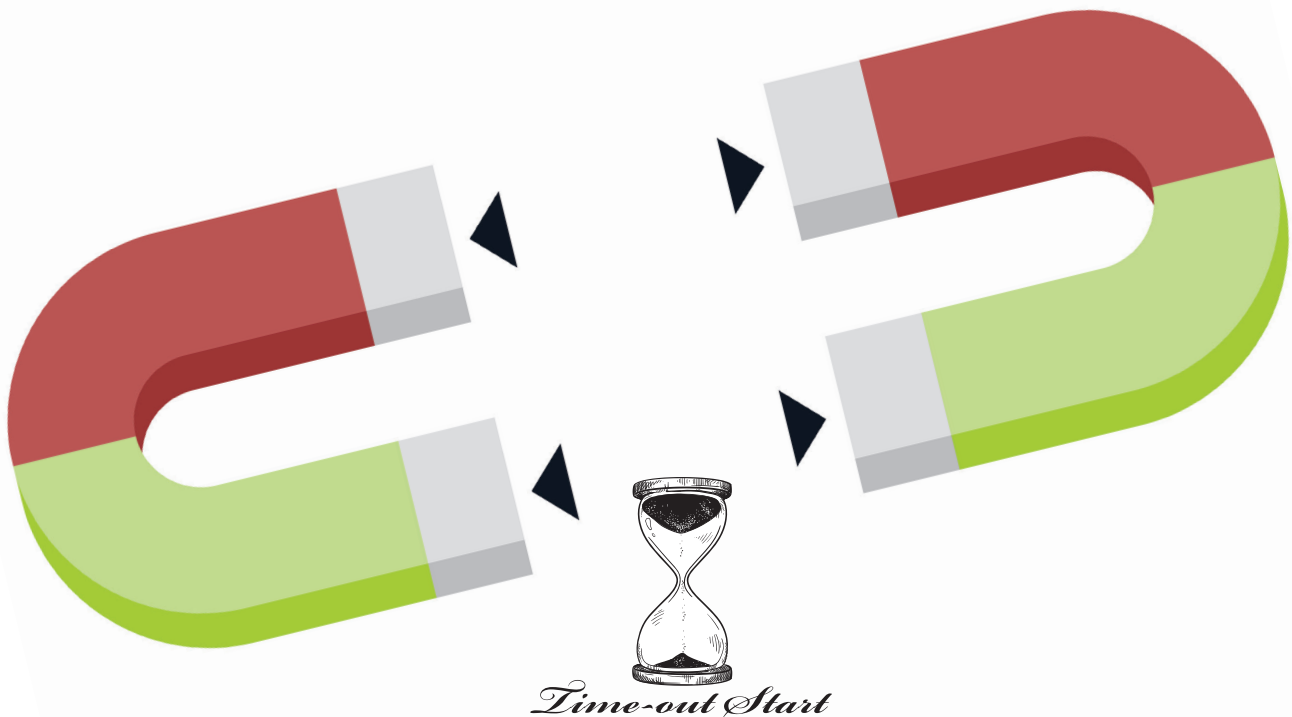
-- Insert 100m records
INSERT INTO LargerTable (Number)
SELECT generate_series(1, 100000000);

-- Shows the size of a given table
SELECT
    pg_size_pretty(pg_relation_size(
        'largertable')),
    relname,
    reltuples,
    relpages,
    *
FROM pg_class WHERE relname = 'largertable';
```


TIME-OUT

Exzellenz versus Glück

Wo bewegen Sie sich in diesem Spannungsfeld?



Es ist Samstagmorgen, und ich könnte entweder im Bett liegen bleiben und das Nichtstun genießen oder aufstehen, eine Runde Joggen gehen und mich dann fortbilden. Ich könnte schauen, welche neuen Posts es gibt, ausprobieren, wie man ein KI-API anprogrammiert, und mir Gedanken machen, ob wir unsere Software nicht komplett anders schreiben müssen, da es ja jetzt die Möglichkeit gibt, Spracheingaben der Benutzer zu verstehen und in sinnvolle Aufgaben umzuformulieren. Sicher würde ich damit Neuland betreten. Ich könnte mir psychologische Studien anschauen, wie denn eine solche neue Oberfläche gestaltet sein muss, um das Vertrauen der Benutzer zu gewinnen. Würde ein paar Nächte dran sitzen und dazu sicher auch etwas erfinden, wenn ich dranbleibe. Das traue ich mir zu. Könnte damit den Innovationspreis gewinnen und der Menschheit einen wichtigen Dienst erweisen. All das könnte ich.

Stattdessen drehe ich mich noch einmal im Bett um und bin einfach nur glücklich. Glücklich, dass die Sonne draußen strahlt und ich einfach faul sein darf. Ich schaue in die Augen meiner Frau und denke mir, wie schön es ist, dass wir uns haben. Dass wir Zeit füreinander haben. Und sie genießen können.

Vor zwei Jahrzehnten war ich in einem Hamsterrad gefangen. War damals für 30 Mitarbeiter zuständig, und jede Woche kamen sicher 60 bis 80 Arbeitsstunden auf die Uhr. Von den ersten Lebensmonaten meines ältesten Sohnes habe ich wenig mitbekommen. Dann kam eines Tages mein Chef zu mir und meinte, er müsse Insolvenz anmelden. Ich habe um den Lohn

der letzten Monate gekämpft, war gezwungen, ihn einzuklagen, und habe mir dann geschworen: Ich werde niemals mehr mein Leben so komplett auf eine Karte setzen, niemals mehr jedem sonnigen Tag nur vom Bürofenster aus verträumt hinterherschauen. Es gibt so etwas wie Glück im Moment, und nichts auf der Welt ist es wert, dass man ihm alles opfert.

Gegenpole

Wenn ich Interviews mit Elon Musk verfolge, dann fällt mir auf, dass es ein krankhafter Wahn sein muss, der ihn verfolgt. Da scheint irgendwann in der Kindheit etwas schiefgelaufen zu sein, und jetzt glaubt er, er kann sich durch Exzellenz sein Glück zurückkaufen. „Ich glaube, dass kein Mensch mit meinem Leben tauschen sollte, denn ich bin ein sehr unglücklicher Mensch“, sagte er einmal auf die Frage, wie es sei, Elon Musk zu sein. Sind nicht alle Genies irgendwie von einem Wahn verfolgt, von einer Sucht, die sie antreibt, mehr zu geben als jeder andere Mensch, mehr zu wagen, mehr zu forschen und Grenzen zu überschreiten?

Mein nächster Blick richtet sich auf dieses Land. Seine Satttheit und Trägheit. Der wirtschaftliche Abschwung, den wir bereits seit rund drei Jahren feststellen und der so langsam auch dem letzten Zweifler offenbar wird. Und ich muss an meine Besuche in Indien denken, wie Millionen junger Menschen sich nach Wohlstand und einer besseren Zukunft sehnen und jeden Tag zwölf Stunden oder mehr damit verbringen, besser zu sein als ihre zahlreichen Konkurrenten. Sie su-

Bild: Shutterstock / Nadia Snopek

chen Exzellenz. Oder ich sehe japanische Musiker, die üben, üben, üben und damit international Anerkennung finden für ihre außergewöhnlichen Leistungen.

Sind wir also die Glücklichen, und die anderen folgen nur einem Wahn? Haben wir diese Zeiten hinter uns? Ist das eine Entwicklungsstufe? Oder geht es wieder bergab, werden wir bald wieder total unglücklich sein, weil keiner mehr exzellent ist? Was sollen wir also tun: Ausschlafen und die Zeit genießen? Oder aufstehen? Oder anders gefragt: Wie lange können wir uns das Ausschlafen noch leisten?

Wir wissen, dass wir in Deutschland ohne eine Elite, die neue Patente hervorbringt, Impfstoffe entwickelt oder Motoren sparsamer macht, nicht weiterkommen. Aber muss ausgerechnet ich das sein? Reicht es nicht, wenn andere das machen?

Sind Glück und Exzellenz Gegensätze? Kann man nicht auch Glück empfinden, wenn man sich einer Sache hingibt? Sich ihr verschreibt und mit Leidenschaft und Freude Menschheitsprobleme löst?

Wie ist das mit der Spannung zwischen „Ich hab schon, passt“ und „Ich sehne mich nach ...“? In der Theologie gibt es den Bereich der Eschatologie, der Wissenschaft von den „letzten Dingen“. Der Mensch lebe in einem Schwebezustand zwischen „schon“ und „noch nicht“. Wir sind schon erlöst, aber noch nicht im Paradies. Man mag auf ein Paradies hoffen oder nicht, richtig ist, dass wir es uns ganz gut eingerichtet haben und es uns sehr gut geht. Bei der derzeitigen Lage kann man aber sicher nicht davon sprechen, dass wir im Schlaraffenland leben. Denn es gibt so viel zu tun, zu heilen, zu versöhnen, zu reparieren ...

Ist es also der ständige Wandel zwischen Dankbarkeit und Unzufriedenheit, zwischen Glück und Exzellenz, der uns wach hält? Kann Exzellenz besser gelingen, nachhaltiger, wenn wir auch einmal glücklich sind?

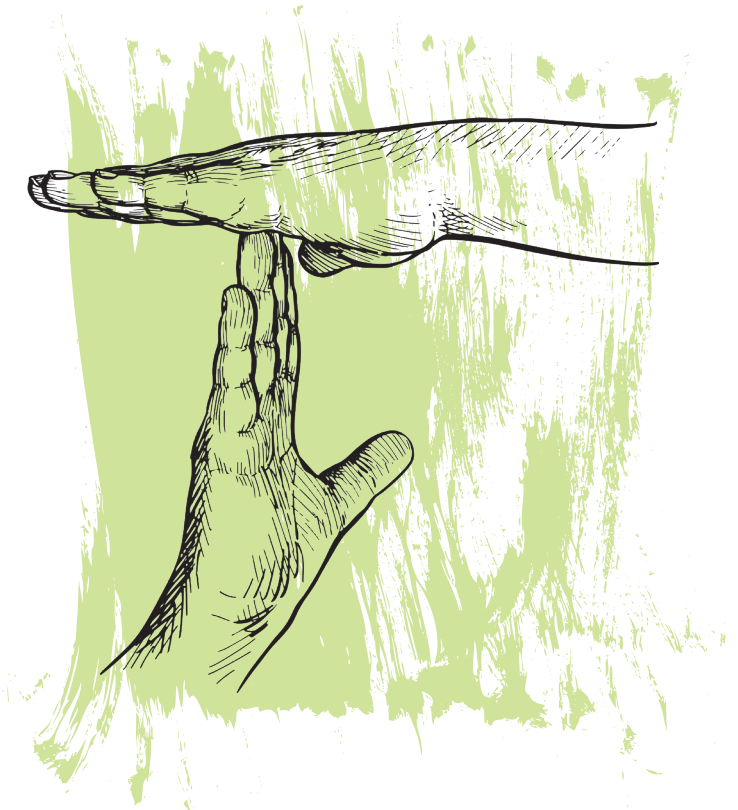
Sicher ist es so, dass man nur richtig gut ist, wenn man etwas gerne tut, aber es gibt dann auch immer die Momente, in denen man sich entscheiden muss: Mache ich jetzt weiter, oder gehe ich stattdessen heute Abend mit Freunden schön essen? Ja, man könnte einwenden, dass einem vielleicht gerade dann die besten Ideen kommen, wenn man etwas Abstand hat, wenn man nicht nur in seinem Gebiet ein Experte ist, sondern sich etwas breiter aufstellt. Mag sein. Ist hin und wieder sicher auch so. Aber wenn es hart auf hart kommt und man sich entscheiden muss zwischen Exzellenz oder Glück, zwischen 80 Stunden arbeiten oder 40 Stunden Dienst nach Vorschrift, wie würde Ihre Wahl ausfallen?

Ich gehe davon aus, dass Sie jünger sind als ich, dass Sie es tatsächlich noch zur Weltspitze schaffen könnten. Wollen Sie das? Und was geben Sie dafür auf? Ein Spitzensportler hat einmal gesagt, die entscheidende Frage ist nicht, wie gut man ist. Sondern vielmehr, was man bereit ist aufzugeben, um besser zu werden. Solange 22 Spieler ihre Jugend opfern, um die Besten im Fußball zu sein, können 60 000 zuschauen und sich bei Bier und Bratwurst entspannen.

Aber wer will zu den 22 gehören, die es in der Softwareentwicklung in die Spitze schaffen? Die in ihrem Beruf Exzellenz zeigen und den Innovationspreis für das beste User Interface bekommen? Sie? Es würde mich freuen. Und ich wünsche Ih-

nen dabei auch einige Momente Glück. Denn ganz ohne Glück sollten Sie nicht durchs Leben ziehen. Ob das unbeschreibliche Gefühl, am Ende der Erste oder Einzige zu sein, der etwas erreicht hat, ausreicht, viele andere Glücksmomente aufzugeben, müssen Sie letztendlich selbst entscheiden.

Und ich selbst? Ich versuche, immer wieder zwischen „schon“ und „noch nicht“ zu wandeln, beides im Blick zu ha-



ben und dabei nicht zu extrem zu sein. Nicht zur Weltspitze aufzuschließen, aber trotzdem Exzellenz zu erreichen. Ich hoffe, Sie nehmen mir das nicht übel. In diesem Sinne wünsche ich Ihnen: Happy Coding. ■



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.

dnpCode

A2503Timeout

BOUNDED CONTEXTS IDENTIFIZIEREN UND VISUALISIEREN

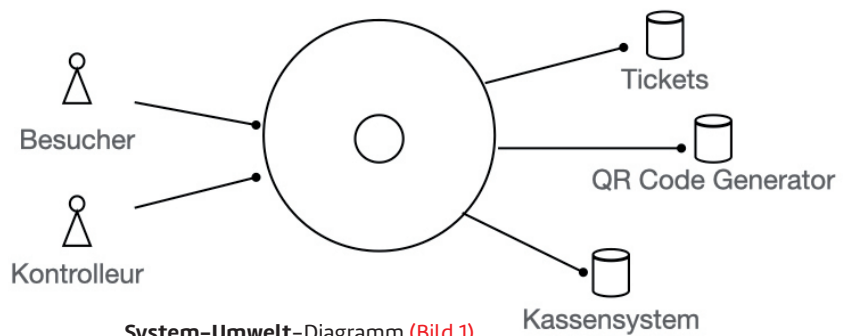
Sauber zerlegt

Bounded Contexts sind ein wichtiges Konzept aus dem Domain Driven Design. Wie man ein System in unabhängige Bestandteile zerlegen kann und wie man diese visualisiert.

Der Begriff Bounded Context stammt aus dem Domain Driven Design [1]. Mit Bounded Contexts kann ein komplexes Softwaresystem in kleinere, unabhängige Teile zerlegt werden. Dies bietet wichtige Vorteile:

- Domänenkonzepte können innerhalb eines Bounded Contexts differenzierter betrachtet werden als auf globaler Ebene des Systems. Innerhalb eines Contexts werden Domänenkonzepte auf die Anforderungen in diesem Context reduziert, was die Komplexität reduziert.
- Jeder Bounded Context hat die eigene Hoheit über seine Datenhaltung. Dadurch müssen Entscheidungen nicht global auf Systemebene getroffen werden. Innerhalb eines jeden Contexts kann entschieden werden, welche Daten gespeichert werden und wie diese gespeichert werden. Im einen Fall mag die technische Lösung in einer SQL-Datenbank liegen, im anderen Fall in einer NoSQL-Lösung oder dergleichen.
- Die einzelnen Bounded Contexts sind getrennt entwickel- und deploybar. Auch das reduziert die Komplexität und fördert eine agile Vorgehensweise, weil sich Entscheidungen jeweils nur auf einen kleineren Ausschnitt des Systems beziehen.
- Wird zwischen den Bounded Contexts Messaging verwendet, um Informationen auszutauschen, passt das Konzept der Bounded Contexts sehr gut zum populären Event Sourcing, bei dem statt des jeweils aktuellen Stands der Informationen die einzelnen Ereignisse persistiert werden.

In der Regel bringen Konzepte wie Bounded Contexts neben den Vorteilen auch Herausforderungen mit sich. Im Fall der Bounded Contexts liegt eine solche Herausforderung darin, dass die Aufteilung des Systems in kleinere Teile letztendlich doch wieder ein Gesamtsystem ergeben muss. Es stellt sich also die Frage, wie Informationen zwischen den Contexts transportiert werden. Dazu können Events verwendet werden, um Informationen über Ereignisse innerhalb der Domäne mitzuteilen. So könnte ein Bounded Context beispielsweise veröffentlichen, dass eine neue Bestellung vorliegt. Ist dies in einem anderen Context von



System-Umwelt-Diagramm (Bild 1)

Interesse, registriert sich dieser Context für den Event. So kann zwischen den Contexts über eine Messaging-Infrastruktur ein Informationsaustausch stattfinden. Auf der einen Seite sorgt dies für eine Entkopplung der Contexts. Innerhalb eines Contexts können Änderungen vorgenommen werden, ohne dass andere Contexts davon betroffen sind, sofern der Kontrakt der Events weiterhin eingehalten wird. Das sorgt für Flexibilität. Auf der anderen Seite wird das Gesamtsystem damit etwas komplexer, weil das Messaging zwischen den Contexts geplant und realisiert werden muss. Auch im Betrieb erhöht das die Komplexität, weil bei der Fehlersuche die Kommunikation über Events betrachtet werden muss. Man kann hier bereits erkennen, dass die Einführung von Bounded Contexts wohlüberlegt erfolgen muss.

Beispiel Kinoverwaltung

Die folgenden Betrachtungen sollen an einem Beispiel erläutert werden: einer Kinoverwaltung. Das System soll diverse Anforderungen erfüllen, die hier nur skizziert werden:

- Kinobesucher sollen Tickets für eine Vorstellung online kaufen können.
- Im Kino soll der Kauf von Tickets sowohl an einem Schalter möglich sein als auch an Self-Service-Automaten.
- Am Eingang jedes Saals sollen die Tickets mit einer Smartphone-App auf Gültigkeit kontrolliert werden können.
- Das Kinoprogramm für die Folgewoche soll geplant werden können.
- Statistiken über Umsatz, Auslastung, Gewinn pro Vorstellung und Ähnliches sollen erstellt werden können.

Rollen	

Kinobesucher	
Kontrolleur	
Betreiber	
Ressourcen	Data vs. API

Tickets	Data
QR Code Generator	API
QR Code Scanner	API
Saalplan	Data

Einfache tabellarische Darstellung von Rollen und Ressourcen (Bild 2)

Dieses System ist ausreichend groß, um es sinnvoll in Bounded Contexts aufteilen zu können. Das bedeutet zwar noch lange nicht, dass die Einführung von Bounded Contexts zwingend ist. Doch als Beispiel für eine Aufteilung in Bounded Contexts kann die Kinoverwaltung allemal dienen.

Wir verwenden dieses Beispiel häufig in unseren Architekturtrainings, um der Frage nachzugehen, wie man Bounded Contexts identifizieren kann und wie man sie darstellt. Im Folgenden wird beides erläutert.

Bounded Contexts identifizieren

Um mögliche Bounded Contexts eines Systems identifizieren zu können, beginnen wir mit einem System-Umwelt-Diagramm (Bild 1). Darin werden die Rollen sowie die Ressourcen dargestellt, die mit dem System in Zusammenhang stehen.

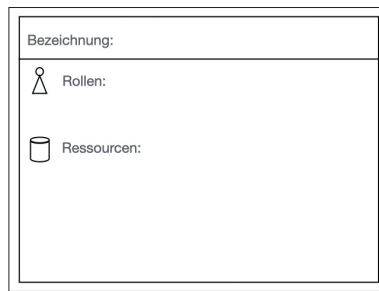
Ein System-Umwelt-Diagramm hat die Aufgabe, das zu erstellende System in seiner Umwelt darzustellen. In der Umwelt gibt es auf der einen Seite Rollen und auf der anderen Seite Ressourcen. Beide gehören nicht zum System, sondern liegen in der Umwelt des Systems. Rollen sind vom System abhängig, das System ist von Ressourcen abhängig.

Rollen

Rollen sind vom System abhängig, erhalten also vom System eine bestimmte Leistung. So kann beispielsweise ein Kinobesucher online Tickets kaufen, während der Kontrolleur die Tickets mithilfe des Systems auf Gültigkeit prüfen kann. Für beide Rollen bietet das System Kinoverwaltung Leistungen an.

Zwischen den Anforderungen beider Rollen gibt es Kontaktpunkte, an denen Dinge zusammenspielen müssen. Dies betrifft beispielsweise die Tickets. Erstellt man etwa für die beiden Rollen Kinobesucher und Kontrolleur getrennte Bounded Contexts, muss sichergestellt sein, dass die Gültigkeit der Tickets, die im einen Context erstellt werden, im anderen Context geprüft werden kann. Es müssen also manche Informationen, die im einen Context anfallen, dem anderen Context zur Verfügung gestellt werden. Aber eben nur manche Informationen, denn der Kontrolleur ist nicht an allen Daten eines Tickets interessiert, wie etwa dem Preis.

Eine Aufteilung der gesamten Anforderungen eines Systems auf verschiedene Rollen ist wichtig, um den einzelnen Rollen jeweils eine geeignete Benutzerschnittstelle anbieten zu können. Jede Rolle erhält später eine eigene Anwendung zur Verfügung gestellt. Dadurch können rollenspezifische Anforderungen an die Benutzerschnittstelle bedient werden. Zum einen können wir so die Wünsche nach Smartphone versus Desktop versus Web abdecken. Gäbe es keine Unterscheidung der Rollen, wäre eine einzige Anwendung für

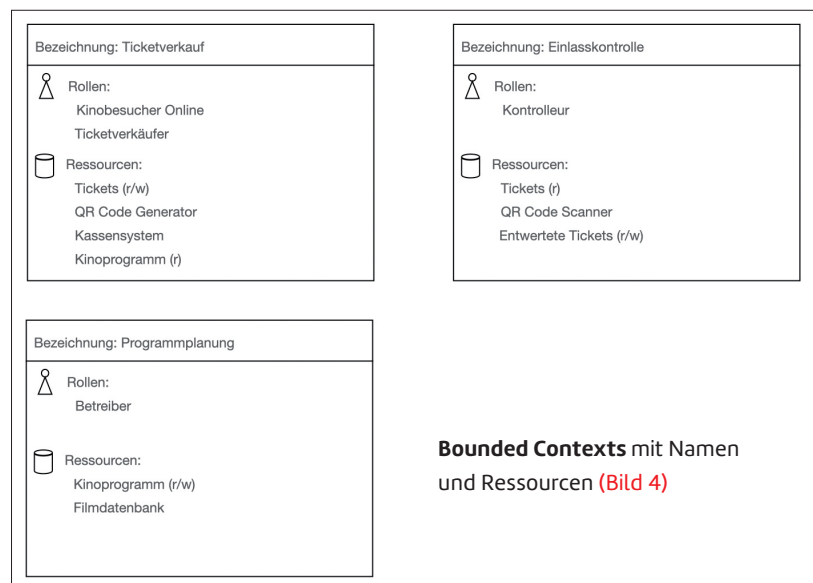


Template „Bounded Context“ (Bild 3)

sämtliche Anforderungen zuständig. Dann wäre eine Differenzierung in Smartphone, Desktop oder Web nicht möglich. Zum anderen können auch innerhalb derselben Plattform, beispielsweise Desktop, unterschiedliche Anforderungen der Rollen an die Benutzerschnittstelle bedient werden. Tätigkeiten, die täglich zig-fach ausgeführt werden, erhalten eine andere Schnittstelle (etwa Tastaturkürzel) als Tätigkeiten, die nur gelegentlich durchgeführt werden müssen (etwa ein Wizard, der neue Benutzer Schritt für Schritt durch einen Prozess führt).

Ressourcen

Die im Beispiel genannten Tickets sind aus Sicht des Systems eine Ressource. Von dieser Ressource ist das System abhän-



Bounded Contexts mit Namen und Ressourcen (Bild 4)

gig. Ohne die Ressource Tickets kann das System Ticketverwaltung seine Leistung nicht bereitstellen.

Eine weitere Ressource stellt ein QR-Code-Generator dar. Dieser wird vom System verwendet, um einen QR-Code zu erzeugen, der in das Ticket eingefügt wird. Ein solcher QR-Code-Generator kann ein Softwarepaket sein, das vom System verwendet wird. Auch ein Webservice wäre für diesen Zweck denkbar. Die beiden Ressourcen Tickets und QR-Code-Generator unterscheiden sich in einem wichtigen Punkt: Tickets sind Daten, während der QR-Code-Generator eine Dienstleistung bereitstellt. Dies wird im weiteren Verlauf der Visualisierung wichtig, wenn es um die Frage geht, welche Ereignisse ein Bounded Context auslösen muss, damit ein anderer Bounded Context Informationen empfangen kann. Datenressourcen können solche Ereignisse auslösen und durch sie beeinflusst werden. Für Dienstleistungsressourcen ist diese Frage unerheblich. ►

Bei den Datenressourcen ist es wichtig, auf den Inhalt abzielen statt auf eine konkrete Technologie. Aus Sicht des System-Umwelt-Diagramms ist das System von Tickets abhängig, nicht von einer Datenbank. Möglicherweise werden die Tickets in einer Datenbank persistiert. Doch das ist eine technische Entscheidung, die erst viel später durchgeführt wird. In dieser frühen Phase geht es darum, die Domäne zu verstehen und mit Domänenbegriffen zu arbeiten. Im Domain Driven Design wird dies die Ubiquitous Language genannt. Auf Deutsch: die allgegenwärtige Sprache.

Im System-Umwelt-Diagramm ist eine dritte Ressource dargestellt: das Kassensystem. Dies ist eine spezielle Hardware, die vom System angesteuert werden muss, um beispielsweise die Schublade für Bargeld zu öffnen oder den Betrag auf einem Display anzuzeigen. Auch Hardware stellt gegenüber dem System eine Dienstleistung bereit und wird über ein API angesteuert.

Als Ergebnis des System-Umwelt-Diagramms verfügen wir nun über eine Liste von Rollen und Ressourcen. Bei den Ressourcen können wir zusätzlich zwischen Daten und Dienstleistungen (APIs) unterscheiden.

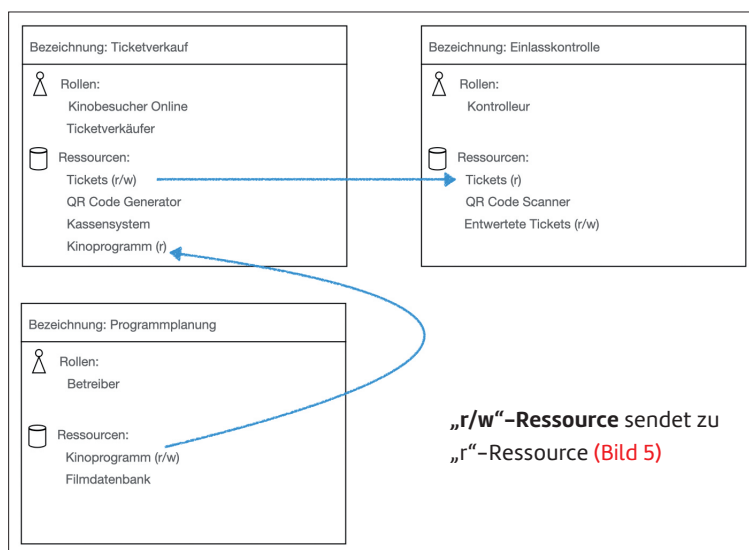
Darstellung in zwei Tabellen

Rollen und Ressourcen können tabellarisch dargestellt werden. **Bild 2** zeigt dies beispielhaft für die schon genannte Kinoverwaltung. Das Beispiel ist hier nur angerissen und deckt eine Kinoverwaltung nicht vollständig ab. Es soll lediglich gezeigt werden, dass eine tabellarische Darstellung von Rollen und Ressourcen möglich ist.

Von den Rollen zu Bounded Contexts

Nachdem nun die Rollen und Ressourcen identifiziert sind, können wir beginnen, die Bounded Contexts zu planen.

Im ersten Schritt ist es zweckmäßig, jeder Rolle einen eigenen Bounded Context zuzuordnen. Damit hat man eine feingranulare Aufteilung des Systems in Bounded Contexts getroffen. Sollte sich dies später als nachteilig erweisen, können Contexts zusammengefasst werden.



Die erste Aufgabe besteht darin, Namen für die Bounded Contexts zu finden.

Nachdem dadurch eine erste Idee der Contexts entstanden ist, beginnt man damit, die Ressourcen den Bounded Contexts zuzuordnen. Manche Ressourcen werden nur in einem einzigen Bounded Context benötigt, andere sind in mehreren erforderlich. Die Frage, die wir uns zu jeder Ressource stellen können, lautet: Benötigt ein Context die Ressource, um seine Leistung erbringen zu können?

In dieser Phase passiert es übrigens häufig, dass man weitere Ressourcen identifiziert, die zuvor noch nicht bedacht wurden. Das ist bei einem kreativen Prozess völlig normal. Man ergänzt die Ressource in der Auflistung, und das Problem ist erledigt.

Zur Darstellung der Bounded Contexts kann jetzt bereits ein Template verwendet werden, in dem nach und nach weitere Details ergänzt werden. Im ersten Schritt geht es um die Namen der Contexts, im zweiten Schritt um die Zuordnung der Ressourcen. Die Ergebnisse könnten so aussehen wie in **Bild 3** und **Bild 4**.

Ressourcen führen zu den Events

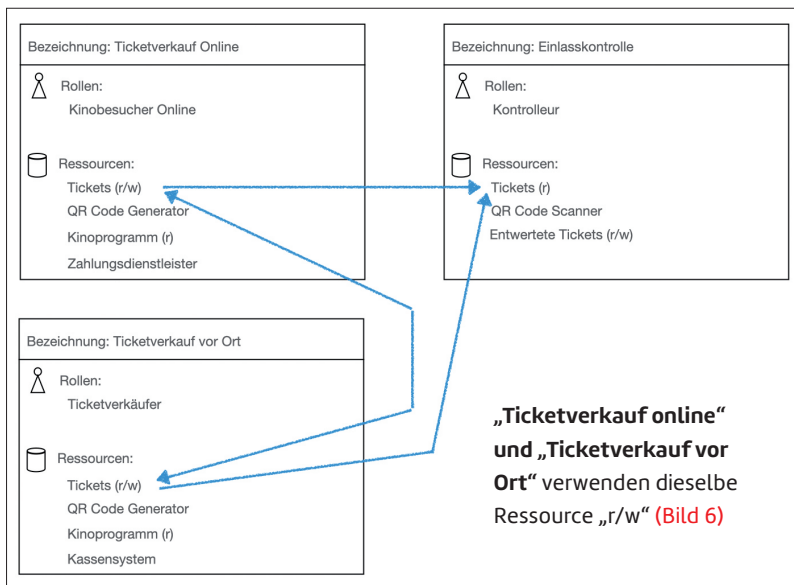
Bei den Datenressourcen ist es jetzt wichtig, festzustellen, ob die Daten innerhalb eines Bounded Contexts nur gelesen werden oder ob diese hier auch entstehen beziehungsweise geändert werden. In der Praxis genügt es, zwischen *read* und *read-write* zu unterscheiden.

Interessant sind vor allem die Fälle, in denen eine Ressource in mehr als einem Bounded Context benötigt wird. Hier ist zunächst zu prüfen, in wie vielen Contexts die Ressource mit *read-write* markiert ist. Ist dies nur in einem Context der Fall, erleichtert es die Modellierung der Events. In diesem Fall sendet nur der eine Context Events über diese Daten aus (**Bild 5**). Alle anderen Contexts, in denen die Ressource ebenfalls verwendet wird, aber eben nur *read-only*, sind reine Empfänger der Daten. Sie subscriben sich auf den Event und aktualisieren jeweils ihre lokale Sicht auf die Ressource.

Knifflig kann es werden, wenn eine Ressource in mehreren Bounded Contexts mit *read-write* markiert wird. Das bedeutet nämlich, dass Änderungen an mehreren Stellen anfallen können und somit die Frage nach der Konsistenz beantwortet werden muss.

Nehmen wir das Beispiel der Kinoverwaltung: Können hier Tickets in zwei Bounded Contexts erstellt werden, muss ein Abgleich stattfinden, um zu verhindern, dass ein Platz in einer Vorstellung mehrfach verkauft wird (**Bild 6**).

In solchen Fällen ist es meist hilfreich, den Schnitt der Bounded Contexts noch einmal zu überdenken. Gegebenenfalls hilft es hier, zwei Contexts zusammenzufassen. Alternativ kann man eine Verantwortlichkeit an einen Service auslagern, den beide Bounded Contexts nutzen. Dann liegt die Sicherstellung der Konsistenz in diesem einen Service, was einfacher ist, als die Konsistenz verteilt sicherzustellen (**Bild 7**). Die Bounded Contexts sind dann abhängig vom Service.



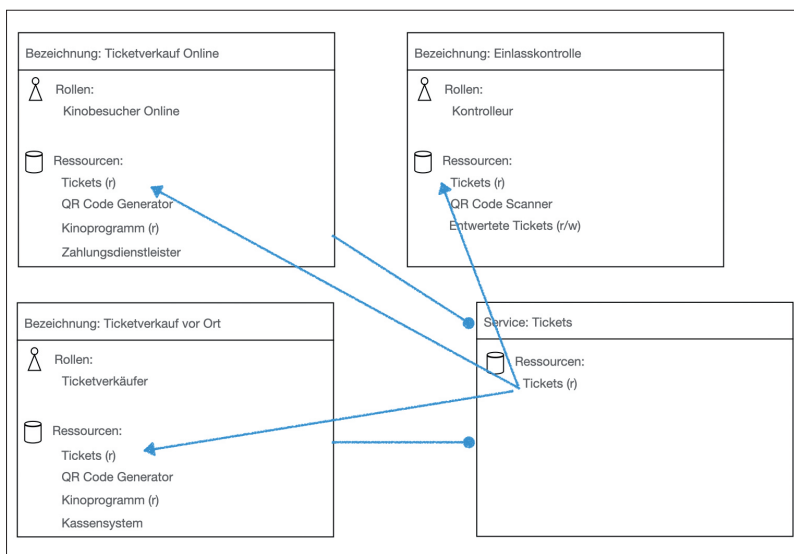
cherweise nicht, für welchen Platz das Ticket verkauft wurde. Die Repräsentation der Tickets kann also in den Bounded Contexts unterschiedlich sein.

Hier kommt es bei der Modellierung der Ressourcen darauf an, zu welchem Zeitpunkt man sich mit diesen Details befasst. Im ersten Schritt ist es hilfreich, auf hoher Flughöhe zu bleiben. Es wird zunächst nur festgehalten, dass diverse Contexts die Ressource *Ticket* benötigen. Wenn die Bounded Contexts gefunden und visualisiert sind, kann man die Ressourcen näher spezifizieren. Anschließend kann dann pro Context geprüft werden, ob dort eine andere Sicht auf die Ressource sinnvoll ist.

Fazit

Das Identifizieren der Bounded Contexts über die Rollen hat sich in der Praxis sehr gut bewährt. Ist ein Context für mehr als eine Rolle verantwortlich, werden den Rollen jeweils eigene Anwendungen zur Verfügung gestellt. Die Anwendungen innerhalb eines Contexts können auf die Daten des Contexts unmittelbar zugreifen. Ein Zugriff auf die Daten eines anderen Contexts ist dagegen verboten, weil so die Unabhängigkeit der Contexts nicht mehr gegeben wäre. Anstelle eines direkten Zugriffs sorgen Events dafür, dass Domäneninformationen von einem zum anderen Context transportiert werden.

Die Visualisierung ist zunächst bewusst einfach gehalten, um zügig voranzukommen. Sind die Contexts erst einmal identifiziert und visualisiert, können zusätzlich auch andere Darstellungsformen hilfreich sein, um weitere Details zu erfassen. Ein Beispiel dafür ist der Bounded Context Canvas [2]. Für den Einstieg in die Modellierung wäre mir diese Darstellung zu detailliert. Zu einem späteren Zeitpunkt dagegen mag sie hilfreich sein. ■



Service für Tickets (Bild 7)

Visualisierung

Die einzelnen Bounded Contexts können in einem sehr einfachen Template dokumentiert werden, das in Bild 3 zu sehen ist. Nachdem man die Contexts benannt hat, ordnet man die Rollen und Ressourcen zu. Anschließend beginnt man, die Events als Datenflüsse einzuzeichnen. Diese beginnen jeweils an einer *read-write*-Datenressource und enden in den *read*-Repräsentationen der Ressource.

So werden beispielsweise bei den Tickets als *read-write*-Ressource Events ausgelöst. Im Context Ticketverkauf wird dazu der Event *Neues Ticket wurde verkauft* gepublikt. Die *read*-Versionen der Ressource empfangen dieses Ereignis und können ihre eigene Repräsentation der verkauften Tickets aktualisieren.

Dabei müssen nicht alle Daten relevant sein. So könnte beispielsweise ein Bounded Context für Reportingzwecke alle verkauften Tickets speichern. Dabei interessiert aber mögli-

[1] Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2003, ISBN 978-0-321-12521-7

[2] The Bounded Context Canvas auf GitHub, www.dotnetpro.de/SL2503BoundedContexts1



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>

dnpCode

A2503BoundedContexts

WAS AUS VERSÄUMNISSEN FOLGT

Die Konsequenz von Schulden

Im Verlauf wohl eines jeden Projekts entstehen Schulden in der Softwareentwicklung. Zu welchen Problemen das führen kann, zeigt der Artikel.

Schulden in der Softwareentwicklung sind ein auf den ersten Blick recht einfaches, aber dann doch hochkomplexes Problem. In der ersten Folge [1] unserer Reihe zu diesem Thema haben wir uns die Natur der Schulden und die Unterteilung in externe und interne Schulden angeschaut. Im zweiten Teil [2] sind wir auf die verschiedenen Arten von Schulden eingegangen. In diesem Artikel nun wollen wir uns mit dem eigentlichen Problem der Schulden beschäftigen: mit den Konsequenzen – denn da wird es erst richtig spannend.

Finanzen: Der Vergleich hinkt

Wie in [1] bereits geschildert, werden bei der Beschreibung von Schulden in der Softwareentwicklung oft Vergleiche mit der Finanzwelt herangezogen. Was auf den ersten Blick logisch erscheint, entpuppt sich dann aber doch als „zu einfach gedacht“. Sehen wir uns das an einem Beispiel an. Angenommen, Sie haben für Ihr Traumhaus einen Kredit in einer bestimmten Höhe aufgenommen, so hat dies Auswirkungen auf Ihr weiteres Leben. Der Kreditgeber möchte das geliehene Geld selbstverständlich zurückhaben, und so vereinbaren Sie schon zu Beginn die monatlichen Raten, mit denen Sie Ihren Kredit an das Finanzinstitut zurückbezahlen.

Das ist Ihre persönliche Konsequenz: Durch die monatlichen Raten sind Ihre finanziellen Möglichkeiten begrenzter als vor der Kreditaufnahme. Diese Konsequenz wirkt sich für eine bestimmte Zeit auf Ihre Kaufkraft aus und hindert Sie beispielsweise an bestimmten Urlauben, am Kauf eines teureren Autos und so weiter. Die Beziehung zwischen den Schulden und der finanziellen Konsequenz ist offensichtlich und leicht nachzuvollziehen. Wollen Sie zu einem bestimmten Zeitpunkt wieder zu alter Finanzkraft zurückkehren, so müssen Sie lediglich sicherstellen, dass Sie bis zu diesem Zeitpunkt den Kredit beglichen haben.

Gefahren durch mangelnde Kontrolle

Diese einfache Art der Konsequenz existiert in der Softwareentwicklung leider nicht. Während im Finanzbereich ein Geldgeber, in der Regel eine Bank, dafür sorgt, dass Sie jeden Monat einen kleinen Betrag zurückzahlen, gibt es einen solchen „Geldeintreiber“ in der Softwareentwicklung nicht. Hier zwingt Sie niemand dazu, Ihre Schulden zu begleichen.

Auf der anderen Seite wird Ihnen keine Bank einen weiteren Kredit geben, wenn beispielsweise Ihr Haus bereits als Sicherheit für einen anderen Kredit hinterlegt ist. Dadurch stellt die Finanzwelt sicher, dass Sie keine Schulden in unbegrenzter Höhe aufnehmen können. In der Softwareentwick-

lung hingegen hindert Sie potenziell niemand daran, jeden Tag weitere Schulden auf Ihren Schuldenberg zu packen. „Wo kein Kläger, da kein Richter“, mögen Sie sich jetzt denken, also warum nicht Schulden in Ihrem Softwareprojekt in unbegrenzter Höhe anhäufen?

Konsequenzen

Natürlich haben auch Schulden in der Softwareentwicklung Konsequenzen.

Anders als in der Finanzwelt sind die Konsequenzen in der Softwareentwicklung jedoch sehr diffus und oft unklar. Der Grund: Selbst mit sehr viel Wissen und Erfahrung ist es in vielen Fällen nur schwer möglich, ein in der Softwareentwicklung aufgetretenes Problem auf eine ganz bestimmte Art von Schulden zurückzuführen.

Ein Beispiel gefällig? Nehmen wir an, Sie stellen ein neues Entwicklungsteam ein, welches nur aus Studenten besteht. Dieses Team sperren Sie standesgemäß im Keller ein und lassen es über Jahre eine Softwareanwendung schreiben. Diese wird am Markt aber irgendwann nur schwer zu verkaufen sein, weil jede Änderung daran so teuer ist, dass Sie kaum noch auf Marktänderungen reagieren können.

Warum die Änderungen so teuer sind? Weil Ihre Entwickler stundenlang im Quellcode die richtige Stelle suchen müssen, um die Erweiterung vorzunehmen. Warum das so lange dauert? Weil die Struktur im Projekt sehr inhomogen ist und jeder Entwickler seinen Code anders strukturiert hat. Und warum haben die Entwickler das gemacht? Weil sie es von Anfang an nicht richtig wussten und immer wieder dazugelernt haben. Warum wussten sie es von Anfang an nicht? Weil sie nach dem Studium nicht wirklich Ahnung von der Softwareentwicklung hatten und einfach „drauflosprogrammiert“ haben. Und warum hatten sie das Wissen nicht? Weil keine Weiterbildung geplant und durchgeführt wurde.

Damit haben wir einen möglichen Schuldentyp (von vermutlich mehreren) identifiziert: Weiterbildungsschulden. Weil am Anfang an der Weiterbildung der Entwickler gespart wurde, hat man nach zehn Jahren ein Produkt, das nicht mehr weiterentwickelt werden kann.

Hätte ich Ihnen als Problem allein die schwer verkäufliche Anwendung geschildert, hätten Sie schon einige Analysen durchführen und viele Details eruieren müssen, um zu dieser Erkenntnis zu kommen. Genau das ist die Gefahr der Schulden in der Softwareentwicklung. Es existieren viele verschie-



dene Arten (siehe [2]), und diese Arten können sehr viele unterschiedliche Konsequenzen haben. In der Realität ist es aber noch viel komplizierter, da oft mehrere Schuldenarten parallel angehäuft werden und diese sich in den Konsequenzen verstärken oder in Kombination vollkommen neue Konsequenzen auslösen.

Die Suche nach genau so einer Ursache zu einem Problem ist mein täglicher Job seit über einem Jahrzehnt. Lassen Sie sich versichern, damit ist nicht zu spaßen.

Fehlendes Wissen

Verstehen Sie das nun bitte nicht falsch: Ich bin nicht der Peter Zwegat der Softwareentwicklung. Doch ich habe mir ganz spezielles Wissen angeeignet, um diese Analysen machen zu können: Wissen über die Softwareentwicklung. Ich will Ihnen natürlich in diesem Bereich keine Kompetenzen absprechen, aber meine Erfahrung aus Hunderten solcher Projekte zeigt: Entwickler, Architekten, Product Owner und Co. sind meist auf zwei Dinge fokussiert: ihre Arbeit, die sie jeden Tag erledigen müssen (Codierung, Architektur, Anforderungen und so weiter), und die eigentliche Fachdomäne, in der die Anwendung betrieben wird.

Der Aufbau von Wissen um das „big picture“, also wie Softwareentwicklung abstrakt und im Allgemeinen funktioniert, kommt dabei meist zu kurz. Ich möchte an dieser Stelle nicht den „Angeber“ spielen, sondern Sie auf ein entscheidendes Problem hinweisen: In vielen Teams existiert nicht genug Wissen über die Softwareentwicklung, um mit Schulden auf der einen Seite umgehen zu können und auf der anderen Seite die dadurch entstehenden Konsequenzen richtig analysieren, einschätzen und behandeln zu können. Zu jedem Problem in der Softwareentwicklung gibt es eine Ursache, die so gut wie immer einer Schuldenart entweder direkt oder indirekt zugeschrieben werden kann. Aber aus Mangel an Wissen – und natürlich auch an Zeit – bleibt dieser Zusammenhang oft lange oder für immer unentdeckt, und während die Schuldenberge auf der einen Seite immer größer werden, wächst auf der anderen Seite der Berg an Problemen immer weiter, bis irgendwann der große Knall kommt.

Wie groß ist das Problem?

Haben Sie dann in Ihrem Projekt irgendwann zehn verschiedene Schuldenarten zu zehn verschiedenen Schuldenbergen aufgetürmt, kann mit den vorhandenen Ressourcen in einem realistischen Zeitrahmen meist keine Lösung mehr gefunden werden. Diese Situation erlebe ich sehr oft, viel öfter, als mir lieb ist. Das Erschreckende dabei ist aber häufig das Ergebnis der Analyse: Seit Jahren wurde in vielen Bereichen unbewusst – oder manchmal auch bewusst – nicht ordentlich gearbeitet, und damit wurde das Problem immer schlimmer. Während dieser Zeit hat niemand in den Projekten erfasst, welche Art und welche Höhe an Schulden aufgebaut wurde.

In einem meiner ersten Projekte dieser Art wurde ich einmal vom Geschäftsführer verabschiedet mit dem Satz „Herr Tielke, bis Freitag sagen Sie mir bitte, wie groß das Problem tatsächlich ist.“ Das kann ich natürlich ermitteln, doch sollte es in jedem Projekt bereits einen internen Mitarbeiter geben,

der genau diese Frage beantworten kann: Wie hoch sind unsere Schulden in welchen Bereichen? Nur wenn über die Art und Menge der Schulden „Buch geführt“ wird, existiert überhaupt eine realistische Chance, diese Schulden in naher Zukunft Schritt für Schritt abzubauen.

Ist der wahre Schaden dann ermittelt, schließt sich immer direkt die nächste Frage an: „Wie kommen wir von dem Schuldenberg runter?“. Genau hier zeigt sich, dass man das Problem der Schulden nicht erkannt hat. Schulden entstehen in der Softwareentwicklung, weil Fehler gemacht wurden. Es wurde nicht weitergebildet, eine Rolle nicht besetzt, ein Dokument nicht erstellt oder was auch immer. Um die Konsequenzen zu lindern, ist für viele Mitarbeiter die oberste Priorität, die Schulden so schnell wie möglich abzubauen. Aber statt sie abzubauen, sollte man zunächst den oder die Gründe finden, warum diese Schulden überhaupt entstanden sind. Es bringt nichts, wenn Sie pro Tag zehn Stunden Schulden abbauen, aber auf der anderen Seite 15 Stunden neue Schulden aufbauen, weil die Mitarbeiter noch immer nicht geschult sind oder die Architektur nicht die richtige ist.

Fazit

Nahezu alle Probleme, die Sie in der Softwareentwicklung tagtäglich erleben, sind direkte oder indirekte Konsequenz der Schulden, die im Projekt angehäuft wurden. Diese Konsequenzen können Ihr Projekt auf viele verschiedene Weisen negativ beeinflussen oder sogar zum Scheitern bringen. Deshalb ist es wichtig, strukturiert vorzugehen. Schnellschüsse mit Halbwissen lösen oft nicht nur das Problem nicht, sondern verbrennen auch noch mehr Geld als das ursprüngliche Problem. Versuchen Sie zu jedem Problem in der Softwareentwicklung die Ursache zu ermitteln und die Art der Schulden zu bestimmen. Wenn die Ursache gefunden wurde, stellen Sie diese unverzüglich ab – und erst dann machen Sie sich Gedanken dazu, wie Sie den existierenden Schuldenberg portionsweise abtragen können.

Was einfach klingt, ist natürlich in der Praxis wesentlich komplizierter: Sie benötigen an den richtigen Stellen das richtige Wissen und auch die richtigen Rollen, die sich um verschiedene Schuldenarten kümmern. Aber das ist Stoff für die nächsten Artikel zu diesem Thema – seien Sie gespannt! ■

[1] David Tielke, *Der Schuldensumpf*, dotnetpro 1/2025, Seite 48 f., www.dotnetpro.de/A2501DDD

[2] David Tielke, *Nicht nur in der Technik...*, dotnetpro 2/2025, Seite 54 f., www.dotnetpro.de/A2502DDD



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

dnPCode

A2503DDD

LOKALISIERUNG FÜR ASP.NET CORE UND BLAZOR

Saluton Mondo!

ASP.NET Core und Blazor internationalisieren – gewusst wie!

Bereits in den beiden zurückliegenden Ausgaben der dotnetpro [1][2] haben wir das Thema Internationalisierung von Webanwendungen mit dem Microsoft-Stack behandelt. Alle relevanten Begrifflichkeiten sind geklärt, und wir sind nun in der Lage, Ressourcendateien mit Übersetzungen anzulegen und diese in ASP.NET-Core-Anwendungen zu integrieren. In diesem abschließenden Teil unserer kleinen Serie gehen wir auf einige Möglichkeiten ein, den Prozess der Lokalisierung per Code zu beeinflussen, und vervollständigen den Stack: Blazor fehlt noch (aber, Spoiler, so viel gibt es da gar nicht).

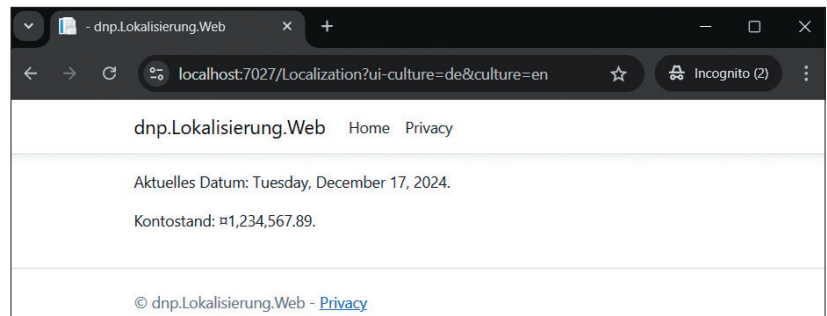
Noch einmal kurz zur Rekapitulation: Die ASP.NET-Core-eigene Middleware, per `builder.Services.AddLocalization()` in `Program.cs` aktiviert, holt sich Übersetzungen aus entsprechend angelegten Ressourcendateien (*.resx). Per `app.UseRequestLocalization()` sorgen wir dafür, dass Informationen aus dem HTTP-Request herangezogen werden, um die zu unterstützende Culture zu ermitteln. In unseren Razor Pages und MVC-Views sorgen `IStringLocalizer` und `IHtmlLocalizer` dafür, dass an den entscheidenden Stellen Platzhalter mit den Übersetzungen aus den Ressourcen gefüllt werden. Wir haben also das „Hallo Welt“ geschafft – so, wie die Überschrift dieses Artikels in der Universalsprache Esperanto heißt.

Welche Sprache soll es sein?

Bei der Sprachauswahl haben wir bisher den Standardmechanismus von ASP.NET Core eingesetzt. Wie in [1] beschrieben, schaut die Lokalisierungs-Middleware auf den HTTP-Header `Accept-Language` und nimmt sich dort die erste Sprache heraus, die unterstützt wird. Die unterstützten Sprachen haben wir im Aufruf von `UseRequestLocalization()` angegeben, inklusive einer Standard-Culture, die dann zum Zuge kommt, wenn das Framework im `Accept-Language`-Header nichts Passendes findet.

Dieses Vorgehen passt in den meisten Fällen, doch es gibt noch weitere Möglichkeiten. Standardmäßig hat ASP.NET Core mehrere sogenannte Request-Culture-Provider im Angebot, die in der folgenden Prioritätenreihenfolge zum Einsatz kommen:

- `QueryStringRequestCultureProvider`: Sucht im Querystring nach einer Culture (Parameter `culture`) oder UI-Culture (Parameter `ui-culture`).
- `CookieRequestCultureProvider`: Sucht in der HTTP-Anfrage nach einem Cookie für die Culture (Bezeichner im Cookie: `c`) oder UI-Culture (Bezeichner im Cookie: `uic`).



Mehrsprachig: Culture und UI-Culture per Querystring setzen (Bild 1)

- `AcceptLanguageHeaderRequestCultureProvider`: Nutzt den `Accept-Language`-Header als Quelle für die gewünschte Culture und UI-Culture.
- `DefaultRequestCulture`: Wenn keiner der anderen Provider fündig wird, verwendet ASP.NET Core die Standard-Culture und -UI-Culture.

Beginnen wir mit der Culture im Querystring. Die MVC-Beispielanwendung etwa verwendet den URL `https://localhost:7027/Localization`. Der folgende URL setzt die UI-Culture auf Deutsch und die Culture auf Englisch:

```
https://localhost:7027/Localization?ui-culture=de&culture=en
```

Bild 1 dokumentiert das Ergebnis. Die Texte sind auf Deutsch, aber Datums- und Zahlenformat die US-amerikanischen.

Wichtig fürs Verständnis: Diese Einstellung ist nicht dauerhaft. Der `QueryStringRequestCultureProvider` liest den aktuellen URL aus, aber persistiert diese Informationen nicht. Ist Persistenz gewünscht, gilt es den `CookieRequestCultureProvider` zu nutzen. Dieser prüft, ob im HTTP-Request ein Cookie mit dem (standardmäßigen) Namen `.AspNetCore.Culture` vorhanden ist. Hier ein typischer Wert für dieses Cookie:

```
c%3Dde%7Cuic%3Dde
```

Das sieht auf den ersten Blick nach Kauderwelsch aus, aber es ist URL-Codierung im Spiel. Decodiert würden wir folgenden Wert erhalten:

```
c=de&uic=de
```

Es werden also die Culture (`c`) und die UI-Culture (`uic`) auf jeweils „de“ gesetzt.

Sprachwahl durch Benutzer

Um diesen Provider einmal im konkreten Beispiel zu zeigen, setzen wir ein etwas umfangreicheres Feature um, bei dem wir quasi nebenbei noch einige andere Lokalisierungsfeatures von ASP.NET Core im Einsatz sehen. Wir möchten in der Applikation eine Sprachauswahl implementieren. Er erscheint also eine Liste aller unterstützten Sprachen, nach Auswahl wird diese Option persistiert und beim nächsten Aufruf automatisch angewandt.

Um das darstellen zu können, benötigen wir zwei Informationen:

- Die aktuell gewählte Sprache.
- Eine Liste aller unterstützten Sprachen.

Informationen über die aktuelle Sprache – nach Auswahl durch den ersten Request-Culture-Provider, der fündig wird – kommen aus dem *HttpContext* (zum Zugriff darauf ist mit *builder.Services.AddHttpContextAccessor()* eine Middleware notwendig):

```
var requestCultureInfo = HttpContext.Features.Get<
    IRequestCultureFeature>();
```

Jetzt liefert *requestCultureInfo?.RequestCulture.Culture* die verwendete Culture.

Die Ermittlung aller unterstützten Sprachen ist etwas komplizierter. Wir könnten hier billig abkürzen, indem wir die Liste der Sprachen, die wir in der *Program.cs* setzen, einfach duplizieren oder gar in einer Konfigurationsdatei hinterlegen würden. Allerdings ist es natürlich ebenfalls programmatisch möglich, an diese Werte heranzukommen, wenn auch mit einer kleinen Hürde. Zunächst zur eigentlichen Implementierung. Wir injizieren in unsere View oder Razor Page *RequestLocalizationOptions* nach folgendem Muster:

```
@inject IOptions<RequestLocalizationOptions>
    LocalizationOptions
```

Der Ausdruck *LocalizationOptions.Value.SupportedCultures* liefert jetzt die gewünschte Liste. Es gibt dabei jedoch einen Haken: Diese enthält genau ein Element, nämlich ausschließlich die aktuelle Culture. Das liegt an der Art und Weise, wie wir die unterstützten Kulturen angegeben haben – im Aufruf *app.UseRequestLocalization()*. Dagegen ist prinzipiell nichts einzuwenden, aber um wie gerade gezeigt dynamisch auf die Liste der Sprachen zugreifen zu können, müssen wir es anders machen:

```
var cultures = new[] { "de", "en" };
builder.Services.Configure<RequestLocalizationOptions>(
    _ =>
    {
        _.AddSupportedCultures(cultures);
        _.AddSupportedUICultures(cultures);
        _.SetDefaultCulture(cultures[0]);
    });
```

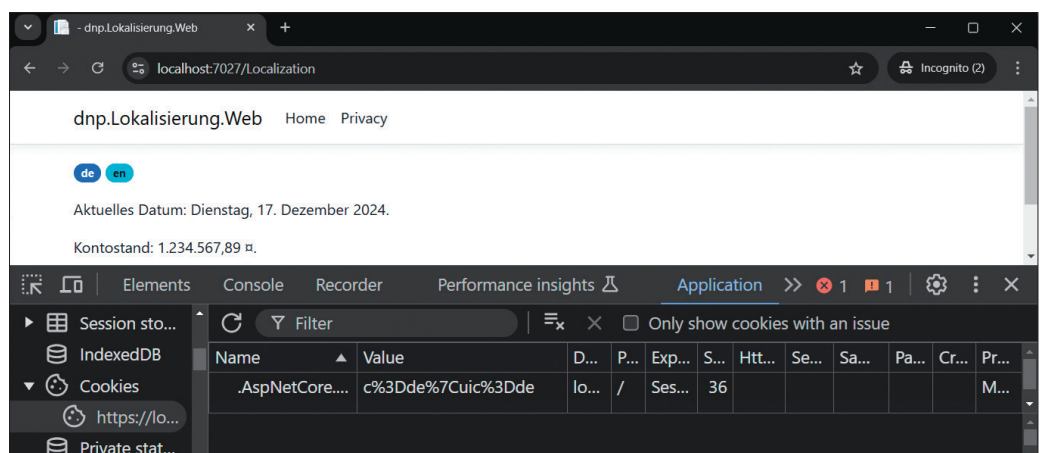
Damit können wir arbeiten. Der folgende Code erzeugt für jede Sprache einen Button; die Schaltfläche für die aktuelle Sprache wird farblich anders markiert, Bootstrap sei Dank:

```
<div>
    @foreach (var culture in LocalizationOptions.Value
        .SupportedCultures)
    {
        <a asp-action="SetCulture" asp-route-cultureName=
            "@culture" class="text-decoration-none">
            <span class="badge rounded-pill
                @((culture.ToString() == requestCultureInfo?
                    .RequestCulture.Culture.ToString()) ? "bg-primary"
                    : "bg-info text-dark")">@culture</span>
        </a>
    }
</div>
```

Per *asp-action* sorgen wir dafür, dass bei Betätigung des Links die Action-Methode *SetCulture()* aufgerufen wird; die gewählte Kultur wird im Parameter *cultureName* übergeben. Gerendert sieht das Markup für eine Culture ungefähr so aus:

```
<a class="text-decoration-none"
    href="/Localization/SetCulture?cultureName=de">
    <span class="badge rounded-pill bg-info text-dark">
        de</span>
</a>
```

Das letzte Puzzlestück ist eben dieser Endpunkt. Hier setzen wir das Cookie *.AspNetCore.Culture*. ASP.NET Core bietet ►



Fernbedienung: Sprachwahl und Persistierung in einem Cookie (Bild 2)

hierfür ein entsprechendes API im *CookieRequestCultureProvider*: die Methode *MakeCookieValue()*, die den URL-codierten Wert von oben erzeugen kann. Damit ergibt sich folgende Implementierung für den Endpunkt:

```
public IActionResult SetCulture(string cultureName)
{
    var culture = new RequestCulture(cultureName);
    Response.Cookies.Append(
        CookieRequestCultureProvider.DefaultCookieName,
        CookieRequestCultureProvider.MakeCookieValue(
            culture));

    return RedirectToAction("Index");
}
```

In **Bild 2** sehen Sie das Ergebnis: Die Inhalte werden auf Deutsch angezeigt, weil es so im Cookie steht.

Eigene Culture-Provider sind übrigens ebenfalls möglich. Hier ein kleines Beispiel: Angenommen, die *dotnetpro* hätte drei landesspezifische Domains, *dotnetpro.de*, *dotnetpro.es* und *dotnetpro.fr*. Der Code für alle Sites ist derselbe, es gibt nämlich nur einen Webserver. Abhängig von der Domain im Request wird die Culture gesetzt.

Basis für die Implementierung ist eine Instanz von *CustomRequestCultureProvider*. Diese erhält per Lambda-Ausdruck die Ermittlung der Culture, die dann als *ProviderCultureResult* zurückgegeben wird. Hier eine mögliche Umsetzung:

```
builder.Services.Configure<RequestLocalizationOptions>(
    _ =>
    {
        // ...
        _.AddInitialRequestCultureProvider(
            new CustomRequestCultureProvider(context =>
            {
                var culture = "de";
                if (context.Request.Host.ToString()
                    .EndsWith(".fr"))
                {
                    culture = "fr";
                }
                else if (context.Request.Host.ToString()
                    .EndsWith(".es"))
                {
                    culture = "es";
                }
            })
        );
    }
);
```

```
return Task.FromResult<ProviderCultureResult?>(
    new ProviderCultureResult(culture));
});
});
```

Validierung lokalisieren

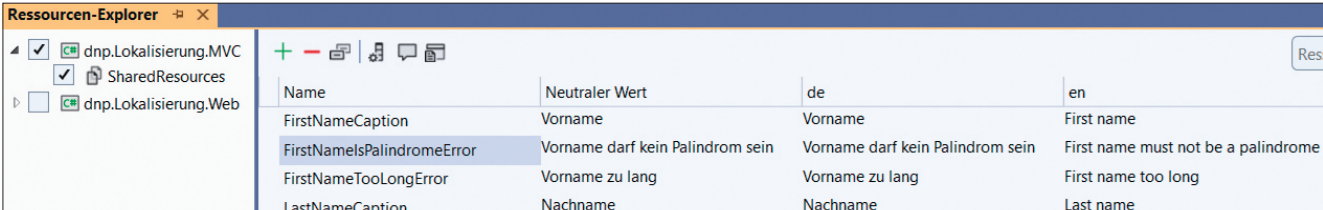
Auch in die Validierung von Data Annotations kann per Code eingegriffen werden. Was das mit Lokalisierung zu tun hat? Dass eingebaute Data Annotations mit Übersetzungen arbeiten können, haben wir in [2] bereits gesehen. Bei selbst implementierten Attributen ist leider etwas Zusatzcode notwendig. Der ist nicht aufwendig, muss aber gemacht werden.

Beginnen wir mit dem Attribut selbst. Als einfaches Beispiel implementieren wir *[NoPalindrome]*, einen Validator, der überprüft, ob ein String kein Palindrom ist (also ob er von vorne und von hinten gelesen nicht die gleiche Schreibung aufweist, wobei wir Groß- und Kleinschreibung ignorieren). Hier eine mögliche Umsetzung:

```
public class NoPalindromeAttribute : ValidationAttribute
{
    protected override ValidationResult? IsValid(
        object? value, ValidationContext validationContext)
    {
        if (value is string &&
            !String.Equals(new string(
                value.ToString().Reverse().ToArray()),
                value.ToString(),
                StringComparison.OrdinalIgnoreCase))
        {
            return ValidationResult.Success;
        }
        else
        {
            return new ValidationResult(
                FormatErrorMessage(ErrorMessageString));
        }
    }
}
```

Im Erfolgsfall läuft die Validierung ohne Beanstandung durch; andernfalls wird eine Fehlermeldung zurückgegeben. Diese Fehlermeldung geben wir wie gehabt bei der Verwendung des Attributs an.

In der MVC-Lösung aus dem vorangegangenen Artikel [2] gibt es die Modellklasse *Person*. Die dortige Eigenschaft *FirstName* darf nun dank Attribut kein Palindrom mehr sein:



Name	Neutraler Wert	de	en
FirstNameCaption	Vorname	Vorname	First name
FirstNamesPalindromeError	Vorname darf kein Palindrom sein	Vorname darf kein Palindrom sein	First name must not be a palindrome
FirstNameTooLongError	Vorname zu lang	Vorname zu lang	First name too long
LastNameCaption	Nachname	Nachname	Last name

Wörterbuch: Die Ressourcendateien enthalten jetzt auch Übersetzungen für den Palindrom-Validator (**Bild 3**)

```
public class PersonViewModel
{
    [Required]
    [DisplayName("FirstNameCaption")]
    [MaxLength(20, ErrorMessage =
        "FirstNameTooLongError")]
    [NoPalindrome(ErrorMessage =
        "FirstNameIsPalindromeError")]
    public string FirstName { get; set; } = string.Empty;
    [DisplayName("LastNameCaption")]
    public string LastName { get; set; } = string.Empty;
}
```

Die Fehlermeldung *FirstNameIsPalindromeError* soll natürlich nicht in dieser Form im Frontend erscheinen. Hier kommt Lokalisierung ins Spiel; wir benötigen unter diesem Schlüssel die zugehörigen übersetzten Texte (siehe **Bild 3**). Doch leider scheint das nicht zu funktionieren: Beim Testen erscheinen nicht die sprachspezifischen Texte, sondern eben *FirstNameIsPalindromeError*.

Es fehlt der bereits angesprochene Zusatzcode. Wie benötigen zwei Bestandteile. Zunächst einmal einen Adapter auf Basis von *AttributeAdapterBase*, der insbesondere *GetErrorMessage()* implementiert, also die Methode, über die die Fehlermeldung bezogen wird. Dabei wird auch ein *IStringLocalizer* per Konstruktor-Injection bereitgestellt:

```
public class NoPalindromeAttributeAdapter :
    AttributeAdapterBase<NoPalindromeAttribute>
{
    public NoPalindromeAttributeAdapter(
        NoPalindromeAttribute attribute, IStringLocalizer
        localizer) : base(attribute, localizer) { }

    public override void AddValidation(
        ClientModelValidationContext context)
    {
    }

    public override string GetErrorMessage(
        ModelValidationContextBase context)
    {
        return GetErrorMessage(context.ModelMetadata,
            context.ModelMetadata.GetDisplayName());
    }
}
```

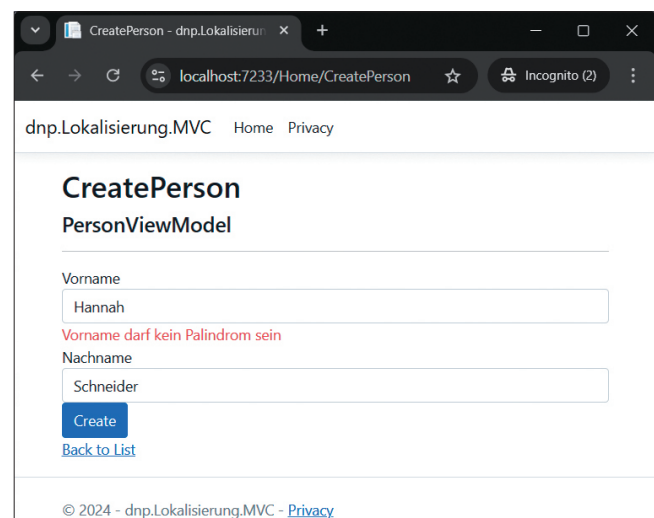
Im zweiten Schritt fügen wir einen Adapter-Provider hinzu, der für alle eigenen Attribute zum Einsatz kommt und im Fall von *[NoPalindrome]* den gerade erstellten Adapter verwendet. Da dieser Provider erweiterbar ist, beispielsweise wenn es zusätzliche Attribute gibt, erhält dieser einen generischen Namen:

```
public class CustomValidationAttributeAdapterProvider :
    IValidationAttributeAdapterProvider
{
}
```

```
private readonly IValidationAttributeAdapterProvider
    _provider = new ValidationAttributeAdapterProvider();

public IAttributeAdapter GetAttributeAdapter(
    ValidationAttribute attribute, IStringLocalizer?
    localizer)
{
    if (attribute is NoPalindromeAttribute)
        return new NoPalindromeAttributeAdapter((
            NoPalindromeAttribute)attribute, localizer!);
    else
        return _provider.GetAttributeAdapter(
            attribute, localizer!);
}
```

Letzter Schritt: In der Datei *Program.cs* müssen wir den Provider noch registrieren. Das sieht dann so aus:



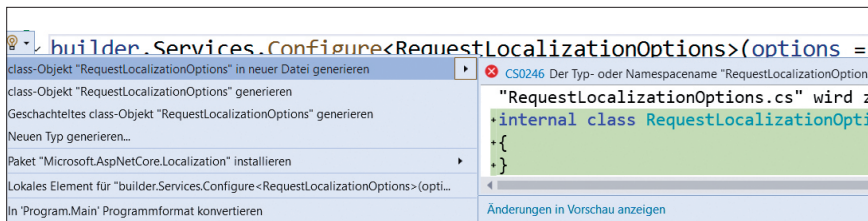
Regelwerk: Dieser Name ist ungültig, so will es das Attribut (**Bild 4**)

```
builder.Services.AddSingleton<
    IValidationAttributeAdapterProvider,
    CustomValidationAttributeAdapterProvider>();
```

Jetzt funktioniert alles (siehe auch **Bild 4**): Bei Namen, die ein Palindrom sind, erscheint eine – lokalisierte – Fehlermeldung.

Blazor lokalisieren

Zum Abschluss werfen wir noch einen kurzen Blick auf Blazor. Microsofts Single-Page-Application-Technologie ist Bestandteil von ASP.NET Core und hat somit Zugriff auf dieselbe Infrastruktur. Allerdings gibt es keine vollständige Feature-Parität zwischen den Frameworks. Lediglich *IStringLocalizer* wird unterstützt – *IHtmlLocalizer* nicht (und ebenfalls nicht alles aus dem MVC-Dunstkreis: lokalisierte Data Annotations und *IViewLocalizer*). Die verwendete Culture wird automatisch auf Basis des *Accept-Language*-HTTP-Headers ermittelt. Achtung: Im Optimalfall wird ein solcher Header ►



Vorsicht Falle: Das vorgeschlagene NuGet-Paket gibt nur in einer veralteten Version für Blazor (Bild 5)

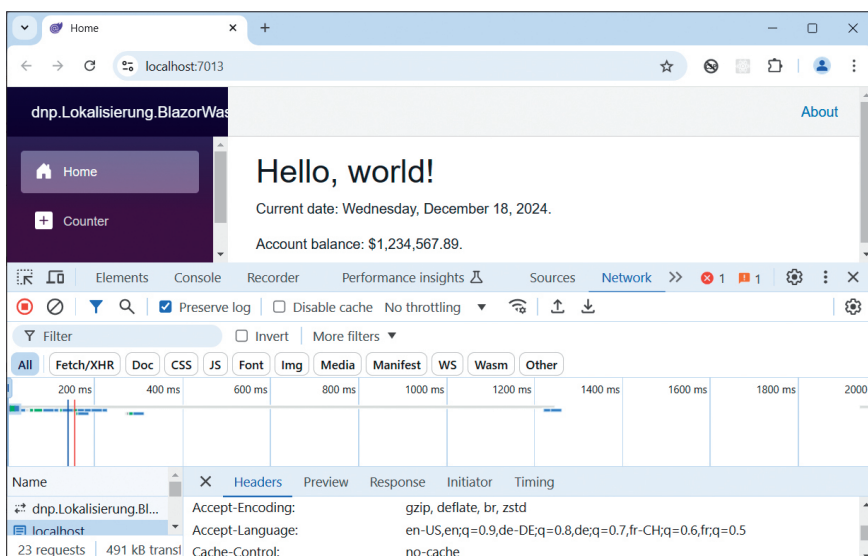
exakt einmal zur Anwendung geschickt, das liegt in der Natur einer SPA – spätere HTTP-Requests gehen gegebenenfalls zu statischen Ressourcen oder APIs. Allerdings kann die Culture auch im Code zur Laufzeit gesetzt werden. Das wie-

In der Datei *Program.cs* wird wie gehabt der Lokalisierungs-Support aktiviert. Für Blazor WebAssembly reicht diese Anwendung:

```
builder.Services.AddLocalization(
    options => options.ResourcesPath =
        "Resources");
```

Lokalisierte Blazor-Komponenten holen sich per *@inject* den zugehörigen *StringLocalizer* und setzen diesen dann wie auch bei „herkömmlichen“ ASP.NET-Core-Anwendungen ein:

```
@inject IStringLocalizer<SharedResources> SharedLocalizer
```



Delivered as ordered: Der HTTP-Header (unten) sorgt für übersetzte Texte (oben) (Bild 6)

derum führt beim Einsatz von Blazor WebAssembly zu Herausforderungen. Bei diesem Render-Modus ist bekanntlich gar kein (dynamischer) Webserver vonnöten, die Anwendung kann komplett aus statischen Assets bestehen. In diesem Fall müssen wir den Compiler explizit anweisen, alle zugehörigen Kulturen und Ressourcen entsprechend bereitzustellen und nicht nur die Standard-Culture. Visual Studio macht das unter Umständen automatisch, wenn eine Ressource angelegt wird, aber Kontrolle ist besser. Die *.csproj*-Datei muss folgendes Element enthalten:

```
<PropertyGroup>
  <BlazorWebAssemblyLoadAllGlobalizationData>true
  </BlazorWebAssemblyLoadAllGlobalizationData>
</PropertyGroup>
```

Alles Weitere erledigt das NuGet-Paket *Microsoft.Extensions.Localization*. Achtung: Wichtig ist, nicht *Microsoft.AspNetCore.Localization* einzusetzen, auch wenn Visual Studio das vorschlagen sollte (Bild 5), denn dieses Paket ist veraltet.

```
<p>
  @SharedLocalizer[
    "CurrentDateCaption"]:
    @DateTime.Now.ToLongDateString().
</p>
<p>
  @SharedLocalizer[
    "AccountBalanceCaption"]:
    @(1_234_567.89.ToString("C")).
</p>
```

Bild 6 zeigt das Ergebnis: englische Texte bei *en-us* im *Accept-Language*-Header.

Bei Einsatz von Blazor Server bedarf es entsprechend eines Aufrufs von *app.UseRequestLocalization()* und der Angabe der unterstützten Cultures und UI-Cultures.

Damit sind wir am Ende unseres Rundumschlags bezüglich der Lokalisierung von ASP.NET-Core-Anwendungen angekommen – seien es Razor Pages, MVC-Sites oder Blazor-Apps. Gis la venontan fojon! [3]

[1] Christian Wenz, *Salamu, Dunia!*, *dotnetpro* 1/2025, Seite 6 ff., www.dotnetpro.de/A2501WebApps

[2] Christian Wenz, *qo' vlvani!*, *dotnetpro* 2/2025, Seite 60 ff., www.dotnetpro.de/A2502WebApps

[3] *Esperanto auf Deutsch übersetzen*, www.dotnetpro.de/SL2503WebApps1



Christian Wenz

ist Spezialist für die Architektur von Webanwendungen und für Web Application 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

dnpCode

A2503WebApps



HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN



Hands-on Workshops Webentwicklung

Webgrundlagen: Programmierung
mit HTML, CSS und JavaScript (2 Tage)

Next Level TypeScript (2 Tage)

Clean-Code-Entwicklung für JavaScript-
und TypeScript-Entwickler (3 Tage)

Einstieg in Angular (3 Tage)

Angular für .NET-Entwickler (3 Tage)

Angular im Enterprise-Projekt meistern:
Redux & NGRX (2 Tage)

Moderne React Patterns (3 Tage)

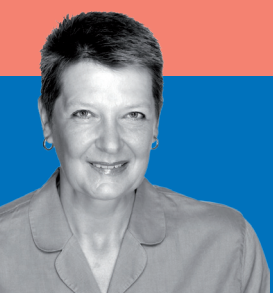
Einführung in Vue.js (2 Tage)

Vue.js 3 - What's new? (2 Tage)

Progressive Web App Bootcamp (3 Tage)

Einführung in GraphQL (2 Tage)

REMOTE
ODER
INHOUSE



Weitere Informationen auf developer-academy.de
Preise auf Anfrage!

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

ZEICHENKETTEN IN .NET 9

Der geteilte String

Zwei Neuerungen von .NET 9, die Entwicklern dabei helfen, Arbeitsspeicher zu sparen.



Es ist wieder einmal an der Zeit, dass diese Kolumne sich dem Thema String widmet. Eigentlich können .NET-Entwickler ja froh sein, dass die Programmiersprache C# für Textrepräsentationen lediglich den Datentyp *String* bereitstellt. Wer mit C++ entwickelt, für den ist es deutlich unübersichtlicher: Er muss sich mit *WCHAR** sowie *CHAR** und auch noch all den anderen Varianten diverser Frameworks herumschlagen.

```
string text = """
    Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
    sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
    sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
    """;
string[] words = text.Split(' ');
Console.WriteLine(words.Length);
```

Das Startbeispiel: Wörter zählen mit C# (Bild 1)

Dennoch ist der scheinbar simple *String* in .NET [1], zumindest wenn man ihn etwas genauer erforscht, zugleich ein besonders komplexer Datentyp. Schon seit den Anfängen der Programmierung gilt *String* als einer der Datentypen, die am meisten Speicher erfordern. Dennoch kann man mit dem Datentyp *String* – sofern korrekt verwendet – viel Performance gewinnen, man denke nur an die Nutzung der Klasse *System.Text.StringBuilder* zum Erzeugen langer Zeichenketten. Jetzt

```
string text = """
    Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
    sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
    sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
    """;

ReadOnlySpan<char> span = text.AsSpan();
int wordCount = 0;

for (int i = 0; i < span.Length; i++)
{
    if (char.IsWhiteSpace(span[i]) || i == span.Length - 1)
    {
        if (i > 0 && !char.IsWhiteSpace(span[i - 1]))
        {
            wordCount++;
        }
    }
}

Console.WriteLine($"Word Count: {wordCount}");
```

Wörter zählen mit *Span<T>*: Effizient, aber unübersichtlich (Bild 2)

kommt mit .NET 9 eine Variante hinzu, die dabei hilft, den Umgang mit Zeichenketten schneller und effektiver zu machen – insbesondere dann, wenn ein String aufgeteilt werden soll.

Das Grundproblem

Das Grundproblem beim Zerlegen einer Zeichenkette in ihre Einzelteile ist recht schnell erklärt, vor allem mit dem Beispiel aus Bild 1: Gegeben ist dort ein etwas längerer Text (*Lorem ipsum...*), und dieser soll vom Programm in einzelne Wörter aufgeteilt werden. Zum Erkennen von Wortanfang und -ende dient das Leerzeichen. Das klappt schon seit der ersten Version von .NET mit der String-Funktion *Split*:

```
string[] words = text.Split(' ')
```

Ergebnis ist ein Zeichenkettenfeld (String Array), in dem jeder Eintrag eines der Wörter enthält. Und genau hier liegt das Problem, das .NET 9 nun löst. Der Mechanismus der beliebten, weil so einfach zu nutzenden *Split*-Funktion erzeugt nämlich im Arbeitsspeicher eine Vielzahl von Strings. Jedes Wort wird dabei in einen eigenen, neuen Speicherbereich kopiert und dabei der eigentlich gleichbleibende Inhalt vervielfältigt. Diese Speicherbelastung ist in der Regel völlig überflüssig, insbesondere wenn es nur darum geht, ein konkretes Wort zu suchen.

Praktisches Problem

In der Praxis kommt das Problem häufig vor, etwa wenn man einen CSV-Parser schreiben will, eine XML-Verarbeitung oder die JSON-Parsing-Funktionalität nachbauen will. Diese Lösungen sollten in der Regel möglichst performant arbeiten. Sogar Microsoft selbst hat dieses Problem recht häufig, zum Beispiel wenn es in ASP.NET Core um das HTTP-Parsing geht, etwa wenn man die Header-Werte auftrennen möchte.

Bisherige Lösung

Um diese Aufgabe effizient zu lösen, musste man bislang auf die .NET-Funktion *Span<T>* [2] zurückgreifen, die in Bild 2 dargestellt ist. Im Vergleich mit der

wirklich einfach zu nutzenden Split-Funktion ist dieser Ansatz deutlich komplizierter. Wie im Bild gut zu erkennen ist, wird mit *AsSpan* auf der String-Instanz ein „Fenster“ geöffnet, für das man einen Datentyp namens *ReadOnlySpan<char>* erhält, mit dem man per *For*-Schleife auf die einzelnen Zeichen des Strings zuzugreifen kann. Dieses Vorgehen belastet den Arbeitsspeicher nicht, weil keine Daten kopiert werden, sondern das Fenster einfach Zeichen für Zeichen im String weiterwandert und dabei die einzelnen Zeichen überprüft werden. Dies war übrigens schon in den Anfängen von .NET möglich, da der Datentyp *String* selbst ein *IEnumerable<char>* ist. Spätestens wenn man jedoch aus einem String beispielsweise einen Integer-Wert oder ein *DateTime*-Objekt herauslesen wollte, musste man den String kopieren. Dank *Span<T>* ist das schon länger nicht mehr erforderlich.

Neu in .NET 9: Split auf Span<T>

Der jüngsten .NET-Variante hat Microsoft einen neuen Ansatz spendiert: Die denkbar einfach zu nutzende Methode *Split* kann jetzt direkt mit *Span<T>* kombiniert werden [3]. Bild 3 zeigt ein Beispiel dafür, was man damit erreichen kann.

Die neue Methode gibt einen sogenannten *SpanSplitEnumerator* zurück, den man einfach mit einer Schleife durchlaufen und dabei die einzelnen Wörter prüfen kann. Dafür liefert der Enumerator eine *System.Range*-Struktur zurück, mit der sich wesentlich komfortabler arbeiten lässt.

Im vorliegenden Fall interessiert nur die Anzahl der Wörter. Deshalb genügt eine einfache *foreach*-Schleife, um die Wörter zu zählen. Da der *SpanSplitEnumerator* nun aber nur eine *Range* zurückgibt, reden wir hier immer von sehr einfachen *structs* und nicht von den Arbeitsspeicher belastenden Heap-Konstrukten. Auch dem Garbage Collector machen die *Range*-Strukturen keine Arbeit.

Ein Blick in den Maschinenraum offenbart zudem, dass selbst der *SpanSplitEnumerator* lediglich eine *ref struct* ist

```
string text = """
    Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
    sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
    sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
    """;

ReadOnlySpan<char> span = text.AsSpan();
MemoryExtensions.SpanSplitEnumerator<char> words = span.Split(separator: ' ');
int wordCount = 0;
foreach (var word :Range in words)
{
    wordCount++;
}

Console.WriteLine(wordCount);
```

.NET 9: Wörter zählen mit *span.Split()* und *Range* (Bild 3)

Enables enumerating each split within a *ReadOnlySpan<T>* that has been divided using one or more separators.

Type params: *T* — The type of items in the *SpanSplitEnumerator<T>*.

[CompilerFeatureRequired("RefStructs")]

IL code

```
public ref struct SpanSplitEnumerator<T> where T : IEquatable<T>
{
    #nullable disable
    IL code
    private readonly ReadOnlySpan<T> _span;
    IL code
    private readonly T _separator;
    IL code
    private readonly ReadOnlySpan<T> _separatorBuffer;
    IL code
    private readonly SearchValues<T> _searchValues;
    IL code
    private MemoryExtensions.SpanSplitEnumeratorMode _splitMode;
```

Der *SpanSplitEnumerator* ist in .NET 9 als *ref struct* angelegt (Bild 4)

und somit im Bezug auf den Arbeitsspeicher besonders effizient arbeitet (Bild 4).

Die optimale Weiterverarbeitung

Soll nicht nur die Anzahl der per *span.Split* erkannten Wörter ermittelt, sondern auch deren Inhalt weiterverarbeitet werden, so könnte man das Beispiel so erweitern, wie es Bild 5 zeigt. In diesem Fall enthält die Zeichenkette namens *text* eine Reihe von Zahlen, die per Strichpunkt voneinander getrennt sind. Mit Hilfe von *Span* und der neuen *Split*-Variante können Sie unter .NET 9 mit einer Schleife über die einzelnen Teile iterieren und die Funktion *int.Parse* dazu nutzen, um die Zeichen in einen Integer-Wert zu konvertieren.

Nachdem *int.Parse* bereits vor einiger Zeit optimiert wurde und mit *Span<T>* zusammenarbeitet, erfordert auch diese Aufgabe keinerlei String-Allokationen auf dem Heap. Und ►

```
string text = "1;2;3;4";

ReadOnlySpan<char> span = text.AsSpan();
MemoryExtensions.SpanSplitEnumerator<char> words = span.Split(separator: ';');
int sum = 0;
foreach (var word :Range in words)
{
    sum += int.Parse(span[word]);
}

Console.WriteLine(sum);
```

Span.Split: Den Inhalt der *Range*-Segmente verarbeiten (Bild 5)

SplitAny: Mehrere
Trennzeichen sind
erlaubt (Bild 6)

```
ReadOnlySpan<char> span = text.AsSpan();
MemoryExtensions.SpanSplitEnumerator<char> words = span.SplitAny( separators: [',', ' ']);
int sum = 0;
```

schwupp – damit haben wir die effizienteste Variante in wenigen Zeilen Code vor uns und können mit den Ergebnissen tun, was auch immer uns vorschwebt.

SplitAny

Mit *SplitAny* [4] ist in .NET 9 noch eine weitere String-Methode hinzugekommen. Der Unterschied zu *Split* liegt darin, dass *SplitAny* nicht nur eines, sondern mehrere Trennzeichen gleichzeitig entgegennimmt. Bei jedem Vorkommen eines der angegebenen Trennzeichen wird eine neue Range zurückgegeben. Damit kann man verschiedene Trennzeichen nutzen, um einen Text zu verarbeiten. Das Beispiel in Bild 6 erzeugt neue Range-Abschnitte nicht nur nach Leerzeichen im Text, sondern auch nach jedem Komma.

Fazit

Mit .NET 9 kommt wieder einmal eine verbesserte Methode für die Arbeit mit Zeichenketten, mit denen man insbesondere den allzu verschwenderischen Umgang mit Arbeitsspeicher bei String-Operationen in den Griff bekommen kann. Für alle Entwickler, die häufig mit der Verarbeitung von Zeichenketten zu tun haben, lohnt es sich, die Neuerungen zu testen.

Schon allein dadurch, dass die Garbage Collection weniger zu tun bekommt, sollten Anwendungen zügiger arbeiten. ■

[1] *System.String* bei Microsoft Learn,
www.dotnetpro.de/SL2503NETirol1

[2] *Span<T>* bei Microsoft Learn,
www.dotnetpro.de/SL2503NETirol2

[3] Microsoft Learn, *span.Split*,
www.dotnetpro.de/SL2503NETirol3

[4] Microsoft Learn, *SplitAny*,
www.dotnetpro.de/SL2503NETirol4



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.

christian@software.tirol

dnpCode

A2503NETirol

DEV ACADEMY

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN

**3 TAGE
REMOTE/INHOUSE**

ASP.NET Blazor - SPA-Anwendungen mit C# und .NET

- Blazor-Grundlagen
- Routing und Binding mit Blazor
- Formularanbindung mit Blazor
- Validierung von Daten mit Blazor
- Datenbereitstellung mit REST und Integration in Blazor

TRAINING!



Christian Giesswein
Trainer

KONTROLLIERTE EXPERIMENTE FÜR DIE PROGRAMMIERUNG

Warum rücken wir Code ein?

Der Nutzen von Codeeintrückung scheint klar. Doch ein Blick in die Literatur offenbart ein diffuses Bild, das erst in jüngster Vergangenheit an Klarheit gewonnen hat.

Codeeintrückung spielt schon in der Programmierausbildung eine Rolle, und die Bedeutung korrekter Einrückung wird in der Industrie nicht mehr ernsthaft hinterfragt. Unkorrekt eingrückter Code dürfte (und sollte) bei jedem Code-Review zu einem Stirnrunzeln führen und den einen oder anderen kritischen Kommentar nach sich ziehen.

Und doch darf man sich fragen, warum wir annehmen, dass Codeeintrückung wichtig ist. Selbst wenn wir es für wichtig halten, sollte eine Folgeüberlegung lauten, wie groß der vermeintliche Effekt ist. Reduziert ein solcher nicht eingrückter Code die Lesbarkeit um fünf, zehn oder hundert Prozent? Oder ist der Effekt nicht der Rede wert? Denn wäre er sehr klein, könnte man sich alle Diskussionen darüber sparen.

Die Antwort lautet jedoch, dass wir es eigentlich nicht wissen. Und daher stellt sich die Frage, warum wir darauf beharren, dass Codeeintrückung wichtig ist, wenn wir doch wenig Handfestes vorzuweisen haben.

So viel vorweg: Der Effekt von Codeeintrückung existiert mit sehr großer Sicherheit, und der Effekt ist groß (Faktor 2 und mehr). Aber bevor wir dazu kommen, müssen wir uns mit der Rolle der Wissenschaftsliteratur in unserer Disziplin beschäftigen, denn es wäre anzunehmen, dass wir derartige Zahlen einfach der Literatur entnehmen können.

Experimente und der Umgang mit Skepsis

Oft wird darauf hingewiesen, dass in der Softwareentwicklung zu selten Aussagen und Techniken empirisch belegt wurden – zu Recht. Doch auch wenn Belege in Form von Experimentaldokumentationen vorliegen, ist die Wissenschaftssprache zu wenig eingängig, um ihre Ergebnisse der breiten Masse zur Verfügung zu stellen. Und selbst dann stellt sich für kritische Leser die Frage, wie glaubwürdig die Resultate sind. Denn wenn ein Experiment mit vielen Teilnehmern durchgeführt wurde, kann ein Leser dieses nicht kurzerhand wiederholen, um sich selbst von den Ergebnissen überzeugen zu lassen oder sich in seiner Skepsis bestätigt zu sehen.

Hinsichtlich der Codeeintrückung stellen sich genau diese Fragen zu einem erheblichen Teil. Denn es existiert eine Reihe von Experimenten, die den Effekt der Einrückung untersucht haben. Doch eine Vielzahl dieser Experimente besagt, dass Einrückung keinen Effekt hat.

Man darf (und sollte) diesen Experimenten durchaus mit Skepsis begegnen. „Das glaube ich nicht“ reicht aber als Einwand nicht. Wo liegt die Grenze zwischen seriösem Zweifel und reiner Ignoranz? Dabei sei hervorgehoben, dass der Zweifel an Forschungsergebnissen durchaus im Sinne der

Wissenschaft ist. Aber es stellt sich die Frage, von welcher grundsätzlichen Ausprägung die Verbindung eines Softwareentwicklers mit der wissenschaftlichen Literatur ist.

Eine schwierige Beziehung

Die Softwareentwicklung hat über die Jahre ein eher schwieriges Verhältnis zur Wissenschaft aufgebaut. Während es für eine Reihe von Berufen undenkbar ist, wesentliche wissenschaftliche Ergebnisse zu ignorieren, ist dies in der Softwareentwicklung gelebte Praxis: Einer Umfrage zufolge, an der mehr als 500 Microsoft-Entwickler teilgenommen haben, beziehen Entwickler ihr Wissen hauptsächlich aus persönlicher Erfahrung und nicht aus wissenschaftlicher Literatur [1].

Dies kann man als Kritik an Softwareentwicklern auffassen, die wissenschaftliche Erkenntnisse ignorieren. Man kann es aber auch als Kritik an der Wissenschaft verstehen, die Entwicklern wenig Handfestes liefert. Denn die Forschung in der Softwareentwicklung hat sich in der Vergangenheit nicht durch ein überbordendes Maß an Wissenschaftlichkeit ausgezeichnet. Zum Beispiel gab es einer Studie zufolge bis zum Jahr 2012 nur 22 sogenannte kontrollierte Experimente zu Programmiersprachenfeatures [2] – eine schockierend geringe Zahl, wenn man betrachtet, wie viele Programmiersprachen bis 2012 industriell eingesetzt wurden, und man sich gleichzeitig Zahlen aus Disziplinen wie zum Beispiel der Medizin ansieht, in der in diesem Zeitraum Hunderttausende von Experimenten durchgeführt wurden (siehe zum Beispiel [3]).

Leider ist unklar, welche Schlüsse aus der geringen Anzahl von Experimenten zu ziehen sind. Sollten wir die wenigen vorhandenen Experimente deswegen besonders wertschätzen? Oder ist die geringe Anzahl Ausdruck einer bisher kaum erwachsenen Disziplin, deren wenige Experimente wir besonders kritisch beäugen sollten?

Bevor wir einen Blick auf frühere Experimente zur Codeeintrückung werfen, verlieren wir zunächst ein paar Worte über empirische Studien im Allgemeinen, um zu verstehen, warum viele Experimente zur Codeeintrückung vermutlich keinen Effekt gezeigt haben [7].

Kurze Einführung in kontrollierte Experimente

Empirische Studien bezwecken, Aussagen durch Beobachtungen zu stützen oder zu widerlegen. Kontrollierte Experimente sind empirische Studien, bei denen Experimentierer sogenannte unabhängige Variablen verändern (wie zum Beispiel die Einrückung), um deren Effekt auf eine oder mehrere abhängige Variablen (zum Beispiel die Zeit zum Lösen ►

einer Aufgabe) zu beobachten. Beobachten bedeutet, dass etwas gemessen und mithilfe von (Inferenz-)Statistik geprüft wird. Dabei drückt der sogenannte p-Wert (wenn er kleiner als 0,05 ist) aus, dass die unabhängige Variable einen Effekt auf die abhängige Variable ausgeübt hat. Es gibt zusätzliche Kennzahlen bei Experimenten. Aber um es zu vereinfachen, reichen p-Werte und die Verhältnisse der Mittelwerte, um zu verstehen, was das Ergebnis eines kontrollierten Experiments ist: Der p-Wert ist ein Evidenzmaß (wie sicher sind wir, dass der Effekt existiert?), und das Verhältnis der Mittelwerte ist ein Maß für die Größe des Effekts.

Aufgabe von Experimentierern ist es, Störeffekte zu beseitigen, also Einflüsse auf die abhängige Variable, die nicht von der unabhängigen Variablen stammen. Für die Interpretation von Experimenten ist es wichtig, zwischen „es gibt keinen Unterschied“ und „das Experiment konnte keinen Unterschied feststellen“ zu unterscheiden. Denn für Experimentierer ist ein Experiment, das keinen Effekt feststellt, erst einmal nur ein Hinweis darauf, dass es vermutlich weitere Störvariablen gibt, die es zu eliminieren gilt: Anstatt zu propagieren, dass etwas keinen Effekt hat, werden Experimentierer weiter am Experiment arbeiten, um den möglichen Effekt sichtbar zu machen.

Reflexartige Reaktionen von Lesern auf kontrollierte Experimente sind oft Zweifel an der Anzahl an Experimententeilnehmern beziehungsweise Zweifel daran, ob die Experimententeilnehmer repräsentativ ausgewählt wurden. Beide Einwände sind, so plausibel sie auch scheinen mögen, wenig überzeugend. Man muss sich dazu nur die Frage stellen, was wohl eine ausreichend große Menge an Experimententeilnehmern ist respektive woran man die Repräsentativität von Experimententeilnehmern ausmacht.

Ein anderer häufig anzutreffender Zweifel ist, ob die Aufgabe, die Experimententeilnehmer lösen müssen, „hinreichend ähnlich zu einer Aufgabe ist, wie man sie auch in der Industrie antreffen würde“. Auch dieser Zweifel ist plausibel. Jedoch ist auch dieser Zweifel problematisch, denn man müsste dazu die Frage beantworten können, was „eine durchschnittliche Aufgabe in der Industrie“ ist.

Zweifel hat jedoch seine Rechtfertigung. Die Frage ist, ob es zu einem Experiment auch Folgeexperimente gibt, die einen festgestellten Effekt oder das Ausbleiben des Effekts bestätigen. Denn möglicherweise war ein singuläres Experimentergebnis lediglich die Folge von glücklichen (oder unglücklichen) situativen Umständen.

Werfen wir vor diesem Hintergrund einen Blick auf die Geschichte der Einrückung von Code aus experimenteller Sicht. Wir schauen uns also an, wann versucht wurde, den Effekt von Einrückung experimentell zu belegen. Dabei betrachten wir vorerst nur Experimente bis zum Jahr 2019; die Liste der Experimente kann unter anderem [4] entnommen werden.

Einrückungsexperimente bis 2019

Weissman hat 1974 eine Reihe von Experimenten durchgeführt. In drei von ihnen wurde an 16 bis 48 Teilnehmern unter anderem Einrückung kontrolliert studiert. Teilnehmer bekamen eingerückte und nicht eingerückte Programme von bekannten algorithmischen Problemen wie dem Damen-Problem in der Sprache PL/I zu lesen und mussten danach einschätzen, wie sich unterschiedliche Variationen im Programm, so etwa Einrückung, auf ihr Verständnis ausgewirkt haben. Weissman fasst zu jeder dieser Studien zusammen, dass kein Effekt von Einrückung nachgewiesen werden konnte.

Shneiderman und McKay führten 1976 eine Studie durch, in der 24 Teilnehmer Fehler in Pascal-Programmen wie der Binärsuche finden und beheben sollten. Auch hier wurde kein Effekt von Einrückung gefunden.

1977 legte Love vier eingerückte und nicht eingerückte Fortran-Programme aus Lehrbüchern 31 Teilnehmern vor und prüfte, inwieweit sie sich an die Programme erinnern konnten. Es wurde kein Effekt von Einrückung festgestellt.

1982 haben zwei Experimente von Norcio an 60 Teilnehmern einen positiven Effekt von Einrückung nachgewiesen. Dabei bekamen die Teilnehmer fünf kurze, nicht näher beschriebene

● Listing 1: Codebeispiele für das JSON-Experiment (stark verkürzt)

```
{
  "tree": "box",
  "house": {
    "chair": {
      "table": "value",
      "beard": "length"
    }
  }
}

{
  "tree": "box",
  "house": {
    "chair": {
      "table": "value", "beard":
        "length"
    }
  }
}
```

ne Fortran-Programme zu lesen und mussten anschließend einen Lückentext zu den Programmen beantworten, bei dem die Anzahl an richtigen Antworten gewertet wurde. Zwar war der Effekt von Einrückung mit ausreichender Evidenz ($p < 0,05$) vorhanden; jedoch sind weder Details zum Lückentext noch zur Effektstärke bekannt. Doch bevor man zu sehr über den möglichen Effekt nachdenkt: 1983 haben Norcio und Kerst ein ähnliches, vielleicht sogar identisches Experiment durchgeführt, ohne einen Effekt festzustellen.

Im Jahr 1983 haben Miara et al. 86 Teilnehmern Pascal-Programme aus Lehrbüchern zu lesen gegeben und anschließend 13 Fragen pro Programm gestellt, wobei die Programme mit unterschiedlicher Einrückungstiefe variiert wurden. Zwar wurde ein Einrückungseffekt festgestellt; dessen Interpretation ist jedoch unklar, auch weil die Effektstärken nicht berichtet wurden: Es scheint, als ob eine Einrückungstiefe von 2 einen positiven Effekt hatte, während Einrückungstiefen von 4 oder 6 einen eher negativen Effekt hatten. Ein

sehr ähnliches Experiment von Kessler et al. hat 1984 an 72 Teilnehmern jedoch keinen Effekt feststellen können. Auch ein Experiment von Bauer et al. aus dem Jahr 2019, das den Autoren zufolge eine Experimentwiederholung des Experiments von Miara et al. darstellte, zeigte keinen Effekt von Einrückung.

Erst im Jahr 2010 zeigt ein Experiment von Albayrak und Davenport einen Effekt von Einrückung, der nicht gleich in einem Folgeexperiment nicht mehr festgestellt wurde – leider aus dem Grund, dass keine ähnliche Experimentdurchführung aus der Literatur bekannt ist. Dort wurde 88 Erstsemesterstudenten ein circa 100 Codezeilen langes Java-Programm gezeigt, in dem die Teilnehmer Defekte finden mussten. Über das Programm und die Defekte ist wenig bekannt, aber die Autoren beschreiben, dass bei nicht eingerücktem Code 25 Prozent weniger Defekte gefunden wurden.

Kein Effekt von Einrückung, oder liegt es an etwas anderem?

Die Experimente bis 2019 liefern ein ernüchterndes Bild. Wohlwollend kann man sagen, dass vier von zwölf Experimenten (zweimal 1982, 1984, 2010) mit ausreichender Evidenz ($p <$

weis für Experimentierer, dass Störvariablen den Effekt verschleiern. Aber was könnten solche Störvariablen sein?

Mögliche Störvariablen in Einrückungs-experimenten

Alle Experimente zeigten Teilnehmern einen mehr oder weniger komplizierten Algorithmus, und es ist anzunehmen, dass dieser womöglich komplizierte Algorithmus eine dieser Störvariablen ist. Die Überlegung dazu ist einfach: Wenn ein Algorithmus ausreichend kompliziert ist, bedarf es einiger Zeit, den Algorithmus zu verstehen.

Nehmen wir als Beispiel das Damen-Problem, das in einem der ersten Experimente verwendet wurde. Es geht darum, auf einem Schachbrett acht Damen so zu positionieren, dass sie sich nicht gegenseitig schlagen können. Der Code dazu ist nicht trivial. Vielleicht macht das Fehlen von Einrückung den Code komplizierter, aber man darf annehmen, dass allein der Algorithmus Teilnehmern schon so viel Kopfzerbrechen bereitet, dass der Effekt von Einrückung demgegenüber nur gering ist. Bedeutet das, dass Einrückung einen geringen Effekt hat? Nein. Es bedeutet nur, dass die Kompliziertheit von Code wahrscheinlich einen größeren Effekt hat. Aus Sicht eines Ex-

perimentierers lautet die Schlussfolgerung, dass der verwendete Code hinreichend trivial sein muss, um keinen Störeffekt darzustellen.

Losgelöst vom Code gibt es mutmaßlich weitere Störeffekte. Dazu gehört auch die Aufgabe, die Teilnehmern in Kombination mit der verwendeten abhängigen Variablen gestellt wurde. Nehmen wir an, dass der Quelltext eine Reihe von Fehlern hat, wie zum Beispiel im Experiment von 1976. Ist die Anzahl an gefundenen Fehlern ein geeignetes Maß, um Unterschiede festzustellen? Dabei muss man sich fragen, ob alle Fehler im Code

– vom fehlenden Semikolon in Java-Anwendungen über falsch initialisierte Zählvariablen bis hin zu fehlerhaften Abbruchkriterien in rekursivem Code – die gleiche Wertigkeit haben. Die Antwort lautet: Nein. Dies wiederum bedeutet, dass die reine Anzahl von gefundenen Fehlern keinen verlässlichen Hinweis darauf liefert, wie gut oder schlecht eine Code-Repräsentation ist: Wenn jemand einen komplizierten Fehler findet, ist das wahrscheinlich besser, als wenn jemand zwei triviale Fehler findet. Und falls manche Versuchsteilnehmer sich primär komplizierten Fehlern widmen, während sich andere auf die trivialen Fehler konzentrieren, führt das zu nicht vergleichbaren Zahlen.

Letztlich lassen sich weitere mögliche Störvariablen in den genannten Experimenten finden, aber wir beenden die Diskussion hier und fassen zusammen, dass zwar bis 2019 keine Experimente den Nutzen von Einrückung belegen konnten, aber Störvariablen den möglichen Effekt der Einrückung vermutlich verschleiern haben. ►

● Listing 2: Codebeispiele für das if-Statement-Experiment (stark verkürzt)

```
if(false) {
    if(true) {
        return 5;
    } else {
        return 6;
    }
} else {
    return 3;
}
```

```
if(false) {
    if(true) {
        return 5;
    } else {
        return 6;
    }
} else {
    return 3;
}
```

0,05) einen Effekt gezeigt haben. Kritischer kann man aber auch sagen, dass wenn einmal ein Experiment einen Einrückungseffekt zeigte, ein Folgeexperiment diesen nicht mehr feststellen konnte. Die einzige Ausnahme davon bildet das Experiment aus dem Jahr 2010, das sowohl einen Effekt feststellte als auch die Stärke des Effekts beschrieb – ein Experiment, zu dem es kein Folgeexperiment gab. Leider sind so wenige Details von diesem Experiment bekannt, dass es nicht möglich ist, dieses nachzustellen. Ohne Folgeexperiment bleiben Zweifel, ob ein solches einen ähnlichen Effekt feststellen würde.

Wenn lediglich ein Experiment, über das wir im Grunde genommen wenig wissen, einen positiven Effekt von Einrückung gezeigt hat, stellt sich die Frage, ob es möglich ist, dass Einrückung keinen positiven Effekt auf die Lesbarkeit hat.

Skepsis scheint auch bei dieser These angebracht. Wir sollten dazu uns noch einmal in Erinnerung rufen, was es bedeutet, wenn ein Experiment keinen Effekt zeigt: Es ist ein Hin-

Geht das alles nicht viel einfacher?

Nun stellt sich die Frage, ob das alles nicht einfacher geht. Wenn wir etwas über den Effekt einer einfachen Technik wie Einrückung erfahren wollen, können wir dazu nicht auch einfache(re) Experimente hinbekommen? Und kann man Experimente nicht so entwerfen, dass sie leicht wiederholbar sind? Im einfachsten Fall sogar an einer Einzelperson? Die Antwort darauf ist: Ja.

Es gibt eine besondere Experimentform, die in vielen Disziplinen angewendet wird, auch wenn sie in der Softwareentwicklung eine untergeordnete Rolle spielt: die sogenannten N-of-1-Experimente. Es handelt sich um Versuche, die an einer Einzelperson alle Kombinationen eines Experiments durchführen. N-of-1-Experimente bringen dabei eine Reihe möglicher Störvariablen mit sich, wie etwa Lerneffekte. Um diese Störvariablen gering zu halten, werden am gleichen Teilnehmer viele Wiederholungen in zufälliger Reihenfolge durchgeführt. Im Idealfall wird in einem solchen Experiment sogar alles zufällig generiert, also nicht nur die Reihenfolge, sondern auch der Code selbst.

Wenn wir diese Überlegungen auf Einrückung anwenden, bedeutet das, dass hinreichend trivialer Code in einem zufälligen Verfahren in ausreichender Menge generiert werden muss. Dieser Code muss ausreichend viele Einrückungen haben, denn sonst wäre er nicht dazu geeignet, Einrückung zu studieren. Die Frage nach ausreichend trivialem Code ist nicht leicht zu beantworten, denn es gibt entgegen der gängigen Meinung wenig Anhaltspunkte, was trivialen von nichttrivialem Code unterscheidet. Aber schauen wir uns das aus zwei unterschiedlichen Perspektiven an.

Nehmen wir an, dass der Code kein funktionaler Code ist, sondern nur eine Datenstruktur, wie zum Beispiel ein JSON-Dokument. Zwar stellt sich die Frage, was ausreichend trivial ist, aber die sprachlichen Mittel von JSON-Dokumenten sind stark beschränkt. Vermutlich spielt die Länge der Dokumente eine Rolle. Wenn wir JSON-Dokumente in gleicher Länge generieren, scheinen sie bereits die Anforderung an trivialen Code zu erfüllen. Und um ausreichend viele Einrückungen zu erhalten, muss sichergestellt werden, dass ausreichend viele Verschachtelungen vorhanden sind.

Blicken wir als Nächstes auf Programmiersprachen, die eine größere Anzahl an sprachlichen Mitteln haben, also Variablen, Schleifen, Methoden, Lambdas und so weiter. Auch hier ist es möglich, trivialen Code zu erzeugen, indem wir nur noch bedingte Verzweigungen, also *if*-Statements, verwenden. Kompliziert wird ein *if*-Statement erst, wenn die boolesche Bedingung kompliziert wird. Aber auch das ist handhabbar, indem wir nur Literale (also *true* oder *false*) verwenden. Und damit irgendwo noch etwas anderes als ein *if*-Statement steht, generieren wir noch zufällige *return*-Statements in den *if*- oder *else*-Zweig, die eine Zahl zurückgeben.

Einrückung in N-of-1 Experimenten

Der Code ist also generierbar. Aber die Frage ist, welche Aufgabe Teilnehmern gestellt werden kann. Sehen wir uns dazu zwei Experimente an, die in jüngerer Vergangenheit durchgeführt wurden. Das erste bezieht sich auf JSON-Dokumen-

te, das andere auf *if*-Statements. Beide stammen aus dem Jahr 2024 [4][5].

Im Experiment zu JSON-Dokumenten wurde ein 27-zeiliger Code, bestehend aus geschachtelten JSON-Objekten, zufällig generiert. Probanden wurde die Frage gestellt, wie viele Kinder das äußere Objekt hat [4]. Im Experiment zu *if*-Statements wurde Probanden ebenfalls 27-zeiliger Code mit sieben geschachtelten *if*-Statements gezeigt und gefragt, welches *integer*-Literal dieser Code zurückgibt [5]. In beiden Experimenten drücken die Teilnehmer auf die Tasten 0 bis 9, um die Antwort zu geben. In beiden Experimenten wurde zusätzlich zur Feststellung, ob die Antwort korrekt war, die Antwortzeit gemessen.

Listing 1 und **Listing 2** zeigen exemplarisch stark verkürzt den Code, der in den beiden Experimenten verwendet wurde. Schon ein erster Eindruck zeigt, dass der nicht eingerückte Code vermutlich schwerer lesbar ist.

Das JSON-Experiment wurde an 40 Teilnehmern durchgeführt. Dabei stellte sich mit großer Evidenz ($p < 0,001$) heraus, dass der nicht eingerückte Code gut 380 Prozent der Lesezeit des eingerückten Codes benötigte; das Verhältnis der Mittelwerte von nicht eingerücktem zu eingerücktem Code betrug 3,823. Das *if*-Statement-Experiment wurde an 27 Teilnehmern durchgeführt. Wieder zeigte sich ein starker ($p < 0,001$) und großer Effekt von Einrückung: Das Verhältnis der Mittelwerte bei der Lesezeit von nicht eingerücktem Code zu eingerücktem Code betrug 2,31.

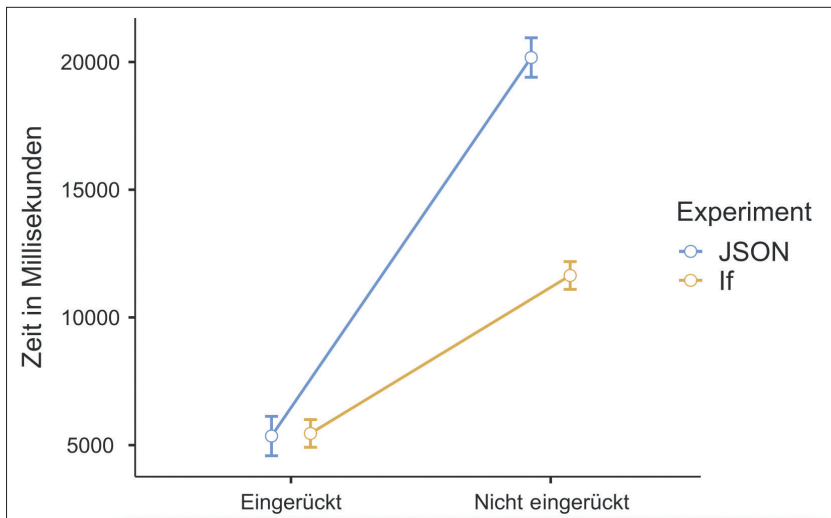
Bei letzterem Experiment lohnt sich ein genauerer Blick, denn das Experiment untersuchte nicht nur den Effekt der Einrückung, sondern auch, welcher weitere Faktor einen Einfluss auf die zeitliche Differenz hat. Es stellte sich heraus, dass das Verhältnis der Lesezeiten von nicht eingerücktem gegenüber eingerücktem Code umso größer war, je mehr Code dabei übersprungen werden kann (wenn also zum Beispiel in einen *else*-Zweig gesprungen werden musste).

Wenn wir beide Experimente zusammenfassen, zeigt sich in beiden Experimenten mit großer Evidenz (jeweils $p < 0,001$) ein großer Faktor als Unterschied zwischen nicht eingerücktem und eingerücktem Code (3,823 beziehungsweise 2,31).

Bild 1 zeigt die Ergebnisse der beiden Experimente in einer grafischen Aufbereitung. Selbst wenn man nicht mit statistischen Methoden vertraut ist, ist klar ersichtlich, dass in beiden Experimenten der Unterschied zwischen eingerücktem und nicht eingerücktem Code sehr groß ist. Ferner ist (auch ohne Statistik) ersichtlich, dass der Effekt in beiden Experimenten unterschiedlich groß ist: Der Effekt der Einrückung ist im JSON-Experiment deutlich größer als beim *if*-Statement-Experiment.

Was ist mit den Zweifeln?

Wie bereits gesagt: Zweifel sind im Sinne der Wissenschaft etwas Positives. Und es gibt Gründe, die Experimentergebnisse anzuzweifeln. Möglicherweise haben die Experimentierer nur ausgewählte Teilnehmer ausgesucht, oder vielleicht wurden missliebige Daten aussortiert. Vielleicht gibt es weitere Restzweifel an den Experimentergebnissen, denn möglicherweise haben die Teilnehmer keineswegs versucht,



Experimentergebnisse des JSON-Experiments und des If-Experiments (Bild 1)

die Aufgabe zu lösen, sondern haben nur einfach wahllos Tasten gedrückt, um das Experiment hinter sich zu bringen.

Hier zeigt sich ein Vorteil von N-of-1-Experimenten, denn sie können mit überschaubarem Aufwand von einer Einzelperson wiederholt werden; im Zweifelsfall besucht man selbst die Website mit den Experimenten und führt diese durch [6]. Beide Experimente liefern als Ergebnis eine CSV-Datei, anhand derer man die Verhältnisse von nicht eingerückten und eingerückten Code selbst berechnen kann. Für den Fall, dass man den p-Wert berechnen möchte, muss man wohl oder übel noch ein Statistik-Werkzeug in die Hand nehmen (R, SPSS, Jamovi oder Ähnliche).

Aber natürlich bleiben Restzweifel, ob sich derartig große Effekte auch in realem Code niederschlagen. Die Antwort darauf: Natürlich nicht. Beide Experimente haben bewusst realen Code außen vor gelassen. Realer Code besitzt eine zusätzliche Kompliziertheit, die den Effekt der Einrückung schmälert: Das *if*-Statement-Experiment hat gezeigt, dass der Effekt der Einrückung durch den Code gesteuert wird, der im eingerückten Fall einfach übersprungen wird. Mit anderen Worten: Der Effekt von Einrückung ist kein fester Faktor, der losgelöst vom konkreten Code auftritt, sondern er tritt zusätzlich zu konkretem Code auf. Sobald der Einrückungseffekt jedoch isoliert betrachtet wird, ist dieser groß.

Der große Kontext

Es stellt sich die Frage, warum noch im Jahr 2024 Effekte wie Einrückung studiert werden – Effekte, von denen angenommen werden darf, dass sie ohnehin seit Jahrzehnten bekannt sind. Die Antwort ist ernüchternd. Leider haben wir abseits unserer persönlichen Erfahrung wenig in der Hand, um jemanden davon zu überzeugen, dass eingerückter Code verständlicher ist. Dieses Problem tritt vor allem in der Lehre auf, wenn Neulingen erklärt wird, warum sie Code einrücken sollen.

N-of-1-Experimente stellen eine Möglichkeit dar, Lernende ohne viel Aufwand vom Einrückungseffekt zu überzeugen. Sie können ein Experiment durchführen und werden rasch feststellen, dass sie für den nicht eingerückten Code deutlich

länger benötigen. Für erfahrene Softwareentwickler sind diese Experimente insofern interessant, als sie empirische Nachweise für den Einrückungseffekt liefern. Und es stellt sich heraus, dass der Einrückungseffekt sehr stark (große Evidenz) und groß ist, wobei dies eher eine Frage eines Faktors als die weniger Prozentpunkte ist.

Es wäre zu wünschen, dass viele solcher N-of-1-Experimente zur Verfügung stünden, anhand derer man sich selbst von Effekten einzelner Techniken überzeugen könnte. Sie stellen die praktische Alternative zum Lesen wissenschaftlicher Abhandlungen dar. Aber man sollte, selbst wenn man nach dem Ausprobieren von der Größe eines Effekts überzeugt ist, dennoch die eine oder andere Minute in die wissenschaftliche Literatur investieren, um zu sehen,

wie groß dieser Effekt bei einer anderen Teilnehmergruppe war. Denn am Ende sollte die wissenschaftliche Literatur Entwicklern helfen, geeignete von nicht geeigneten Werkzeugen zu unterscheiden. Trotz aller Zweifel. ■

- [1] Devanbu et al. (2016), *Belief & evidence in empirical software engineering*, <https://doi.org/10.1145/2884781.2884812>
- [2] Kaijanaho (2015), *Evidence-based programming language design: a philosophical and methodological exploration*, <https://jyx.jyu.fi/handle/123456789/47698>
- [3] Glanville et al. (2006), *How to identify randomized controlled trials*, in MEDLINE: ten years on, <https://pubmed.ncbi.nlm.nih.gov/16636704>
- [4] Hanenberg et al. (2024a), *Indentation and Reading Time: A Controlled Experiment on the Differences Between Generated Indented and Non-indented JSON Objects*, https://doi.org/10.1007/978-3-031-61753-9_4
- [5] Hanenberg et al. (2024b), *Indentation and reading time: a randomized control trial on the differences between generated indented and non-indented if-statements*, <https://doi.org/10.1007/s10664-024-10531-y>
- [6] Git-Repo zu N-of-1-Experimenten, <https://github.com/shanenbe/Experiments>
- [7] Sebastian Kleinschmager, *Statisch oder dynamisch*, dotnetpro 2/2020, Seite 28 ff., www.dotnetpro.de/A2002Typsysteme



Sebastian Kleinschmager

ist seit vielen Jahren als freiberuflicher Softwareentwickler, Trainer und Berater unterwegs. Mit seiner lockeren Art schult er zu Themen wie .NET Core, Angular, Clean Coding oder Test-First und setzt die Techniken in Projekten ein.

himself@schmaga.de

dnpCode

A2503Codeeintrückungen

ZEITANGABEN UNTER .NET RICHTIG HANDHABEN

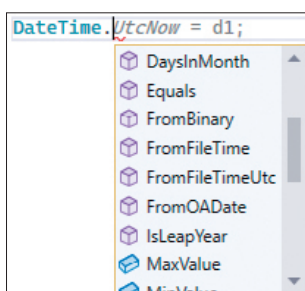
Doing the Time Warp

Die Schritte beim Umgang mit den .NET-Klassen zu Date und Time wollen gelernt sein.

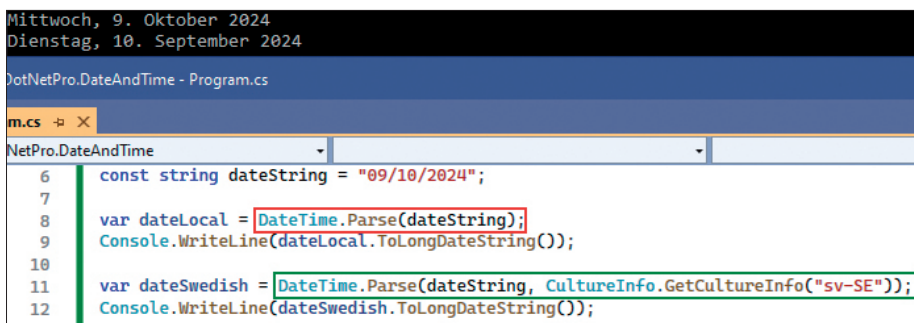
Für die Verwendung von Datums- und Zeitangaben in .NET stellt das .NET Framework einige Klassen zur Verfügung. Was hat es mit relativen Zeitangaben und Zeitperioden auf sich? Wie wirken sich verschiedene Zeitzonen auf die Anwendung aus? Wie kann zwischen den Zeitzonen konvertiert werden? Was ist der *DateTimeOffset*, und wie wird er verwendet? Der Klärung dieser Fragen widmet sich dieser Beitrag.

Ein Zeitpunkt kann in .NET durch den Typ *DateTime* repräsentiert werden, der das Datum und die Uhrzeit eines Tages darstellt und umfangreiche Funktionen anbietet (Bild 1).

Die Klasse *DateTime* hat unter anderem eine Funktion, die einen String parst und in einem *DateTime*-Objekt zur Verfügung stellt. In



Die Klasse *DateTime* (Bild 1)



Unterschiedliche Interpretation von Datumsangaben (Bild 2)

Bild 2 wird diese Funktion zweimal aufgerufen. Der erste Aufruf verzichtet bewusst auf die Angabe einer spezifischen Kultur, in diesem Fall wird die Kultur des Systems angewendet. Hingegen wird beim zweiten Aufruf die schwedische Kultur explizit angegeben. Die Ausgabe im oberen Abschnitt von Bild 2 zeigt, wie sich dieser Parameter auswirkt.

Die beiden ersten Angaben in dem zu parsenden String 09/10/2024, also die Anteile 09 und 10, werden unterschiedlich ausgewertet. Im ersten Fall steht die 09 für den Tag des Monats und 10 für den Monat, es führt zur Ausgabe *Mittwoch, 9. Oktober*. Mit Anwendung der schwedischen Kultur ist es genau umgekehrt, was zur Ausgabe von *Dienstag, 10. September* führt. Schon dies zeigt, dass beim Umgang mit Zeitangaben und bei deren Verarbeitung Sorgfalt gefragt ist.



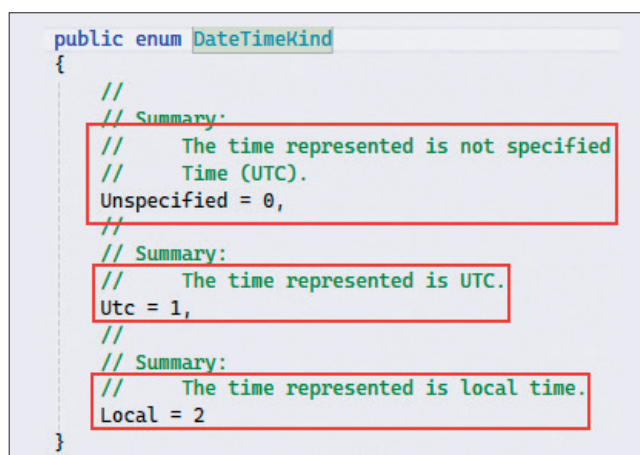
ISO-8601-Format im Überblick (Bild 3)

Problem der Mehrdeutigkeit und dessen Lösung

Das geschilderte Problem der Mehrdeutigkeit muss betrachtet und behandelt werden: Ist mit der Angabe 02/11/2024 der 2. November 2024 oder der 11. Februar 2024 gemeint?

Diese Behandlung wird durch den internationalen Standard ISO 8601 [1] vorgenommen, der besagt, dass das Datumsformat YYYY-MM-DD lautet. Es wird auch als „internationales Datumsformat“ betrachtet. Bild 3 zeigt das durch den Standard vorgegebene Format. Das Datum ist von der Zeit separiert und nutzt zusätzlich einen Zeitzone-Offset. Die verschiedenen Zeitzonen haben unterschiedliche Offsets in Bezug auf die Coordinated Universal Time (UTC). Der Offset kann sowohl positiv als auch negativ sein.

Kommt die Anwendung in verschiedenen Zeitzonen zum Einsatz, sollten Datumsangaben somit definitiv in UTC gespeichert werden.



Darstellungsoptionen von *DateTimeKind* (Bild 4)

```
(UTC+09:00) Osaka, Sapporo, Tokio' | 'Tokyo Standard Time'
(UTC+09:00) Pjöngjang' | 'North Korea Standard Time'
(UTC+09:00) Seoul' | 'Korea Standard Time'
(UTC+09:30) Adelaide' | 'Cen. Australia Standard Time'
(UTC+09:30) Darwin' | 'AUS Central Standard Time'
(UTC+10:00) Brisbane' | 'E. Australia Standard Time'

DotNetPro.DateAndTime - Program.cs
am.cs  X
tNetPro.DateAndTime
9  foreach (TimeZoneInfo z in TimeZoneInfo.GetSystemTimeZones())
10 {
11     Console.WriteLine($"{z.DisplayName} | '{z.Id}');
12 }
```

Zeitzone-IDs abfragen (Bild 5)

Die Mehrdeutigkeit betrifft nicht nur die Datumsangabe, sondern auch die Angabe der Zeit – das gilt jedenfalls immer im Hinblick auf den Dialog zwischen Teilnehmern, die sich in unterschiedlichen Zeitzonen aufhalten. Ist zum Beispiel von „9 Uhr“ die Rede, stellt sich außerdem die Frage, ob diese Angabe vormittags oder abends gemeint ist.

Die Verwendung von *DateTime.Now* stellt das Datum in der lokalen Zeit des jeweiligen Systems zur Verfügung, das heißt im Format *DateTimeKind.Local* (vergleiche Bild 4), während *DateTime.UtcNow* das Datum im Format UTC bereitstellt, also als *DateTimeKind.Utc*. Es ist daher sinnvoll, den Offset der Zeitzone zu vermerken.

Die Verwendung von *DateTime.Now* kann durchaus zu Problemen führen, weil es wie zuvor erwähnt die Zeit des lokalen Systems liefert. So kann es bei Unit-Tests, die lokal fehlerfrei ausgeführt werden, dann zu unerwarteten Problemen kommen, wenn diese zum Beispiel beim Cloud-Provider ausgeführt werden. In welcher Zeitzone sich das verwendete System befindet oder befinden wird, dürfte zudem in vielen Fällen nicht ganz klar sein.

Konvertierung zwischen Zeitzonen

Um die Zeit von einer gegebenen Zeitzone, bei der es sich um eine abgegrenzte geografische Region handelt, in eine andere Zeitzone umzuwandeln, stellt .NET die Klasse *TimeZoneInfo* [2] zur Verfügung. Im Unterschied zur Klasse *TimeZone* [3] kann *TimeZoneInfo* eine beliebige Zeitzone darstellen. Mit deren angebotenen Funktionen kann die Zeitangabe einer Zeitzone in die einer anderen Zeitzone gewandelt werden.

Neben der lokalen oder der UTC-Zeit lässt sich die Zeitinformation auch anhand einer Zeitzone-ID ermitteln. Die Zeitzonen können wie in Bild 5 gezeigt abgefragt werden.

Um zu ermitteln, welche Zeit in Hawaii in Bezug auf die lokale Zeitzone aktuell gültig ist, wird über die hawaiianische ID das hawaiianische Zeitzone-Objekt ermittelt, das mit der lokalen Zeit auf die Konvertierungsroutine angewendet wird. In Bild 6 ist die aktuelle lokale Zeit zu sehen, und darunter, welche Zeit in Hawaii gilt.

Wie bereits zuvor erwähnt ist es durchaus sinnvoll, Zeitangaben in UTC zu speichern; bei der Darstellung hingegen sollte darüber nachgedacht werden, diese in der Zeitzone *local users* darzustellen. Auch ist zu berücksichtigen, dass bei der Nutzung von *DateTime* anstelle von *DateTimeOffset* die Information fehlt, von welcher *CultureInfo* dieser Zeitstempel erfasst wurde.

DateTimeOffset [4] enthält zusätzlich die Information des UTC-Offsets. Über die Funktion *GetUtcOffset* kann der Versatz – der Offset – zwischen den Zeitzonen ermittelt werden. ►

```
Local time: 09.11.2024 15:55:37
Hawaiian time: 09.11.2024 04:55:37

DateAndTime - Program.cs
AndTime
var currentLocalTime = DateTime.Now;
var hawaiianTimeZone = TimeZoneInfo.FindSystemTimeZoneById("Hawaiian Standard Time");
var currentLocalTimeHawaii = TimeZoneInfo.ConvertTime(currentLocalTime, hawaiianTimeZone);

Console.WriteLine($"Local time: {currentLocalTime}");
Console.WriteLine($"Hawaiian time: {currentLocalTimeHawaii}");
```

Lokale Zeit in die hawaiianische Zeitzone wandeln (Bild 6)

```
(UTC+10:00) Brisbane
(UTC+10:00) Guam, Port Moresby
(UTC+10:00) Wladiwostok
(UTC-06:00) Osterinsel
(UTC-05:00) Bogotá, Lima, Quito, Rio Branco
(UTC-05:00) Chetumal
(UTC-05:00) Eastern Zeit (USA & Kanada)
(UTC-05:00) Haiti
(UTC-05:00) Havanna
(UTC-05:00) Indiana (Ost)
(UTC-05:00) Turks- und Caicosinseln

DateAndTime - Program.cs
AndTime
WriteTimeZonesByOffset(DateTimeOffset.Now.ToOffset(TimeSpan.FromHours(10)));
WriteTimeZonesByOffset(DateTimeOffset.Now.ToOffset(TimeSpan.FromHours(-5)));

2 references
void WriteTimeZonesByOffset(DateTimeOffset timeZoneToSearchFor)
{
    foreach(var timeZone in TimeZoneInfo.GetSystemTimeZones())
    {
        if (timeZone.GetUtcOffset(timeZoneToSearchFor) == timeZoneToSearchFor.Offset)
            Console.WriteLine(timeZone);
    }
}
```

Versatz zwischen Zeitzonen ermitteln (Bild 7)


```

Input: 2024-07-01 10:00:00 PM +02:00
Parsed: 01.07.2024 22:00:00

DateAndTime - Program.cs

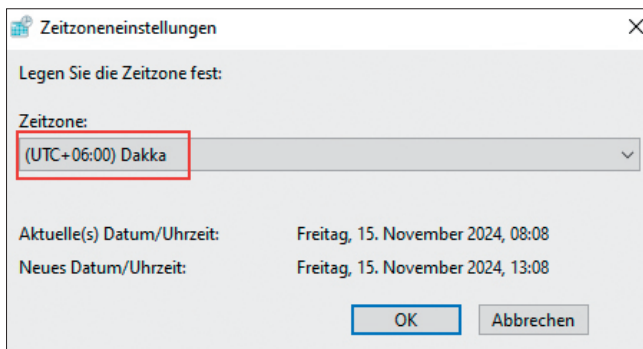
const string inputDate = "2024-07-01 10:00:00 PM +02:00";
var parsed = DateTime.Parse(inputDate);
Console.WriteLine($"Input: {inputDate}");
Console.WriteLine($"Parsed: {parsed} ");

```

Parsen von DateTime (Bild 8)

Dies ist in Bild 7 gezeigt, wo ausgehend von der lokalen Zeit die Zeitzonen ermittelt werden, die 5 Stunden vor der eigenen und 10 Stunden nach der eigenen liegen. *DateTimeOffset* enthält neben dem Datum und der Uhrzeit auch den *TimeZoneOffset*, was den Einsatz über Zeitzonengrenzen hinweg vereinfacht.

Betrachten wir das Beispiel aus Bild 8: Der an *DateTime.Parse* übergebene String wird korrekt geparkt. Welche Ausgabe die Funktion generiert, wenn die Zeitzone auf dem Windows-Betriebssystem geändert wird (Bild 9), ist in Bild 10 zu sehen. Es ist offensichtlich, dass automatisch die lokale Zeitzone an-



Festlegen der Zeitzone (Bild 9)

gewendet wird. Auch fehlen in dem geparkten String sowohl der Offset als auch eine Angabe der Zeitzone.

Einiges zu UTC

Unter [5] ist eine Karte der verschiedenen Zeitzonen verfügbar, aus welcher auch die Bilder 11 und 12 entnommen wurden. In Bild 11 ist einerseits zu sehen, dass Deutschland einen Offset von +1 hat, als auch, dass die Grenzen der Zeitzonen nicht gerade verlaufen.

Mit diesen Grafiken soll nochmals verdeutlicht werden, dass bei Anwendungen, die verteilt über verschiedene Zeitzonen betrieben werden, mit Problemen zu rechnen ist, wenn

```

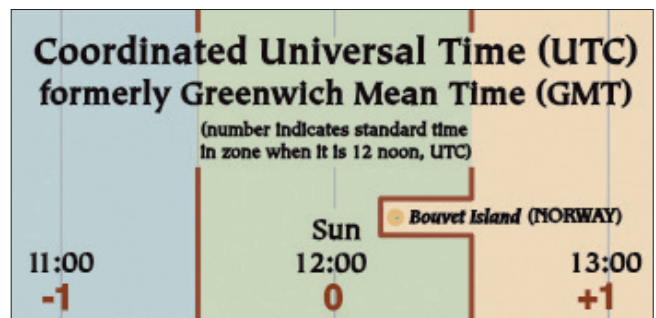
Input: 2024-07-01 10:00:00 PM +02:00
Parsed: 02.07.2024 02:00:00

```

Ändern der Zeitzone (Bild 10)



Kartenansicht der Zeitzone UTC (Bild 11)



UTC beziehungsweise Greenwich Mean Time (Bild 12)

Datums- beziehungsweise Zeitangaben nicht in UTC übermittelt werden. Die Anwenderinnen und Anwender dagegen sind vermutlich daran interessiert, die Darstellung entsprechend ihrer lokalen Zeitzonen zu sehen.

```

UTC now: 17.11.2024 10:03:06 +00:00
Local time: 17.11.2024 11:03:06 +01:00
Local to UTC: 17.11.2024 10:03:06 +00:00

DateAndTime - Program.cs

var utcNow = DateTimeOffset.UtcNow;
Console.WriteLine($"UTC now: {utcNow}");

var localTime = utcNow.ToLocalTime();
Console.WriteLine($"Local time: {localTime}");

var localToUtc = localTime.ToUniversalTime();
Console.WriteLine($"Local to UTC: {localToUtc}");

```

UTC Now und die lokale Darstellung (Bild 13)

```

### English (United States) ###
CalendarWeekRule: FirstDay.
FirstDayOfWeek: Sunday.
Current week: 48
Total number of weeks: 53.

### German (Germany) ###
CalendarWeekRule: FirstFourDayWeek.
FirstDayOfWeek: Monday.
Current week: 48
Total number of weeks: 53.

var cultures = new List<CultureInfo>
{
    new CultureInfo("en-US"),
    new CultureInfo("de-DE")
};

foreach (var culture in cultures)
{
    Console.WriteLine($"### {culture.EnglishName} ###");
    var calendar = culture.Calendar;

    var calendarWeekRule = culture.DateTimeFormat.CalendarWeekRule;
    var firstDayOfWeek = culture.DateTimeFormat.FirstDayOfWeek;
    var weekOfYearNow = calendar.GetWeekOfYear(DateTime.Now, calendarWeekRule, firstDayOfWeek);

    Console.WriteLine($"CalendarWeekRule: {calendarWeekRule}.");
    Console.WriteLine($"FirstDayOfWeek: {firstDayOfWeek}.");
    Console.WriteLine($"Current week: {weekOfYearNow}.");

    DateTime lastDay = new DateTime(DateTime.Now.Year, 12, 31);
    var weekOfYearLastDay = calendar.GetWeekOfYear(lastDay, calendarWeekRule, firstDayOfWeek);

    Console.WriteLine($"Total number of weeks: {weekOfYearLastDay}.");
    Console.WriteLine("");
}

```

Kalenderwoche ermitteln (Bild 14)

```

To late: 01.01.2025 00:00:00 +00:00
Correctly: 31.12.2024 23:59:59 +00:00

var contract = new DateTimeOffset(2024, 10, 1, 0, 0, 0, TimeSpan.Zero);

const int expiresInThreeMonths = 3;
var endsToLate = contract.AddMonths(expiresInThreeMonths);
var endsCorrectly = contract.AddMonths(expiresInThreeMonths).AddTicks(-1);

Console.WriteLine($"To late: {endsToLate}");
Console.WriteLine($"Correctly: {endsCorrectly}");

```

Kalenderwoche berechnen (Bild 15)

Wie die lokale Zeit zum Beispiel in einem Client dargestellt und im nächsten Schritt in UTC gewandelt und bei Bedarf wieder entsprechend der lokalen Zeit zurückgewandelt wird, zeigt **Bild 13**.

Wochennummer berechnen

Die Wochennummer wird häufig bei der Planung von Terminen benutzt. Sehen wir uns an dieser Stelle eine Möglichkeit der Wochennummer-Ermittlung mit dem Typ *Calendar* an.

Hierzu ist das Beispiel aus **Bild 14** zu betrachten. Die farblich markierten Ausgaben deuten auf Unterschiede bei den verschiedenen Regionen hin.

Die Werte der Eigenschaft *CalendarWeekRule* haben folgende Bedeutung:

- *FirstDay*: Die erste Woche, die auf diesem Wert basiert, kann einen bis sieben Tage haben.

- *FirstFourDayWeek*: Die erste Woche, die auf diesem Wert basiert, kann vier bis sieben Tage haben. Der erste Tag des Jahres, der 1. Januar, wird als Beginn der ersten Woche des Jahres definiert. Zwischen dem 1. Januar und dem Tag vor dem folgenden Sonntag liegen mehr als vier Tage.

Ein kleiner Tipp

Angenommen, es soll das Ablaufdatum eines Vertrags mit einer Laufzeit von drei Monaten berechnet werden. **Bild 15** zeigt hierzu eine Implementierung, bei der mittels der Funktion *AddMonths(..)* die Laufzeit addiert wird. Die Ausgabe zeigt jedoch, dass das Ablaufdatum eine winzige Kleinigkeit zu spät ist. Der Vertrag muss nämlich noch im aktuellen Jahr enden. Mit *AddTicks(-1)* wird das gewünschte Resultat erzielt.

Fazit

Der Umgang mit Datums- und Zeitangaben ist nicht ganz trivial. Es gibt dabei einiges zu beachten, aber die .NET-Klassen bieten hierfür entsprechende Unterstützung an. ■

- [1] ISO 8601 bei Wikipedia, www.dotnetpro.de/SL2503DateAndTime1
- [2] Microsoft Learn, *TimeZoneInfo*, www.dotnetpro.de/SL2503DateAndTime2
- [3] Microsoft Learn, *TimeZone*, www.dotnetpro.de/SL2503DateAndTime3
- [4] Microsoft Learn, *DateTimeOffset*, www.dotnetpro.de/SL2503DateAndTime4
- [5] UTC bei Wikipedia, www.dotnetpro.de/SL2503DateAndTime5



Christian Havel

wohnt in einem Vorort von München und ist seit mehr als zwanzig Jahren in der Softwareentwicklung tätig. Er programmiert hauptsächlich in C#. Sie erreichen ihn unter christian.havel@googlemail.com.

MESSAGING MIT WOLVERINE UND .NET, TEIL 9

Luftschlange

Im abschließenden Teil der Serie zu Wolverine machen wir unsere Messaging-Anwendung bereit für die Cloud.

Seit Beginn der Serie zu Wolverine 1] haben wir nur Transportmechanismen verwendet, die lokal oder optional in der Cloud (RabbitMQ) betrieben werden können. In diesem abschließenden Teil der Serie werden wir nun einen Mechanismus nutzen, der Cloud-native ist, also ausschließlich in der Cloud läuft. Hierbei handelt es sich um den Simple Queue Service, kurz SQS, aus den Amazon Web Services (AWS).

SQS bietet eine hochverfügbare, persistente (durable) Messaging Queue. Neben der für Cloud-Dienste obligatorischen Verfügbarkeit und Skalierbarkeit und der damit erreichten Verlässlichkeit ermöglicht SQS das Verschlüsseln von Nachrichten. Dies kann entweder mit vollständig von SQS verwalteten Schlüsseln geschehen (das ist der Default) oder mit eigenen, im AWS Key Management Service verwalteten Schlüsseln.

Bei der Persistenz gibt es zwei Optionen: Standard-Queues unterstützen lediglich At-least-once Delivery, während FIFO-Queues Exactly-once Message Processing gewährleisten.

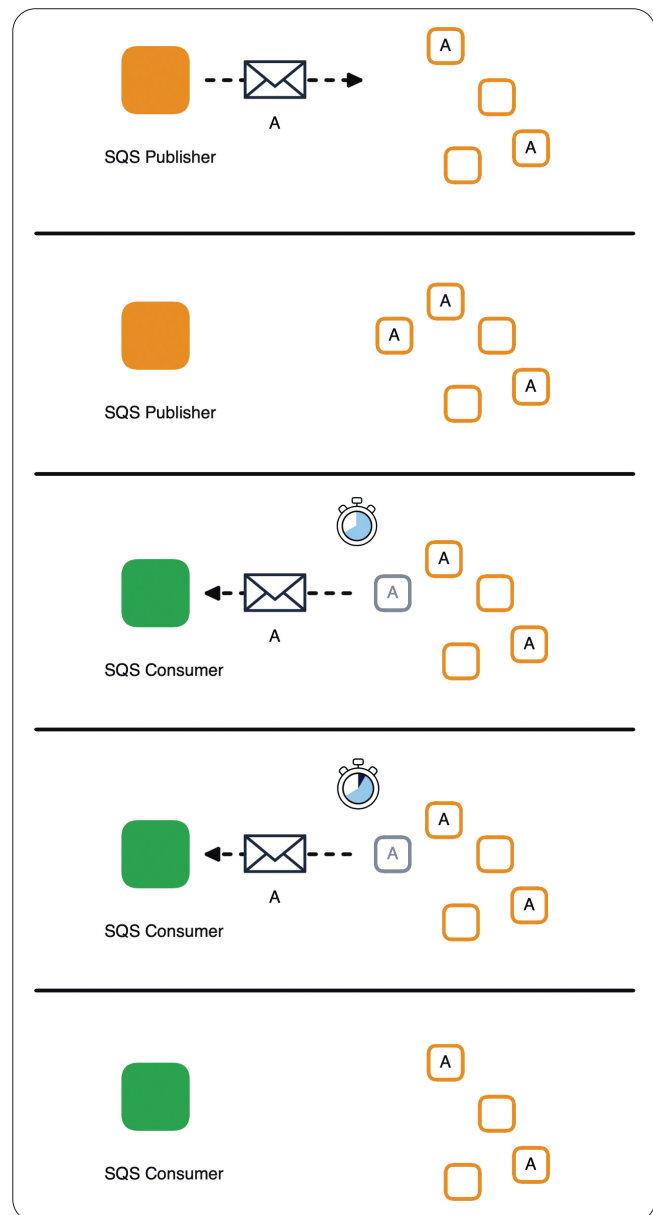
Queue, (eigentlich) ganz einfach

Wie verwendet man SQS nun in der Praxis? Zunächst sendet ein Producer eine Nachricht an eine Queue. Die Nachricht wird mit einem sogenannten Visibility Timeout versehen. Ein Consumer empfängt die Nachricht aus der Queue und bearbeitet diese, wodurch der Visibility Timeout gestartet wird. Nach dem Verarbeiten löscht der Consumer die Nachricht aus der Queue. Der Prozess ist in Bild 1 dargestellt.

Was genau ist nun der Visibility Timeout? Damit Nachrichten in SQS nicht von mehreren Consumern abgerufen werden, werden sie nach dem Abruf durch den ersten Consumer in der entsprechenden Queue unsichtbar geschaltet. Verarbeitet der erste Consumer die Nachricht innerhalb des Visibility Timeouts, wird die Nachricht nach dem Verarbeiten gelöscht. Ist die Verarbeitung der Nachricht nach dem Ablauf des Visibility Timeouts nicht abgeschlossen, wird die Nachricht in der Queue wieder sichtbar und die Nachricht kann erneut abgerufen werden. Das Diagramm in Bild 2 verdeutlicht dieses Verfahren.

Mit dem Visibility Timeout lassen sich auch spezifisch gelagerte Anwendungsfälle abbilden:

- **Lang laufende Prozesse:** Indem der Timeout für Nachrichten, die lang laufende Prozesse anstoßen, erhöht wird, wird vermieden, dass die Nachricht an weitere Consumer zugestellt wird, bevor die Bearbeitung durch den ursprünglichen Consumer der Nachricht abgeschlossen wurde. Das Limit für den Visibility Timeout beträgt zwölf Stunden. Müssen Prozesse länger als zwölf Stunden laufen, muss man den



Senden und Empfangen von Nachrichten mit SQS (Bild 1)

Prozess kleinteiliger gestalten oder andere AWS-Dienste verwenden, um den Prozess abzubilden.

- **Retry-Pattern:** Wenn eine Nachricht innerhalb des Visibility Timeouts nicht verarbeitet werden konnte, erhöht man den Visibility Timeout (per Code) und versucht die Verarbeitung der Nachricht erneut.

Das Löschen der Nachrichten nach der Verarbeitung erfolgt entweder automatisch, wenn man ein offizielles AWS-SDK für SQS verwendet, oder durch die Verwendung der *Delete-Message*-Aktion des SQS-API.

Grenzen setzen

Wie bei jedem Cloud-Service gibt es auch bei AWS SQS Quotas, die den Nutzungsumfang definieren. Bei SQS sind dies die sogenannten In-flight Messages pro Queue. Hierbei handelt es sich um Nachrichten, die von einem Consumer bereits abgerufen, aber noch nicht gelöscht wurden. Für Standard-Queues liegt die Obergrenze zum Zeitpunkt der Abfassung dieses Artikels bei circa 120000 In-flight Messages pro Queue. Beim Überschreiten dieser Grenze erhält man einen *OverLimit*-Fehler, der signalisiert, dass keine weiteren Nachrichten aus der Queue abgerufen werden können, bevor nicht einige vorhandene In-flight Messages gelöscht werden.

Da die Standard-Queues At-least-once Delivery garantieren, kann es trotz Visibility Timeout zur mehrfachen Zustellung einer Nachricht kommen. Um zu vermeiden, dass Nachrichten endlos erneut zugestellt und nicht verarbeitet werden (und damit die Zahl der In-flight Messages ständig wächst), empfiehlt sich der Einsatz von Dead Letter Queues (DLQ).

Themenwechsel

Im Konzept des Visibility Timeouts ist implizit eine Einschränkung von SQS versteckt: Der Service kann, anders als zum Beispiel RabbitMQ, keinen Fan-out-Mechanismus, das heißt das Senden einer Nachricht an mehrere Consumer, bereitstellen. Wie viele Cloud-Services besitzt auch SQS ein beschränktes Feature-Set, das bei Bedarf dadurch erweitert wird, indem man es mit einem weiteren Cloud-Service kombiniert. Der Cloud-Service, der die fehlende Fan-out-Funktionalität für SQS liefert, ist der Simple Notification Service (SNS) von AWS.

SNS erlaubt es Publishern, Nachrichten über sogenannte Topics, die als Channel dienen, zu senden. Die Nachrichten werden über die Channels an Endpunkte zugestellt, wobei es mehrere Typen von Endpunkten geben kann. Neben SQS Queues sind zum Beispiel HTTP-Endpunkte, E-Mail oder AWS-Lambda-Functions als Endpunkte möglich. **Bild 3** veranschaulicht die genannten SNS-Konzepte.

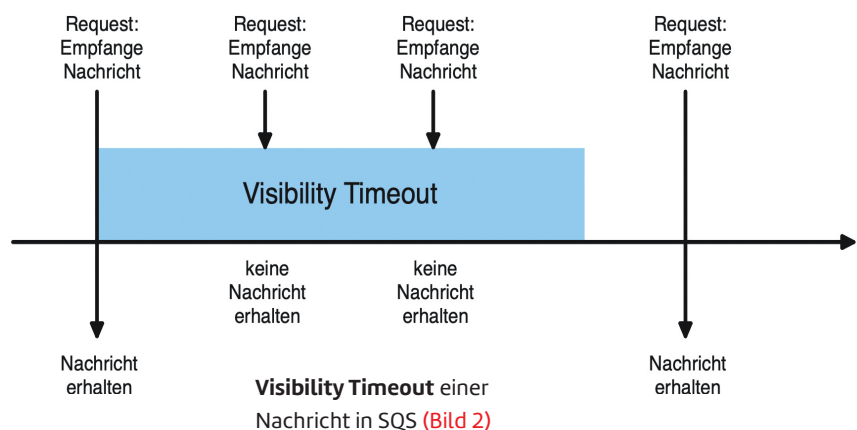
Umsetzung

Für den Betrieb der Queues benötigen wir selbstredend einen AWS-Account. Nachdem dieser eingerichtet ist, empfiehlt es sich, nicht mit dem root-Account zu arbeiten. Stattdessen legen wir uns eine Benutzergruppe *dotnetpro-wolverine* in AWS IAM an, die Vollzugriff auf AWS SQS bekommt. Die entsprechende Policy lautet *AmazonSQSFullAccess*.

Nach dem Anlegen der Gruppe legen wir einen Benutzer *dotnetpro-wolverine* an und weisen ihn der Gruppe zu. Für den Benutzer erzeugen wir Access-Key-Credentials. Da-

durch erhalten wir eine Access-Key-ID (zum Beispiel *AKIAIOSFODNN7EXAMPLE*) und einen Secret Access Key (zum Beispiel *wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY*). Die Kombination aus beiden wird benötigt, um einen Request auf die AWS Services zu authentifizieren. Beide Credentials werden sicher aufbewahrt und nicht im Code gespeichert.

Um die AWS-Credentials an unsere Anwendung zu übergeben, gibt es mindestens zwei Wege: Man legt mit dem AWS-CLI ein AWS-Benutzerprofil auf dem Rechner an. Dieses erzeugt zwei Dateien *config* und *credentials* in einem Ordner *.aws* im Home-Verzeichnis des angemeldeten Benutzers. Über diesen Mechanismus können mehrere benannte Profile erzeugt werden. Der Standardname für das Profil lautet *default*. Verwendet man das AWS-SDK (unabhängig von



der Entwicklerplattform), wird standardmäßig im genannten Ordner nach dem *default*-Profil gesucht.

Ein weiterer Ansatz, den wir verwenden werden, ist, die Credentials in der Anwendungsconfiguration zu hinterlegen und den Code entsprechend zu konfigurieren. Da AWS SQS ein regionaler Service ist, muss entsprechend auch die Region, die verwendet werden soll, angegeben werden, zum Beispiel *eu-central-1* (Frankfurt).

SQS mit Wolverine

Unser Ziel ist, die Anwendung *WolverineMartenCreditLimitSaaS*, die wir im vorangegangenen Artikel [2] erarbeitet haben, nun mit SQS statt mit RabbitMQ zu betreiben. Ausgehend vom vorhandenen Code werden wir die notwendigen Änderungen durchführen, um dieses Ziel zu erreichen.

Zunächst fügen wir sowohl im **Customers**- als auch im **Orders**-Projekt eine *appsettings.json*-Datei hinzu. Der Inhalt sieht wie folgt aus:

```
{
  "aws": {
    "AccessKeyId": "",
    "SecretAccessKey": ""
  }
}
```


Die fehlenden Werte für die Credentials sind vom Leser gemäß den oben genannten Schritten zu befüllen.

Im nächsten Schritt lesen wir die Konfiguration – auch dies findet wieder in beiden Services statt:

```
var configuration = new ConfigurationManager()
    .AddJsonFile(
        "appsettings.json",
        optional: false,
        reloadOnChange: true
    )
    .AddJsonFile(
        $"appsettings.{Environment
            .GetEnvironmentVariable(
                "ASPNETCORE_ENVIRONMENT")}.json",
        optional: true,
        reloadOnChange: true
    )
    .Build();
```

Da wir als Transport nun AWS SQS statt RabbitMQ verwenden, benötigen wir die entsprechende Unterstützung für Wolverine. Diese installieren wir uns mittels NuGet:

```
dotnet add package WolverineFx.AmazonSqs
```

Danach teilen wir Wolverine mit, dass wir SQS in der Region Frankfurt (*eu-central-1*) verwenden und wie die Credentials lauten:

```
var accessKeyId =
    configuration.GetSection("aws")["AccessKeyId"];
var secretAccessKey =
    configuration.GetSection("aws")["SecretAccessKey"];

options.UseAmazonSqsTransport(config =>
    config.RegionEndpoint =
        RegionEndpoint.EUCentral1)
    .Credentials(
        new BasicAWSCredentials(
            accessKeyId, secretAccessKey)
    )
    .AutoProvision();
```

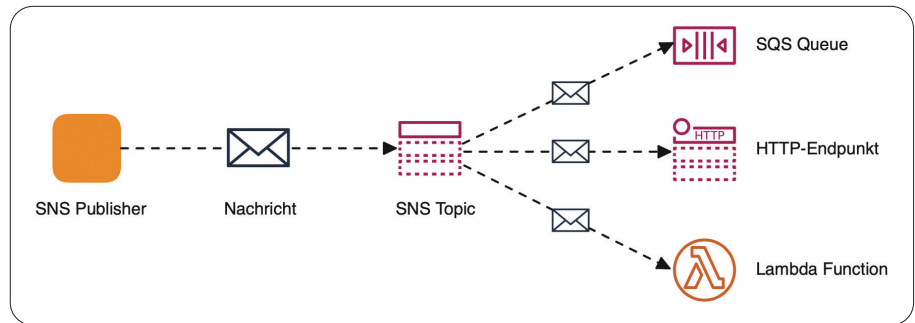
Die bisherigen Aufrufe von `.ListenToRabbitQueue` ersetzen wir durch `.ListenToSqsQueue`. Die Aufrufe von `.ToRabbitQueue` werden durch `.ToSqsQueue` ersetzt. Die Code-Blöcke mit `rabbitMq.DeclareExchange(...)` entfernen wir ersatzlos.

Damit können wir unsere Anwendung in gewohnter Manier starten und per HTTP-Request Bestellungen auslösen. Abhängig davon, ob das Kreditlimit ausreicht oder nicht, wird die Bestellung genehmigt oder zurückgewiesen.

Aufgepasst?

Damit scheint die Implementierung bereits abgeschlossen und die Anwendung auf SQS umgestellt zu sein. Erzeugen

wir mehrere Bestellungen und werfen anschließend einen Blick in die Marten-Datenbanken, fällt uns allerdings ein Fehler auf: Die Daten zwischen *Customers*- und *Orders*-Service sind nicht konsistent: Entweder fehlt im *Customers*-Service der *OrderApproved*-Event und die darauf basierende Projektion *Credit* ist fehlerhaft, oder im *Orders*-Service ist die



SNS Topics, Nachrichten und Endpunkte (Bild 3)

Order-Saga nicht korrekt beendet und damit nicht aus der entsprechenden Datenbanktabelle *mt_doc_order* gelöscht. Diese Fehler treten immer wechselseitig auf, aber nie zusammen – woran liegt das?

Der Grund dafür liegt in SQS: Wie wir eingangs gelernt haben, beherrscht SQS kein Fan-out, und die Nachricht *OrderApproved* wird immer nur an einen der beiden Services zugestellt und danach aus der Queue gelöscht.

Die Funktionalität für Fan-out in RabbitMQ steckte in den Aufrufen von *DeclareExchange*, mit denen wir die Fan-out-Exchanges an die entsprechenden Queues gebunden haben. Wie wir ebenfalls in diesem Artikel gelernt haben, benötigen wir für Fan-out beziehungsweise Publish/Subscribe mit AWS SQS zusätzlich AWS SNS. Allerdings unterstützt WolverineFx.AmazonSqs (wie der Name vermuten lässt) AWS SNS nicht. Darüber hinaus gibt es zum derzeitigen Zeitpunkt noch keine SNS-Unterstützung für Wolverine. Wie können wir unsere Anwendung aktuell dennoch betreiben?

Bei RabbitMQ sah die entsprechende Exchange-Konfiguration für den *Orders*-Service wie nachfolgend dargestellt aus:

```
options
    .UseRabbitMq()
    .DeclareExchange(
        "approved-orders",
        exchange =>
        {
            exchange.ExchangeType = ExchangeType.Fanout;
            exchange.BindQueue("approved-orders-orders");
        }
    )
    .AutoProvision();
```

Die Exchange-Konfiguration für den *Customers*-Service lautete bei RabbitMQ:

```
rabbitMq.DeclareExchange(
    "approved-orders",
    exchange => { exchange.BindQueue(
        "approved-orders-customers"); }
);
```

Definieren wir nun im *Orders-Service* den *PublishMessage<OrderApproved>()* ein weiteres Mal, jedoch für die Queue des *Customers-Service* wie folgt:

```
options.PublishMessage<OrderApproved>()
    .ToSqsQueue("approved-orders-customers");
```

Damit ist das geschilderte Problem behoben, und die Anwendung funktioniert nun wieder wie gewohnt.

Allerdings muss man sich im Klaren darüber sein, dass man jetzt eine sehr starke Kopplung erzeugt hat, da der *Orders-Service* nun dafür zuständig ist, die Queue für den *Customers-Service* zu erzeugen. Bis die entsprechende SNS-Unterstützung in Wolverine implementiert sein wird, bleiben jedoch nicht viele Ausweichmöglichkeiten, wenn man den Simple Queue Service verwenden möchte beziehungsweise muss.

Dieser zugegebenermaßen unschöne Zustand muss allerdings nicht zwingend zum Tragen kommen, abhängig davon, welche Art von Anwendung man mit Wolverine und SQS entwickelt.

Fazit

Mit dem Blick auf die AWS-SQS-Integration in Wolverine endet diese Artikelserie. Zusammenfassend kann man Wolverine und den zugehörigen Integrationen – insbesondere Marten – einen hohen Reifegrad attestieren. Die erfrischend schlanken Lösungsansätze für etablierte Aufgabenstellungen wie die transaktionale Outbox machen es in der Praxis sehr angenehm, Messaging-basierte Anwendungen mit Wolverine zu entwickeln. ■

[1] Alexander Zeitler, *Einsteigen, bitte!*,
dotnetpro 7/2024, Seite 66 ff.,
www.dotnetpro.de/A2407Wolverine

[2] Alexander Zeitler, *The Saga continues*,
dotnetpro 2/2025, Seite 72 ff.,
www.dotnetpro.de/A2502Wolverine



Alexander Zeitler

entwickelt als Cloud Solutions Architect B2B-Plattformen im industriellen Umfeld mit Schwerpunkt AWS, .NET und Node.js. Er unterstützt Unternehmen mit Architekturberatung und Entwicklung für Event-basierte Systeme.
alexander.zeitler@pdmlab.com

dnpCode

A2503Wolverine

DEV ACADEMY

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN



2 TAGE
REMOTE/KÖLN

Semantisches Datenmanagement

- Einführung in das semantische Web und OWL-Ontologien
- Generelle Ansätze von Wissensrepräsentation und Wissensmanagement
- Generierung und Abfrage von Wissen, Regeln und Restriktionen in OWL
- Konzeptionelles und praktisches Wissen über Reasoner und Inferenz
- Einführung in Nutzen und Einsatz semantischer Graph-Datenbanken
- SPARQL als Sprache für Wissensmanagement mit praktischen Beispielen

TRAINING!

Alexander Schulze
Trainer



REGEXTRACT: DATEN EXTRAHIEREN LEICHT GEMACHT

Aber bitte typsicher

RegexExtract ist eine moderne Bibliothek zur Extraktion von Daten aus Texten. Das Projekt kombiniert die Mächtigkeit regulärer Ausdrücke mit der Typsicherheit von C#.

Mit regulären Ausdrücken verhält es sich wie mit Trauben-Nuss-Schokolade: Die einen lieben sie, die anderen verabscheuen sie. Anspruchsvoll ist die komplexe Syntax allemal und führt nicht selten zu schwer lesbarem Code. Verschachtelte Gruppen und Capture-Groups erschweren noch dazu die Wartbarkeit. Fehlerhafte Muster bleiben oft bis zur Laufzeit unentdeckt, da die statische Typsicherheit von .NET hier nicht greift. Tests für die regulären Ausdrücke sind zwar möglich und wichtig, aber aufgrund der Komplexität der Muster ebenfalls aufwendig und komplex. Die Standard-Implementierung von Regex erfordert zudem umständliche Typkonvertierungen, wenn extrahierte Werte in stark typisierte Objekte überführt werden sollen.

RegexExtract [1] bietet einen neuen Lösungsansatz für diese Probleme. Die Bibliothek steht im Fokus dieser Episode der

Zusätzlich bietet RegexExtract erweiterte Fehlerbehandlung mit aussagekräftigen Meldungen bei Extraktionsfehlern oder ungültigen Mustern.

Im Vergleich zur klassischen Regex-Nutzung in .NET entfällt bei RegexExtract die manuelle Nachbearbeitung der Ergebnisse. Herkömmliche Ansätze erfordern mehrere Zeilen Code für die Extraktion und Typkonvertierung eines einzelnen Werts. Mit RegexExtract reduziert sich dieser Aufwand auf einen Methodenaufruf. Die frühe Fehlererkennung durch den Compiler minimiert zudem Laufzeitfehler, die bei Standard-Regex durch fehlerhafte Muster oder Konvertierungen entstehen können.

RegexExtract richtet sich primär an .NET-Entwickler, die regelmäßig Daten aus Texten extrahieren müssen. Typische Anwendungsfälle umfassen die Verarbeitung von Log-Datei-

● Listing 1: Erstes Beispiel zur typsicheren Extraktion von Daten

```
using RegexExtract;

// Definition eines Extraktionsmusters
const string text = "Alter: 25 Jahre";
const string pattern = @"Alter: (?<age>\d+) Jahre";

// Extraktion und Typkonvertierung in einem Schritt
var age = text.Extract<int>(pattern);

// Ergebnis: age enthält den Wert 25 als Integer
Console.WriteLine(age);
```

Kolumne Frameworks und mehr und ergänzt die bekannten Regex-Funktionen von .NET um ein typsicheres API. Sie wandelt reguläre Ausdrücke automatisch in stark typisierte Extraktoren um. Muster lassen sich weiterhin als Strings definieren, die aber von Compile-Zeit-Prüfungen und direkter Konvertierung in Zieltypen profitieren. Die Bibliothek integriert sich nahtlos in bestehende .NET-Projekte und ermöglicht eine prägnante, wartbare Implementierung von Textextraktionen.

Was ist RegexExtract?

Die Kernfunktionalität von RegexExtract liegt in der automatischen Umwandlung von Capture Groups in typisierte Objekte. Die Bibliothek analysiert die Struktur des regulären Ausdrucks zur Kompilierzeit und generiert passende Extraktoren. Diese wandeln gefundene Textfragmente direkt in primitive Datentypen oder komplexe Objekte um.

en, das Parsen von strukturierten Texten oder die Analyse von Benutzerangaben. Die Bibliothek eignet sich besonders für Szenarien, in denen die extrahierten Daten direkt in das Typsystem der Anwendung überführt werden sollen.

Die Integration von RegexExtract vereinfacht besonders die Entwicklung von Systemen zur Textanalyse oder Datenextraktion. Durch die Compile-Zeit-Prüfungen und automatische Typkonvertierung reduziert sich die Fehleranfälligkeit der Implementierung. Der Code bleibt dabei übersichtlich und wartbar, da komplexe Extraktionslogik in wenigen, leicht verständlichen Zeilen ausgedrückt werden kann.

Die Bibliothek ist Open Source und steht unter der MIT-Lizenz. Der Quelltext liegt in einem Repository [1] auf GitHub. Das Projekt wird aktuell gepflegt, allerdings nicht mehr so aktiv wie zu Beginn. Die zuletzt veröffentlichte Version [2] trägt die Nummer 2.3.0 und ist am 23. Dezember 2023 erschienen.

● Listing 2: Extraktion auf Basis eines Tupels

```
const string text2 = "Name: Max Alter: 25";
const string pattern2 =
    @"Name: (?<name>\w+) Alter: (?<age>\d+)";

var result = text2.Extract<(string name, int age)>
    (pattern2);

// result.name enthält "Max", result.age enthält 25
Console.WriteLine(result);
```

Installation und erste Beispiele

RegexExtract setzt .NET Standard 2.0 oder höher voraus und läuft auf allen kompatiblen Plattformen wie .NET Core, .NET Framework und .NET 5+. Die Bibliothek benötigt keine zusätzlichen Abhängigkeiten und integriert sich nahtlos in bestehende Projekte. Die Installation erfolgt über den NuGet-Paketmanager oder durch Ausführung des folgenden Befehls in der Konsole:

```
dotnet add package RegexExtract
```

Informationen über fehlgeschlagene Extraktionen oder Typkonvertierungen.

Funktionsweise der Bibliothek

RegexExtract nutzt intern die .NET-Regex-Engine und erweitert diese um eine typsichere Abstraktionsschicht. Wie oben schon erwähnt analysiert die Bibliothek reguläre Ausdrücke zur Kompilierzeit und generiert daraus stark typisierte Extraktoren. Diese wandeln gefundene Textfragmente automatisch in die gewünschten Zieltypen um.

Das API basiert auf Extension-Methoden für den String-Typ. Die zentrale Methode *Extract<T>* extrahiert und konvertiert Werte in einem Schritt.

Die Klassenhierarchie gliedert sich in drei Hauptbereiche: das öffentliche API mit den Extension-Methoden, die internen Extraktoren für verschiedene Typen sowie die Infrastruktur für Kompilierung und Validierung der Muster.

Listing 3 zeigt dazu ein Beispiel mit einer benutzerdefinierten Datenklasse. RegexExtract extrahiert die Daten direkt typsicher in die Datenklasse, sodass diese im weiteren Verlauf der Implementierung zum Einsatz kommen kann.

Im Vergleich zur manuellen Regex-Verarbeitung eliminiert RegexExtract redundanten Code. Statt mehrerer Zeilen für Extraktion, Validierung und Konvertierung genügt ein einzelner Methodenaufruf. Der Code bleibt dadurch übersichtlich

● Listing 3: Eigene Datenklasse zur Verarbeitung der Daten aus den Regex-Captures

```
// Eigene Datenklasse für die Captures.
public class UserData
{
    public string Name { get; set; }
    public int Age { get; set; }
    public override string ToString()
    {
        return $"Name: {Name}, Alter: {Age}";
    }
}

// Beispiel für die API-Nutzung mit
// benutzerdefinierter Klasse.

const string text3 = "Name: John, Age: 30";

var data = text3.Extract<UserData>(
    @"Name: (?<Name>\w+), Age: (?<Age>\d+)");

Console.WriteLine(data);
```

Das einfache Beispiel in **Listing 1** demonstriert die grundlegende Verwendung von RegexExtract. Der Code zeigt die drei wesentlichen Schritte: Import des Namespaces, Definition des Musters mit benannten Capture Groups und Extraktion mit direkter Typkonvertierung.

RegexExtract erkennt den gewünschten Zieltyp automatisch anhand der Typparameter. Im Fall des ersten Beispiels in **Listing 1** ist das ein Integer.

Komplexere Extraktionen folgen dem gleichen Muster. Die Bibliothek erlaubt auch die Extraktion mehrerer Werte in einem Durchgang, wie **Listing 2** zeigt. Hier werden die Capture Groups für den Namen und das Alter definiert und als typisiertes Ergebnis in Form eines Tupels zurückgegeben. Die Fehlerbehandlung erfolgt durch Exceptions mit detaillierten

und fokussiert auf die eigentliche Extraktionslogik. Die Compile-Zeit-Validierung der Muster und automatische Typkonvertierung reduzieren potenzielle Laufzeitfehler. Die benannten Capture Groups im regulären Ausdruck entsprechen direkt den Eigenschaften der Zieltypen, was die Wartbarkeit des Codes verbessert.

Grundlegende Datenextraktionen an Beispielen

RegexExtract ermöglicht verschiedene Arten der Datenextraktion aus Texten. Die folgenden Beispiele zeigen die wichtigsten Anwendungsfälle und deren Implementierung.

Die einfachste Form der Extraktion verwendet ValueTuples als Zieltyp. Diese eignen sich besonders für Ad-hoc-Extraktionen ohne eigene Klassendeklaration. Die vorherigen ►

Listing 4: Batch-Extraktion von Daten aus einer Liste

```
// Batch-Extraktion aus mehreren Zeilen
var lines = new[]
{
    "User: admin Role: Administrator",
    "User: guest Role: Viewer"
};

var pattern3 = @"User: (?<username>\w+) Role: (?<role>\w+)";
IEnumerable<(string, string)> users =
    lines.Extract<(string username, string role)>
        (pattern3);

// users enthält [("admin", "Administrator"),
//                ("guest", "Viewer")]
Console.WriteLine(users);
```

Beispiele mit Alter und der Kombination aus den Daten zu Namen und Alter haben diese ValueTuples bereits gezeigt.

Diese Form der Extraktion ist unkompliziert zu implementieren und einfach in der Verwendung. Das ist wichtig, da Bibliotheken wie RegExtract potenziell einen Overhead bei der Implementierung erzeugen: Andere Methoden sind ebenso zu nutzen wie andere Ergebnistypen. Die Bibliothek möchte genau das vermeiden.

lässt sich der Typ der Ergebnismenge auf *IEnumerable<(string, string)>* festlegen – eine sehr einfache und typisierte Möglichkeit, um die Daten strukturiert zu erfassen und weiterzuverarbeiten.

Komplexere Muster profitieren von benannten Gruppen. Diese machen den Code selbstdokumentierend und vereinfachen spätere Anpassungen, insbesondere in Kombination mit den bereits vorgestellten benutzerdefinierten Datenklassen. Listing 5 zeigt ein Beispiel für ein komplexes Muster mit benannten Gruppen und der Datenklasse *LogEntry*.

Das verarbeitete Resultat ist eine gefüllte Instanz der *LogEntry*-Klasse, wie in Bild 1 gezeigt.

Die Lesbarkeit und Wartbarkeit verbessern sich durch Auslagerung häufig verwendeter Muster in Konstanten oder separate Klassen, wie Listing 6 verdeutlicht. Diese Beispiele zeigen, dass die Verarbeitung von Log-Dateien ein gutes Anwendungsfeld für RegExtract ist. Reguläre Ausdrücke schaffen diese Aufgaben auch, aber die Typisierung der Ergebnisse bringt Sicherheit und Komfort bei der Verarbeitung.

Fortgeschrittene Funktionen

RegExtract bietet fortgeschrittene Funktionen für komplexe Extraktionsszenarien. Die Bibliothek extrahiert Daten direkt in verschiedene Sammlungstypen, wie zum Beispiel *List<T>*.

Screenshot der gefüllten Datenklassen (Debug-Modus in JetBrains Rider) (Bild 1)

Bei der Verarbeitung mehrerer Eingabezeichenfolgen bietet RegExtract die Ergebnismenge auf Basis von *IEnumerable<string>* für die Batch-Verarbeitung. *Extract* holt hier die Daten aus einer Sequenz von Texten und gibt eine Liste von Ergebnissen zurück. Listing 4 zeigt dazu ein Beispiel.

Das Array definiert mehrere Zeilen, die vom RegEx-Muster erfasst werden. Da das Ergebnis aus einer Menge besteht,

Listing 5: Anwendungsbeispiel zur Extraktion von Log-Daten

```
public class LogEntry
{
    public DateTime Timestamp { get; set; }
    public string Level { get; set; }
    public string Message { get; set; }
    public string Source { get; set; }
}

// Komplexes Muster mit benannten Gruppen

var pattern4 = @"(?<Timestamp>\d{4}-\d{2}-\d{2} " +
    "\d{2}:\d{2}:\d{2})\s+[(?<Level>\w+)\s]" +
    "\s+(?<Source>[\w\.\s]+):\s+(?<Message>.+>";
const string log = @"2024-01-15 10:30:45 [ERROR] " +
    "App.Service: Database connection failed";
var entry = log.Extract<LogEntry>(pattern4);

Console.WriteLine(entry);
```

● Listing 6: Eine Klasse mit vorbereiteten RegEx-Mustern

```
public static class Patterns
{
    public const string DateTime =
        @"(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})";
    public const string Time =
        @"(?<hour>\d{2}):(?<minute>\d{2})" +
        ":(?<second>\d{2})";

    public static string BuildLogPattern(
        string messagePattern) =>
        $"{DateTime}\s+{Time}\s+{messagePattern}";
}
```

Listing 7 zeigt dazu den entsprechenden Code. Das Ergebnis ist eine gefüllte Liste vom Typ *List<string>* mit den drei Einträgen *red*, *green* und *blue*.

Bei nullable-Typen erkennt RegExtract optional fehlende Werte und setzt diese auf null. In Kombination mit dem Operator „|“ für alternative reguläre Ausdrücke lässt sich das Ergebnis in jeweils unterschiedliche Typen extrahieren – basierend auf dem konkreten RegEx-Match. Die nachfolgende Zeile C#-Code verdeutlicht das:

```
var result2 =
    "Zeichenkette".Extract<(int?, string)>(@"(\d+)|(.*)");
```

In diesem Beispiel ist in der Variablen *result2* das Tupel (*Zeichenkette*) gespeichert. Der *int*-Wert ist optional und in der Eingabezeichenkette nicht vorhanden. Durch den alternativen Operator wird die Zeichenkette trotzdem gematcht und im Ergebnistupel gespeichert.

Verschachtelte Typen und Sammlungen erlauben die Modellierung komplexer Datenstrukturen. Listing 8 zeigt ein einfaches Beispiel basierend auf einem ValueTuple in Kombination mit einem HashSet. Das Ergebnis besteht aus dem ersten Element, in diesem Fall die Nummer 1, und anschließend dem gefüllten HashSet mit allen nachfolgenden Werten.

In Kombination mit benutzerdefinierten Datenklassen lassen sich so direkt komplexe Datenstrukturen aus RegEx-Matches herauslösen und typisiert weiterverarbeiten.

Diese Verschachtelung von Daten funktioniert nicht nur bei Tupeln und Collections, sondern auch mit Collections, die ein Tupel enthalten, einer Collection mit weiteren Collections oder mit einer Collection und mehreren Argumenten, also einem Dictionary.

Das gilt ebenso für die Kombination von Records und anderen, benutzerdefinierten Typen, die sich nahtlos integrieren lassen. Die Kombination mit LINQ ermöglicht darüber hinaus eine komplexe Datenverarbeitung, wenn LINQ für das

Filtern von Eingabedaten oder Ausgabewerten zum Einsatz kommt. Listing 9 zeigt ein Beispiel mit verschachtelten Records, die aus einer in der Zeichenkette codierten Liste gefüllt werden.

Zusätzlich zu den zahlreichen Tupeln, benutzerdefinierten Typen und Collections, wie oben beschrieben, unterstützt RegExtract alle primitiven Typen von .NET sowie jede Enum und jeden Typ, der eine statische *Parse(string)*-Methode implementiert. Darüber hinaus unterstützt das Projekt jeden Typ mit einem öffentlichen Konstruktor, der eine einzelne

● Listing 7: Direkte Extraktion in einen Listen-Typ

```
// Extraktion in List<T>
const string text4 = "red,green,blue";
const string pattern5 = @"((\w+),?)+";
var tags = text4.Extract<List<string>>(pattern5);

Console.WriteLine(tags);
```

Zeichenkette als einzigen Parameter akzeptiert. Das erlaubt eine sehr breite Nutzung und eine nahezu beliebige Kombinationsmöglichkeit von Mustern zu Datentypen.

Über Templates lassen sich vorgefertigte RegEx-Muster einbinden und nutzen, wie das Listing 10 zeigt. Das Beispiel definiert eine Datenklasse für das RegEx-Resultat und ein Muster, in diesem Fall einen regulären Ausdruck für eine E-Mail. Dieses Template ist direkt in der Datenklasse über die Eigenschaft *REGEXTRACT_REGEX_PATTERN* integriert.

Zusätzlich besteht die Möglichkeit, RegEx-Optionen über eine benannte Eigenschaft *REGEXTRACT_REGEX_OPTIONS* einzubinden, die in diesem Beispiel auf *RegexOptions.None* ►

● Listing 8: Extraktion von Daten in eine verschachtelte Datenstruktur

```
const string text5 = "Item #1: 27 61 49 58 44 2 " +
    "69 78";
const string pattern6 = @"Item #(\d+): (\d+ ?)+";

var result3 =
    text5.Extract<(int itemno, HashSet<int> set)>
    (pattern6);
```

gesetzt ist. Das Beispiel zeigt den Vorteil, dass sich darüber sowohl das Muster einbinden als auch die extrahierten Daten speichern lassen, was ein gewisses Level an Modularisierung und dadurch bessere Wartbarkeit erlaubt.

und auf jeden Fall eine Empfehlung, einen Blick auf das Projekt zu werfen.

Etwas unsicherer scheint aktuell die Weiterentwicklung zu sein. Der letzte Release ist mehr als ein Jahr her, was erst ein-

Listing 9: Komplexe Extraktion von Daten in eine verschachtelte Datenstruktur

```
record BagDescription(string name)
{
    public List<IncludedBags> contents {get;set;}
};

record IncludedBags
{
    public int? num { get; set; }
    public string type { get; set; }
}
```

```
const string text6 = "faded yellow bags contain 4 " +
    "mirrored fuchsia bags, 4 dotted indigo bags, " +
    "3 faded orange bags, 5 plaid crimson bags.";
const string pattern7 = @"^(.+) bags " +
    "contain(<contents> (?<num>\d+) (?<type>.*?) " +
    "bags?[.])+$";
var result4 = text6.Extract<BagDescription>(pattern7);
```

Listing 10: Datenklasse mit integriertem RegEx-Template und -Optionen

```
public class WithEmailTemplate
{
    public const string REGEXTRACT_REGEX_PATTERN =
        @"(?<Email>[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+)" +
        "\.[a-zA-Z]{2,})";

    public const RegexOptions REGEXTRACT_REGEX_OPTIONS =
        RegexOptions.None;
}
```

```
public string Email { get; set; }

const string text7 = "Bitte kontaktieren Sie uns " +
    "unter support@example.com für weitere " +
    "Informationen.";
// Email-Pattern importieren und anwenden
var email = text7.Extract<WithEmailTemplate>();
```

Fazit

RegExtract ist eine innovative .NET-Bibliothek, die reguläre Ausdrücke mit typsicherer Datenextraktion kombiniert. Sie reduziert die Komplexität traditioneller RegEx-Verwendungen, indem sie Capture Groups automatisch in typisierte Objekte umwandelt.

Die Bibliothek verbessert die Lesbarkeit, Wartbarkeit und Fehlersicherheit des Codes durch Compile-Zeit-Validierung und nahtlose Integration in bestehende Projekte. Entwicklern bietet sie eine effiziente Lösung für Datenextraktion, etwa aus Log-Dateien oder strukturierten Texten.

Mit den Eigenschaften der einfachen Installation, ihren flexiblen Einsatzmöglichkeiten und einer umfangreichen Unterstützung für komplexe Datenstrukturen ist RegExtract ein sehr vielseitiges Werkzeug für prägnante und sichere Textextraktionen.

Die Bibliothek macht einen rundum guten Eindruck. Die Dokumentation bei GitHub ist für den Einstieg mehr als ausreichend.

Die Beispiele sind schnell verständlich und unterstützen den zügigen und direkten Einstieg in die Bibliothek. Insgesamt verdient sich RegExtract als Bewertung ein „Sehr gut“

mal nicht weiter schlimm ist. Eine Roadmap gibt es, und eine Weiterentwicklung scheint zumindest geplant zu sein. Wie es weitergeht, muss also die Zeit zeigen.

Wünschenswert ist eine weitere Entwicklung und Pflege, da der Einsatz der Bibliothek viel Spaß bereitet und in vielen Anwendungsfällen sinnvoll ist. ■

[1] Repository von RegExtract auf GitHub,
www.dotnetpro.de/SL2503Frameworks1

[2] Das NuGet-Paket von RegExtract,
www.dotnetpro.de/SL2503Frameworks2



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter bei der Sportnavi.de GmbH. Darüber hinaus ist er Gründer im Bereich der digitalen Bildung sowie als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig. Sie erreichen ihn über deitelhoff.me oder fabian@deitelhoff.me.

dnPCode

A2503Frameworks



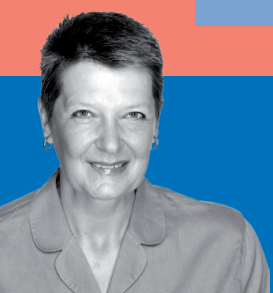
HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN



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)

**REMOTE
ODER
INHOUSE**



Weitere Informationen auf developer-academy.de
Preise auf Anfrage!

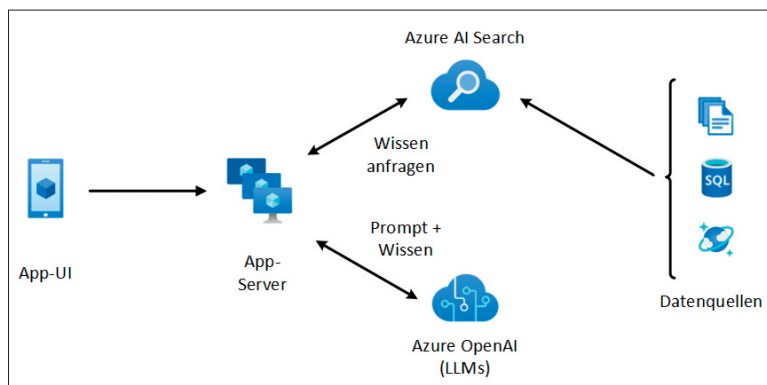
Ihre Ansprechpartnerin: Susanne Herl • +49 731 88005-8835 • susanne.herl@ebnermedia.de

KI UND DATA, TEIL 5

RAG-Time bei Microsoft

Systeme auf Basis von Retrieval-Augmented Generation erfreuen sich großer Beliebtheit. Wie sie sich in der Microsoft-Welt umsetzen lassen, erfahren Sie in diesem Artikel.

Retrieval-Augmented Generation (RAG) ist eine wichtige Komponente im Kontext der aktuellen Nutzung generativer KI. Das Microsoft-Ökosystem mit M365, Azure und .NET ist hier keine Ausnahme. Aufbauend auf dem vorangegangenen Artikel [1], der die Grundlagen von RAG und Fine-Tuning beleuchtete, widmen wir uns nun den spezifischen Vorteilen von RAG-Systemen in KI-Anwendungen. Wie in [1] beschrieben, entfällt mit RAG die Notwendigkeit, Modelle ständig neu trainieren zu müssen. Stattdessen greift das System dynamisch auf relevante Informationen zurück und liefert kontextgenaue Antworten. Dies ist insbesondere in Unternehmensanwendungen mit sich schnell ändernden Informationslandschaften von Vorteil.



Beispiel-Architektur für ein einfaches RAG-System (Bild 1)

Der Fokus dieses Artikels liegt auf den Einsatzmöglichkeiten von RAG-Systemen in der Microsoft-Welt. Im Mittelpunkt stehen technische Details, Implementierungen und Werkzeuge wie Semantic Kernel und Azure AI Services. Der Beitrag richtet sich an Entwickler und Entscheider, die praktische Anwendungen von RAG verstehen und nutzen möchten. Er basiert auf einem realen Use Case, der in abstrakterer Form vorgestellt wird. Die Implementierung ist testweise umgesetzt und aktuell in der Evaluationsphase.

Zunächst eine kurze Beschreibung des verwendeten Use Case: Ein Unternehmen benötigt ein intelligentes FAQ-System für seine Mitarbeiter. Das System soll Informationen zu internen Workflows und Unternehmensprozessen sowie Software-Anleitungen bereitstellen. Die bestehende Wissensbasis umfasst im ersten Schritt 500 Dokumente in verschiedenen Formaten wie PDF, Word und HTML-Seiten. Dieses System

soll dazu beitragen, dass die Mitarbeiter täglich weniger Zeit bei der Suche nach relevanten Informationen verlieren und die IT-Abteilung weniger wiederkehrende Anfragen zu Standardprozessen bearbeiten muss. Sobald konkrete Ergebnisse aus internen Tests vorliegen, werden diese gern in einem weiteren Artikel nachgereicht und beschrieben, was gut und was weniger gut funktioniert hat.

Das Microsoft-Ökosystem als Grundlage für RAG-Systeme

Das Microsoft-Ökosystem bietet mit Azure OpenAI, Azure AI Search und verschiedenen Datenspeicherlösungen die technische Basis für RAG-Systeme. Azure OpenAI stellt leistungsstarke Sprachmodelle bereit, während Azure AI Search Dokumente indiziert und relevante Inhalte auf Basis semantischer Analysen liefert. Als Datenspeicher kommen Azure SQL Database für strukturierte Daten, Fabric SQL Database für Analytics-Szenarien und Azure Blob Storage für unstrukturierte Dokumente zum Einsatz.

Zentraler Bestandteil vieler RAG-Systeme ist der Semantic Kernel oder artverwandte Lösungen, die den Informationsfluss steuern, Benutzeranfragen mit Kontext anreichern und die Kommunikation zwischen den Azure-Diensten orchestrieren. Der Semantic Kernel übernimmt Funktionen wie Prompting, Speicherverwaltung und Pipeline-Steuerung. Azure AI Search identifiziert relevante Dokumente durch Vektorrepräsentationen und leitet diese als Kontext für die Antwortgenerierung an das Sprachmodell weiter. Eine häufige Architektur solcher Systeme zeigt Bild 1.

Die OpenAI-Dienste generieren daraufhin kohärente und relevante Antworten. Der Semantic Kernel sorgt für die Erstellung der Prompts sowie die Nachbearbeitung der Ergebnisse. Ergänzend speichert das System den Kontextverlauf, um die Konsistenz bei wiederholten Interaktionen zu gewährleisten.

Die Architektur ermöglicht die nahtlose Integration von KI-Funktionen in Microsoft-Produkte wie Office, Teams und Visual Studio Code, etwa durch die Copilot-Funktionen. Mit Copilot Studio können Entwickler Plug-ins, Prompts und Datenquellen definieren sowie RAG-Anwendungen testen, überwachen und iterieren. Dass eine Implementierung aber nicht zwangsläufig Frameworks wie Semantic Kernel nutzen muss, ist ebenfalls Ziel dieses Artikels. ►

● Listing 1: Implementierung des FAQ-Systems (Ausschnitt)

```
public class FAQSystem
{
    private readonly IKernel _kernel;
    private readonly SearchClient _searchClient;

    public FAQSystem(IKernel kernel, SearchClient
        searchClient)
    {
        _kernel = kernel;
        _searchClient = searchClient;
    }

    public async Task<string> ProcessQuery(
        string userQuery)
    {
        // Vektorisierung der Benutzeranfrage
        var queryEmbedding = await
            _kernel.Memory.GenerateEmbeddingAsync(userQuery);

        // Suche nach relevanten Dokumenten
        var searchOptions = new SearchOptions
        {
            VectorSearchConfiguration =
                new VectorSearchConfiguration
                {
                    Vectors = { new VectorSearchField(
                        "embedding", queryEmbedding) },
                    KNearestNeighborsCount = 3
                }
        };

        var searchResults =
            await _searchClient.SearchAsync<FAQDocument>(
                null, searchOptions);

        // Kontext für das Sprachmodell aufbauen
        var context = new StringBuilder();
        await foreach (var result
            in searchResults.Value.GetResultsAsync())
        {
            context.AppendLine(result.Document.Content);
        }

        // Antwort generieren
        var prompt = $"Basierend auf folgenden
            Informationen: {context}\n\nBeantworte die
            Frage: {userQuery}";
        var completion = await _kernel.InvokeAsync(
            "CompletePrompt", prompt);

        return completion.ToString();
    }
}
```

● Grounded versus Ungrounded Prompts

Eine häufige Herausforderung bei der Implementierung von Sprachmodellen über einen Chat ist die sogenannte Erdung, die sich darauf bezieht, ob eine Antwort in der Realität oder einem bestimmten Kontext verwurzelt, verbunden oder verankert ist. Mit anderen Worten gibt die „Bodenständigkeit“ der Antwort eines Sprachmodells Auskunft darüber, ob diese Antwort auf faktischen Informationen beruht.

Die Grundlage für RAG-Systeme ist die Notwendigkeit, fundierte Antworten liefern zu können. Kommt ein Sprachmodell zum Einsatz, um eine Antwort auf eine Eingabeaufforderung zu generieren, stammen die einzigen Informationen, auf die das Modell die Antwort stützen kann, aus den Daten, auf denen es trainiert wurde – und das sind oft nur große Mengen an kontextlosem Text aus dem Internet oder einer anderen Quelle. Das Ergebnis wird wahrscheinlich eine grammatikalisch kohärente und logische Antwort auf die Aufforderung sein, aber da sie nicht auf relevanten, faktischen Daten beruht, ist sie nicht kontextbezogen und kann sogar ungenau sein und erfundene Informationen enthalten. Die Frage „Welches Produkt sollte ich für X verwenden?“ könnte zum Beispiel Details zu einem fiktiven Produkt enthalten.

Das macht diese Art der Informationserzeugung ungenau oder gar unnütz im Kontext von wissensbasierten Systemen.

RAG-Systeme verbessern die Antwortqualität, indem sie es ermöglichen, auf zusätzliche Datenquellen zuzugreifen, um die Aufforderung mit einem relevanten Kontext zu unterlegen. Die Aufforderung lässt sich dann an ein Sprachmodell weiterleiten. Dieses Modell enthält die zugrunde liegenden Daten, um eine kontextbezogene, relevante und genaue Antwort zu generieren. Die Datenquelle kann ein beliebiges Repository mit relevanten Daten sein. Beispielsweise lassen sich Daten aus einer Produktkatalogdatenbank verwenden, um die Frage „Welches Produkt soll ich für X verwenden?“ so zu gestalten, dass die Antwort relevante Details zu den im Katalog vorhandenen Produkten enthält.

Das ist der Unterschied zwischen „ungrounded“ und „grounded“ Prompts; übersetzbar mit „verankert“ beziehungsweise „nicht verankert“, bezogen auf die konkrete Datengrundlage wie Daten, Fakten oder Kontextinformationen. Häufig werden auch die Begriffe „kontextbezogene“ und „kontextlose“ Prompts verwendet, um deutlich zu machen, dass der Kontext mit weiteren Daten wichtig für die Antwortqualität eines LLM ist.

Erster Test: Semantic Kernel als Orchestrator

Die Integration von Azure AI Search [2] und Semantic Kernel [3] erfolgt über eine C#-Anwendung. Der Code in Listing 1 zeigt die grundlegende Struktur. Die Klasse *FAQSystem* verarbeitet Benutzeranfragen zu sportbezogenen Themen mithilfe von Vektorrepräsentationen, semantischer Suche und KI-gestützter Textgenerierung. Sie umfasst zwei Hauptkomponenten: den Kernel für KI-Operationen wie Vektoreinbettungen und Textgenerierung sowie den Suchclient, der auf eine Dokumentendatenbank zugreift.

Der Konstruktor der Klasse erhält den Kernel und den Client als Abhängigkeiten, damit diese in den Methoden verwendet werden können. Die zentrale Methode verarbeitet Benutzeranfragen in mehreren Schritten:

- **Der Kernel** wandelt die Anfrage in eine Vektordarstellung um, die die semantische Bedeutung numerisch repräsentiert.
- **Der Suchclient** identifiziert die drei am besten passenden Dokumente in der Datenbank basierend auf den Vektorrepräsentationen.
- **Der Code** sammelt die Inhalte der gefundenen Dokumente und fasst diese zu einem Text zusammen, der als Kontext für die Antwortgenerierung dient.
- **Der Kernel** erstellt mithilfe dieses Kontexts und der ursprünglichen Anfrage eine passende Antwort.

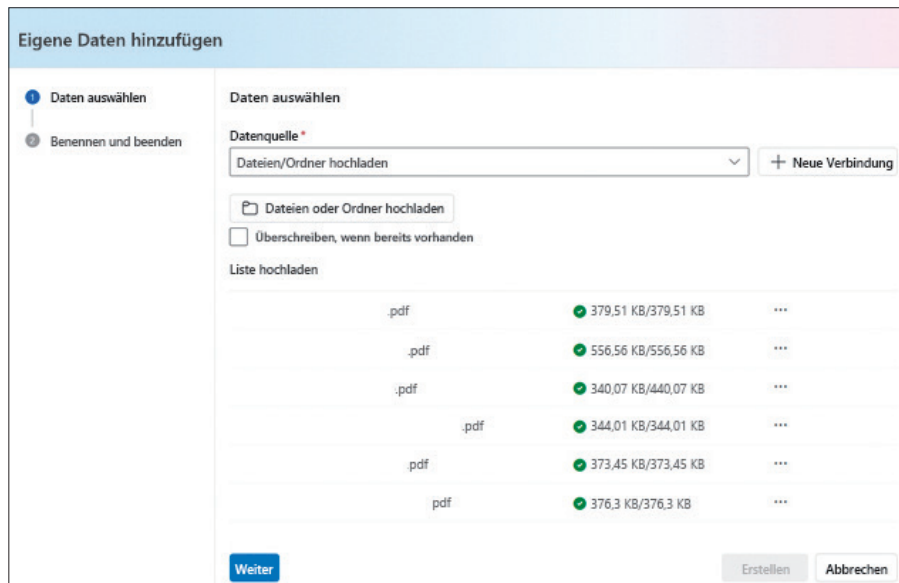
Das System liefert Ergebnisse, die auch bei nicht exakt passenden Suchbegriffen relevant bleiben, da die Vektorisierung semantische Ähnlichkeiten berücksichtigt. Es kombiniert KI-gestützte Suche und Textgenerierung zu einer effizienten Lösung für komplexe Spracheingaben, etwa in Wissensdatenbanken oder sportbezogenen FAQ-Systemen.

In ersten Tests hat diese Implementierung gut funktioniert. Der Prototyp diente als erste testweise Evaluation und war mehr für das Ausprobieren im IT-Team gedacht. Produktiv kam diese erste Iterationsstufe nicht zum Einsatz.

Zweiter Test: RAG-System mit Azure und .NET als nächste Iterationsstufe

Die Implementierung eines RAG-Systems mit Microsoft Azure und .NET bietet eine leistungsfähige Lösung für die Verarbeitung und Generierung von Informationen. Dieses System kombiniert die Stärken von Azure AI Search als robuste Such-Engine und Wissensspeicher mit den fortschrittlichen Sprachmodellen des Azure OpenAI Service, orchestriert durch eine .NET-Anwendung. Dieses Mal nicht auf Basis des Frameworks Semantic Kernel oder einer Alternative.

Der Prozess beginnt mit der sorgfältigen Aufbereitung und Indizierung der Daten. Zunächst extrahiert ein Verarbeitungsschritt alle relevanten Informationen aus verschiedenen



Der Upload von Daten in ein Azure-AI-Foundry-Projekt (Bild 2)

Quellen und unterteilt diese in kleinere, semantisch sinnvolle Abschnitte. Ein Vorgang, der als Chunking bezeichnet wird und im vorangegangenen Teil [1] dieser Serie bereits erläutert wurde. Diese Chunks indiziert dann Azure AI Search, wobei Vektor-Embeddings zum Einsatz kommen, die eine effiziente semantische Suche ermöglichen.

Bei einer Benutzeranfrage durchläuft das System mehrere Schritte, um eine präzise und kontextrelevante Antwort zu generieren. Der Code wandelt die Anfrage zunächst in einen Vektor um, der dann für eine Vektorähnlichkeitssuche in Azure AI Search zum Einsatz kommt. Die relevantesten Suchergebnisse werden als Kontext zusammengestellt und in einen strukturierten Prompt integriert. Diesen Prompt, der sowohl die ursprüngliche Anfrage als auch den generierten Kontext enthält, schickt der Code dann an das LLM des Azure OpenAI Service, um eine fundierte Antwort zu erzeugen. Die Integration dieses Systems in eine .NET-Anwendung erfolgt durch die Verwendung der entsprechenden Azure-SDKs. Listing 2 zeigt dazu eine Implementierung in C#.

In diesem Codebeispiel werden zunächst die notwendigen Clients für Azure AI Search und Azure OpenAI initialisiert. Die Benutzeranfrage wird dann für eine semantische Suche verwendet, wobei die *VectorSearchOptions* konfiguriert werden, um die drei ähnlichsten Dokumente zu finden. Die Suchergebnisse werden zu einem Kontext zusammengefasst, der zusammen mit der ursprünglichen Anfrage in einen Prompt integriert wird. Das Sprachmodell erhält dann diesen Prompt, um eine Antwort zu generieren, die schließlich ausgegeben wird.

Die Stärke dieses Ansatzes liegt in der Kombination von präziser Informationsabfrage durch Azure AI Search und der kontextbezogenen Sprachgenerierung durch Azure OpenAI. Dies ermöglicht es dem System, auf eine breite Wissensbasis zuzugreifen und gleichzeitig natürliche, kontextrelevante Antworten zu generieren. Die Verwendung von .NET als Entwicklungsplattform bietet dabei eine robuste und flexible Grundlage für die Implementierung und Skalierung des RAG-Systems.

Durch die Nutzung der Azure-Dienste profitiert das System von der Skalierbarkeit und Zuverlässigkeit der Cloud-Infrastruktur. Dies ermöglicht es Entwicklern, sich auf die Optimierung der Anwendungslogik und die Verfeinerung der Benutzererfahrung zu konzentrieren, während die zugrunde liegende Infrastruktur von Microsoft verwaltet wird. Die kontinuierliche Weiterentwicklung der Azure-Dienste und der .NET-Plattform eröffnet zudem ständig neue Möglichkeiten zur Verbesserung und Erweiterung des RAG-Systems, was es zu einer zukunftssicheren Lösung für anspruchsvolle Informationsverarbeitungs- und Generierungsaufgaben macht.

Schritt 1: Vorbereitung einer RAG-basierten Lösung mit Azure AI Foundry

Um sicherzustellen, dass sich ein Agent auf bestimmte Daten stützt, um genaue und domänenspezifische Antworten zu geben, können wir Retrieval-Augmented Generation (RAG) verwenden. Hier kommt das Abrufen von Erdrungsdaten auf der Grundlage der vom Benutzer eingegebenen Eingabeaufforderung zum Einsatz. Anschließend reichert das System die Eingabeaufforderung mit Daten aus der zusätzlichen Datenquelle an. Jetzt kommt erneut ein Sprachmodell zum Einsatz, um eine fundierte Antwort zu erzeugen. Indem der Kontext aus einer bestimmten Datenquelle stammt, ist sichergestellt, dass das Sprachmodell relevante Informationen verwendet, wenn es antwortet, statt sich ausschließlich auf seine Trainingsdaten zu verlassen.

Um diese Datenquelle hinzuzufügen, eignen sich verschiedene Datenverbindungen, die wir im Kontext eines Azure-

AI-Projekts nutzen. Wir können Azure AI Foundry [3] verwenden, um einen benutzerdefinierten Copiloten zu erstellen, der eigene Daten nutzt, um Prompts zu grounden. Azure AI Foundry unterstützt dazu eine Reihe von Datenverbindungen einschließlich Azure Blob Storage, Azure Data Lake Storage Gen2 und Microsoft OneLake. Alternativ lassen sich Dateien oder Ordner manuell in das AI-Foundry-Projekt hochladen, wie es [Bild 2](#) zeigt.

Schritt 2: Daten durchsuchbar gestalten

Wenn wir einen Agenten erstellen möchten, der eigene Daten nutzt, um präzise Antworten zu generieren, müssen diese Daten effizient durchsuchbar sein. Durch die Nutzung von Azure AI Foundry lässt sich eine Integration mit Azure AI Search einsetzen, um den relevanten Kontext im Chatverlauf abzurufen. Azure AI Search ist ein Retriever, den wir bei der Erstellung einer Anwendung auf Basis eines Sprachmodells mit Prompt Flow einbeziehen können. Azure AI Search ermöglicht es, eigene Daten einzubringen, diese Daten zu indizieren und den Index abzufragen, um alle benötigten Informationen abzurufen. Weitere gängige Lösungen beinhalten LangChain, um den Workflow zu koordinieren. LangChain lässt sich in Azure AI Search integrieren, was es einfacher macht, Azure AI Search als Retriever in einen Workflow einzubinden. Llama-Index und Semantic Kernel sind weitere Optionen.

Während ein textbasierter Index die Suche effizienz verbessert, lässt sich in der Regel eine bessere Lösung für die Datenabfrage erzielen, indem ein vektorbasierter Index zum Einsatz kommt, der Embeddings (Einbettungen) enthält, ►

Listing 2: Beispiel einer RAG-Implementierung in .NET (C#)

```
using Azure;
using Azure.AI.OpenAI;
using Azure.Search.Documents;
using Azure.Search.Documents.Models;

// Azure AI Search-Client initialisieren
var searchClient = new SearchClient(new Uri(
    "https://your-search-service.search.windows.net"),
    "your-index-name", new AzureKeyCredential(
    "your-api-key"));

// Azure OpenAI-Client initialisieren
var openAIClient = new OpenAIClient(new Uri(
    "https://your-openai-resource.openai.azure.com"),
    new AzureKeyCredential("your-api-key"));

// Benutzeranfrage verarbeiten
string userQuery = "Wie funktioniert RAG?";

// Semantische Suche durchführen
var searchOptions = new SearchOptions {
    VectorSearch = new VectorSearchOptions {

        Queries = { new VectorQuery(userQuery) {
            KNearestNeighborsCount = 3 } }
    };
};

SearchResults<SearchDocument> results =
    searchClient.Search<SearchDocument>(
        userQuery, searchOptions);

// Kontext aus Suchergebnissen erstellen
string context = string.Join("\n",
    results.GetResults().Select(r =>
        r.Document["content"].ToString()));

// Prompt erstellen und an LLM senden
string prompt = $"Basierend auf folgendem Kontext,
    beantworte die Frage: {userQuery}\n\nKontext:
    {context}";

Response<Completions> response = await openAIClient.
    GetCompletionsAsync("your-deployment-name", prompt);

// Antwort ausgeben
Console.WriteLine(response.Value.Choices[0].Text);
```


die die Text-Tokens in einer Datenquelle darstellen. Eine Einbettung ist ein spezielles Format der Datendarstellung, das eine Suchmaschine verwenden kann, um die relevanten Informationen leicht zu finden. Genauer gesagt ist ein Embedding ein Vektor aus Gleitkommazahlen.

Der Abstand zwischen Vektoren lässt sich durch Messung des Kosinus des Winkels zwischen zwei Vektoren berechnen, auch bekannt als Kosinusähnlichkeit. Mit anderen Worten, die Kosinusähnlichkeit berechnet die semantische Ähnlichkeit zwischen Dokumenten und einer Abfrage. Durch die Darstellung von Wörtern und ihren Bedeutungen mit Vektoren lässt sich der relevante Kontext aus einer Datenquelle extrahieren, selbst wenn die Daten in unterschiedlichen Formaten, wie Text oder Bild, und in verschiedenen Sprachen gespeichert sind. Wenn die Vektorsuche für die Suche in Daten zum Einsatz kommen soll, müssen wir bei der Erstellung des Suchindex Embeddings erstellen. Hierzu können wir ein Azure-OpenAI-Modell für Embeddings verwenden, das in Azure AI Foundry verfügbar ist.

In Azure AI Search beschreibt ein Suchindex, wie Inhalte organisiert sind, um sie durchsuchbar zu machen. Das ist vergleichbar mit einer Bibliothek, bestehend aus vielen Büchern. Wir möchten in der Lage sein, die Bibliothek zu durchsuchen

Konfiguration eines Index für die Suche in Daten (Bild 3)

und das gewünschte Buch einfach und effizient zu finden. Um die Bibliothek durchsuchbar zu machen, erstellen wir einen Katalog, der alle relevanten Daten zu den Büchern enthält, damit jedes Buch leicht zu finden ist. Der Katalog einer Bibliothek dient als Suchindex.

Es gibt zwar verschiedene Ansätze zur Erstellung eines Index, aber die Integration von Azure AI Search in Azure AI Foundry macht es recht leicht, einen Index zu erstellen, der für Sprachmodelle geeignet ist. Wir können Daten zu Azure AI Foundry hinzufügen und dann Azure AI Search verwenden, um einen Index im Azure-AI-Foundry-Portal mithilfe eines Einbettungsmodells zu erstellen. Das Index-Asset ist in Azure AI Search gespeichert. Azure AI Foundry fragt den Index ab, wenn es in einem Chatverlauf zum Einsatz kommt. Bild 3 zeigt, wie ein solcher Index auf Basis von Embeddings in Azure AI Foundry erzeugt wird.

● Listing 3: Beispiel für modularisierte Skills

```
public class SoftwareInstallationSkill
{
    [SKFunction("Anleitung zur Software-Installation")]
    public async Task<string> GetInstallationGuide(
        [SKParameter("Name der Software")] string
        softwareName)
    {
        var documents = await
            _searchClient.SearchAsync<FAQDocument>(
                $"software:{softwareName} AND
                type:installation");

        return FormatInstallationSteps(documents);
    }

    [SKFunction("Lizenzschlüssel-Verwaltung")]
    public async Task<string> GetLicenseInformation(
        [SKParameter("Software-ID")] string softwareId)
    {
        var licenseInfo = await
            _licenseService.GetLicenseDetails(softwareId);
        return FormatLicenseResponse(licenseInfo);
    }
}
```

Schritt 3: Einen Agenten über Flows erstellen

Nach dem Hochladen von Daten in Azure AI Foundry und dem Erstellen eines Index für die Daten mithilfe der Integration mit Azure AI Search lässt sich das RAG-Muster mit Prompt Flow implementieren, um eine generative KI-Anwendung zu erstellen. Prompt Flow ist ein Entwicklungsframework zur Definition von Flows, die Interaktionen mit einem LLM orchestrieren.

Ein Ablauf beginnt mit einer oder mehreren Eingaben, in der Regel einer Frage oder einem Prompt, die/der von einem Benutzer eingegeben wird, und im Fall von iterativen Unterhaltungen mit dem Chatverlauf bis zu diesem Punkt. Der Ablauf wird dann als eine Reihe von verbundenen Tools definiert, von denen jedes eine bestimmte Operation an den Eingaben und anderen Umgebungsvariablen durchführt. Es gibt mehrere Arten von Werkzeugen, die wir in einen Prompt Flow einbinden können, um Aufgaben auszuführen, wie zum Beispiel das Ausführen von benutzerdefiniertem Python-Code, das Nachschlagen von Datenwerten in einem Index, das Erstellen von Prompt-Varianten, um damit mehrere Versionen eines Prompts für ein großes Sprachmodell (LLM) zu definieren, und das Übermitteln eines Prompts an ein LLM, um Ergebnisse zu generieren.

Der Kern in einem solchen RAG-Muster für einen Prompt Flow ist es, das Index-Lookup-Werkzeug [4] zu nutzen, um

Daten aus einem Index abzufragen. Diese Daten lassen sich dann im weiteren Verlauf des Prompts nutzen, beispielsweise, um ein Ergebnis durch ein Sprachmodell zu erzeugen. Für diese Anwendungsfälle bietet Azure AI Foundry zahlreiche Beispiel-Flows an, um einen Chat-Flow inklusive Lookup zu implementieren. Diese Beispiele beinhalten bereits alle notwendigen Schritte und Werkzeuge, um einen Lookup für ein RAG-System zu implementieren. **Bild 4** zeigt diesen Flow.

Zunächst verarbeitet ein LLM den Chatverlauf und die letzte Nutzerfrage, um daraus eine präzisierte Frage mit allen relevanten Informationen zu generieren. Dies sorgt für eine kompakte Eingabe für den weiteren Ablauf. Mit dem Index-Lookup-Tool werden relevante Daten aus dem integrierten Azure-AI-Search-Index abgefragt, um die benötigten Informationen zu finden. Der Output des Index-Lookup-Tools wird als Kontext aufbereitet, der in einem Prompt für das LLM verwendet wird. Dazu können Python-Nodes genutzt werden, um die Inhalte mehrerer Dokumente zusammenzuführen und in ein passendes Format umzuwandeln.

Um die Antworten des Chatbots stärker auf Fakten zu stützen, lassen sich unterschiedliche Varianten des Prompts erstellen. Diese enthalten verschiedene Systemnachrichten, um die Relevanz und Präzision der Antworten zu testen. Abschließend erhält das LLM den generierten Prompt, um auf Basis des abgerufenen Kontexts eine Antwort zu erstellen. Diese Antwort bildet das Ergebnis des gesamten Flows. Der konfigurierte Ablauf lässt sich in Anwendungen integrieren, um eine interaktive Nutzererfahrung zu ermöglichen.

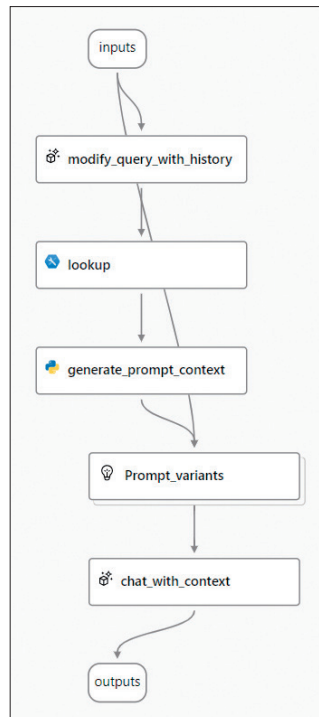
Hiermit steht nun der zweite Prototyp. Ersten Auswertungen zufolge ist die Antwortqualität mindestens gut, sodass weitere Tests erfolgen. Ob eine konkrete Integration in den Workflow einer Abteilung erfolgt, wird sich in den nächsten Wochen zeigen.

Best Practices

Eine modulare Struktur der Semantic-Kernel-Skills verbessert die Wartbarkeit. Jeder Skill fokussiert sich auf eine spezifische Aufgabe und folgt dem Single-Responsibility-Prinzip. Die Skills kommunizieren über definierte Schnittstellen, wodurch sich Komponenten einfach austauschen lassen. **Listing 3** zeigt dazu als Beispiel zwei Semantic-Kernel-Funktionen, um eine Modularisierung zu erreichen.

Die Performance-Optimierung in Azure beginnt bei der Indizierung. Ein ausgewogenes Chunking der Dokumente reduziert Speicherbedarf und Abfragezeiten. Das Caching häufiger Anfragen senkt die Azure-OpenAI-Kosten. Die Skalierung der Cognitive-Search-Instanzen orientiert sich an den tatsächlichen Lastspitzen.

Ein mehrstufiges Monitoring erfasst System-Metriken und Nutzerinteraktionen. Azure Monitor protokolliert technische KPIs wie Antwortzeiten und Ressourcenauslastung. Applica-



Ein Beispiel für einen Prompt-Flow mit RAG-Pattern (**Bild 4**)

tion Insights analysiert das Nutzerverhalten und identifiziert problematische Anfragemuster. Die gesammelten Daten fließen in die kontinuierliche Optimierung der Prompts und Retrievalstrategien ein.

Fazit und Ausblick

RAG-Systeme zeichnen sich durch ihre Flexibilität und Skalierbarkeit aus. Technologien wie Semantic Kernel, vektorbasierte Indexe und graphbasierte Ansätze unterstützen Wissensmanagement und Automatisierung. Die Anbindung an Microsoft-Produkte wie Office und Teams erleichtert den praktischen Einsatz. Die Umsetzung von RAG-Systemen mit Microsoft-Technologien zeigt, wie KI-Methoden in Unternehmenslösungen integriert werden können. Azure AI Search, Azure

OpenAI und verwandte Dienste ermöglichen semantische Suche und kontextualisierte Antwortgenerierung.

Erste interne Tests mit den Prototypen bestätigen die Anpassungsfähigkeit von RAG-Systemen, sei es mit Frameworks wie Semantic Kernel oder eigenständigen .NET-Lösungen. Die Weiterentwicklung von Azure-Diensten bietet zudem Raum für Optimierungen.

Wie gut sich das System in der Praxis macht, wird das erste Quartal des Jahres 2025 zeigen. Geplant sind hierzu weitere Tests, technische Optimierungen und weitere Integrationen. Beispielsweise in eigene Webanwendungen, andere Anwendungen wie Salesforce oder Microsoft Teams sowie der schrittweise Ausbau der Datenbasis auf deutlich mehr Dokumente. ■

[1] Fabian Deitelhoff, *Anpassen oder erweitern?*, dotnetpro 2/2025, Seite 84 ff., www.dotnetpro.de/A2502KIundData

[2] Webseite zu Azure AI Search, www.dotnetpro.de/SL2503KIundData1

[3] Übersicht zum Semantic Kernel, www.dotnetpro.de/SL2503KIundData2

[4] Webseite zu Azure AI Foundry, <https://ai.azure.com/>

[5] Weiteres Wissen zum Index-Lookup-Tool, www.dotnetpro.de/SL2503KIundData3



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter bei der Sportnavi.de GmbH. Darüber hinaus ist er Gründer im Bereich der digitalen Bildung sowie als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig. Sie erreichen ihn über deitelhoff.me oder fabian@deitelhoff.me.

dnpCode

A2503DDD

ENTWICKLUNG DER STELLEN FÜR ENTWICKLER

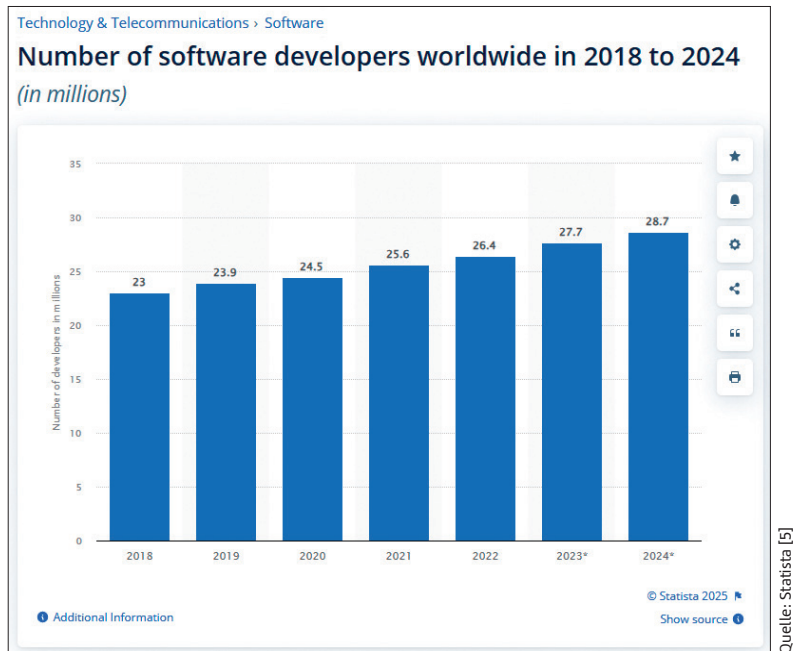
Übernimmt KI deinen Job?

Widersprüchliche Signale und Zahlen kommen zur Zukunft für Softwareentwickler aus dem Markt. Eine Einordnung.

Softwareentwickler:innen hassen stupide Tätigkeiten. Statt in einer halben Stunde 30 Daten einzeln per Hand zu importieren, schreiben sie dafür lieber eine Routine – in vier Stunden. Okay, na gut, das mag ein Klischee sein, aber etwas Wahres ist sicher dran.

Völlig klar, dass KI-Tools deshalb bei Entwickler:innen hoch im Kurs stehen. Die genannte Routine ist mit KI-Hilfe in zwei Minuten fertig. Wenn nur nicht das längliche Prompt-Eintippen wäre – aber das ist eine andere Geschichte.

Ebner-Media-Group-Geschäftsführer Marco Parillo prognostizierte kurz nach Erscheinen von ChatGPT dessen Auswirkungen auf die Branche der Softwareentwicklung so: Softwareentwickler:innen würden zum einen die Ersten sein, die KI-Tools exzessiv in ihrem Job einsetzen, aber zum anderen auch diejenigen, deren Arbeitsplätze am meisten durch den KI-Fortschritt gefährdet sein würden. War seine Prognose damals übertrieben?



Entwicklung der Anzahl der Softwareentwickler:innen weltweit (Bild 1)

Zahlengrundlage

Wie eingangs schon beschrieben scheint die erste Aussage unstrittig: Entwickler:innen sind Technologie-affin und scheuen den Einsatz neuer Tools nicht. Zudem erfahren sie durch die KI in großem Maße Unterstützung.

Ein Teilnehmer der .NET Developer Conference 2024 [1] brachte es so auf den Punkt: „Endlich gibt es die KI-Tools, die bei der Softwareentwicklung helfen. Die riesige Menge an APIs und Klassen wäre anders künftig nicht mehr zu managen.“

Hat die Nutzung von KI-Tools in der Softwareentwicklung aber auch Auswirkungen auf den Jobmarkt für Softwareentwickler:innen, und wenn ja, welche?

Folgt man der Analyse des World Economic Forum (WEF), sieht die Zukunft für Softwareentwickler:innen unverändert rosig aus, denn laut der WEF-Umfrage [2] nach den künftig fragtesten Berufsbildern bis 2030 finden sich die „Software and applications developers“ auf Platz drei.

Vor dem Hintergrund des zunehmenden Einflusses der Large Language Models erscheint die Prognose des World Economic Forum für die Nachfrage nach Softwareentwickler:innen allerdings überraschend. Haben die Befragten diese Entwicklung etwa übersehen? Denn wenn die Produktivität dank Tools wie GitHub Copilot oder ChatGPT wesentlich steigt, wäre eine schwindende Nachfrage im Hinblick auf die Zahl der an den Projekten beteiligten Entwickler:innen zu erwarten.

Einschätzung der Experten im Markt

Im Vorgespräch zur Panel-Diskussion „Mein Kopilot treibt mich in den Wahnsinn“ auf der .NET Developer Conference Ende November 2024 verriet Dr. Damir Dobric, CEO und Lead Architect der Daenet GmbH, dass er mithilfe von KI-Tools innerhalb von nur vier Tagen ein Proof of Concept umgesetzt habe, für das er früher zwei bis drei Wochen veranschlagt hätte. Allerdings bedürfe es eines Senior Engineers, der den Entwicklungsprozess aus dem Effeff kenne und KI-Tools optimal einsetzen könne. „Solche Charaktere zaubert aber noch keine KI herbei“, meinte er grinsend.

Nun könnte man einwenden, dass Entwickler:innen vielleicht doch nicht so Technologie-neugierig sind oder KI-Tools gar nicht einsetzen wollen. Auch könnten Firmen-Policies den Einsatz womöglich verhindern.

Dem scheint der Trend beim Frage-Antwort-Portal Stackoverflow zu widersprechen: Dort sinkt die Anzahl an eingestellten Fragen kontinuierlich und in großem Ausmaß [3]. Das dürfte zwar nicht ausschließlich KI-Tools wie GitHub Copilot, ChatGPT und Co. anzulasten sein. Aber die unmittelbare und schnelle Hilfe im passenden Kontext dürfte KI-Tools in vielen Fällen zum wesentlich nützlicheren Hilfsmittel machen als die Antworten in Stackoverflow.

Dazu passt die Aussage von Marc Benioff, einem der Gründer und CEO des CRM-Marktführers Salesforce, wonach die Produktivität bei der Softwareentwicklung dank der eigenen KI „Agent Force“ und anderer KI-Tools um 30 Prozent gestiegen sei [4]. Folgerichtig will Salesforce im Jahr 2025 auf jeden Fall keine neuen Entwickler:innen einstellen.

Im Gegenteil soll es dort vielmehr einen Stellenabbau geben: Die Zahl der Support Engineers würde reduziert, kündigte Benioff an, denn der „Agentic Layer“ würde viele ihrer Aufgaben übernehmen. Im Sales-Bereich hingegen würde die Mannschaft aufgestockt: „And then we will have less support engineers next year (2025) because we have an agentic layer. We will have more salespeople next year because we need to really explain to everybody exactly the value that we can achieve with AI so we will add another probably 1 to 2000 salespeople in the short term.“

Weltweit ist die Anzahl der Softwareentwickler:innen laut einer Statista-Erhebung in den vergangenen Jahren kontinuierlich gestiegen: von 23 Millionen im Jahr 2018 auf prognostiziert 28,7 Millionen im Jahr 2024 (Bild 1) [5]. In Deutschland allerdings ist ganz aktuell ein gegenläufiger Trend zu verzeichnen, nämlich ein Minderangebot an freien Stellen für Entwickler:innen. So schreibt der Dekra-Arbeitsmarktreport 2024 zu den in Stellenanzeigen nachgefragten Jobprofilen: „Softwareentwickler:innen sind zum ersten Mal seit 2010 nicht mehr auf einem der vorderen zehn Plätze vertreten (Platz 19).“ Hier spielt natürlich nicht nur die KI-Entwicklung hinein, sondern auch die schwächelnde Wirtschaft mit damit verbundenen nicht stattfindenden Investitionen und Softwareprojekten.

Dualität der Trends in der Softwareentwicklung

Hat Marco Parrillo nun recht behalten? Fakt ist: Es gibt zwei gegenläufige Bewegungen. Zum einen ist da der immer größere Bedarf an Software aufgrund der digitalen Transformation in jedem Bereich.

Wer meint, ein Bereich würde von passender Hardware mit passender Software nicht ergänzt beziehungsweise verändert, der irrt. Zu einfach sind die Möglichkeiten, durch Softwareprodukte auch beim Kunden nachzubessern und durch entsprechende Abonnements die Kundenbindung zu stärken. Der Ausdruck „Bananensoftware, die beim Kunden reift“ erfährt dadurch eine neue Bedeutung.

Zum anderen hilft die KI bei der Softwareentwicklung und erhöht die Produktivität um ein Vielfaches. Das benötigt aber Entwickler:innen, die in der Lage sind, alle Hilfsmittel und Tools bestmöglich einzusetzen.

Insofern gehört die Zukunft Mensch und Maschine gemeinsam – denjenigen Entwickelnden, die die Anforderungen verstehen und mit viel Erfahrung sowie den passenden KI-Tools Software schneller und besser umsetzen können, als das vielleicht ein klassisches Team von Entwickler:innen könnte.

Auch die Gefahr eines Developer-Lock-ins ist dank der KI nicht mehr so groß: Der Ausfall einer Entwicklerin oder eines Entwicklers ist keine Katastrophe mehr, wenn er oder sie eine Code-Basis alleine gepflegt hat. Die KI kann das Projekt einlesen und neuen Entwickler:innen erklären.

Ob die eine oder die andere Bewegung die Nase vorn haben wird, muss sich zeigen. Drei Aspekte gelten auf jeden Fall:

- Das Wissen und Können der Softwareentwickler:innen wird weiterhin unbedingt benötigt. Für die Übersetzung von Anforderungen in ein fertiges Softwareprodukt ist, von einfachen Projekten abgesehen, deren Expertise immer noch erforderlich. Freilich werden die Tools immer besser, und sie werden künftig mehr und mehr Aufgaben wie das Aufsetzen der Architektur übernehmen können. Im Endeffekt muss das fertige Produkt aber funktionieren und gewartet werden. Ob jemand ohne jedes Entwicklungswissen das nur mit der Unterstützung durch KI leisten kann, ist stark zu bezweifeln.
- Es braucht gut ausgebildete Entwickler:innen. Der Job des „Code Monkeys“, der bislang einfache Entwicklungsarbeiten übernommen hat, wird aussterben. Fortbildung wird aufgrund der zunehmenden Verwendung von KI-Tools noch viel wichtiger, und lebenslanges Lernen gilt nun auch besonders für den effektiven Einsatz von KI-Tools. Helfen können hierbei Veranstaltungen wie die Developer Week.
- Um am Arbeitsmarkt bestehen zu können oder ein höheres Gehalt erzielen zu können, gehört die Expertise „Kann mit KI umgehen“ bereits heute unbedingt dazu. Fachinformatiker:innen, Programmierer:innen und Informatikstudent:innen sollten unbedingt schon während der Ausbildung mit KI-Tools arbeiten und lernen, sie im Entwicklungsprozess einzusetzen.

Fazit

Der Leitgedanke zum Abschluss soll in Abwandlung des Spruchs von Focus-Ex-Chefredakteur Helmut Markwort formuliert werden. Statt „Fakten, Fakten, Fakten. Und immer an den Leser denken“ heißt es für Softwareentwickler: „Lernen, lernen, lernen. Und immer die KI verwenden.“ ■

- [1] .NET Developer Conference 2024, www.dotnet-developer-conference.de
- [2] World Economic Forum, Umfrage, www.dotnetpro.de/SL2503KIundDev1
- [3] Stackoverflow, www.dotnetpro.de/SL2503KIundDev2
- [4] Salesforce, Produktivität bei der Softwareentwicklung dank der eigenen KI „Agent Force“ und anderer KI-Tools um 30 Prozent gestiegen, www.dotnetpro.de/SL2503KIundDev3
- [5] Statista, Anzahl der Softwareentwickler:innen weltweit, www.dotnetpro.de/SL2503KIundDev4



Tilman Börner

war 20 Jahre Chefredakteur der dotnetpro. Heute unterstützt er das Team von Developer Education bei der Ebner Media Group. Daneben bringt er den Kolleginnen und Kollegen den Einsatz der KI nahe.

tilman.boerner@ebnermedia.de

dnpCode

A2503KIundDev



DATA ACCESS

Die Datenzugriffsberatung mit DR. HOLGER SCHWICHTENBERG

WEITERE NEUERUNGEN IN SYSTEM.TEXT.JSON 9.0

Nachgelegt

Microsoft hat in System.Text.Json 9.0 zwischen dem Release Candidate vom September 2024 und der final veröffentlichten Version noch einige weitere Neuerungen ergänzt.

In der Datenzugriffskolumne der dotnetpro-Ausgabe 12/2024 berichtete ich, dass Microsoft in der Version 9.0 von System.Text.Json erst vier neue Features eingebaut hat:

- Einrückungen anpassen mit *IndentCharacter* und *IndentSize*,
- Einstellungen wie bei Web-APIs mit *JsonSerializerOptions.Web*,
- Berücksichtigung von Nullable-Kontext und *[DisallowNull]*,
- JSON-Schema exportieren mit *JsonSchemaExporter*.

Diese Aussagen basierten auf dem Preview-Stand vom 10. September 2024. Bis zur Release-Version am 12. November 2024 hat Microsoft noch einiges nachgelegt.

Berücksichtigung verpflichtender Konstruktorparameter

Neben der Option *RespectNullableAnnotations = true* bietet System.Text.Json in Version 9.0 eine weitere optionale Regelverschärfung: *RespectRequiredConstructorParameters =*

true sorgt dafür, dass beim Deserialisieren alle verpflichtenden Konstruktorparameter befüllt werden. Den Einsatz zeigt Listing 1.

Genau wie die Option *RespectNullableAnnotations* kann man auch dieses Prüf-Feature in der Projektdatei global einschalten:

```
<ItemGroup>
  <RuntimeHostConfigurationOption
    Include="System.Text.Json.Serialization.
      RespectRequiredConstructorParametersDefault"
    Value="true" />
</ItemGroup>
```

Microsoft empfiehlt im Dev-Blog unter [1], diese Option für neue Projekte zu nutzen. Zur Vermeidung eines Breaking Change hat Microsoft an dieser Stelle darauf verzichtet, den Standard zu ändern. ►

● Listing 1: Einsatz von *RespectRequiredConstructorParameters*

```
public void JSON_RequiredConstructorParameters()
{
    JsonSerializerOptions options = new() {
        RespectRequiredConstructorParameters = true };

    string json = """
        {"Name":"Dr. Holger Schwichtenberg"}
        """; // ID ist Pflicht im Konstruktor, fehlt aber
           // hier im JSON

    CUI.H1("Deserialisieren von einem Objekt mit
        verpflichtenden Konstruktorparametern");
    try
    {
        // Laufzeitfehler: "was missing required
        // properties including: 'ID'."
        Entwickler e = JsonSerializer.Deserialize<
            Entwickler>(json, options);
        Console.WriteLine(e);
    }
    catch (Exception ex)
    {
        CUI.Error(ex);
    }
}
```



● Listing 2: Einsatz von [JsonStringEnumMemberName]

```
[JsonConverter(typeof(JsonStringEnumConverter))]
enum Note
{
    [JsonStringEnumMemberName("Sehr gut")] // NEU!
    SehrGut = 1,
    Gut = 2,
    Befriedigend = 3,
    Ausreichend = 4,
    [JsonStringEnumMemberName("Nicht bestanden")]
    // NEU!
    Mangelhaft = 5,
}

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

    // Serialisierung
    CUI.H2("Serialisierung");
    string json1 =
        JsonSerializer.Serialize(Note.SehrGut);
    Console.WriteLine(json1); // "Sehr Gut"

    string json2 =
        JsonSerializer.Serialize(Note.Gut);
    Console.WriteLine(json2); // "Gut"

    string json3 =
        JsonSerializer.Serialize(Note.Mangelhaft);
    Console.WriteLine(json3); // "Nicht bestanden"

    // Deserialisierung
    CUI.H2("Deserialisierung");
    Note note1 =
        JsonSerializer.Deserialize<Note>(json1);
    Console.WriteLine(note1.ToString()); // SehrGut

    Note note2 =
        JsonSerializer.Deserialize<Note>(json2);
    Console.WriteLine(note2.ToString()); // Gut

    Note note3 =
        JsonSerializer.Deserialize<Note>(json3);
    Console.WriteLine(note3.ToString()); // Mangelhaft

    try
    {
        string json4 = @""Mangelhaft"";
        // Das geht nicht mehr, weil
        // [JsonStringEnumMemberName] verwendet wird!
        Note note4 =
            JsonSerializer.Deserialize<Note>(json4);
        Console.WriteLine(note4.ToString());
    }
    catch (Exception ex)
    {
        CUI.Error(ex); // The JSON value could not be
        // converted to
        // NET9_Console.FCL90.FCL9_JSON+Note
    }
}
```

```
JSON_EnumNames
Serialisierung
"Sehr gut"
"Gut"
"Nicht bestanden"
Deserialisierung
SehrGut
Gut
Mangelhaft
System.Text.Json.JsonException: The JSON value could not be converted to NET9_Console.FCL90.FCL9_JSON+Note.
```

Ausgabe von Listing 2 (Bild 1)

● Listing 3: Einsatz von AllowOutOfOrderMetadataProperties=true

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

    // Erstelle Objekt
    var consultant = new Consultant()
    {
        ID = 42,
        FullName = "Holger Schwichtenberg",
        Salutation = "Dr.",
        PersonalWebsite = "www.dotnet-doktor.de"
    };
    consultant.Languages = ["C#", "Visual Basic .NET",
        "JavaScript", "TypeScript"];

    // Serialisierungsoptionen festlegen
    var options = new JsonSerializerOptions
    {
        // ReferenceHandler = ReferenceHandler.Preserve,
        WriteIndented = true,
        IndentCharacter = ' ', // NEU! --> ACHTUNG: Als
        // IndentCharacter werden aber nur ein Leerzeichen
        // ' ' oder ein Tabulator '\t' unterstützt!!!
        IndentSize = 4,
    };

    // Serialisieren in JSON
    CUI.H2("Serialisieren --> $type am Anfang");

    var json = JsonSerializer.Serialize<Person>(
        consultant, options);
    Console.WriteLine(json);

    // Ausgabe:
    // {
    //   "$type": "Consultant",
    //   "Languages": [
    //     "C#",
    //     "Visual Basic .NET",
    //     "JavaScript",
    //     "TypeScript"
    //   ],
    //   "PersonalWebsite": "www.dotnet-doktor.de",
    //   "ID": 42,
    //   "FullName": "Holger Schwichtenberg",
    //   "Salutation": "Dr.",
    //   "Address": null
    // }

    // Deserialisieren aus JSON
    CUI.H2("Deserialisieren mit $type am Anfang");
    var obj = JsonSerializer.Deserialize<Person>(json);
    Console.WriteLine(obj); // Consultant!

    string json2 = ""
    {
        "Languages": [
```

JSON_OutOfOrderMetadataProperties

Serialisieren --> \$type am Anfang

```
{
  "$type": "Consultant",
  "Languages": [
    "C#",
    "Visual Basic .NET",
    "JavaScript",
    "TypeScript"
  ],
  "PersonalWebsite": "www.dotnet-doktor.de",
  "ID": 42,
  "FullName": "Holger Schwichtenberg",
  "Salutation": "Dr.",
  "Address": {
    "Country": null,
    "City": "Essen"
  }
}
```

Deserialisieren mit \$type am Anfang

Consultant 42: Dr. Holger Schwichtenberg wohnt in Essen (www.dotnet-doktor.de)
-> Sprachen: C#+Visual Basic .NET+JavaScript+TypeScript

Deserialisieren mit \$type am Ende

Consultant 42: Dr. Holger Schwichtenberg wohnt in Essen (www.dotnet-doktor.de)
-> Sprachen: C#+Visual Basic .NET+JavaScript+TypeScript

Ausgabe
von Listing 3
(Bild 2)

```

        "C#",
        "Visual Basic .NET",
        "JavaScript",
        "TypeScript"
    ],
    "PersonalWebsite": "www.dotnet-doktor.de",
    "ID": 42,
    "FullName": "Holger Schwichtenberg",
    "Salutation": "Dr.",
    "Address": null,
    "$type": "Consultant"
}
"";

// Deserialisieren aus JSON
// Führt ohne AllowOutOfOrderMetadataProperties=
// true zum Fehler: 'The metadata property is
// either not supported by the type or is not
// the first property in the deserialized JSON
// object
CUI.H2("Deserialisieren mit $type am Ende");
JsonSerializerOptions options2 = new() {
    AllowOutOfOrderMetadataProperties = true };
var obj2 = JsonSerializer.Deserialize<Person>(
    json2, options2);
Console.WriteLine(obj2); // Consultant!
}

```

Anpassung der Serialisierung von Enumerationsmitgliedsnamen

System.Text.Json kann man ab Version 9.0 mit der neuen Annotation `[JsonStringEnumMemberName]` vermitteln, dass bei der Serialisierung von Namen aus Aufzählungstypen (Enumerationen) nicht der im Programmcode vergebene Name, sondern ein anderer Text genutzt werden soll. Auch bei der Deserialisierung werden diese Namen verwendet.

In dem folgenden Beispiel soll es in dem JSON-Dokument etwas abweichende Texte für zwei Noten geben: Beim Aufzählungsmitglied *SehrGut* soll es im Dokument die korrekte Schreibweise „Sehr gut“ geben, die im Programmcode nicht erlaubt wäre. Beim Aufzählungsmitglied *Mangelhaft* soll im Dokument der Text „Nicht bestanden“ stehen.

Wenn Namen mit `[JsonStringEnumMemberName]` geändert werden, werden die ursprünglichen Enumerationsmitgliedsnamen nicht mehr bei der Deserialisierung erkannt, siehe den vergeblichen Versuch in [Listing 2](#), den Text „Mangelhaft“ zu deserialisieren, und die zugehörige Ausgabe in [Bild 1](#).

Anpassen der Position von Typ-Metadaten

In System.Text.Json Version 7.0 hatte Microsoft die Unterstützung für Polymorphismus eingeführt, indem man beim Seri-

Listing 4: Einsatz von DeepEquals

```

public void JSON_DeepEquals()
{
    // Erstelle Objekt
    var consultant = new Consultant()
    {
        ID = 42,
        FullName = "Holger Schwichtenberg",
        Salutation = "Dr.",
        PersonalWebsite = "www.dotnet-doktor.de"
    };
    consultant.Languages = ["C#",
        "Visual Basic .NET", "JavaScript",
        "TypeScript"];

    // Serialisieren in JSON
    CUI.H2("Objekt serialisieren");
    var json =
        JsonSerializer.Serialize<Person>(consultant);
    Console.WriteLine(json);

    // Lade in JSElement
    JsonDocument doc1 = JsonDocument.Parse(json);
    JsonElement root1 = doc1.RootElement;

    // Lade in noch ein JSElement
    JsonDocument doc2 = JsonDocument.Parse(json);
    JsonElement root2 = doc1.RootElement;

    bool deepEqual1 =
        JsonElement.DeepEquals(root1, root2);
    Console.WriteLine(deepEqual1); // true

    CUI.H2("Geändertes Objekt serialisieren");
    consultant.Address =
        new Address() { City = "Essen" };
    var json3 =
        JsonSerializer.Serialize<Person>(consultant);
    Console.WriteLine(json3);

    // Lade geändertes Objekt in ein JSElement
    JsonDocument doc3 = JsonDocument.Parse(json3);
    JsonElement root3 = doc3.RootElement;

    bool deepEqual2 =
        JsonElement.DeepEquals(root1, root3);
    Console.WriteLine(deepEqual2); // false
}

```

alisieren einen Typnamen mit `$type` angeben kann, der dann beim Deserialisieren zur Instanziierung des korrekten Typs führt. System.Text.Json setzt `$type` immer vor die erste ►

JSON_MultiDocReader

```

Number 42
StartObject
PropertyName ID
Number 123
PropertyName FullName
String Holger Schwichtenberg
PropertyName Languages
StartArray
String C#
String Visual Basic .NET
String JavaScript
String TypeScript
EndArray
EndObject
StartArray
Number 1
Number 2
Number 3
EndArray
Null
StartObject
EndObject
StartArray
EndArray

```

● Listing 5: Beispiel für ein JSON-Dokument ohne eindeutigen Wurzelknoten

```

42
{ "ID": "123", "FullName": "Holger Schwichtenberg",
  "Languages": [
    "C#",
    "Visual Basic .NET",
    "JavaScript",
    "TypeScript"
  ]
}
[1,2,3]
null {} []

```

Ausgabe
von Listing 6
(Bild 3)

Property und erwartete bisher dies auch beim Deserialisieren genau in dieser Reihenfolge. Mit der neuen Option *AllowOutOfOrderMetadataProperties = true* kann System.Text.Json auch JSON-Dokumente deserialisieren, in denen die Position von *\$type* eine andere ist, weil andere JSON-Bibliotheken

● Listing 6: Einsatz von AllowMultipleValues beim Utf8JsonReader

```

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

    ReadOnlySpan<byte> utf8Json1 =
        """
42
{ "ID": 123, "FullName": "Holger Schwichtenberg",
  "Languages": [
    "C#",
    "Visual Basic .NET",
    "JavaScript",
    "TypeScript"
  ]
}
[1,2,3]
null {} []
""";

    JsonSerializerOptions options = new() {
        AllowMultipleValues = true;
        // NEU für Multi-Dokumente!
    };
    Utf8JsonReader reader =
        new(utf8Json1, options);

    while (reader.Read())
    {
        Console.WriteLine(reader.TokenType);

        switch (reader.TokenType)
        {
            case JsonTokenType.PropertyName:
            case JsonTokenType.String:
            {
                string? text = reader.GetString();
                Console.WriteLine(" ");
                Console.WriteLine(text);
                Console.WriteLine(text);
                break;
            }

            case JsonTokenType.Number:
            {
                int intValue = reader.GetInt32();
                Console.WriteLine(" ");
                Console.WriteLine(intValue);
                Console.WriteLine(intValue);
                break;
            }

            // ... (weitere Token-Typen)

            Console.WriteLine("\n");
            Console.WriteLine();
        }
    }
}

```

● Listing 7: Einsatz von `DeserializeAsyncEnumerable()` mit `topLevelValues = true`

```
public async Task JSON_MultiDocDeserialisierung()
{
    CUI.Demo();

    CUI.H2("Liste von Zahlen");
    ReadOnlySpan<byte> utf8Json1 = """ 1 2 3""""u8;
    using var stream1 =
        new MemoryStream(utf8Json1.ToArray());

    await foreach (int item in
        JsonSerializer.DeserializeAsyncEnumerable<int>(
            stream1, topLevelValues: true))
    {
        Console.WriteLine(item);
    }

    CUI.H2("Liste von Arrays");
    ReadOnlySpan<byte> utf8Json2 =
        """[42] [42,43] [42,43,44] [42,42,42] []""""u8;
    using var stream2 =
        new MemoryStream(utf8Json2.ToArray());

    await foreach (int[] array in
        JsonSerializer.DeserializeAsyncEnumerable<int[]>(
            stream2, topLevelValues: true))
    {
        Console.WriteLine(
            "Array mit " + array.Length + " Element(en):");
        foreach (var num in array)
        {
            CUI.LI(num);
        }
    }

    CUI.H2("Liste von Objekten");
    ReadOnlySpan<byte> utf8Json3 =
        """
        { "ID": 1, "FullName": "Holger Schwichtenberg",
          "Languages": [
            "C#",
            "Visual Basic .NET",
            "JavaScript",
            "TypeScript" ],
          "Address": { "City" : "Essen" }
        }
        { "ID": 2, "FullName": "Rainer Stropek",
          "Languages": [
            "C#",
            "Rust",
            "Go",
            "JavaScript",
            "TypeScript" ],
          "Address": { "City" : "Linz" }
        }
        """"u8;

    using var stream3 =
        new MemoryStream(utf8Json3.ToArray());

    await foreach (Consultant c in
        JsonSerializer.DeserializeAsyncEnumerable<
            Consultant>(stream3, topLevelValues: true))
    {
        Console.WriteLine(c);
        foreach (var l in c.Languages)
        {
            CUI.LI(l);
        }
    }
}
```

das Dokument erzeugt haben, siehe [Listing 3](#) und die zugehörige Ausgabe in [Bild 2](#).

Microsoft warnt davor, dass die Aktivierung von `AllowOutOfOrderMetadataProperties` zu Leistungseinbußen führen kann, da der Serialisierer mehr puffern muss, denn er muss den Typ ja kennen, bevor er das Objekt instanzieren kann.

Falls man auch den Namen `$type` ändern will oder muss: Das geht schon seit Version 7.0 mit:

```
[JsonPolymorphic(TypeDiscriminatorPropertyName = "$xyz")]
```

Neue Methode `DeepEquals()` in der Klasse `JsonElement`

Die Klasse `JsonElement` bietet eine neue Methode `DeepEquals()`, mit der man zwei Elemente und alle darunterliegenden

den Elemente (tiefer Vergleich) vergleichen kann. Den Einsatz dieser Methode zeigt [Listing 4](#). Analog gab es zuvor schon `DeepEquals()` in der Klasse `JsonNode`.

Lesen von Multi-JSON-Dokumenten

Ein JSON-Dokument benötigt normalerweise einen eindeutigen Wurzelknoten. Aber was tun, wenn es diesen nicht gibt, sondern man ein JSON-Dokument bekommt, das technisch gesehen aus mehreren JSON-Dokumenten besteht? Ein Beispiel ist in [Listing 5](#) gezeigt.

`System.Text.Json` bietet seit Version 9.0 eine Lösung an, auch solche „Multi-JSON-Dokumente“ zu lesen. Das geht zum einen im `Utf8JsonReader` mit Aktivierung der neuen Option `AllowMultipleValues = true`, siehe [Listing 6](#) und dessen Ausgabe in [Bild 3](#). ▶

JSON_MultiDocDeserialisierung

Liste von Zahlen

```
1
2
3
```

Liste von Arrays

Array mit 1 Element(en):

```
- 42
```

Array mit 2 Element(en):

```
- 42
```

```
- 43
```

Array mit 3 Element(en):

```
- 42
```

```
- 43
```

```
- 44
```

Array mit 3 Element(en):

```
- 42
```

```
- 42
```

```
- 42
```

Array mit 0 Element(en):

Liste von Objekten

Consultant 1: Holger Schwichtenberg wohnt in Essen -> Sprachen: C#+Visual Basic .NET+JavaScript+TypeScript

```
- C#
```

```
- Visual Basic .NET
```

```
- JavaScript
```

```
- TypeScript
```

Consultant 2: Rainer Stropek wohnt in Linz -> Sprachen: C#+Rust+Go+JavaScript+TypeScript

```
- C#
```

```
- Rust
```

```
- Go
```

```
- JavaScript
```

```
- TypeScript
```

Ausgabe von Listing 7
(Bild 4)

Quelle: Microsoft [2]

Method	Runtime	_value	Mean	Ratio	Allocated	Alloc Ratio
Serialize	.NET 8.0	Default	38.67 ns	1.00	24 B	1.00
Serialize	.NET 9.0	Default	27.23 ns	0.70	-	0.00
Deserialize	.NET 8.0	Default	73.86 ns	1.00	-	NA
Deserialize	.NET 9.0	Default	70.48 ns	0.95	-	NA
Serialize	.NET 8.0	Instance, NonPublic	37.60 ns	1.00	24 B	1.00
Serialize	.NET 9.0	Instance, NonPublic	26.82 ns	0.71	-	0.00
Deserialize	.NET 8.0	Instance, NonPublic	97.54 ns	1.00	-	NA
Deserialize	.NET 9.0	Instance, NonPublic	70.72 ns	0.73	-	NA

Leistungsverbesserungen in System.Text.Json 9.0 (Bild 5)

Zum anderen gibt es auch bei der Klasse *JsonSerializer* eine neue Überladung von *DeserializeAsyncEnumerable()*, bei der man das Flag *topLevelValues* auf *true* setzen kann. Das Listing 7 zeigt die Deserialisierung von Dokumenten mit Listen von Zahlen, Listen von Arrays von Zahlen und Listen von Objekten; die zugehörige Ausgabe ist in Bild 4 zu sehen.

Für das Schreiben von Multi-JSON-Dokumenten gibt es noch keine spezielle Lösung in System.Text.Json. Sie können einfach mehrere Objekte einzeln serialisieren und die entstehenden JSON-Dokumente dann zusammensetzen (zum Beispiel bei *StringBuilder* oder *Stream*).

Leistungsverbesserungen in System.Text.Json 9.0

Microsoft hat die Leistung von System.Text.Json beim Serialisieren und Deserialisieren verbessert. Die Leistungsverbesserungen der Version 9.0 von System.Text.Json sind im Rahmen eines sehr langen Dokuments unter [2] erwähnt, das alle Leistungsverbesserungen in .NET 9.0 auflistet (Bild 5).

[1] Microsoft Dev Blogs, Eirik Tsarpalis, What's new in System.Text.Json in .NET 9, www.dotnetpro.de/SL2503DataAccess1

[2] Microsoft Dev Blogs, Stephen Toub, Performance Improvements in .NET 9/ System.Text.Json 9.0, www.dotnetpro.de/SL2503DataAccess2

Dr. Holger Schwichtenberg

alias der „Dotnet-Doktor“ ist Leiter des Expertennetzwerks www.IT-Visions.de, das mit 53 renommierten Experten mittlere und große Unternehmen durch Beratungen und Schulungen sowie bei der Softwareentwicklung unterstützt. Durch seine Fachbücher und Vorträge auf Fachkonferenzen gehört er zu den bekanntesten Experten für .NET und Webtechniken in Deutschland.

www.dotnet-doktor.de

dnpCode

A2503DataAccess



HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN

Hands-on Workshops Azure

REMOTE
ODER
INHOUSE

Einführung in Azure OpenAI - Von der Datenintegration bis zur Entwicklung eigener KI-Agenten (3 Tage)

Cloud-native basierte Entwicklung mit Azure-Diensten (2 Tage)

Message-basierte Azure-Dienste (2 Tage)

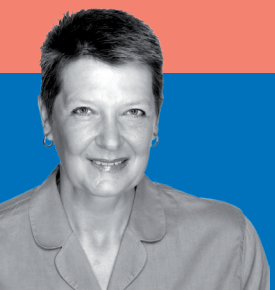
Azure: Architektur und Dienstausswahl (2 Tage)

Verteilte Anwendungen bei Azure mit Docker und Kubernetes (3 Tage)

Azure Pipelines - Multistage YAML Pipelines für Kubernetes (3 Tage)

Azure DevOps - CI/CD mit Azure Pipelines und Git (3 Tage)

Microsoft Azure DB (2 Tage)



Weitere Informationen auf developer-academy.de
Preise auf Anfrage!

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



LERNSTOFF

dnpCode A2503Lernstoff

ENTSCHEIDUNGSHILFE

Datenarchitekturen

Das enorme Wachstum der digital verfügbaren Datenmengen hat bei den zu ihrer Beherrschung angebotenen Architekturen zu zahlreichen konkurrierenden Konzepten geführt. Fehlentscheidungen erweisen sich häufig als teuer. James Serra schiebt Hypes und Marketinggetöse beiseite und stellt eine die Vor- und Nachteile abwägende Darstellung der vorhandenen Alternativen zusammen, die bei der Auswahl und bei der Einführung einer Datenarchitektur helfen soll, die dem tatsächlichen Bedarf am besten entspricht.

Zunächst werden die Eigenschaften von Big Data und die Grade ihrer Nutzung im Unternehmen unterschieden. Darauf folgt eine kurze Übersicht über die verschiedenen Arten von Architekturen, ausgehend vom relationalen Data Warehouse. Eingeschoben ist ein Vorschlag, wie die Kommunikation und die Entscheidung über die bevorzugte Architektur am besten ausgeführt wird. Die folgenden Teile behandeln die Konzepte und Datenarchitekturen im Detail: Data Warehouse, Data Fabric, Data Lakehouse sowie die Grundlagen zu Data Mesh.



James Serra

Datenarchitekturen. Modern Data Warehouse, Data Fabric, Data Lakehouse und Data Mesh richtig einsetzen

O'Reilly, 2024, 318 Seiten, Deutsch, ISBN 978-3-96009-254-4, 39,90 Euro

Zielgruppe: Datenverwaltung

Voraussetzungen: keine

Leseprobe:

www.dotnetpro.de/SL2503Lernstoff1

Zu dieser dezentralisierten Architektur gibt es noch ein zusätzliches Kapitel, das mögliche Probleme bei der Einführung benennt und passende Lösungen anbietet.

Besondere Aufmerksamkeit gilt der Einbindung der Menschen in die durch die Datenarchitektur gegebenen Form des Umgangs mit den anfallenden Daten. Dabei geht es um eine passende Ausgestaltung der verschiedenen Rollen, die belegt sein sollten. Anhand typischer Fallstricke wird demonstriert, was schiefgehen kann und was dann zu tun ist. (vh) ■

CI/CD-WORKFLOWS AUTOMATISIEREN

GitHub Actions

Hinter der Funktion GitHub Actions steckt eine leistungsstarke Workflow-Engine, mit deren Hilfe man wiederkehrende Aufgaben beim Aufsetzen von CI/CD-Workflows innerhalb des GitHub-Ökosystems automatisieren kann. Das Buch erklärt die Workflow-Syntax von GitHub Actions, die verschiedenen Arten von Aktionen und die Arbeitsweise von auf GitHub gehosteten (Linux, Windows und macOS) und selbst gehosteten Workflow-Runnern. Der Autor liefert außerdem Tipps zum Verfassen und Debuggen von GitHub-Aktionen und -Workflows mit VS Code. Er erklärt auch, wie man Aktionen lokal ausführt und mit GitHub Copilot arbeitet.

Am Beispiel realer Anwendungsfälle wird gezeigt, wie man den gesamten Release-Prozess automatisieren kann. Die Beispiele reichen vom Automatisieren von Versionshinweisen bis zum Erstellen und Testen von Software. Zudem wird die sichere Bereitstellung auf Azure, Amazon Web Services (AWS) oder Google Cloud mithilfe von OpenID Connect



Michael Kaufmann

GitHub Actions Cookbook. A practical guide to automating repetitive tasks and streamlining your development process

Packt Publishing, 2024, 250 Seiten, Englisch, ISBN 978-1-83546-894-4, 29,99 Euro, E-Book 23,99 Euro

Zielgruppe:

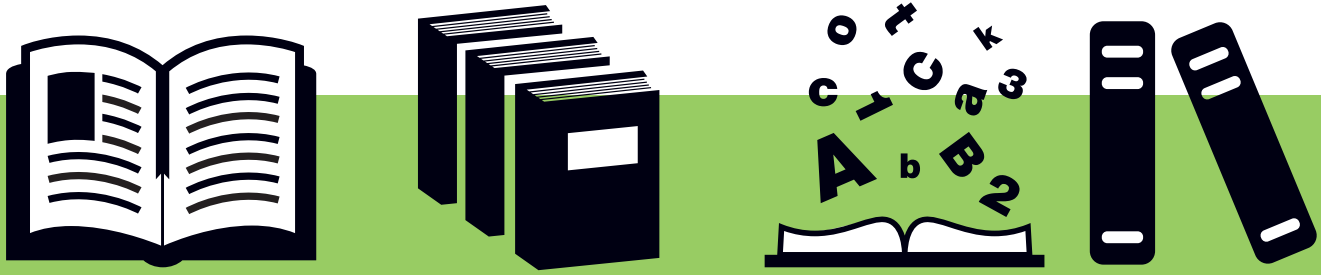
Entwickler, DevOps-Ingenieure

Voraussetzungen: Kenntnisse im Umgang mit Git und Kommandozeilen

Leseprobe:

www.dotnetpro.de/SL2503Lernstoff2

(OIDC), Secrets, Variablen, Umgebungen und Genehmigungsprüfungen behandelt. Rezepte zur Ausführung von IssueOps und zur Automatisierung anderer sich wiederholender Aufgaben werden ebenfalls vorgestellt. Dabei kommen GitHub-CLI, -APIs, -SDKs und GitHub-Tokens zum Einsatz. Der Leser erfährt, wie er eigene Aktionen und wiederverwendbare Workflows entwickeln kann. (bl) ■



IT-SICHERHEIT FÖRDERN

Awareness

Gestützt auf Erkenntnisse der Sicherheitswissenschaft wirbt Kerstin Weber für einen Paradigmenwechsel. Menschen werden nicht mehr in erster Linie als Sicherheitsfaktor eingestuft, sondern als wesentlicher Teil der Gewährleistung sicherer Abläufe. Kernanforderung dafür ist Awareness, also die Sensibilität für Vorgänge, die die IT-Sicherheit betreffen.

Solange der Mensch als größtes Sicherheitsrisiko betrachtet wird, gehen Gegenmaßnahmen in Richtung Ausschaltung des menschlichen Faktors durch Automatisierung oder Vermeidung durch starre Regelungen, die ein Mitdenken gar nicht erst aufkommen lassen. Die Alternative baut auf die menschlichen Fähigkeiten, auch in nicht eindeutigen Situationen überlegte und erfolgreiche Entscheidungen zu treffen.

Zweifellos können von Menschen Bedrohungen ausgehen, die Gründe reichen von Leichtsinn und Unwissenheit bis Börsartigkeit. Die möglichen Maßnahmen sollten sorgfältig auf die speziellen Sicherheitsbedürfnisse der Organisation abgestimmt werden, damit sie Zustimmung finden.



Kristin Weber

Mensch und Informationssicherheit.
Verhalten verstehen, Awareness fördern,
Human Hacking erkennen

Hanser, 2024, 204 Seiten, Deutsch
ISBN 978-3-446-47645-5, 34,99 Euro

Zielgruppe: Verantwortliche für
IT-Sicherheit und Führungskräfte

Voraussetzungen: Keine

Leseprobe:

www.dotnetpro.de/SL2503Lernstoff3

Ausführlich werden die verschiedenen Szenarien abgehandelt, in denen Menschen Opfer von Sicherheitsattacken werden. Die immer raffinierteren Methoden des Social Engineering werden an zahlreichen Beispielen demonstriert und Hilfen zur Erkennung gegeben. Der überwiegende Teil des Buches widmet sich den Fragen, aus welchen Elementen sich Awareness zusammensetzt, wie sie gefördert werden kann und welche Messverfahren eine gute Einschätzung erlauben, wie erfolgreich die Maßnahmen sind. (vh) ■

● Kurz vorgestellt

Mark J. Price

C# 13 and .NET 9

Die aktuelle Auflage dieses Leitfadens für .NET-Entwickler wurde für .NET 9 sowie C# 13 aktualisiert. Nach dem Standardteil zu OOP geht es um neueste Erweiterungen und Verbesserungen. Das Buch stellt auch Blazor mit seinem neuen Hosting-Modell vor.

Packt Publishing, 9. Auflage, 2024, 828 Seiten, Englisch,
ISBN 978-1-83588-123-1, 49,99 US-Dollar, E-Book 39,99 US-Dollar

Natascha Windholz

Praxishandbuch KI-VO

Das Handbuch will seine Leser praxisnah und rechtskonform auf die Anforderungen der europäischen KI-Verordnung vorbereiten. Es informiert über die Auswirkungen auf die Anwendungsbereiche von KI im privaten und öffentlichen Sektor. Mit Einordnung der Inhalte der KI-Verordnung anhand von Risikoklassen.

Hanser, 2024, 537 Seiten, Deutsch, ISBN 978-3-446-48194-7,
69,99 Euro

Roland Schwaiger, Joachim Steinwendner

Neuronale Netze programmieren mit Python

Die Autoren erklären Neuronale Netze – die Technologie hinter Deep Learning, Machine Learning und generativer KI. Anhand anschaulicher Beispiele lernen die Leser, mit Python verschiedene Netztypen zu programmieren. Inklusive Online-Lernumgebung und Einstieg in TensorFlow.

Rheinwerk, 3. Auflage 2025, 510 Seiten, Deutsch,
ISBN 978-3-367-10254-9, 34,90 Euro

Helmut Vonhoegen

Excel – Formeln und Funktionen

Dieses Nachschlagewerk bietet eine vollständige Referenz aller Formeln und Funktionen in Excel bis einschließlich Version 2024. Sämtliche mathematischen Hilfen des Tabellenkalkulationsprogramms werden anhand von Praxisbeispielen erklärt.

Rheinwerk, 8. Auflage 2025, 1008 Seiten, Deutsch,
ISBN 978-3-8421-1045-8, 24,90 Euro

POLYGLOT NOTEBOOKS IN VS CODE UND C#, TEIL 2

Nachvollziehbarer Code statt Excel-Chaos

Schnelle und professionelle Datenanalysen.

Vielleicht geht es ja nur mir so, aber Excel ist für mich ein süßes Gift. Man bekommt die Aufgabe, eine bestimmte Fragestellung zu beantworten. Der dafür benötigte Datensatz existiert, ist mir aber unbekannt. Am schnellsten beantworte ich die Frage mit ein paar Klicks in Excel. Die Daten sind leicht importiert (wenn keine datumsähnlichen Werte enthalten sind, sogar richtig) [1], und schon geht es los: Pivot-Tabellen, Filter, Formeln oder Wenn-dann-Bedingungen, um komplexere Filter umzusetzen, und nach relativ kurzer Zeit ist die Auswertung – inklusive recht ansehnlicher Grafiken – präsentierbar.

Schnell und bequem, aber keine Historie

Ab hier geht es abwärts. „Wie bist du darauf gekommen?“ „Was wäre, wenn du diese Auswertung ohne C-Kunden machen würdest?“ „Wieso hast du diesen Filter bei 80 % gesetzt und nicht auf 75 %? Wie würde das das Ergebnis verändern?“ Alles berechnete Fragen, aber leider nicht mehr so leicht zu beantworten. Entscheidungen zu bestimmten Filtern sind bestenfalls rudimentär dokumentiert, verworfene Analyse-Ideen schon gar nicht. Weil zwischen Auswertung und Präsentation einige Zeit verstrichen ist, ist das Gedächtnis auch nur noch eingeschränkt hilfreich. Neue Auswertungswünsche ziehen größere Aufwände nach sich, weil zwischen den Rohdaten und der schönen Grafik mehrere Pivot-Tabellen und Verweise liegen, die alle aktualisiert werden müssen. Es braucht viel Disziplin, sich sämtliche Erkenntnisse und Entscheidungen aufzuschreiben, und selbst dann ist das Problem mit dem Wunsch angepasster Filter weiterhin aufwendig.

Der Super-GAU

Die genannten Probleme lassen sich nach der Präsentation meist lösen. Peinlich wird es, wenn bei der Ergebnisvorstellung

```
Name, BirthDate, Department, Salary, IsManager, JoinedCompany
John Doe, 1985-04-23, HR, 50000, False, 2013-05-14
Jane Smith, 1990-07-14, IT, 72000, True, 2015-08-22
Alice Johnson, 1982-12-01, HR, 65000, True, 2011-11-29
Robert Brown, 1995-03-22, IT, 47000, False, 2018-03-17
Emily Davis, 1988-10-19, HR, 54000, False, 2012-07-10
Michael Wilson, 1979-08-07, IT, 83000, True, 2016-04-18
Jessica Moore, 1993-05-11, HR, 51000, False, 2019-09-23
Thomas Taylor, 1987-11-30, IT, 69000, True, 2014-10-05
Laura Anderson, 1983-09-15, HR, 60000, True, 2017-06-15
Daniel Thomas, 1992-02-05, IT, 46000, False, 2020-02-01
```

Die ersten zehn Beispieldaten aus Employees.csv (Bild 1)

Nur vier Datentypen beim CSV-Import

`DataFrame.LoadCsv()` identifiziert beim Import aktuell nur vier Datentypen: *String*, *Boolean*, *Single* und *DateTime*. Auch *Integer* und *Decimals* werden im Moment (Package-Version 0.22) als *Single* behandelt und als *SingleDataFrameColumn* abgelegt. Gerade beim Rechnen mit Geldbeträgen sollte man hier eine Umwandlung in *DecimalDataFrameColumn* vornehmen. Alternativ bietet es sich an, die Daten erst als *IEnumerable<T>* zu laden und dann in einen *DataFrame* umzuwandeln. Dazu habe ich im GitHub-Repo unter [6] einige Hilfsmethoden bereitgestellt.

lung Auswertungsfehler auffallen. Besonders gern passiert dies durch Ausreißer oder – noch schlimmer – Code-Werte in Zahlenfeldern. In einem Datensatz wurde zwischen „nicht verfügbar“, codiert als NULL-Wert, und „nicht bekannt“, codiert als -1, unterschieden. Wenn man Letzteres nicht berücksichtigt, können Mittelwerte ziemlich große Überraschungen bergen. Schlimmstenfalls führt dies zu Fehlentscheidungen. Ein guter Fachbereich kennt seine Zahlen und bemerkt solche Fehler oft schnell – auch kein gutes Gefühl, wenn das im Meeting vor großer Runde passiert.

Ziele des Artikels

Damit weder die kleinen noch die großen Probleme entstehen, ist es ratsam, sich eine saubere Arbeitsweise für Datenauswertungen anzugewöhnen. In der Wissenschaft haben sich Python und Jupyter Notebooks dafür etabliert. Für dotnet hat Microsoft dieses Konzept auf Polyglot Notebooks übertragen. Deren grundsätzliche Funktion wurde in einem vorangegangenen Artikel [2] bereits beschrieben.

In diesem Folgeartikel soll die Analyse von Daten mit C# an einem praktischen Beispiel ebenso vorgestellt werden wie einige Tools zur Visualisierung der Ergebnisse und Daten. Abschließend werden die Grenzen von C# für die Datenanalyse im Vergleich zu Python und R aufgezeigt.

Arbeiten mit Daten

Als Datenstruktur für Analysen ist der *DataFrame* das Mittel der Wahl. Zumindest, wenn man sich an Python und R orientiert. In dotnet gibt es diesen Datentyp als Teil von Microsofts

```

1 #r "nuget:Microsoft.Data.Analysis,0.22.0"
2 using Microsoft.Data.Analysis;
3 var df = DataFrame.LoadCsv(@"Employees.csv");
4 df.Head(5)

```

✓ 0.1s csharp - C# Script Code

Installed Packages

- Microsoft.Data.Analysis, 0.22.0

index	Name	BirthDate	Department	Salary	IsManager	JoinedCompany
0	John Doe	1985-04-23 00:00:00Z	HR	50000	False	2013-05-14 00:00:00Z
1	Jane Smith	1990-07-14 00:00:00Z	IT	72000	True	2015-08-22 00:00:00Z
2	Alice Johnson	1982-12-01 00:00:00Z	HR	65000	True	2011-11-29 00:00:00Z
3	Robert Brown	1995-03-22 00:00:00Z	IT	47000	False	2018-03-17 00:00:00Z
4	Emily Davis	1988-10-19 00:00:00Z	HR	54000	False	2012-07-10 00:00:00Z

```

1 df.Info()

```

✓ 0.1s csharp - C# Script Code

index	Info	Name	BirthDate	Department	Salary	IsManager	JoinedCompany
0	DataType	System.String	System.DateTime	System.String	System.Single	System.Boolean	System.DateTime
1	Length (excluding null values)	20	19	20	20	20	20

Der Code zum Datenimport und die Ausgabe der ersten fünf Zeilen (Bild 2)

Machine Learning Framework ML.NET [3]. Das entsprechende NuGet-Paket und auch der Namespace lauten *Microsoft.Data.Analysis*.

Der DataFrame ist eine zweidimensionale Datenstruktur mit Daten in Zeilen und Spaltenüberschriften und einem fixen Datentyp je Spalte. Im Prinzip also vergleichbar mit einem Excel-Sheet oder eine Datenbanktabelle. Fehlende Werte werden als Null-Values akzeptiert. Vorteile eines DataFrames gegenüber den gewöhnlichen Listen oder Arrays sind die einfache Erweiterbarkeit um zusätzliche Spalten, ganz wie aus Excel gewohnt, sowie die höhere Flexibilität bei der Anwendung von analytischen Funktionen. Doch dazu später mehr.

Los geht es mit dem Beispiel und der Frage, wie man sich einem wenig bekannten beziehungsweise weitgehend unbekannten Datensatz nähert. Die ersten zehn Zeilen der Daten sind in **Bild 1** dargestellt.

Das Laden in einen DataFrame erfolgt am einfachsten mittels *LoadCsv()*. Das funktioniert in den meisten Fällen problemlos. Der Datentyp je Spalte wird dabei auf Basis der ersten zehn Zeilen ermittelt, dieser Wert lässt sich aber auch anpassen, sofern es beim Import Probleme gibt. Alternativ können die Spalten aber auch explizit gemappt werden. Die API-Dokumentation ist leider noch sehr knapp, ein Blick in den Code [4] ist hier oft zielführender.

Der Code für den Import sowie die Anzeige der ersten fünf Zeilen mittels *df.Head(5)* sind in **Bild 2** dargestellt (*df.Tail(n)* liefert analog die letzten *n* Zeilen). Hier kann und sollte direkt ein erster Sichttest erfolgen, ob der Import korrekt ausgeführt wurde. Anschließend können mit *df.Info()* die Datentypen überprüft werden und auch, ob NULL-Werte vorhanden sind. Im Beispiel wurden 20 Datensätze geladen (überprüfbar mit *df.Rows.Count*). In der Spalte *BirthDate* sind nur 19 Werte enthalten, hier scheint es also einen NULL-Wert zu geben. Bemerkenswert ist, dass für *Salary* als Datentyp *Single* verwendet wurde statt *Decimal*. Für dieses Beispiel soll das nicht weiter stören, nur generell sollte man gerade bei Währungen den richtigen Datentyp verwenden. Siehe hierzu auch den Kasten **Nur vier Datentypen beim CSV-Import**.

Erkundende Datenanalyse

Nachdem die Daten sich jetzt im DataFrame im Speicher befinden, liefern deskriptive Statistiken einen ersten Eindruck über den Inhalt der Daten und zeigen die ersten Stärken des

DataFrames. **Bild 3** zeigt den Code und auch das Ergebnis der Ausführung. *df.Description()* liefert für alle numerischen Werte Minimum, Maximum und Mittelwert. Die Methode *ValueCounts()* liefert für einzelne Spalten die Häufigkeit der einzelnen Werte.

Damit können die Daten auf Plausibilität geprüft werden, und nebenbei erweitert man sein Wissen über die Daten beziehungsweise die darin enthaltenen Informationen, was mindestens genauso wichtig ist. Im Beispieldatensatz sehen wir, dass die Gehälter zwischen 45000 und 83000 Euro schwanken, sechs Mitarbeiter bei HR und 14 in der IT-Abteilung arbeiten und es insgesamt fünf Manager gibt. Durch Betrachten solcher Ergebnisse ergeben sich in der Regel weitere Fragen wie zum Beispiel:

- Existiert ein Gehaltsunterschied zwischen Managern und Nichtmanagern oder zwischen den Abteilungen?
- Wie viele Manager und Mitarbeiter (also Nichtmanager) hat welche Abteilung?

Die ersten beiden Fragen lassen sich mittels *GroupBy()* leicht beantworten. Code und Ergebnis sind in **Bild 4** dargestellt. Zuerst erfolgt eine Gruppierung nach der Spalte *IsManager*, bevor der Mittelwert über die *Salary*-Spalte berechnet wird. Aktuell ist eine Gruppierung mit dem DataFrame leider nur über eine einzelne Spalte möglich. Pivot-Tabellen lassen sich noch nicht ohne Weiteres erstellen – als Feature-Request existiert dies bereits. Das GitHub-Issue unter [5] liefert eine gute Übersicht über die noch fehlenden Funktionen.

An dieser Stelle müssen wir noch einmal über einen Elefanten im Raum sprechen: die Verwendung von Strings, ►

```

1 #r "nuget:Microsoft.Data.Analysis,0.22.0"
2 using Microsoft.Data.Analysis;
3 var df = DataFrame.LoadCsv(@"Employees.csv");

```

✓ 0.1s csharp - C# Script Code

Installed Packages

- Microsoft.Data.Analysis, 0.22.0

```

1 df.Description()

```

✓ 0.0s csharp - C# Script Code

index	Description	BirthDate	Salary	JoinedCompany
0	Length (excluding null values)	19	20	20
1	Max	<null>	83000	<null>
2	Min	<null>	45000	<null>
3	Mean	<null>	58300	<null>

```

1 df.Columns["Department"].ValueCounts().Display();
2 df.Columns["IsManager"].ValueCounts()

```

✓ 0.0s csharp - C# Script Code

index	Values	Counts
0	HR	6
1	IT	14

index	Values	Counts
0	False	15
1	True	5

Description() liefert statistische Basiswerte, **ValueCounts()** die Häufigkeiten für einzelne Spalten (**Bild 3**)

um die Spalten im DataFrame anzusprechen. Als Nutzer einer typisierten Sprache fühlt sich das wie ein Rückschritt an, weil (Tipp-)Fehler erst zur Laufzeit auffallen. Das Gute ist, dass die Laufzeit bei Polyglot Notebooks meist nur Sekundenbruchteile entfernt ist. Zum anderen ist es möglich, jederzeit neue, berechnete Spalten zu ergänzen. Hierfür müssen auch Namen vergeben werden, und da würde die Typisierung schnell an ihre Grenzen stoßen.

Für die Beantwortung der Frage zu Mitarbeitern und Managern je Abteilung ist eine neue Spalte zu berechnen. **Bild 5** zeigt den Code und die Ergebnisse. In Zeile 1 wird eine Kopie des DataFrames erzeugt. Das sollte man immer dann machen, wenn man den DataFrame manipuliert (siehe auch den Kasten **Achtung, Immutability!**). So kann man auf der Kopie arbeiten und diese bei Bedarf schnell löschen und neu erzeugen. Ansonsten läuft man Gefahr, dass man irgendwann testweise ausgeführte und wieder gelöschte Befehle vergessen hat und die Daten einen undokumentierten Änderungsstatus haben – Excel lässt grüßen.

In Zeile 3 wird eine neue Spalte erzeugt, deren Felder in den Zeilen 3 bis 7 befüllt werden, bevor die neue Spalte in Zeile 8 zum DataFrame hinzugefügt wird. Wer es kürzer mag, findet im GitHub-Repo unter [6] eine Extension-Methode, die das mittels LINQ in einer Zeile ermöglicht:

```
enhancedDf.AddColumn("IsManagerDepartment",
    row => $"{(row["Department"])} | " + row["IsManager"]});
```

Die neue Zeile ist als Zwischenergebnis in **Bild 5** für die ersten beiden Zeilen des DataFrames ausgegeben. Das Ender-

```
1 var dfManagerSalary = df.GroupBy("IsManager")
2 | .Mean(new [] {"Salary"});
3 dfManagerSalary.Display();
```

✓ 0.0s csharp - C# Script Code

index	IsManager	Salary
0	False	54466.666666666666
1	True	69800

Das Durchschnittsgehalt für Manager und Nichtmanager (**Bild 4**)

● Achtung, Immutability!

Der DataFrame beziehungsweise viele seiner Methoden setzen auf das Konzept der Immutability. Das heißt, State sollte nach Möglichkeit nicht verändert werden. Das hat zur Folge, dass einige Methoden standardmäßig einen neuen DataFrame erzeugen, statt den bestehenden DataFrame zu manipulieren. Das gilt beispielsweise auch für die Methode `DataFrame.Append()`, mit der eine weitere Zeile angefügt werden kann. Manche Methoden verfügen dann jedoch über einen optionalen Parameter `bool inplace`, der auch das Editieren des DataFrames erlaubt.

```
1 var enhancedDf = df.Clone();
2
3 var newRow = new StringDataFrameColumn("IsManagerDepartment");
4 foreach (var row in enhancedDf.Rows)
5 {
6     newRow.Append(row["Department"] + " | " + row["IsManager"]);
7 }
8 enhancedDf.Columns.Add(newRow);
9 "Zwischenergebnis:".Display();
10 enhancedDf.Head(2).Display();
11
12 var dfIsManagerDepartment = enhancedDf.GroupBy("IsManagerDepartment")
13 | .Count(new [] {"Name"});
14 "Ergebnis:".Display();
15 dfIsManagerDepartment.OrderBy("IsManagerDepartment")
```

✓ 0.1s csharp - C# Script Code

Zwischenergebnis:

index	Name	BirthDate	Department	Salary	IsManager	IsManagerDepartment
0	John Doe	1985-04-23 00:00:00Z	HR	50000	False	HR False
1	Jane Smith	1990-07-14 00:00:00Z	IT	72000	True	IT True

Ergebnis:

index	IsManagerDepartment	Name
0	HR False	4
1	HR True	2
2	IT False	11
3	IT True	3

Eine berechnete Spalte, um Kreuztabellen abzubilden (**Bild 5**)

gebnis ist ein `GroupBy()` und anschließender `Count()` über ein beliebiges Feld, das ausdrücklich keine NULL-Werte enthalten sollte. NULLs werden bei `Count()` nicht mitgezählt.

Im Beispiel in **Bild 5** erfolgt das Casting der Werte aus dem DataFrame implizit, was bei Strings in dotnet immer möglich ist. Für den Fall, dass Werte berechnet werden sollen, ist hier ein explizites Casting der Werte nötig, da die Ansprache mittels `row[Spaltenname]` immer ein Nullable-Objekt zurückliefert.

Auf der anderen Seite bieten die DataFrames aber für einfache Rechenoperationen auch die Möglichkeit, neue Spalten auf Basis bestehender Spalten zu definieren. Die Zeile

```
df["MonthlySalary"] = df["Salary"] / 12;
```

erzeugt eine neue Spalte im DataFrame, die das monatliche Gehalt als 1/12 des Jahresgehalts enthält.

Fehlerhafte oder fehlende Werte

Der Umgang mit fehlenden oder fehlerhaften Werten ist bei der Datenanalyse fast unvermeidlich. Sei es, weil ein Sensor streikt oder Informationen (noch) nicht vorhanden sind oder diese falsch erfasst wurden. Grundsätzlich hat man folgende Möglichkeiten:

1. Ausschluss der betroffenen Datensätze für einzelne oder sämtliche Analysen.
2. Überschreiben der Werte mit sinnvollen Default-Werten.
3. Nacherfassen der fehlenden Werte.

Für welche Option man sich entscheidet, hängt vom Einzelfall ab. Option 3 ist natürlich der Idealfall, oft aber unmöglich, teuer oder zeitaufwendig. Für was man sich auch entscheidet, wichtig ist eine gute Dokumentation der Entscheidung. Bei der praktischen Durchführung der ersten beiden Optionen hilft der DataFrame. Fehlende Werte können mit `df[Spalten-`

● Listing 1: Werte im DataFrame können auf Basis von Bedingungen angepasst werden

```
foreach (var row in df.Rows)
{
    if ((string) row["Department"] == "IT") {
        row["Salary"] = (float)row["Salary"] * 1.1f;
    }
}
```

```
1 var managersOnly = df["IsManager"].ElementwiseEquals(true);
2 df.Filter(managersOnly)
```

✓ 0.0s csharp - C# Script Code

index	Name	BirthDate	Department	Salary	IsManager
0	Jane Smith	1990-07-14 00:00:00Z	IT	72000	True
1	Alice Johnson	1982-12-01 00:00:00Z	HR	65000	True
2	Michael Wilson	1979-08-07 00:00:00Z	IT	83000	True
3	Thomas Taylor	1987-11-30 00:00:00Z	IT	69000	True
4	Laura Anderson	1983-09-15 00:00:00Z	HR	60000	True

Filtern von Datensätzen mittels Elementwise-Funktionen (Bild 6)

`name].FillNulls()` befüllt werden. Komplexere Regeln können angewendet werden, indem über die Zeilen iteriert wird. Wie das geht, zeigt Listing 1.

Für das Herausfiltern von Datensätzen gibt es eine Reihe von Methoden, die mit *Elementwise...*() beginnen und in der Regel auf eine Spalte angewendet werden. Die Methoden liefern selbst eine Spalte vom Typ *bool* zurück. Diese Spalte lässt als Filter auf den DataFrame anwenden, wie in Bild 6 zu sehen. Komplexere Filter, die mehrere Werte einer Zeile auswerten, können in LINQ mittels *Select()* erzeugt werden:

```
var filteredRows = df.Rows.Select(row => ((bool)
    row["IsManager"] == false && (string)
    row["Department"] == "IT"));
var filterColumn = new PrimitiveDataFrameColumn<bool>(
    columnName, filteredRows);
df.Filter(filterColumn)
```

Auf den ersten Blick wirkt das merkwürdig, weil die Filterung bei LINQ mittels *Where()* erfolgt. In diesem Fall erzeugt man

mit dem *Select()* erst ein *IEnumerable<bool>* und daraus eine neue DataFrame-Spalte, die dann als Filter angewendet wird.

Die meisten wesentlichen Funktionen des DataFrames sind damit vorgestellt. Erwähnenswert sind noch die Möglichkeiten, mehrere DataFrames zu joinen (*df.Merge()*) und einzelne Spalten mittels

```
df.Columns.Remove(Spaltenname);
```

zu entfernen. Für die Datenmanipulation einzelner Spalten bietet die *DataFrameColumn*-Klasse ebenfalls zahlreiche weitere Methoden.

Grafische Analyse

Ein erster Eindruck der Daten ist gewonnen. Die weiteren Analysen erfolgen in der Regel mittels grafischer Darstellungen. Hat man diese einmal verinnerlicht, kann man Auffälligkeiten in Diagrammen meist schnell erkennen.

Die Anzahl an Diagrammtypen und damit verbundenen Darstellungen ist schier endlos. Für die Präsentation sollte man sich auf wenige, etablierte Diagrammtypen beschränken, da man sonst Gefahr läuft, mehr Zeit mit der Erklärung von Methodik zu verbringen als mit der inhaltlichen Auseinandersetzung. Für diesen Artikel beschränken wir uns auf Boxplots und Streudiagramme mit der Bibliothek Plotly.NET [7]. Die Bibliothek bietet eine Vielzahl weiterer Diagrammtypen.

Plotly.NET setzt auf der Javascript-Library plotly.js [8] auf, die das Rendering der Diagramme übernimmt. plotly.js bezeichnet sich selbst als „data visualization library“ und ist auch die Basis für die plotly-Packages in R und Python. Plotly.NET ist in F# implementiert, was die Arbeit in C# bisweilen etwas kompliziert macht. Das Package Plotly.NET.CSharp liefert ►

● Listing 2: Datenaufbereitung und Darstellung von Alter und Abteilung in einem Boxplot

```
#r "nuget:Microsoft.Data.Analysis,0.22.0"
#r "nuget:Plotly.NET.CSharp,0.13.0"
#r "nuget:Plotly.NET.Interactive,5.0.0"

using Microsoft.Data.Analysis;
using Plotly.NET.CSharp;

var df = DataFrame.LoadCsv(@"Employees.csv");
var dfWork = df.Clone();
//Datensatz mit fehlendem Geburtstag herausfiltern

var filterColumn =
    df.Columns["BirthDate"].ElementwiseIsNotNull();
dfWork = dfWork.Filter(filterColumn);
//Geburtstag in Alter umwandeln
var age = dfWork["BirthDate"].Cast<DateTime>()
    .Select(x => DateTime.Now.Year - x.Year).ToArray();
var department =
    df["Department"].Cast<string>().ToArray();

Chart.BoxPlot<string, int, string>(department, age)
```

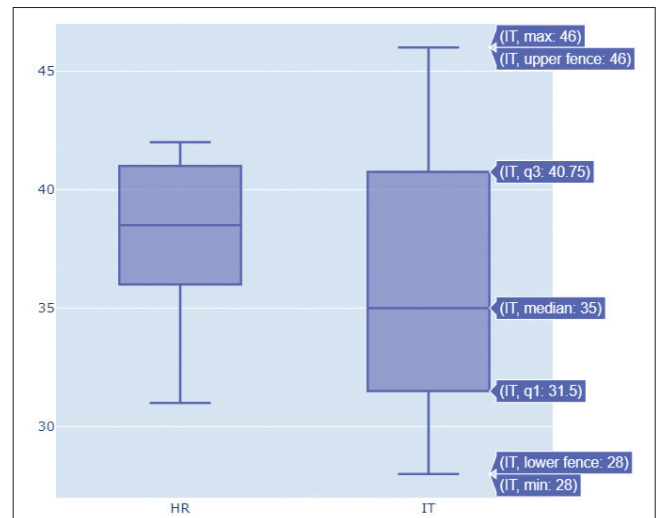
APIs, die den C#-Gewohnheiten entsprechen. Für den Einsatz mit Polyglot Notebooks ist das Package Plotly.NET.Interactive nötig. Dieses Paket stellt Formatter bereit, die die Ausgabe innerhalb von Notebooks ermöglichen.

Als erste Analyse schauen wir uns noch einmal das Alter in Abhängigkeit von der Abteilung in einem Boxplot an. Der Code steht in [Listing 2](#). Die Ausgabe ist in [Bild 7](#) zu sehen. Die Werte für die einzelnen Punkte werden nur beim Mouse-over angezeigt. Die y-Achse repräsentiert das Alter, während die x-Achse die Abteilung zeigt.

Für die IT-Abteilung liegt das mittlere 50-Prozent-Alter (zwischen dem ersten und dritten Quartil) zwischen 31,5 und 40,75 Jahren. Der Median, also das mittlere Alter, beträgt 35 Jahre. Die Alterswerte reichen von 28 Jahren (untere „Antenne“) bis 46 Jahren (obere „Antenne“). Sofern es Ausreißer gäbe, wären diese als einzelne Punkte dargestellt, die unter beziehungsweise über den Antennen liegen. Im Vergleich zur HR-Abteilung ist der Altersunterschied in der IT-Abteilung deutlich größer, das mittlere Alter aber einige Jahre niedriger. Diese Information als solche ist natürlich noch nicht viel wert, in Verbindung zum Beispiel mit dem Krankenstand oder Gehalt könnten sich hier jedoch relevante Informationen verbergen.

Der zweite Diagrammtyp ist das Streudiagramm (auch Punktwolke beziehungsweise Scatterplot). Hiermit können die Werte von zwei numerischen Variablen visualisiert werden. Code und Ergebnis sind in [Bild 8](#) dargestellt. Streudiagramme erlauben es, schnell zu erkennen, ob ein Zusammenhang zwischen zwei Variablen existiert. Im Beispiel sind dies das Gehalt auf der y-Achse und das Geburtsjahr auf der x-Achse. Ein erster Blick lässt einen Zusammenhang zwischen beiden Variablen vermuten. Wäre der Zusammenhang perfekt, könnte man eine gerade Linie (eine sogenannte Regressionsgerade) durch alle Punkte ziehen. Im Beispiel existiert ein Zusammenhang: Je jünger das Geburtsjahr, desto niedriger das Gehalt. Es gibt aber Ausnahmen in beide Richtungen. Würde man eine Regressionsgerade einzeichnen, hätten einige Punkte einen großen vertikalen Abstand zur Linie, während andere relativ dicht daran lägen.

Oft hat man es aber nicht nur mit zwei, sondern sehr vielen numerischen Variablen zu tun. In diesem Fall hilft die Scatterplot-Matrix. In der werden beliebig viele Variablen miteinander verglichen. [Bild 9](#) zeigt das Ergebnis für die drei Vari-



Das Boxplot für Alter in Abhängigkeit von der Abteilung ([Bild 7](#))

ablen *BirthDate*, *Salary* und *JoinedCompanyYear*. Heraus kommt eine 3x3-Matrix ([Bild 9](#), Code in [Listing 3](#)). In der ersten Zeile und Spalte steht jeweils die Variable *BirthDate*. Die Diagramme auf der Diagonale setzen die jeweilige Variable in Bezug zu sich selbst, hier hat man es also immer mit einer Korrelation von 1 zu tun. Interessanter sind aber die anderen Diagramme. Oben in der Mitte ist das Diagramm aus [Bild 8](#) abgetragen. Das gleiche Diagramm mit vertauschten Achsen ist in Zeile 2 ganz links. Es ergibt also Sinn, sich nur die Diagramme oberhalb beziehungsweise unterhalb der Diagonale anzusehen. Dafür verfügt *Chart.Splom()* auch über einen Parameter, der auf Wunsch die doppelten Diagramme verbirgt. Aus fachlicher Sicht erlaubt es dieser Diagrammtyp sehr schnell, mögliche Zusammenhänge zu erkennen.

Korrelation ist nicht Kausalität

Ein Wort der Warnung zum Abschluss. Die hier vorgestellten Analyse-Tools liefern bestenfalls Erkenntnisse darüber, dass ein Zusammenhang zwischen Variablen besteht. Darüber, ob dieser Zusammenhang zufällig oder systematisch ist, sagen die vorgestellten Methoden nichts aus. Hüten Sie sich hier vor voreiligen Schlüssen.

Plotly.NET ist nicht die einzige Bibliothek für Diagramme. Wem Plotly.NET nicht zusagt, der kann einen Blick auf ScottPlot, XPlotX oder SandDance [9] werfen. Besonders

Listing 3: Der Code zum Erstellen einer Scatterplot-Matrix

```
using Plotly.NET.TraceObjects;

var birthDate =
    dfWork["BirthDate"].Cast<DateTime>().ToArray();
var salary = dfWork["Salary"].Cast<float>().ToArray();
var joinedCompanyYear = dfWork["JoinedCompany"]
    .Cast<DateTime>().Select(x => x.Year).ToArray();

var splomDimension1 = Dimension.initSplom<string,
    DateTime>(Label: "BirthDate", Values: birthDate);
var splomDimension2 = Dimension.initSplom<string,
    float>(Label: "Salary", Values: salary);
var splomDimension3 = Dimension.initSplom<string,
    int>(Label: "JoinedCompanyYear", Values:
    joinedCompanyYear);

Chart.Splom<string>([splomDimension1, splomDimension2,
    splomDimension3])
```

Letzteres liefert beeindruckende Animationen auch für große Datenmengen.

Die Grenzen von C#, oder: Es geht auch eleganter

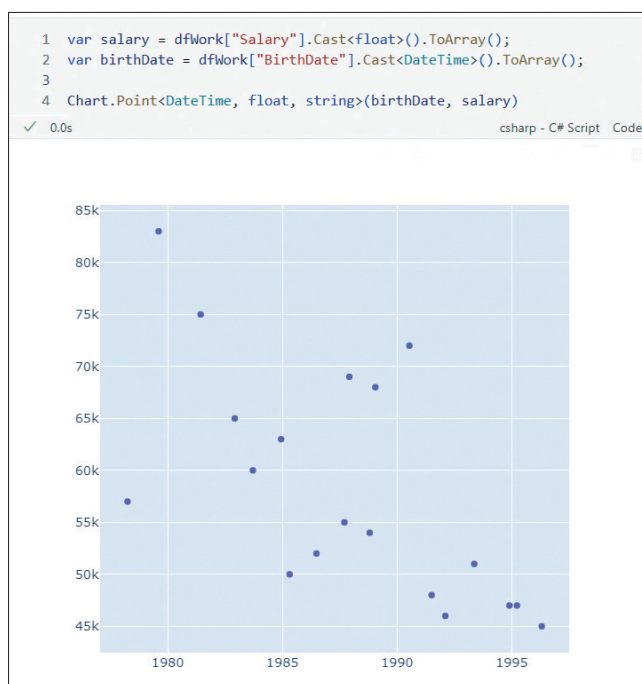
Für dotnet gibt es noch eine konkurrierende Implementierung des DataFrames im Projekt Deedle [10]. Das Projekt wurde 2014 und damit deutlich früher als ML.NET gestartet und ist stark durch F# geprägt. Mittlerweile ist dieses Projekt aber ziemlich eingeschlafen und zeigt kaum noch Aktivität auf GitHub. Wenn man zum Thema DataFrame recherchiert, muss man dennoch aufpassen, dass man den „richtigen“ DataFrame erwisch.

Weitaus häufiger wird einem in den Suchergebnissen das Wort Pandas begegnen. Pandas ist das Basispaket für Datenanalysen in Python und entsprechend omnipräsent. Bemerkbar macht sich die größere Community auch in besserem Tooling, insbesondere auch für grafische Auswertungen. Gleiches gilt für R; hier existiert mit Tidyverse [11] ein komplettes und konsistentes Toolset, das sich mit der Datenanalyse befasst.

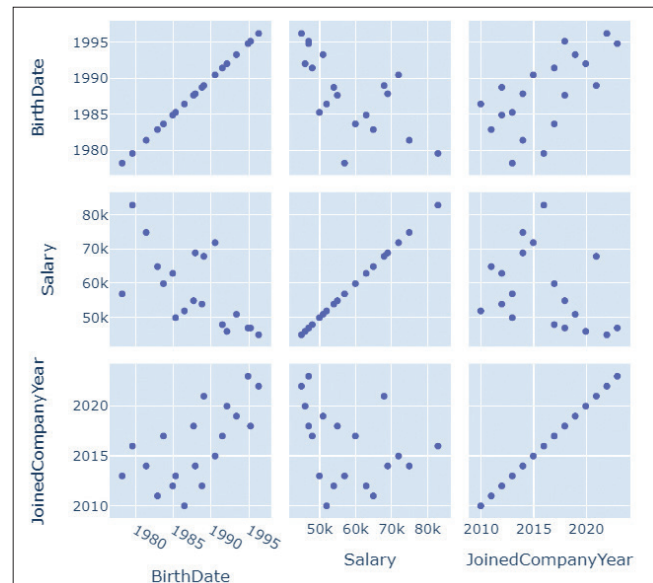
Fazit

In diesem Artikel wurden eine einfache Datenanalyse sowie die grundlegenden Funktionen des DataFrames vorgestellt. Wenn man den dotnet DataFrame mit den etablierten Werkzeugen wie Python und R vergleicht, zeigt sich, dass er noch einige Schwächen aufweist. Für den täglichen Einsatz in der Datenanalyse erzielt man mit Python und R sicherlich bessere Ergebnisse. Dennoch bietet der dotnet DataFrame in Verbindung mit Plotly.NET eine solide Grundlage für gelegentliche Datenanalysen.

Wer tiefer in das Thema Datenanalyse eintauchen möchte, dem sei das Buch „Data Science with .NET and Polyglot Notebooks“ von Matt Eland [12] wärmstens empfohlen. Auch



Streudiagramm und zugehöriger Code in Plotly.NET (Bild 8)



Eine Streudiagramm-Matrix (Scatterplot-Matrix) zeigt auf einen Blick den Zusammenhang zwischen vielen Spalten (Bild 9)

Elands Blog, das man unter [13] findet, bietet wertvolle Einblicke und weiterführende Informationen. ■

- [1] Cecily Mauran, *Microsoft Excel finally lets you disable the auto-conversion of data into dates*, www.dotnetpro.de/SL2503Polyglot1
- [2] Jan Hendrik Schreier, *Dem Koffeinschock entgehen*, dotnetpro 10/2024, Seite 12 ff., www.dotnetpro.de/A2410Polyglot
- [3] ML.NET, www.dotnetpro.de/SL2503Polyglot2
- [4] DataFrame.IO.cs auf GitHub, www.dotnetpro.de/SL2503Polyglot3
- [5] GitHub, *DataFrame (Microsoft.Data.Analysis) Tracking #6144*, www.dotnetpro.de/SL2503Polyglot4
- [6] *Extensions for ML.NET DataFrame* auf GitHub, www.dotnetpro.de/SL2503Polyglot5
- [7] Plotly.NET, <https://plotly.net>
- [8] plotly.js, <https://github.com/plotly/plotly.js>
- [9] SandDance, <https://github.com/microsoft/SandDance>
- [10] Deedle, <https://github.com/fslaborg/Deedle>
- [11] Tidyverse, <https://www.tidyverse.org>
- [12] Matt Eland, *Data Science with .NET and Polyglot Notebooks*, Packt Publishing, 2024, ISBN 978-1-83588-296-2
- [13] Matt Elands Blog, <https://mattonml.net>



Jan Hendrik Schreier

ist Wirtschaftsinformatiker und lebt im Münchner Osten. Beruflich entwickelt er Line-of-Business-Anwendungen mit Blazor, Azure Functions, SQL Server und Cosmos DB.

mail@janschreier.de

dnCode

A2503Polyglot

AZURE COMMUNICATION SERVICE

Kommunikations-Baukasten

Erste Schritte mit dem Azure Communication Service, einem Dienst zur Einbindung von Sprach-, Video- und Textchats in eigene native Applikationen und Web-Apps.

Der Azure Communication Service [1] zählt zu den jüngeren Azure-Diensten und wurde im Jahr 2020 vorgestellt. Mein Interesse an diesem Dienst wurde durch eine zunächst einfache Frage geweckt: Gibt es in Azure einen E-Mail-Dienst? Die Antwort lautet ja – und der Dienst kann noch viel mehr. Verschaffen wir uns daher zunächst einen Überblick.

ServiceLandschaft

Der Azure Communication Service ermöglicht nicht nur den Versand von E-Mails, sondern unterstützt auch deutlich komplexere Kommunikationskanäle:

- **SMS:** Über den Dienst können klassische SMS gesendet und empfangen werden.
- **Telefonie:** Neben SMS ist auch die klassische Telefonie möglich, einschließlich Anrufaufbau und Telefonkonferenzen.
- **Chat-System:** Der Dienst bietet ein modernes Chat-System, das auch Sprach- und Videotelefonie sowie Screensharing unterstützt.
- **Advanced Messaging:** Eine Schnittstelle zum WhatsApp-for-Business-Universum erweitert die Kommunikationsmöglichkeiten.
- **E-Mails:** Wie bereits erwähnt ermöglicht der Dienst auch den Versand von E-Mails.

„Microsoft-Teams-Baukasten“ und Interoperabilität

Wer bei einem Chat-System mit Sprach-, Video- und Screensharing-Funktionalität an „Das ist so was wie Teams, richtig?“ denkt, liegt gar nicht so falsch. Microsoft Teams und der Azure Communication Service scheinen zumindest ähnliche Komponenten und wahrscheinlich auch einen vergleichbaren technischen Unterbau zu nutzen.

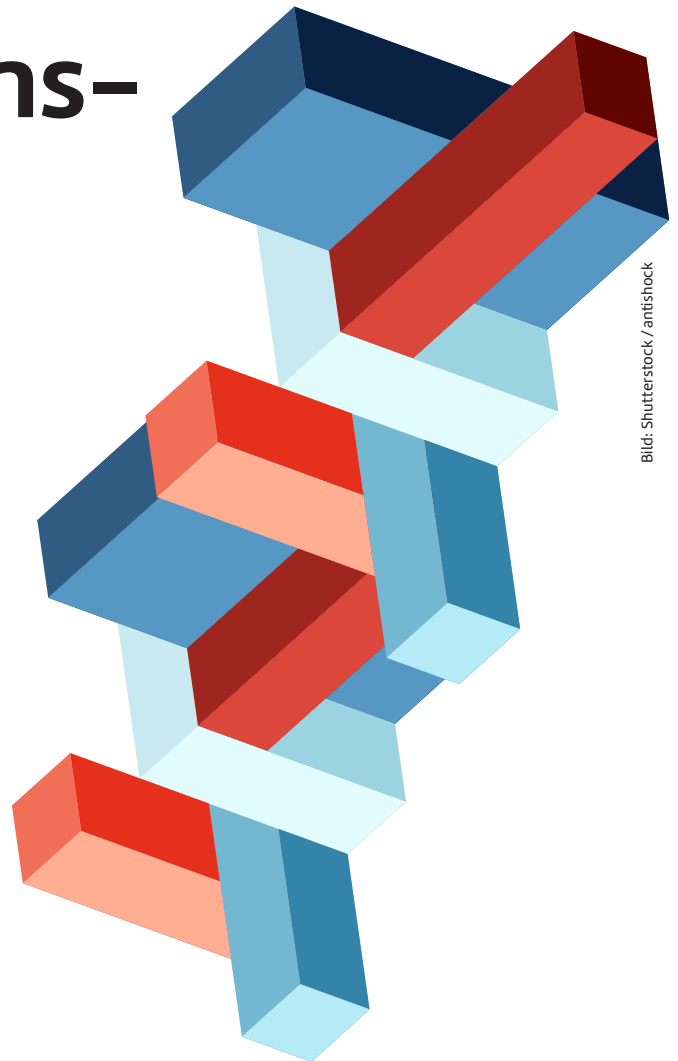


Bild: Shutterstock / antishock

Startpunkt: Anlegen der Ressource (Bild 1)

Microsoft geht mit diesem Dienst sogar noch einen Schritt weiter: Über den Azure Communication Service ist es äußerst einfach, sich mit der Microsoft-Teams-Infrastruktur zu verbinden.

Erster Schritt

Wie alle anderen Dienste in Azure ist auch der Communication Service eine einfache Resource unter einer *Resource group* mit einem Namen in einer bestimmten *Data location* (Bild 1). Die *Data location* bestimmt, wo die Daten später gespeichert werden sollen.

Achtung: Bestimmte Entwickler-Features, wie etwa eine vorkonfigurierte E-Mail-Adresse, scheinen nur mit dem *United States*-Standort zu funktionieren. Es kann auch sein, dass bestimmte Telefonie-Features abhängig von der Data Location sind, allerdings bin ich selbst gar nicht so weit gekommen, später dazu mehr.

Nach dem Erstellen bekommt der Dienst einen Endpunkt und Schlüssel für die Verwendung generiert (Bild 2).

Klassische SMS und Telefonie

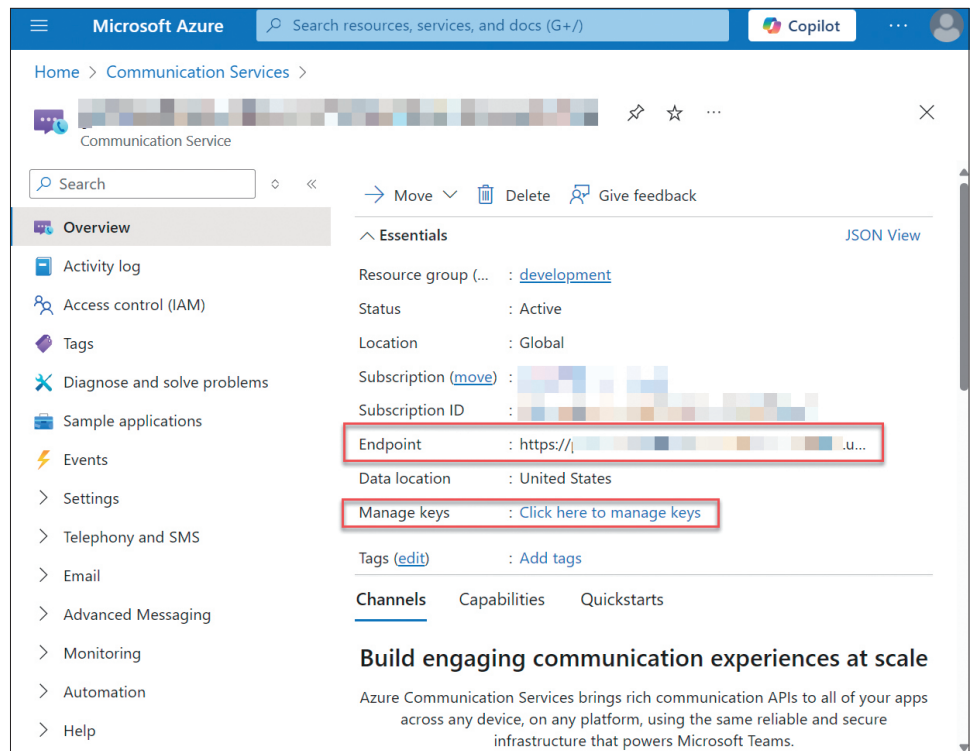
Scrollt man im Menü auf der linken Seite weiter nach unten, findet man als ersten Punkt *Telephony and SMS*. Unter diesem Menüpunkt befinden sich alle Einstellungen rund um die klassische Telefonie sowie den Versand und Empfang von SMS.

Ein positiver Aspekt ist die Vielzahl an Testmöglichkeiten, die direkt in der Benutzeroberfläche bereitgestellt werden, vergleiche Bild 3. Zusätzlich bietet der Dienst in diesem Bereich Beispiel-Code für verschiedene Programmiersprachen, welcher allerdings nicht auf dem Screenshot abgebildet wurde.

Zur Veranschaulichung hier ein Beispiel in C#:

```
using System;
using System.Collections.Generic;

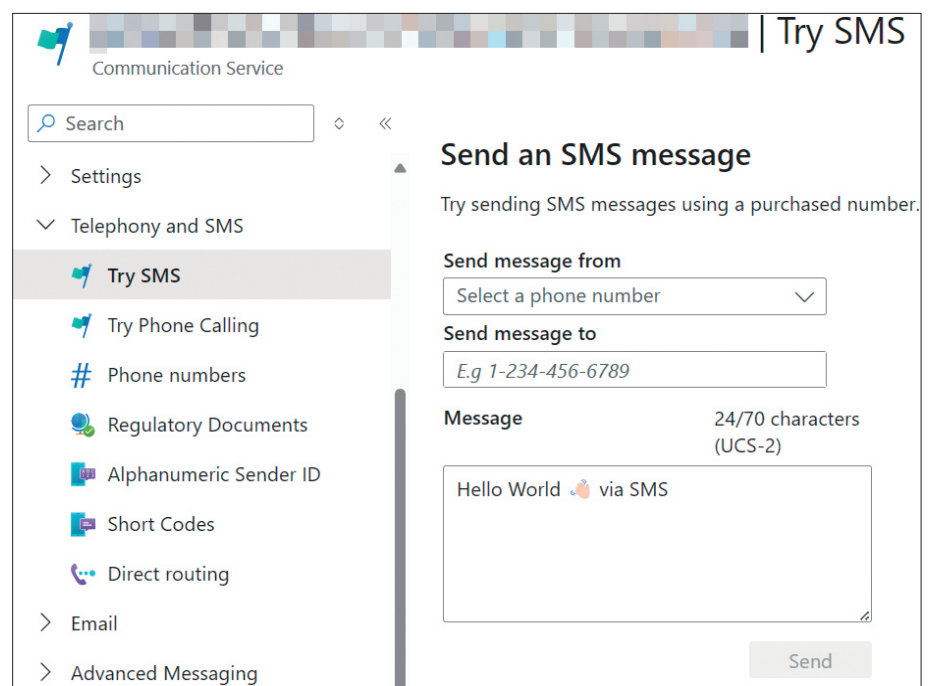
using Azure;
using Azure.Communication;
using Azure.Communication.Sms;
```



Essenziell: Endpunkt und Schlüssel für die Verwendung generieren (Bild 2)

```
// This code retrieves your connection string from an
// environment variable.
string connectionString =
    Environment.GetEnvironmentVariable(
        "COMMUNICATION_SERVICES_CONNECTION_STRING");

SmsClient smsClient = new SmsClient(connectionString); ▶
```



Integriert: Die Oberfläche enthält vielfältige Testmöglichkeiten (Bild 3)

```
smsClient.Send(
    from: "<from_phone_number>",
    to: "<to_phone_number>",
    message: ""Hello World
    📞 via SMS""
);
```

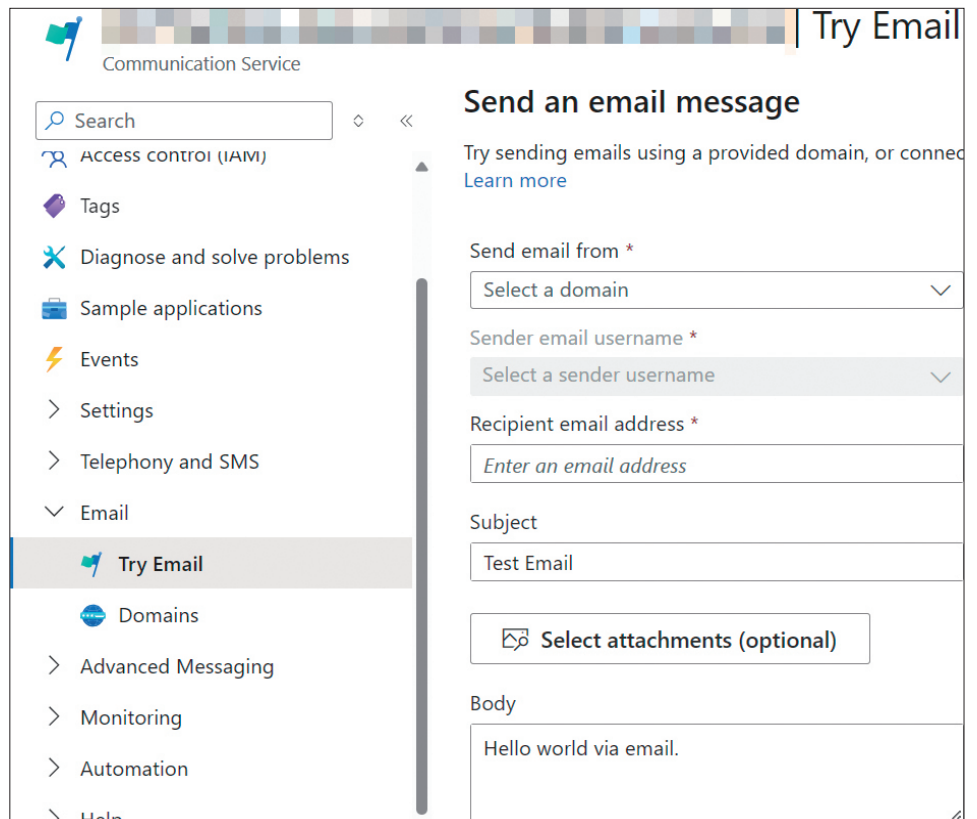
Wie man anhand der knapp 16 Zeilen Code erkennen kann, bietet der Azure Communication Service ein einfaches API. Im Hintergrund verbirgt sich dabei ein HTTP-API, sodass man den Dienst auch mit den unterschiedlichsten Programmiersprachen ansteuern kann. Für C# beziehungsweise .NET, Java, JavaScript und Python gibt es fertige SDKs.

Für die klassische Telefonie und den SMS-Versand ist eine entsprechende Telefonnummer erforderlich. Ab diesem Punkt wird es jedoch etwas komplizierter: Aufgrund von Regulierungen hängt der Erwerb von Telefonnummern mit dem Abrechnungsstandort und der Abrechnungsart des eigenen Azure-Abonnements zusammen. Für bestimmte Länder, zum Beispiel für eine Schweizer Nummer, muss auch ein manueller Prozess durchlaufen werden, oder der Dienst steht im Zweifel gar nicht direkt zur Verfügung. Aus diesem Grund kann ich leider keine eigenen Erfahrungen in diesem Bereich weitergeben.

E-Mail

Scrollt man im Menü weiter nach unten, erreicht man die E-Mail-Einstellungen. Zur Erinnerung: Der E-Mail-Dienst innerhalb des Azure Communication Service dient ausschließlich dem Versand von E-Mails. Ein Empfang von E-Mails ist darüber nicht möglich.

Ähnlich wie im Bereich *Telephony and SMS* steht auch hier eine integrierte Testoberfläche zur Verfügung (Bild 4). Eigene



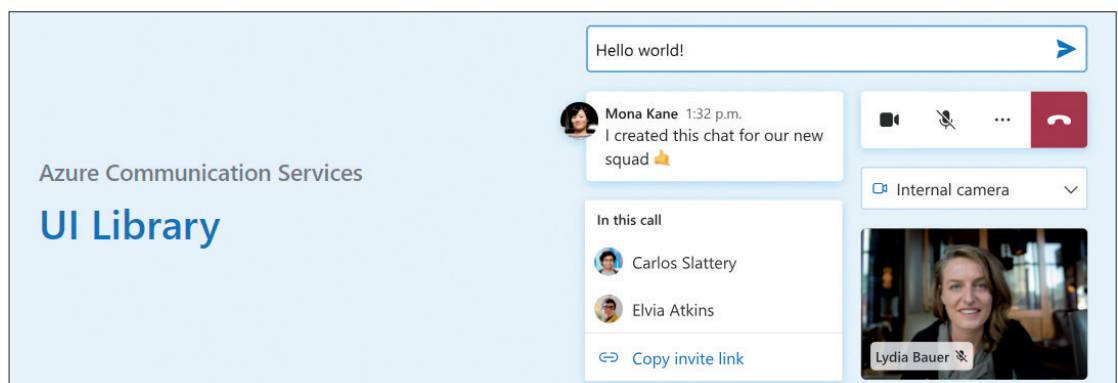
Probelauf: Test-E-Mails direkt aus der integrierten Testoberfläche (Bild 4)

Domains kann man natürlich ebenfalls hinterlegen. Hierfür sind diverse TXT Records (für SPF, DKIM und DKIM2) im DNS zu hinterlegen, und nach kurzer Zeit steht die Domain zur Verwendung im Portal und über das API bereit. Der Code ist – ähnlich wie bei der SMS – sehr schlicht gehalten:

```
using System;
using System.Collections.Generic;
using Azure;
using Azure.Communication.Email;

string connectionString =
    Environment.GetEnvironmentVariable(
        "COMMUNICATION_SERVICES_CONNECTION_STRING");
var emailClient = new EmailClient(connectionString);
```

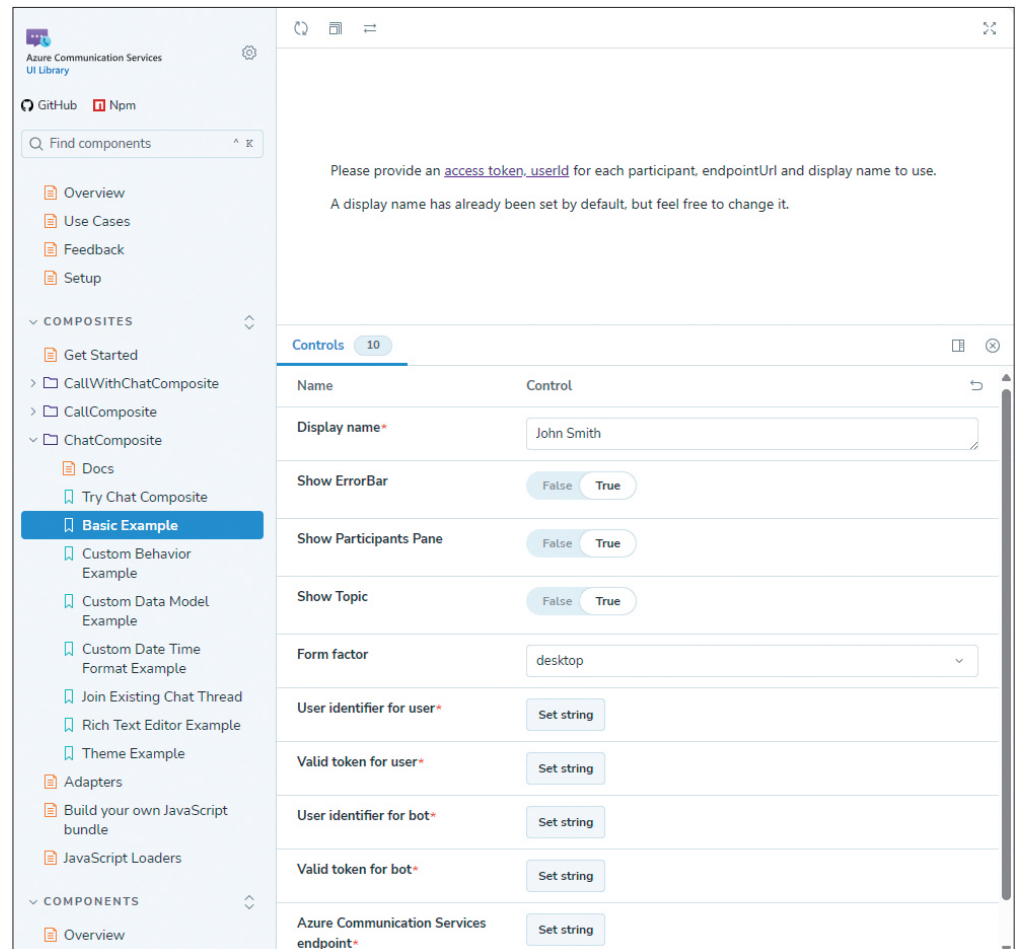
Werkzeugkasten:
Die Azure Communication Service UI Library (Bild 5)



```

var emailMessage =
    new EmailMessage(
        senderAddress:
            "DoNotReply@
            <from_domain>",
        content: new
            EmailContent("Test
            Email")
    {
        PlainText = "Hello
        world via email.",
        Html = @"
        <html>
        <body>
        <h1>Hello world
        via email.
        </h1>
        </body>
        </html>"
    },
    recipients:
        new EmailRecipients(
            new List
            <EmailAddress> {
                new EmailAddress(
                    "<to_email>") })
    );
EmailSendOperation
    emailSendOperation =
        emailClient.Send(
            WaitUntil.Completed,
            emailMessage);

```



Einstieg: Das Basis-Beispiel des Chats (Bild 6)

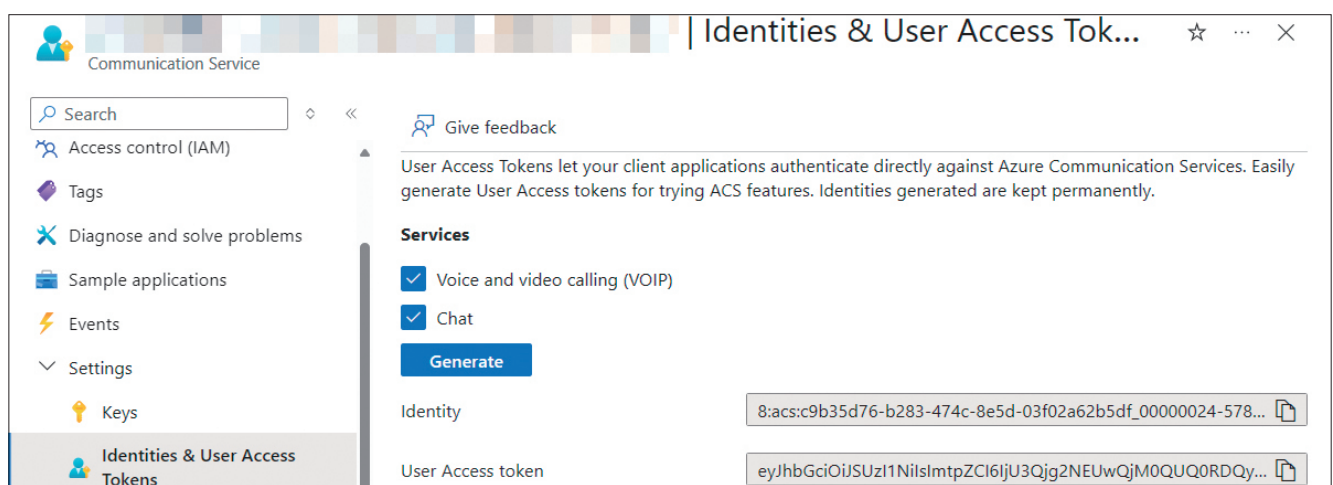
Advanced Messaging

Nach dem Menüpunkt *Email* folgt *Advanced Messaging*, das eine Schnittstelle zu WhatsApp Business bietet. Für die Nutzung dieses Dienstes ist eine Registrierung auf der WhatsApp-Business-Website erforderlich. Zudem benötigt man die WhatsApp-Business-App auf einem Smartphone sowie eine

eigene Rufnummer. Aufgrund dieser Anforderungen war es mir leider nicht möglich, den Dienst selbst auszuprobieren.

Chat, Video und Sprache

Das „Herzstück“ des Dienstes findet sich gar nicht direkt im Menü, sondern erschließt sich erst, wenn man sich die Azure Communication Service UI Library [2] genauer ansieht ►



Authentifizierung: Identifier und User-Tokens werden erzeugt (Bild 7)

(Bild 5). Wie dort bereits unschwer zu erkennen ist, ist die Ähnlichkeit mit Microsoft Teams frappierend.

Die UI Library teilt sich grob in zwei Bereiche ein:

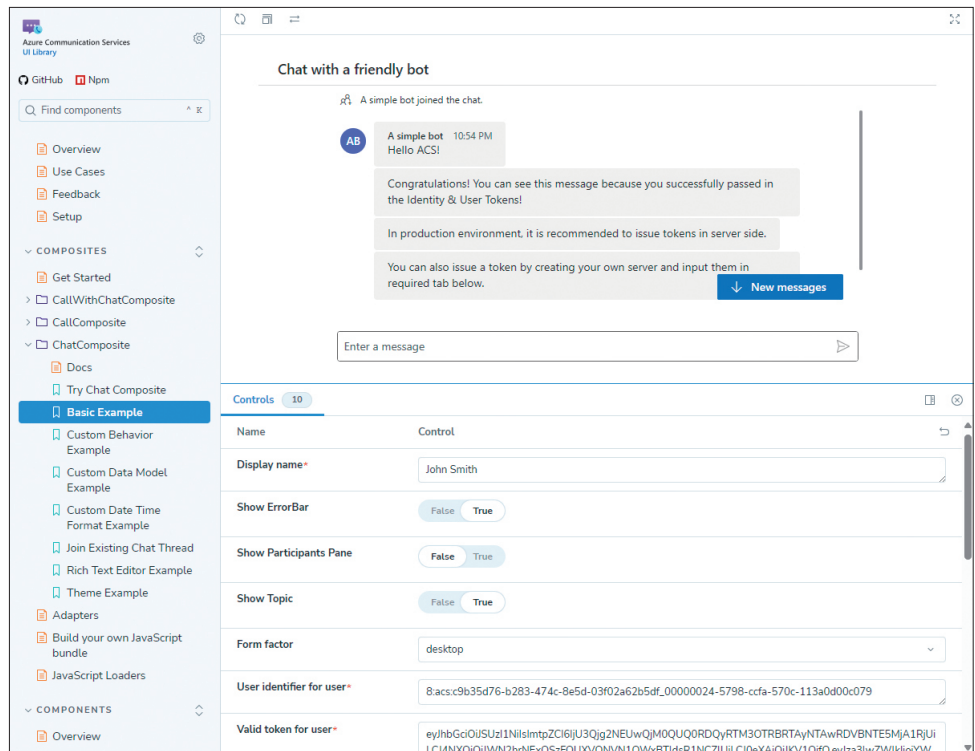
- **Composites:** Dies sind fertige Lösungen, die direkt verwendet werden können.
- **Components:** Dies sind Bausteine für eigene Lösungen, welche stärker an die eigenen Bedürfnisse angepasst werden können.

Die gesamte UI Library gibt es einmal als Web-Variante in Form einer React-Komponente, aber auch für iOS und Android in einer nativen Mobile-Variante.

Lass mal chatten ...

Sehen wir uns jetzt die *Chat-Composite*-Fertiglösung als Einstieg genauer an (Bild 6). Die Komponente benötigt für unsere beiden Gesprächsteilnehmer valide Identifier und Tokens und den Endpoint des Dienstes.

Den Endpoint findet man unter anderem unter dem *Keys*-Menüpunkt. Der Identifier und die Tokens können über *Identities & User Access Tokens* generiert werden (Bild 7).



Bereit: Der Bot bestätigt die erfolgreiche Einbindung der Tokens (Bild 8)

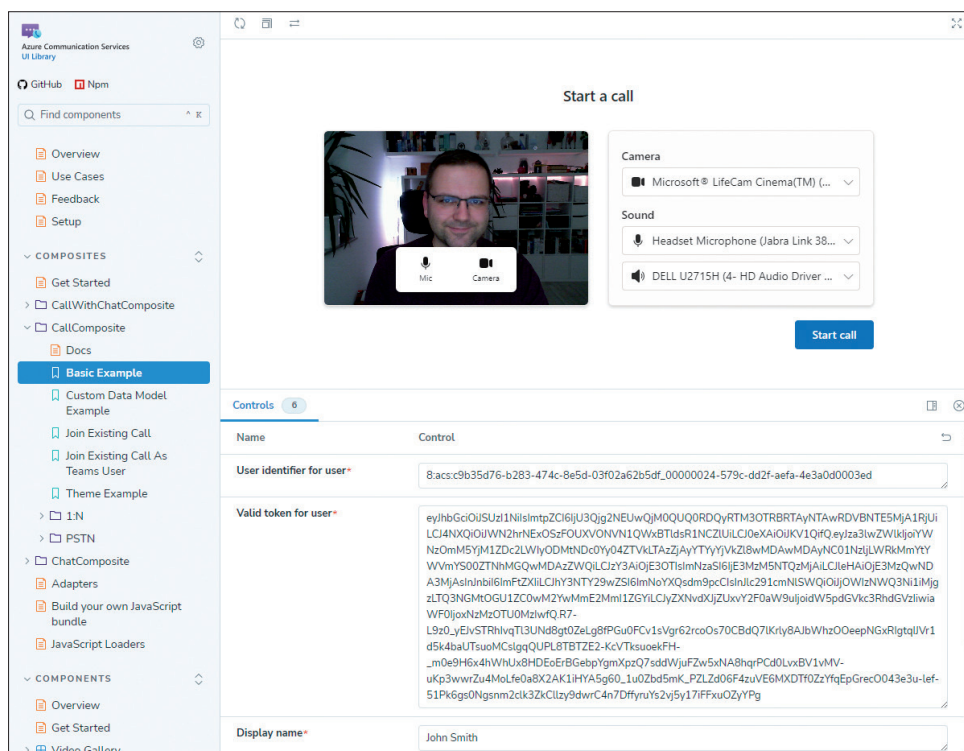
Hat man die benötigten Daten generiert und in die Einstellungen kopiert, begrüßt uns ein Bot, vergleiche Bild 8. Wie in der Abbildung zu sehen ist, stellt uns der Azure Communication Service also in Verbindung mit der UI Library eine vollständige Chat-Umgebung bereit.

... und lass uns sprechen

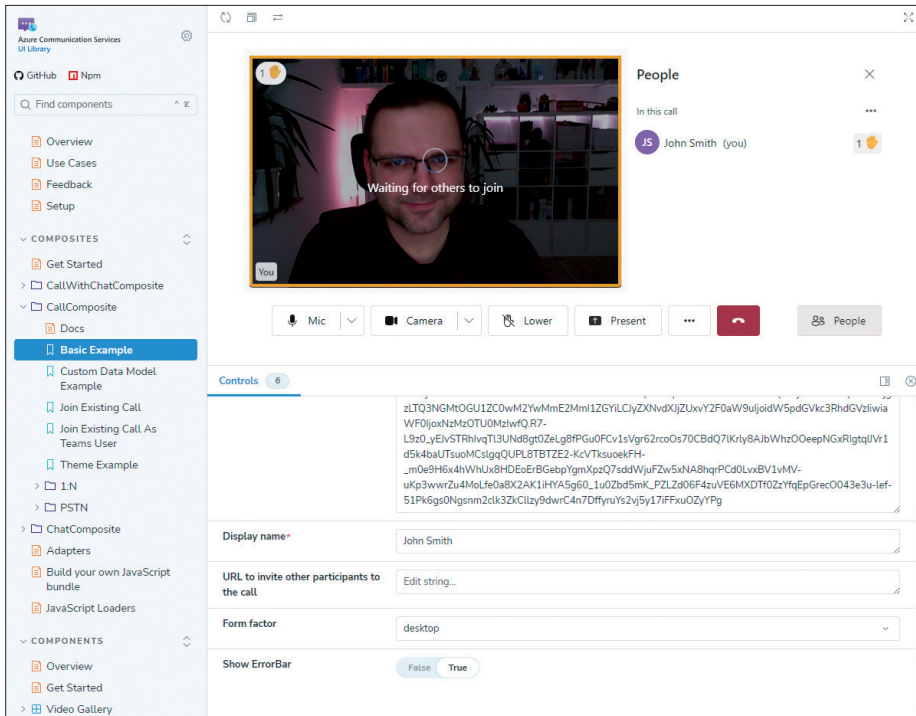
Das Chat-Beispiel mag erst einmal nicht direkt beeindruckend sein, daher geht es weiter zum *CallComposite*-Beispiel. Hier spielt die UI Library eine weitere Stärke aus: Die Integration der vorhandenen Hardware ist direkt gelöst (Bild 9). Nach dem Gesprächsbeginn steht eine sehr vertraute Umgebung mit allen wesentlichen Teams-Features bereit (Bild 10).

Teams-Integration

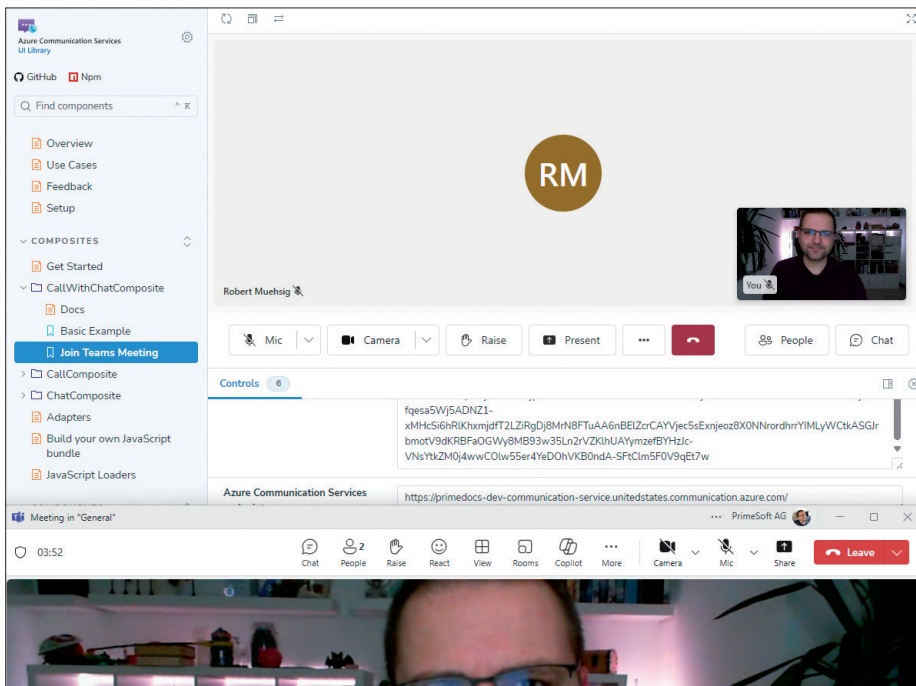
Nicht nur die Oberflächengestaltung ist sehr von Microsoft Teams geprägt, sondern es gibt auch eine direkte Brücke zwischen den Azure Communication Services und Teams. In der Fertiglösung *CallWithChatComposite* kann man sich nämlich auch in ein bestehendes Teams-Meeting einladen



Einfacher Zugriff: Die vorhandene Hardware ist direkt integriert (Bild 9)



Vertraut: Gestaltung und Features gleichen denen von Teams (Bild 10)



Brückenschlag: Chat-, Video- und Sprachtelefonie-Features von Teams direkt im Browserfenster (Bild 11)

und die typischen Chat-, Video- und Sprachtelefonie-Features direkt aus dem Browser heraus nutzen (Bild 11).

Plattformsupport und Co.

Die Beispiele aus der UI Library bieten bereits einen guten Überblick über die Funktionen. Alle diese Features werden – abhängig vom Betriebssystem und Browser – auf allen gängigen

Plattformen unterstützt, einschließlich Linux mit Chrome. Auch Optionen zur Lokalisierung und Barrierefreiheit sind berücksichtigt.

Logging und Monitoring

Wie jeder Azure-Dienst ist auch der Azure Communication Service mit umfangreichen Logging- und Monitoring-Funktionen ausgestattet. Dadurch können beispielsweise die Anzahl der versendeten Chat-Nachrichten, E-Mails, SMS und andere Aktivitäten eingesehen werden.

Fazit

Wie bereits eingangs erwähnt, kann man natürlich E-Mails über einen Azure-Dienst versenden. Doch der Azure Communication Service bietet noch viel mehr und lädt zum Experimentieren ein.

Sowohl die Portalseite als auch die Dokumentation und die UI Library stellen zahlreiche Informationen und Testwerkzeuge bereit. Dadurch lassen sich schnell erste Ergebnisse erzielen. Das API für serverseitige E-Mail- und Telefonie-Szenarien ist ebenfalls sehr benutzerfreundlich gestaltet.

Wer Microsoft Teams im Unternehmen einsetzt und externe Dienste in diese Umgebung integrieren möchte, findet im Azure Communication Service vermutlich eine interessante Option. ■

- [1] Microsoft Learn, What is Azure Communication Services?, www.dotnetpro.de/SL2503AzureCommunication1
[2] Storybook (UI Library), www.dotnetpro.de/SL2503AzureCommunication2



Robert Muehsig

ist Softwareentwickler und arbeitet vornehmlich mit dem Microsoft-Technologie-Stack. Durch sein Community-Engagement – von User Groups bis hin zu Blogs – ist er seit mehr als 15 Jahren Microsoft MVP.

robert.muehsig@codeinside.eu

dnpCode

A2503AzureCommunication

SIMULATOREN UND EMULATOREN FÜR REALITÄTSNAHE TESTS

Fast wie in echt

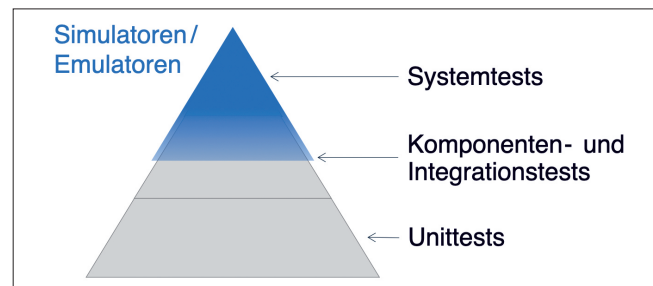
Simulatoren und Emulatoren helfen dabei, im Projekt unabhängiger testen zu können und damit eine höhere Qualität zu einem früheren Zeitpunkt zu erhalten.

Mit diesem Artikel möchte ich über eine Methode berichten, die ich regelmäßig in Softwareprojekten anwenden durfte: Software mithilfe von Simulatoren und Emulatoren testen. Was meine ich damit? Den allermeisten Leserinnen und Lesern ist sicherlich die Testpyramide bekannt. In der untersten Ebene testen wir mittels Unit-Tests einzelne Methoden und Klassen ab. In der Mitte nutzen wir für Komponenten- und Integrationstests meistens das gleiche Unit-Test-Framework und testen damit größere Teile unserer Applikation oder starten diese sogar komplett und testen direkt an den Endpunkten. Ab dieser mittleren Ebene der Testpyramide fängt es nun an, spannend zu werden, sobald ich Simulatoren und Emulatoren mit hineinnehme. Sie helfen uns dabei, unabhängiger von fremden Systemen zu werden. Naturgemäß passen sie dadurch sogar am besten zur höchsten Ebene, den Systemtests. **Bild 1** zeigt das Beschriebene grafisch an der Testpyramide.

Bevor wir tiefer einsteigen, möchte ich erläutern, was ich genau mit Simulatoren und Emulatoren meine. Diese beiden Begriffe kennen wir schließlich eher aus anderen Kontexten und nicht unbedingt aus dem Umfeld von Softwaretests. Eine Simulation etwa meint in der Regel eine abstrakte Darstellung der Wirklichkeit. Videospiele sind gute Beispiele für Simulatoren. Sie simulieren zwar in den meisten Fällen nicht unsere Realität, bilden aber zumindest eine erdachte Spielwelt ab. Einige Videospiele tragen sogar „Simulator“ im Namen. Emulatoren kennen wir etwa aus dem Umfeld der Virtualisierung. Ein virtuelles Betriebssystem läuft nicht selten auf emulierter Hardware. Von all dem spreche ich im Kontext dieses Artikels nicht. Wir können uns an dieser Stelle aber schon mal einen Unterschied der beiden Begriffe merken. „Emulator“ bezeichnet typischerweise eine sehr detaillierte Abbildung der Realität, ein „Simulator“ dagegen ist abstrakter.

● Begriffe Simulator und Emulator

Leider existieren keine klaren Begriffsdefinitionen für Simulator oder Emulator. In aller Regel sind Simulatoren eine gröbere Abbildung der Realität und damit eher „High-Level“. Emulatoren bieten eine detailliertere Abbildung der Realität und sind damit eher „Low-Level“. Im Softwareumfeld sind auch andere Begriffe vorzufinden, etwa „Service-Virtualisierung“ oder etwas moderner „Digital Twin“.



Einordnung der Relevanz von Simulatoren und Emulatoren in der Testpyramide (**Bild 1**)

Was hat das alles nun mit den Tests unserer Software auf den höheren Ebenen der Testpyramide zu tun? Mehr, als man vielleicht zunächst denkt. Sehen wir uns zunächst die unteren Schichten der Testpyramide noch einmal an. Hier nutzen wir alle bereits etwas sehr Ähnliches: Einige nennen sie Stubs, andere Mocks, und ich bin mir sicher, dass es noch einige weitere Bezeichnungen gibt. Wir setzen kleine Bausteine ein, die so tun, als wären sie eine Komponente unserer Software. Wir nutzen sie, damit wir gezielt nur bestimmte Teile unserer Software testen und den Einfluss externer Abhängigkeiten auf das Testergebnis reduzieren können.

Vermutlich wird bereits klar, auf was ich als Nächstes hinausmöchte. Simulatoren und Emulatoren machen genau das, nur sind sie im Regelfall größer und mit mehr Logik versehen. Typischerweise werden damit also nicht nur einzelne Bausteine, sondern ganze Umsysteme nachgebildet.

Im Kasten **Begriffe Simulator und Emulator** ordne ich die Begriffe nochmals ein. Im weiteren Verlauf dieses Artikels verwende ich nur noch „Simulator“, da das nachfolgende Projektbeispiel ein Simulator ist. Das Beschriebene gilt allerdings analog auch für Emulatoren.

Ein praktisches Beispiel

In meiner eigenen Projekthistorie könnte ich viele Beispiele nennen. Eines aus der jüngeren Vergangenheit ist gut geeignet, um einen Einstieg in das Thema zu finden. Es wurde im Rahmen eines Projekts im Umfeld der Digitalisierung des Einzelhandels umgesetzt. Dabei ging es um die Bereitstellung digitaler Kassenbons. Das Ziel des Projekts war es, dem Kunden anstelle eines ausgedruckten Bons einen digitalen Bon anzubieten. Das klingt zunächst nicht sonderlich kompliziert, das Mengengerüst ist aber durchaus nicht ohne. Wir sprechen über mehrere Millionen Bons pro Tag, mehrere Tau-

send Kassen, circa 10 bis 20 KByte an Daten pro Bon und in der Spitze einen dreistelligen Wert von Bons pro Sekunde. Nachdem digitale Bons im System angekommen sind, werden sie wahlweise über eine App, per E-Mail und/oder per Push ausgeliefert.

Die Challenges im Projekt liegen nicht nur auf der vom Mengengerüst abgeleiteten Last, sondern auch bei den vielen Sonderfällen, die es an den Kassen gibt. Beispiel: Einkauf in einer Filiale in der Nähe der Landesgrenze, Kunde zahlt in Fremdwährung, Rückgeld kommt aber in der Landeswährung. Solche Fälle haben entsprechende Auswirkungen auf den digitalen Kassensbon.

Sehen wir uns nun die einzelnen Bestandteile der Teststrategie an, für die wir uns im Projekt entschieden haben:

1. Automatisierte Unit- und Integrationstests: Damit stellen wir sicher, dass unsere APIs wie vorgesehen reagieren und neu hinzukommende Spezialfälle das Verhalten an anderer Stelle nicht ändern.
2. Funktionstests mit einem eigens entwickelten Kassens-Simulator: Ein Simulator sorgt hier dafür, dass sowohl Entwickler und Product Owner als auch Fachanwender den vollständigen Ablauf innerhalb einer Testumgebung testen können – zu jeder Zeit und ohne Abhängigkeit auf eine zur Verfügung stehende Testkasse.
3. Massentests ebenso mit eigens entwickeltem Kassens-Simulator: Ein Simulator unterstützt uns dabei, eine größere Menge an Testdaten zu generieren, und simuliert dabei nicht nur eine, sondern mehrere Tausend Kassen, die unser System unter Stress setzen sollen. Parallel werden ebenso massenhaft Zugriffe auf die digitalen Bons simuliert, um das Benutzerverhalten im Massentest mitzubetrachten.
4. Funktionstests mit echter Kasse: Während des Projekts und am Ende werden Funktionstests an der echten Kasse durchgeführt. Dies dient dazu, sicherzustellen, dass die Systeme auch tatsächlich in der Realität störungsfrei miteinander kommunizieren können.
5. Testphase im Produktivsystem mit Tester-Kreis: Ab hier geht es in die Realität mit einer vorbestimmten Auswahl an Testfilialen und Testbenutzern. Ziel ist es, das System innerhalb von echten Filialen mit echten Einkäufen zu testen.
6. Ausrollen in die Produktion in Stufen: Abschließend wird das System schrittweise auf alle Filialen des Einzelhändlers ausgerollt.

Bei den Punkten 2 und 3 wurde ein eigens entwickelter Kassens-Simulator eingesetzt.

Wie beschrieben, geht es bei Punkt 2 zunächst um funktionale Tests. Der Kassens-Simulator ist technisch als Webapplikation umgesetzt und bietet ein UI ähnlich dem Wireframe in Bild 2. Hier gibt es verschiedene Eingabefelder, etwa für die Filiale (Feld *Location*), für das Benutzerkonto (Feld *User Number*), für die Punktekarte (Feld *Loyalty Card*) und noch einige weitere. Strukturierte Daten wie der Warenkorb erhalten ebenfalls ein Eingabefeld, hier allerdings direkt als JSON. Warum JSON? An einigen Stellen ist das ein guter Kompro-

miss aus Aufwand, Nutzen und Bedienbarkeit. Im konkreten Beispiel sind JSON-Felder sogar nach der tatsächlichen technischen Schnittstelle angelegt. Wieso? Für einen Test können Testdaten direkt aus der Protokollierung des Live-Systems entnommen werden. Sonderfälle können so im Produktivsystem gesucht und die relevanten Daten leicht für einen Funktionstest übernommen werden.

Die Schaltflächen *Start*, *Modify* und *Finish* bilden jeweils Schritte in der Kassenschnittstelle ab. Für diesen Artikel ist das nur insofern relevant, als es bei einem Simulator-Tool durchaus okay ist, technische Aspekte in das UI mit hineinzunehmen. Der Grund: Es handelt sich um ein spezialisiertes Testtool, das etwas Bestimmtes simulieren soll. Es ist nicht nur

Wireframe Kassens-Simulator mit Fokus auf funktionale Tests (Bild 2)

okay, sondern in der Regel auch besser, sich hier an der Technik zu orientieren. Wenn Fehlverhalten gefunden wird, erleichtert das die Kommunikation zwischen Fachanwender, Product Owner und Entwickler ungemein. Ebenso wird die Fehleranalyse deutlich vereinfacht, da kein mentales Mapping zwischen Begriffen im UI und Begriffen in der Technik stattfinden muss.

Im letzten Wireframe fehlt noch ein wichtiger Bestandteil. Der Kassens-Simulator unterstützt den Tester zwar darin, Testfälle zu konfigurieren und auszulösen. Was jedoch noch fehlt, ist ein Einblick darin, was im Hintergrund tatsächlich passiert. Das ist für Benutzer:innen des Kassens-Simulators durchaus wichtig. Zum einen geht es darum, unmittelbar Feedback zu bekommen, dass ein Button-Druck eine Aktion im Hintergrund ausgelöst hat. Zum anderen dient es auch dazu, dass Benutzer:innen beurteilen können, dass der Kassens-Simulator auch tatsächlich das Richtige macht. Ein Punkt, der bei Simulatoren stets wichtig ist, ist, dass auch diese Tools selbst Fehler aufweisen können. Eine möglichst hohe Transparenz darin, was der Simulator im Hintergrund tatsächlich macht, hilft, derartige Fehler frühzeitig als solche zu erkennen. ►

In **Bild 3** sehen wir eine Erweiterung des vorigen Wireframes. Enthalten ist zusätzlich zu den Feldern von vorher auch der Kasten *Communication*. Darin sehen wir, welchen Request der Kassen-Simulator ausgelöst und welche Response unser System als Antwort geliefert hat. Mittels dieser einfachen Informationen lässt sich beurteilen, ob der Kassen-Simulator die Eingaben korrekt in einen Request verpackt und ob die Antwort unseres Systems zu dem passt, was die Kasse erwartet.

Der nächste Punkt 3 in der weiter oben beschriebenen Teststrategie ist der Massentest. Hierbei eignet sich der Kassen-Simulator hervorragend, um die umgesetzte Funktionalität auf Masse hochzuskalieren. Viele Teile der Logik können in einen neuen Massentest-Modus übernommen werden. Für den Massentest ist es – anders als bei funktionalen Tests – nicht wichtig, dass die Benutzer:innen Testfälle beliebig konfigurieren können. Hier kommt es eher darauf an, das Lastverhalten konfigurieren zu können und einen Einblick in Antwortzeiten und Fehler des Systems zu erhalten.

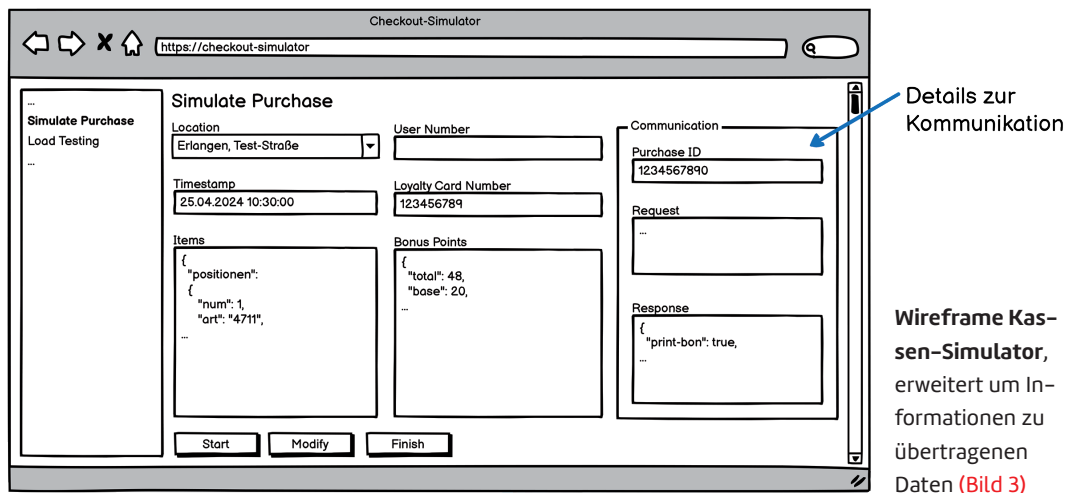
Das Wireframe in **Bild 4** soll einen Einblick darin geben, wie wir es im Projekt umgesetzt haben. Die Benutzer:innen können den Massentest hier zunächst starten und stoppen. Sie stellen außerdem die Anzahl der Einkäufe pro Sekunde ein (Feld *Purchases per Second*), die von den im Hintergrund simulierten Kassen generiert werden.

Eine neue Funktionalität für den Massentest, die auch nicht mehr ganz zum Begriff Kassen-Simulator passt, befindet sich hinter dem Feld *Receipt Views per Second*. An dieser Stelle wurde der Kassen-Simulator auch um das Verhalten des Endkunden erweitert, und zwar in Gestalt der Anzahl der Views auf den digitalen Kassenbonn pro Sekunde. In Summe geht es darum, dass man unser System kontrolliert von zwei Seiten unter Last setzt:

- Generierung von Einkäufen, die die Kassen an unser System übertragen.
- Generierung von Zugriffen auf die digitalen Kassenbons in unserem System.

Im unteren Teil des Wireframes sehen wir abschließend, was der Massentest im Hintergrund macht. Eine fortlaufende Liste der generierten Einkäufe, ein Verlauf der Response-Zeiten unseres Systems und erkannte Fehler. Diese Anzeigen erfüllen mehrere Zwecke:

- Die Benutzer:innen auf mögliche Fehler im System aufmerksam machen.
- Den Benutzer:innen aktiv zeigen, dass der Kassen-Simulator wie vorgesehen arbeitet.



Wieso nicht mit der Realität testen?

Nachdem wir uns mit einem Beispiel im Detail beschäftigt haben, stellt sich die Frage, wieso wir nicht einfach mit realen Dingen testen? Das kann mehrere Gründe haben. Ganz oben auf der Liste stehen Abhängigkeiten. Mit Abhängigkeiten meine ich nicht nur technische Themen, sondern auch fachliche. Ist man etwa abhängig von anderen Teams oder gar anderen Bereichen im Unternehmen? Teilt man sich Test-Hardware mit anderen? Ist spezielles Personal notwendig, um testen zu können? Alle diese Punkte können für teure Verzögerungen im Projekt sorgen. Simulatoren reduzieren solche Abhängigkeiten, da diese direkt zur Verfügung stehen.

Ein weiterer Punkt sind unterschiedliche Fertigstellungstermine. Häufig ist es so, dass nicht nur das eigene System in einem Projekt entwickelt wird, sondern auch Umsysteme werden parallel entwickelt oder um neue Features erweitert. Simulatoren helfen dabei, dennoch frühzeitig und unabhängig testen zu können.

Abschließend möchte ich noch einen in unserem Beispiel wichtigen Punkt nennen: Die Realität ist häufig nicht so einfach auf Masse skalierbar. Simulatoren in der Regel aber schon. Das ermöglicht uns, im Projekt frühzeitig die Reaktion unseres Systems auf massenhafte Anfragen zu testen.

Es gibt noch viele weitere Punkte, die ich hier aufzählen könnte. All diese Punkte sind je nach Projekt unterschiedlich relevant. Im beschriebenen Beispiel etwa waren die Abhängigkeit zu anderen Abteilungen und die Skalierbarkeit auf

Kein endgültiger Ersatz für die Realität

Simulatoren und Emulatoren ermöglichen frühzeitige und leicht wiederholbare Tests. Sie helfen dementsprechend, sehr früh im Projekt bereits eine höhere Qualität der entwickelten Software zu erhalten. Den Test an der Realität ersetzen Simulatoren und Emulatoren aber nicht. Sie reduzieren lediglich den dafür notwendigen Zeitaufwand, da das zu testende System bereits einen höheren Qualitätsgrad aufweist.

Masse zwei wichtige Punkte. Es gab zwar Testkassen in Hardware, diese waren aber an einem anderen Ort und in der Verantwortung einer anderen Abteilung. So kann es schon einmal vorkommen, dass sie spontan nicht verfügbar sind. Zudem ist es schwierig, jeden Sonderfall hier exakt und schnell wiederholen zu können. Der Kassen-Simulator ermöglichte es, regelmäßig ein Set von Regressionstestfällen auf Ebene der Systemtests durchzuführen.

Technische Umsetzung von Simulatoren und Emulatoren

Bei der technischen Umsetzung gilt es zu prüfen, ob vorhandene Software verwendet werden kann oder ob es vorteilhafter ist, wie im Beispiel etwas Eigenes zu bauen. Für verschiedene Branchen und Einsatzzwecke existieren Standardprodukte, auf die aufgesetzt werden kann. Demgegenüber existieren auch einige Tools, die abstrakter aufgebaut sind und zum Beispiel vereinfacht immer OK bei Anfragen liefern.

Bei dem obigen Projektbeispiel hat man sich für eine Eigenentwicklung entschieden. Ein Grund war, dass der dafür notwendige Zeitaufwand sehr überschaubar war. Technisch wurde das Beispiel mit ASP.NET Core Blazor (Server Side) umgesetzt. Insgesamt kann man für Entwicklung inklusive Test, Qualitätssicherung und Betrieb von wenigen Wochen Aufwand sprechen, die auf der anderen Seite eine deutliche Produktivitätssteigerung bei den funktionalen Tests gebracht hat. Massentests waren durch den Einsatz eines Simulators überhaupt erst möglich.

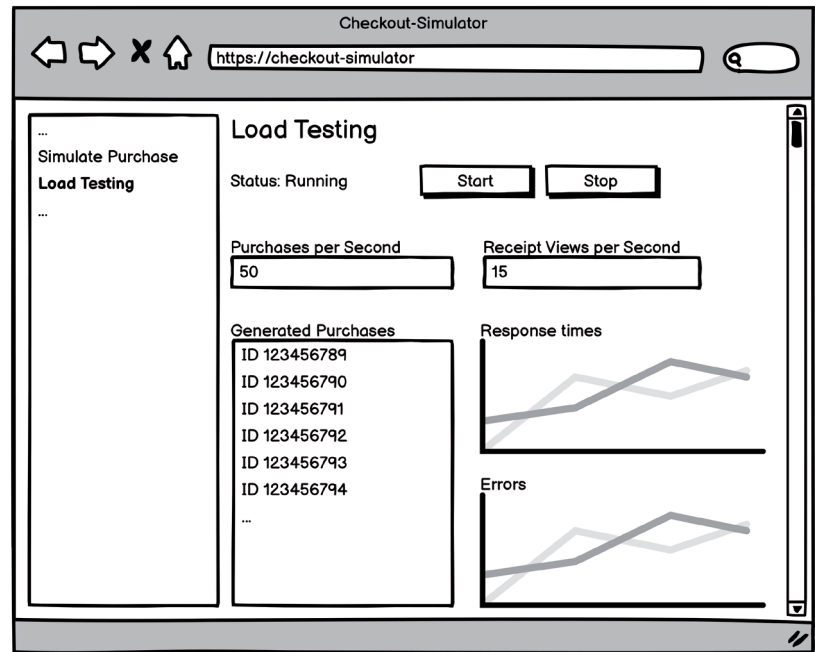
Simulatoren oder Emulatoren für eigene Projekte

Ob der Einsatz von Simulatoren für ein Projekt Sinn ergibt, ist schwierig pauschal zu beantworten. Primär geht es zunächst um eine Kosten-Nutzen-Betrachtung. Ich möchte dazu einige Punkte mitgeben, die an der Stelle wichtig sind:

- Es geht häufig darum, das Verhalten einer Schnittstelle nachzubilden. Sind Schnittstellenbeschreibungen vorhanden, aus denen Code generiert werden kann, erleichtert das die Arbeit erheblich. Beispiele wären Swagger-, WSDL- oder Proto-Dateien.
- Fehlerfälle sollten nur nach Bedarf abgebildet werden. In den meisten Projekten sind für Simulatoren eher weniger Fehlerfälle relevant. Wichtiger ist es, die auf der Leitung kommunizierten Daten sichtbar zu machen. So ist für die allermeisten Tests keine automatische Behandlung von Fehlerfällen notwendig, da man als Benutzer:in den Fehler an den übertragenen Daten an der Schnittstelle lesen kann.
- Möglichkeiten für manuelle Einflussnahme schaffen, um möglichst wenig Edge-Cases im Code abbilden zu müssen. Die Eingabe des Warenkorbs als JSON im Beispiel oben folgt genau diesem Gedanken. Es geht also nicht darum, ein möglichst benutzerfreundliches UI zu bauen, sondern darum, den Benutzer:innen ein Werkzeug in die Hand zu

geben, mit dem sich effizient die relevanten Testfälle durchführen lassen.

- Nachvollziehbarkeit für die Benutzer:innen des Simulators schaffen. Damit meine ich etwa die Anzeige von Requests und Responses. Das hilft dabei, das korrekte Verhalten der betroffenen Systeme im Detail beurteilen zu können.



Wireframe Kassen-Simulator mit Fokus auf Massentests (Bild 4)

Fazit

Simulatoren und Emulatoren helfen uns ab der mittleren Ebene der Testpyramide, unabhängiger testen zu können. Abhängigkeiten wie spezielle Test-Hardware oder -Software, spezifisches Personal, Ressourcen oder auch eine Abhängigkeit von einem bestimmten Ort verschwinden. Man erhält höhere Flexibilität bei der Testplanung und bei der Wiederholung von Tests. Ebenso werden Massentests in vielen Einsatzszenarien erst möglich.

Der Einstieg in Simulatoren und Emulatoren kann je nach Fachgebiet einfach sein, sofern entsprechende Tools bereits am Markt existieren. Eine Eigenentwicklung kann aber meist ebenso häufig mit geringen Zusatzaufwänden gestemmt werden. Solche Zusatzaufwände werden von den Vorteilen im Projekt meist übertroffen. ■



Roland König

ist freier Softwarearchitekt, -Entwickler und Berater. Er ist seit 2007 in der Welt von C#/.NET zu Hause und beschäftigt sich seit 2012 intensiv mit aktuellen Trends im Umfeld moderner Softwarearchitektur.

roland.koenig@rolandk.de

dnpCode

A2503Simulatoren

HYBRIDE ECHTZEITSTEUERUNG MIT EINER SPS UND C#

Eine Frage der Echtzeit

Echtzeitsteuerungen mit C# scheitern am fehlenden deterministischen Zeitverhalten des .NET Frameworks und Betriebssystems. Wie eine Hybridlösung das Problem umgeht.

Wer schon einmal versucht hat, eine Echtzeitsteuerung mit Standard-Windows und C# zu realisieren, ist wahrscheinlich in Schwierigkeiten geraten. Der Grund: Sowohl Windows als auch das .NET Framework haben in der Standardausführung kein deterministisches Zeitverhalten. Es gibt jedoch verschiedene Ansätze, um das Problem zu lösen beziehungsweise zu umgehen. Dieser Artikel beschreibt eine gängige Hybridlösung, die eine speicherprogrammierbare Steuerung (SPS) in Kombination mit einem Windows- oder Linux-Computer verwendet.

Zunächst soll geklärt werden, was genau Echtzeit ist, welche besondere Herausforderungen Echtzeit mit sich bringt und welche gängigen kommerziellen Lösungen auf dem Markt verfügbar sind. Als Echtzeitsteuerung wird eine SPS verwendet, wie sie in der Praxis typischerweise für größere Maschinen und Geräte eingesetzt wird. Anhand eines einfachen Beispiels wird eine Softwarearchitektur implementiert, welche die modernen OOP-Konzepte verwendet, die wir aus Hochsprachen wie C# kennen.

Anschließend werden wir uns mit dem Kommunikationsprotokoll OPC Unified Architecture (OPC UA) befassen, das eingesetzt wird, um die Daten zwischen der SPS und dem Computer auszutauschen. Für das Beispiel im Artikel wird die OPC-UA-Schnittstelle definiert und gezeigt, wie man mit einem kommerziellen OPC-UA-Client die Echtzeitfunktionalität in der SPS auslöst. Eine in C# definierte Proxy-Klasse greift dazu mit einer OPC-UA-Client-Bibliothek auf die SPS zu. Mit den Methoden und Eigenschaften der Proxy-Klasse lässt sich die SPS-Software so ausführen, als wäre sie in C# implementiert.

„Harte“ und „weiche“ Echtzeit

Echtzeit wird nicht definiert durch die Geschwindigkeit der Berechnungen, sondern dadurch, dass die Berechnungen rechtzeitig ausgeführt werden. Wikipedia definiert Echtzeitanforderungen durch „ein Ergebnis, welches innerhalb eines vorher fest definierten Zeitintervalls garantiert berechnet ist“. Beispiele von Echtzeitsystemen sind Airbag-Steuerungen, Antiblockiersysteme, Frequenzumrichter sowie Geräte- und Maschinensteuerungen.

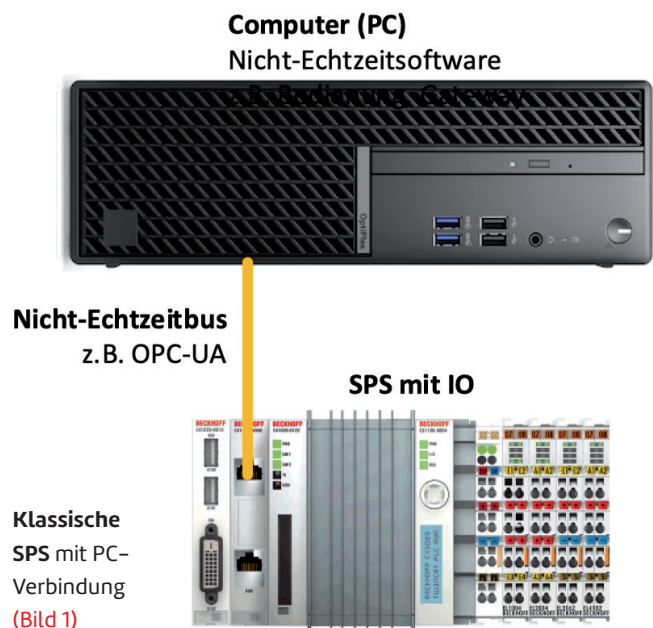
Neben „harten“ Echtzeitanforderungen gibt es auch „weiche“ Echtzeitanforderungen. Bei „weicher“ Echtzeit wird lediglich ein ungefährender Mittelwert an Reaktionszeiten angestrebt. Für derartige Aufgaben reicht in der Regel C#-Software aus, die auf einem Windows- oder Linux-Computer ausgeführt wird.

Echtzeitlösungen

Abhängig von der Größe der Anlage ist die Steuerung entweder eingebettet (Embedded Control) oder in einem separaten Gehäuse untergebracht. Eingebettete Controller bestehen typischerweise aus kundenspezifischer Elektronik mit Mikrocontrollern oder handelsüblichen Minicomputern wie Raspberry Pi, Beagleboard und Arduino. Mikrocontroller werden in der Regel in C/C++ oder – in zeitkritischen Fällen – sogar direkt in Assembler programmiert. Je nach System und Größe wird die Hardware-Peripherie direkt oder per Firmware angesteuert. Die größeren Prozessoren und Minicomputer unterstützen meist eingebettete Echtzeitbetriebssysteme wie Embedded Linux, QNX, VxWorks und INTEGRITY.

Größere Anlagen hingegen werden per speicherprogrammierbarer Steuerung (SPS; englisch: programmable logic controller, PLC) oder per Industrie-PC angesteuert. Eine SPS wird gemäß der EN/IEC-61131-3-Norm in einer von fünf Sprachen programmiert. Die meisten SPS-Steuerungen unterstützen jedoch auch C oder C++, und mit der neuen PLC-next-Technologie ist es endlich möglich, eine SPS auch in C# zu programmieren. Bis heute wird diese Technologie jedoch nur von einzelnen Lieferanten angeboten, so beispielsweise von Beckhoff oder Phoenix Contact.

Wird ein Industrie-PC eingesetzt, läuft auf dem Gerät entweder eine Soft-SPS oder ein echtzeitfähiges Betriebssystem



wie RTLinux. Auch in diesem Echtzeit-Umfeld sind C und C++ immer noch die meistverwendeten Programmiersprachen.

Die speicherprogrammierbare Steuerung (SPS)

Die erste SPS kam vor etwa 50 Jahren auf den Markt und hat seitdem fest verdrahtete Steuerungen ersetzt. Eine SPS besteht aus einem Prozessormodul und einer Reihe von Peripheriemodulen, die mit den Sensoren, Aktuatoren und Motoren verdrahtet sind (Bild 1).

Bei einer Soft-SPS läuft das Prozessormodul auf einem Industrie-PC. Die Echtzeitsoftware wird auf dem Industrie-PC ausgeführt und steuert die IO-Module über einen proprietären Echtzeitbus an, vergleiche Bild 2. Seit einigen Jahren versucht man, mit OPC UA over TSN (Time-Sensitive Network) die proprietären Feldbusse zu standardisieren. OPC UA over TSN soll eine standardkonforme und herstellerunabhängige Vernetzung von Maschinen und Anlagen über ein einheitliches Protokoll für Echtzeitanforderungen ermöglichen.

Eine SPS ist sehr robust und hat eine Lebensdauer von mehr als 25 Jahren. Dank jahrelanger Erfahrung in unterschiedlichen Anwendungsbereichen kann der Anwender aus einer schier endlosen Vielfalt von Off-the-Shelf-Peripheriemodulen wählen.

Ein klassischer SPS-Programmierer ist kein IT-Spezialist mit Bachelor- oder Masterabschluss, sondern eine Elektrofachperson oder ein Elektroniker mit Programmiererfahrung aus der Praxis. Die SPS-Lieferanten und SPS-Standards wie IEC 61131 und OpenPLC versuchen bewusst, die Komplexität für die Programmierer durch Standardisierung und Abstraktion so niedrig wie möglich zu halten. Dennoch erfordert die Programmierung einer SPS erhebliche Erfahrung und gute Einblicke in den zu steuernden Prozess sowie sein Echtzeitverhalten. Wie bei herkömmlicher Software besteht auch bei einer SPS die Möglichkeit von Race Conditions und Data Corruption, da zahlreiche Prozesse parallel und in unterschiedlichen Zykluszeiten ablaufen.

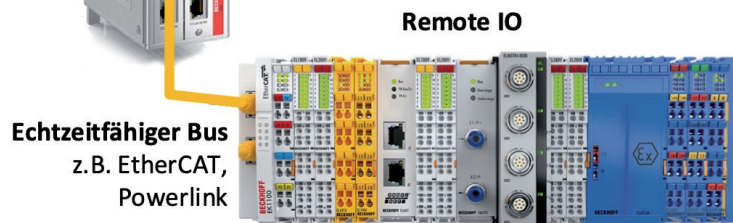
SPS-Software

Eine SPS erfordert ein anderes Programmierparadigma als herkömmliche Software. Software in der PC-, Mobil- und Internetwelt ist prozedural und ereignisgesteuert. Externe Interaktionen wie Benutzereingaben, Systemereignisse und Webanfragen lösen die Ausführung des Codes aus. SPS-Code hingegen wird zyklisch in definierten festen Zeitintervallen ausgeführt. Die Codeausführungszeit muss kürzer sein als die Intervallzeit. Andernfalls kommt es zu Zykluszeitverletzungen. Je nach Echtzeitanforderungen entscheidet der Programmierer, wie das System mit Zykluszeitverletzungen umgehen soll. Typischerweise wird jeder Ablauf, der länger als die Zykluszeit dauert, durch eine Zustandsmaschine mit kurzer Ausführungszeit modelliert. Schleifen und rekursive Aufrufe sind dabei mit Vorsicht zu genießen.

Wie bereits weiter oben erwähnt, definiert der IEC-61131-Standard fünf verschiedene Sprachen, um eine SPS zu

Industriecomputer (IPC) SoftSPS mit Echtzeitsoftware

SoftSPS mit Remote-IO-Modulen (Bild 2)



programmieren. Dabei ist Strukturierter Text (STX) [1] den herkömmlichen Programmiersprachen wie Pascal am ähnlichsten und wird deshalb im Beispiel dieses Artikels verwendet.

Die Hauptbestandteile der SPS-Software sind Programme, Funktionen und Funktionsblöcke.

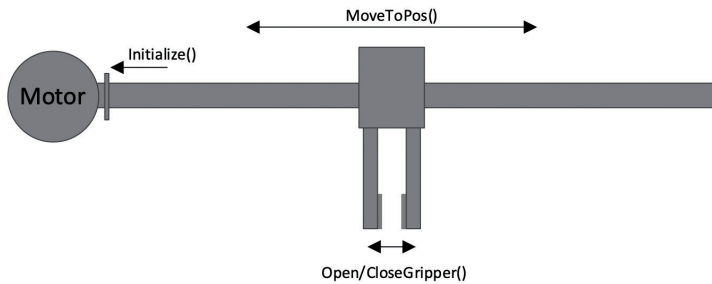
- Programme bilden die Einstiegspunkte der SPS-Software (wie 'main()' in der PC-Software) und werden durch Tasks mit vordefinierten Zykluszeiten ausgeführt.
- Funktionen gleichen den Methoden in der PC-Welt: Sie sind zustandslos, haben einen optionalen Rückgabewert und werden für einfache Aufgaben verwendet.
- Ein Funktionsblock ist bei den meisten SPS-Lieferanten eine Funktion, die sich Zustände merken kann. In einer Variablen wird der Schritt der Zustandsmaschine gespeichert und beim nächsten Aufruf wieder aktiviert.

Mit dem Codesys Development System ist es schon seit 2010 möglich, Funktionsblöcke wie Klassen aus der PC-Welt zu gestalten und so die OOP-Basiskonzepte in STX anzuwenden. Mit dieser Erweiterung können Funktionsblöcke Schnittstellen implementieren und an andere Funktionsblöcke vererben. Sie unterstützen Eigenschaften, Methoden, einen Konstruktor und virtuelle Methoden, womit sich eine Form von Polymorphismus realisieren lässt.

Enttäuschend ist, dass nur wenige Anbieter wie Beckhoff und Phoenix Contact die grundlegenden OOP-Konzepte auch in STX umgesetzt haben.

Das Beispiel

Ein einfaches Beispiel soll erläutern, wie man SPS-Software so gestalten kann, dass sie eins zu eins durch eine C#-Proxy-Klasse dargestellt werden kann. Der zu steuernde Prozess besteht aus einem Greifer, der auf einer linearen Achse montiert ist. Der Greifer kann über die C#-Proxy-Klasse geöffnet, geschlossen und positioniert werden. Weil die Greiferposition nach dem Einschalten nicht bekannt ist, muss die Einheit erst initialisiert werden. Während der Initialisierung wird die Achse mithilfe eines Sensors referenziert, um die absolute Position des Greifers feststellen zu können. Die Achse fährt dabei langsam in die Richtung des Sensors und setzt die ►



Beispieleinheit: Greifer auf einer linearen Achse (Bild 3)

absolute Position des Greifers auf 0, sobald der Sensor detektiert wird (Bild 3).

Das Beispiel wird in Strukturiertem Text auf einer SPS von Beckhoff realisiert, weil dieser Lieferant wie zuvor erwähnt die OOP-Basiskonzepte in Strukturiertem Text unterstützt, wodurch diese Lösung PC-Software am ehesten ähnelt.

Die Schnittstelle der SPS-Softwarelösung besteht aus Befehlen und Eigenschaften, die einfach mit einer Klasse in C# abgebildet werden können. Es handelt sich um eine pure Datenschnittstelle, die über OPC UA zyklisch gepollt wird. Eventuelle Ereignisse wie Änderungen von Eigenschaften werden durch die C#-Proxy-Klasse generiert. Details hierzu finden Sie in einen späteren Abschnitt dieses Artikels.

Jeder Befehl hat die gleiche Basisstruktur mit einem *Execute*-, *Status*- und *ErrorID*-Feld, wie in Bild 4 gezeigt. Ein Befehl kann nur ausgeführt werden, wenn der Status *Idle* ist. Kann der Befehl nicht ausgeführt werden, ist sein Zustand *Locked*. Ein Befehl wird mit dem *Execute*-Flag gestartet. Die Dauer der Ausführung eines Befehls hängt dabei vom jeweiligen Prozess ab und kann mehrere Minuten dauern. Wurde der Befehl vollständig ausgeführt, wechselt der Zustand von *Busy* auf *Done*. Tritt während der Ausführung jedoch ein Fehler auf, wechselt der Zustand auf *Error*, und das *ErrorID*-Feld enthält den Fehlercode. Die Ausführung eines Befehls kann je nach Befehl durch die parallele Ausführung des Befehls *Cancel* unterbrochen werden. Der neue Zustand des Befehls wird dann auf *Canceled* gesetzt, um anzugeben, dass die Ausführung unterbrochen wurde. Das Endresultat des Befehls (Status mit eventueller *ErrorID*) wird erst zurückgesetzt, wenn der Aufrufer die Information verarbeitet hat und das *Execute*-Flag auf *FALSE* setzt.

Die PLC-Software

Die kostenfreie Entwicklungsumgebung von Beckhoff ist in Visual Studio integriert. Genau wie bei der Programmierung mit C# gibt es bei der Beckhoff-IDE eine Solution mit einem oder mehreren Projekten (Bild 5).

Im Ordner *DUTs* (Data Unit Types) werden die Datentypen für die Schnittstelle der Einheit definiert. Als Erstes wird ein Enumerator angelegt, der die Zustände der Befehle abbildet:

```
TYPE ECmdState :
(
```

```
Idle,      // Command can be executed
Busy,      // Command is executing
Done,      // Command was successfully executed
Error,     // Command is in error state
Locked,    // Command is locked
Canceled   // Command was canceled while being busy
);
END_TYPE
```

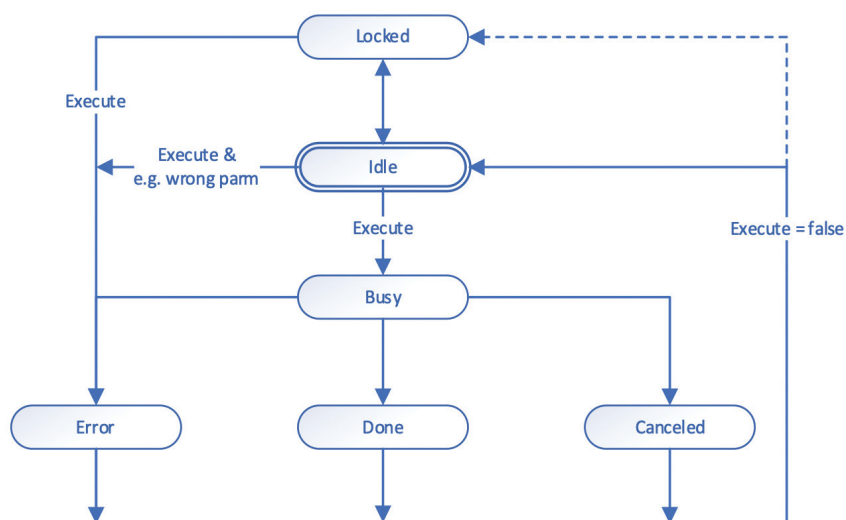
Die Basisklasse für jeden Befehl hat ein boolesches *Execute*-Feld zur Ausführung und einen *State* mit *ErrorID* zur Rückmeldung:

```
TYPE STCmd :
STRUCT
    Execute : BOOL;
    State : ECmdState;
    ErrorID : UDINT; // Identifier of error if command
failed
END_STRUCT
END_TYPE
```

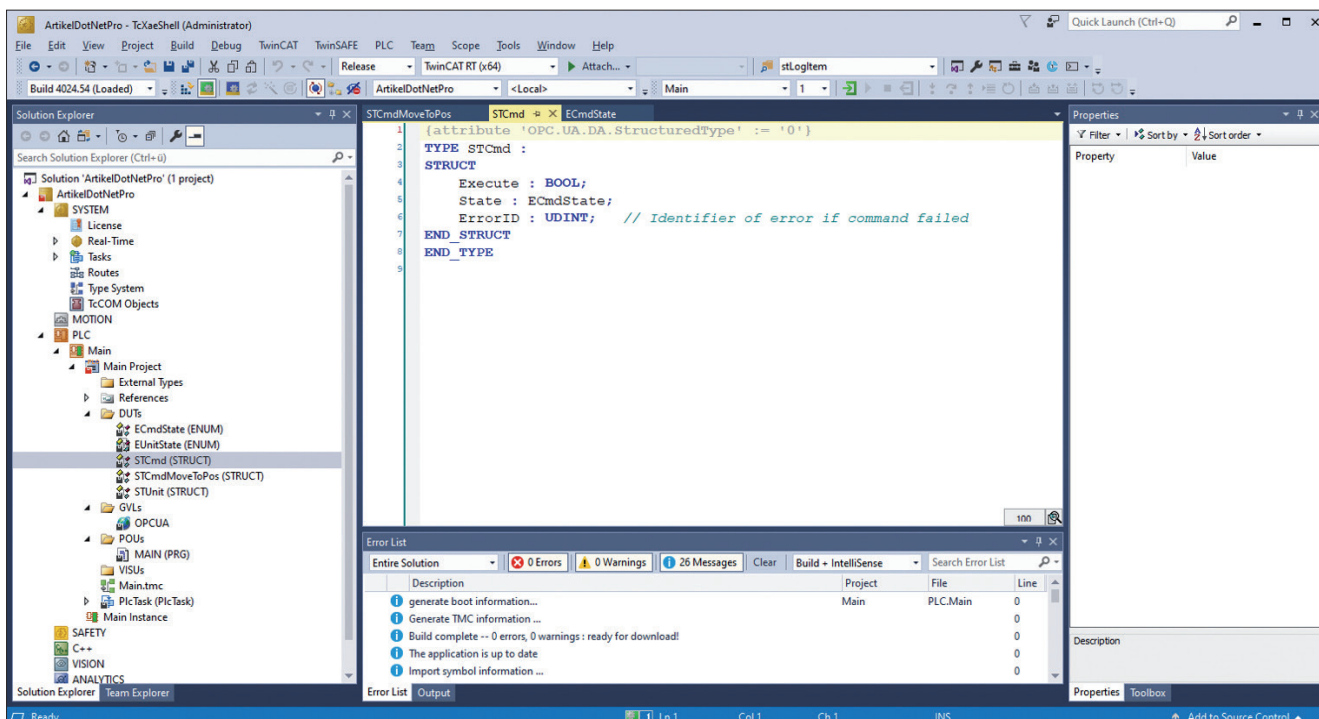
Wenn jetzt ein Befehl einen separaten Parameter und/oder Rückgabewert benötigt, wird eine dedizierte Struktur definiert, die vom *STCmd* ableitet. Der Befehl *CmdMoveToPos*, um den Greifer zu positionieren, ist ein derartiger Fall:

```
TYPE STCmdMoveToPos EXTENDS STCmd :
STRUCT
    Position : REAL; // Position to move gripper to
END_STRUCT
END_TYPE
```

Wie bereits erwähnt besteht die Schnittstelle der Einheit aus einer Datenstruktur. Die Einheit selbst hat ebenfalls einen Zustand (und einen Fehlercode). Die Reihenfolge der Zustän-



Mögliche Befehlszustände: Befehle können optional Parameter und/oder Rückgabewerte haben. Diese werden in einer Struktur definiert, die von der Befehls-Basisklasse abgeleitet wird (Bild 4)



Visual Studio als Basis für die Entwicklungsumgebung (Bild 5)

de spricht für sich; zu erwähnen ist hierbei, dass nach dem Zustand *ResettingError* die Einheit in den Zustand *Uninitialized* geht und neu initialisiert werden muss, um die Achse nach einem eventuellen Crash neu zu referenzieren:

```
TYPE EUnitState :
(
    Uninitialized, // Unit is powered on and not
                  // initialized
    Initializing, // Unit is initializing
    Operation,    // Unit is initialized and
                  // operational
    Error,        // Unit is in error state. ErrorID
                  // holds the reason.
    ResettingError // Unit is resetting error
);
END_TYPE
```

Die Datenstruktur, welche die Schnittstelle der Einheit definiert, sieht letztlich folgendermaßen aus:

```
TYPE STUnit :
STRUCT
    Version : STRING;
    State : EUnitState;
    ErrorID : UDINT;
    CurrentPosition : REAL;
    IsGripperClosed : BOOL;
    CmdCancel : STCmd;
    CmdCloseGripper : STCmd;
    CmdErrorReset : STCmd;
    CmdInitialize : STCmd;
```

```
CmdMoveToPos : STCmdMoveToPos;
CmdOpenGripper : STCmd;
END_STRUCT
END_TYPE
```

Das Feld *Version* wird hierbei verwendet, um die Kompatibilität der Schnittstelle von Server und Client zu gewährleisten.

Im Ordner *GVLs* (Global Variables Lists, globale Variablenlisten) wird eine Instanz *Unit* vom *STUnit* als globale Variable definiert. Auf diese Variable wird sowohl von außen über OPC UA durch die C#-Proxy-Klasse zugegriffen als auch von innen durch den PLC-Code im Verzeichnis *POUs* (Programming Organization Unit, Programmierobjekte). Die eigentliche Implementierung der Befehle und das Setzen der Eigenschaften der Instanz *Unit* sind für diesen Artikel nicht relevant und werden daher nicht weiter erklärt.

Zwischenstand

Anhand eines einfachen Beispiels wurde bisher die Implementierung des Echtzeit-Anteils der Software mit einer Beckhoff-SPS erläutert. Angewendet wurde dabei eine Softwarearchitektur, die die OOP-Basisansätze verkörpert. Die Datenschnittstelle ist so aufgebaut, dass sie intuitiv mit einer C#-Proxy-Klasse abgebildet werden kann.

Nun widmen wir uns dem OPC-UA-Protokoll, einem etablierten Industriestandard für den plattformunabhängigen Datenaustausch, mit dessen Hilfe die Daten zwischen der SPS einerseits und der C#-Software auf der anderen Seite ausgetauscht werden. Gezeigt wird, wie sich mit einem kommerziellen C#-OPC-UA-Client eine Proxy-Klasse implementieren lässt, die es ermöglicht, die Funktionalität in der SPS so auszuführen, als wäre sie in C# selbst programmiert. ►

Was ist eigentlich die OPC UA?

OPC UA ist das Kürzel für Open Platform Communication Unified Architecture und definiert einen Standard für plattformunabhängigen Datenaustausch für die Industrie. Der Standard wurde durch die OPC Foundation geschaffen; 1994 gegründet, zählen die führenden Automatisierungsfirmen zu ihren Mitgliedern. Ihre hauptsächliche Aufgabe ist es, Standards für die offene Konnektivität von industriellen Automatisierungsgeräten und -systemen zu schaffen und zu pflegen.

Die 2008 veröffentlichte erste OPC-UA-Version löst OPC Classic ab. OPC Classic hatte viele eigenständige Teilbereiche – zum Beispiel DA (Data Access), AE (Alarm and Events) oder HDA (Historical Data Access) – und nutzte Microsoft OLE oder COM/DCOM als Kommunikationstechnologie. OPC UA verwendet einen eigens entwickelten plattformübergreifenden Kommunikationsstack, der auf allen Betriebssystemen wie Linux, iOS und Windows genutzt werden kann. Im Jahr 2008 erschienen die ersten Produkte von Beckhoff, SAP und Unified Automation mit OPC UA auf dem Markt.

Die OPC-UA-Spezifikation definiert verschiedene Transportarten wie SOAP über HTTP/HTTPS und ein Binärprotokoll über TCP. Letzteres ist für Servergeräte gedacht, die nicht über die notwendigen Ressourcen zur Implementierung von XML-Codierung und HTTP/SOAP verfügen. Eine SPS ist ein Beispiel eines solchen Geräts.

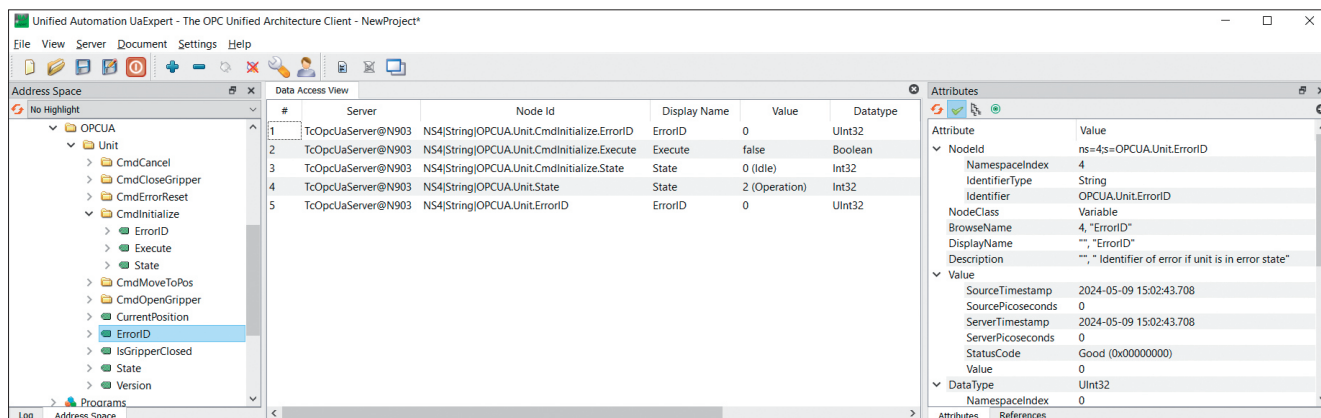
OPC UA definiert die folgenden Eigenschaften und Funktionalitäten:

- **Sicherheit:** Mehr dazu unten.
- **Internetfähigkeit:** OPC UA ist in der Lage, Daten über das Internet zu übertragen.
- **Lesen und Schreiben von Werttypen:** OPC UA ermöglicht dem Client das Lesen und Schreiben von Skalar-Werten, die an der OPC-UA-Schnittstelle des Servers, das heißt der SPS, als sichtbare Nodes definiert sind.
- **Lesen und Schreiben von Strukturen:** OPC UA erlaubt dem Client das Lesen und Schreiben von Strukturen, die an der OPC-UA-Schnittstelle des Servers, also der SPS, definiert sind. Abhängig von den Fähigkeiten des Treibers kann der Client die Felder der Strukturen dynamisch abfragen, ohne dass das Strukturlayout redundant im Client definiert werden muss.
- **Lesen und Schreiben von ein- oder mehrdimensionalen Arrays:** Ein-, zwei- oder mehrdimensionale Arrays eines Typs wie beispielsweise Skalare oder Strukturen können zwischen OPC-UA-Client und -Server übertragen werden.
- **Subscription (Wertüberwachung):** Um eine umfangreiche Kommunikation aufgrund von Polling zu vermeiden, ist ein Benachrichtigungsmechanismus auf dem Server implementiert, der die abonnierten Variablen lokal überwacht und den Client benachrichtigt, wenn sich ihr Wert ändert.
- **Node (Knoten) Browsing:** OPC UA bietet einen Mechanismus, um die Schnittstelle des Servers dynamisch zu durchsuchen.
- **Remote-Methodenaufrufe:** Der Client kann eine Methode des Servers aufrufen.
- **Historie:** Der Server lässt sich so konfigurieren, dass er die Historie der Werte eines einzelnen Knotens aufbewahrt. Diese Historie kann der Client abrufen.
- **Alarmierung und Ereignisverwaltung:** OPC UA bietet einen hochgradig konfigurierbaren Mechanismus zur Bereitstellung von Alarmen sowie Ereignisbenachrichtigungen für Clients.
- **Integration mit branchenspezifischen Standard-Datenmodellen:** Die OPC Foundation arbeitet mit mehreren Branchenverbänden zusammen, die spezifische Informationsmodelle für ihre Branchen definieren, um diese Informationsmodelle in UA zu unterstützen.

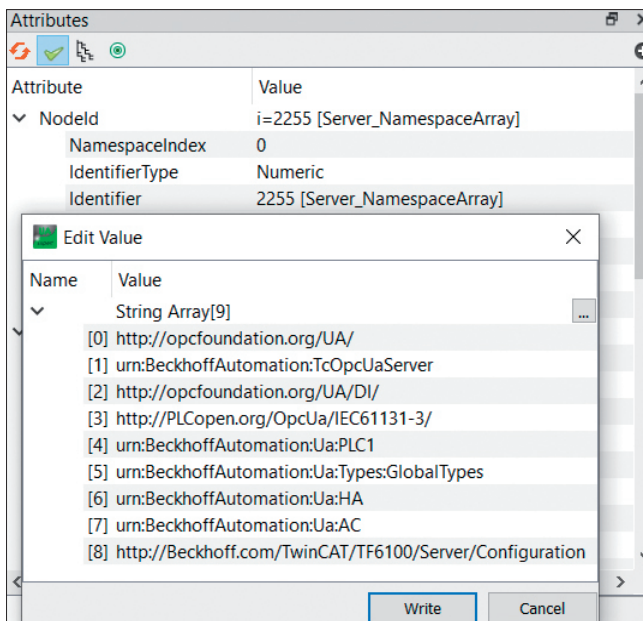
Sicherheit

Die Architektur von OPC UA integriert alle modernen Mechanismen zur Gewährleistung der Sicherheit. Die Sicherheit umfasst die Authentifizierung von Benutzern und deren Rollen, die Authentifizierung von OPC-UA-Clients, die Integrität und Vertraulichkeit der ausgetauschten Nachrichten sowie die Validierung von Funktionsprofilen.

Die Option, digital signierte X.509-Zertifikate zu verwenden, macht die Authentifizierung und Verschlüsselung der Kommunikation möglich. Da in unserem Anwendungsfall jedoch OPC UA als interner Bus des Schaltschranks verwendet wird, schießt diese Art der Absicherung über das Ziel hinaus und schafft zusätzliche Komplexität, die später im Feld zu Problemen führen kann. Erst wenn die OPC-UA-Kommunikation den Schaltschrank verlässt, etwa weil die Anlage an



Die OPC-UA-Schnittstelle des einfachen Artikel-Beispiels (Bild 6)



NodeID 2255 enthält alle verwendeten Adressräume (Bild 7)

die Cloud angebunden werden soll, sind obige Sicherheitsmechanismen zwingend zu implementieren. In unserem Beispiel aber wird auf gesicherte Kommunikation verzichtet.

OPC UA und die SPS

Bei einer SPS ist der OPC-UA-Server ins Betriebssystem integriert und kann durch die OPC-UA-Clients über TCP angesprochen werden. Dabei wird in der Regel der dedizierte Port 4840 verwendet, oder Port 4843 für die TLS/SSL-Verschlüsselung. Die SPS ist also im Standardfall der OPC-UA-Server.

Nahezu jedes Element im OPC-UA-Server wird durch einen „Node“ (Knoten) definiert. Nodes sind hierarchisch geordnet und haben Attribute, durch die Eigenschaften wie Zugangsrechte, Verlauf (HDA – Historical Data Access), Datentyp und Grenzwerte definiert werden. Jeder Node ist einem Adressraum (Namespace) zugeordnet und wird durch eine NodeID in diesem Adressraum identifiziert. Der NodeID-Typ ist entweder eine GUID (128 Bit), ein String, ein Integer (32 Bit) oder ein adressraumspezifisches Format, dargestellt als Byte-String.

Einer der wichtigen Vorteile, einen etablierten Standard wie OPC UA zu nutzen, ist die breite Unterstützung der großen OPC-UA-Community im Internet. So gibt es Dutzende kostenfreier OPC-UA-Viewer, mit denen man die Nodes eines OPC-UA-Servers durchstöbern und manipulieren kann.

Bild 6 zeigt die Schnittstelle des Beispiels vom Beginn des Artikels mit UaExpert, einem kostenfreien OPC-UA-Viewer der Firma Unified Automation. Im linken Teilfenster sieht man die Hierarchie der Nodes, im mittleren Hauptfenster einige Nodes und deren aktuellen Werte, und im rechten Teilfenster werden die Attribute des selektierten Nodes dargestellt. Die hier dargestellten NodeIDs sind alle vom Typ *String* mit einem leicht verständlichen Namen.

Erklärungsbedürftig ist der Namespace *Index 4*. Aus Effizienzgründen wird auf den Adressraum, in dem der Node de-

finiert ist, mit einem Index verwiesen, anstatt den Adressraumnamen als Zeichenkette permanent hin- und herzusenden. Die verwendeten Adressräume sind auf OPC-UA-Server-Ebene in einem Array definiert und können unter der NodeID 2255 abgefragt werden (Bild 7). Da die SPS-Konfiguration im Zeitverlauf möglicherweise geändert wurde, sollte dieser Knoten nach dem Aufbau einer neuen Verbindung immer zuerst abgefragt werden, um den aktuellen Namensraum-Index für den Adressraum zu erhalten.

OPC-UA-C#-Client

C# bietet von Haus aus keinen OPC-UA-Client an, und es ist eine Wissenschaft für sich, einen OPC-UA-Client selbst zu implementieren. Mögliche Alternativen sind der Open-Source-Client der OPC Foundation oder Produkte von Automatisierungsspezialisten wie zum Beispiel Unified Automation oder Traeger. Leider ist die Dokumentation der OPC Foundation recht lückenhaft. Zudem gilt, dass nur Corporate Member der OPC Foundation von der Freigabe ihres gesamten Quellcodes aufgrund der GPL-2.0-Lizenz befreit sind.

Für das Beispiel in diesem Artikel wird daher der gut dokumentierte OPC-UA-Client der Firma Traeger verwendet.

Herstellen der Verbindung

Damit man sich mit dem OPC-UA-Server verbinden kann, muss zuerst eine Instanz des *OpcClients* erstellt werden. Bei der Erstellung werden der Lizenzschlüssel und die Adresse vom Server übergeben. Das Format einer OPC-UA-Server-Adresse ist wie folgt: *opc.tcp://IPAdresse:4840* (zum Beispiel *opc.tcp://192.168.1.30:4840*)

Die *OpcClient*-Instanz kann jetzt optional für die gewünschte Sicherheit konfiguriert werden. Ein „KeepAlive Interval“ definiert die Intervallzeit für das zyklische Pinggen des Servers, um zu kontrollieren, ob die Verbindung noch besteht und funktionstüchtig ist.

Im *Connect()* der Beispielklasse wird zunächst das *Connect()* des *OpcClients* aufgerufen. Wenn die Verbindung erfolgreich aufgebaut ist, wird der Knoten wie oben bereits erläutert mit den Adressräumen vom OPC-UA-Server gelesen. Alle Knoten unseres Beispiels befinden sich im Adressraum *urn:BeckhoffAutomation:Ua:PLC1*, der mit Index 4 übereinstimmt.

Als nächsten Schritt sollte man kontrollieren, ob die SPS im Running Mode ist. Die SPS kann sich gegebenenfalls im Konfigurations- oder Service-Mode befinden, in dem es nicht möglich ist, den SPS-Code auszuführen. Weil in diesem Sonderzustand die Programm-Knoten ebenfalls unerreichbar sind, reicht es aus, den Knoten *Version* der Struktur *Unit* zu lesen. Die Versionsinformation braucht man ohnehin, um zu verifizieren, ob die SPS-Softwareversion mit der Client-Software kompatibel ist. Eine Diskrepanz wäre möglich, wenn nur einer der beiden Kommunikationsteilnehmer aktualisiert wurde, etwa nach einem Bugfix.

Erwähnt sei noch, dass man damit rechnen muss, dass die SPS auch während des normalen Betriebs durch einen Software- oder Hardwarefehler in den Service Mode wechseln kann. Ein zyklisches Pollen des Versionsknotens würde ►

● Listing 1: Herstellen der Verbindung mit dem OPC-UA-Server

```
private int _opcUaDefaultNamespaceIndex { get; set; }
    = -1;

private string _defaultNamespace { get; set; }

private OpcClient _opcClient;

public bool IsConnected { get; set; }

public OpcUaClient(string serverUrl,
    string defaultNamespace)
{
    defaultNamespace = defaultNamespace;
    Opc.UaFx.Client.Licenser.LicenseKey =
        Constants.LicenseKey;
    opcClient = new OpcClient(serverUrl);
    // Setup security here...
    opcClient.KeepAlive.Interval =
        Constants.KeepAliveInterval;
}

public bool Connect()
{
    IsConnected = false;
    try
    {
        _opcClient.Connect();
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex);
    }

    if (_opcClient.State != OpcClientState.Connected)
    {
        return false;
    }

    // Get the namespace index for the default
    // namespace...
    OpcValue namespacesNode =
        _opcClient.ReadNode(2255, 0);
    if(namespacesNode.Value == null)
    {
        Debug.WriteLine($"Could not read namespaces " +
            "node'2255' of OPC-UA server");
        return false;
    }
    List<string> namespaces = new List<string>((
        string[])namespacesNode.Value);
    _opcUaDefaultNamespaceIndex =
        namespaces.IndexOf(_defaultNamespace);
    if (_opcUaDefaultNamespaceIndex < 0)
    {
        Debug.WriteLine($"Namespace " +
            "'{_defaultNamespace}' is not listed in " +
            "namespaces node '2255' of OPC-UA server");
        return false;
    }

    // Read PLC software version node to check if PLC is
    // running and if version is correct...
    OpcValue swVersionNode = _opcClient.ReadNode(
        "OPCUA.Unit.Version", _opcUaDefaultNamespaceIndex);
    if(swVersionNode.Value == null)
```

dieses Problem entdecken. Des Weiteren muss man damit rechnen, dass nach dem Einschalten der Anlage die SPS zunächst noch nicht erreichbar ist, weil sie erst aufstartet. Und schließlich müssen Verbindungsunterbrüche (etwa durch Neustarten der SPS) durch den OPC-UA-Client abgefangen werden können (Listing 1).

Lesen und Schreiben von Werten

Die Werte von Knoten werden mit *ReadNode()* und *WriteNode()* beziehungsweise *ReadNodes()* und *WriteNodes()* gelesen und geschrieben. Dabei unterscheidet man zwischen eingebauten und anwendungsspezifischen Datentypen.

Eingebaute Datentypen sind Basistypen und vergleichbar mit Werttypen in C#. Anwendungsspezifische Datentypen sind dagegen komplexer und entsprechen den Referenztypen in C#. Grundsätzlich sind in OPC UA von mehrdimensionalen Arrays bis zu Enumeratoren alle Typen möglich. Die Definition aller anwendungsspezifischen Datentypen ist über Node Browsing durch den Client zur Laufzeit erhältlich. Die

Erklärung des Datentypensystems in OPC UA würde jedoch den Rahmen dieses Artikels sprengen; Infos dazu liefern das Kapitel „ExtensionObject“ der OPC-UA-Spezifikation unter [2] sowie [3].

Eingebaute Datentypen werden im C#-Client als Typ *object* übergeben und mittels statischem Casting dem ursprünglichen Typ zugewiesen. Komplexe Typen sind in OPC UA sogenannte Extension Objects und können zur Laufzeit in C# mit dynamischem Casting ausgelesen und geschrieben werden. Wem das zu heikel ist, der kann die einzelnen Felder des komplexen Typs auch mit *ReadNodes()* oder *WriteNodes()* einzeln lesen und schreiben und so eine mögliche Runtime Exception verhindern.

Der Overhead beim Lesen und Schreiben von Werten mit OPC UA ist relativ groß. Ein Roundtrip mit hundert Werten dauert nur einige Millisekunden länger als ein Roundtrip mit nur einem Wert. Deshalb sollte man, wo immer es geht, alle Knoten sammeln und mit *ReadNodes()* oder *WriteNodes()* in einem Roundtrip übermitteln.

```

    {
        Debug.WriteLine($"Node 'OPCUA.Unit.Version' " +
            "cannot be read. Is the PLC running?");
        return false;
    }
    IsConnected = true;
    return true;
}

public void Disconnect()
{
    try
    {
        _opcClient?.Disconnect();
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex);
    }
    IsConnected = false;
}

public void Dispose()
{
    if (IsConnected)
    {
        Disconnect();
    }
    _opcClient?.Dispose();
    _opcClient = null;
}

```

Ausführen eines SPS-Befehls

Nachdem nun die Verbindung zur SPS hergestellt ist und wir wissen, wie wir Werte lesen und schreiben können, ist es an der Zeit, einen Befehl aus dem Beispiel vom Artikelbeginn in der SPS auszulösen. Zur Erinnerung: Jeder Befehl hat die Basisstruktur mit den Feldern *Execute*, *State* und *ErrorID*. Wenn der Befehl den Status *Idle* hat, kann er ausgelöst werden durch Setzen des Feldes *Execute*. Nach der Ausführung ist der Status entweder *Done*, *Error* oder *Canceled*. Im Fehlerfall kann die Ursache durch das Lesen des Feldes *ErrorID* ermittelt werden. Hat der OPC-UA-Client das Resultat verarbeitet, setzt dieser das *Execute*-Feld zurück auf *FALSE*, und der Status des Befehls wird entweder wieder *Idle* oder gegebenenfalls *Locked*.

Die Methode *ExecuteCmd()* hat als Parameter den Basis-knoten des Befehls und einen Wert, der verhindert, dass die Methode unendlich lange auf die Änderung vom Feld *State* wartet. Für den Befehl *CmdInitialize* ist *OPCUA.Unit.CmdInitialize* die Basis-ID aller Knoten (siehe **Bild 6** der OPC-UA- ▶

Listing 2: OPC-UA-Methodenaufrufe (Teil 1)

```

public enum EPlcResult
{
    Undefined,
    Ok,
    TargetNotConnected,
    PlcCommandFailed,
    TargetReturnedError,
    FunStateNotDoneOrErrorWithinTime,
    InternalError,
    Canceled
}

public class PlcResult
{
    public UInt32 ErrorId { get; set; }
    public EPlcResult Result { get; set; } =
        EPlcResult.Ok;
}

public OpcStatusCode WriteNode(string nodeId,
    object value)
{
    OpcStatus opcStatus;
    try
    {
        opcStatus = _opcClient.WriteNode(
            nodeId, _opcUaDefaultNamespaceIndex, value);
    }
    catch
    {
        return OpcStatusCode.BadUnexpectedError;
    }
    return opcStatus.Code;
}

public PlcResult ExecuteCmd(string cmdNode,
    int timeoutMs = 10000)
{
    PlcResult plcResult = new PlcResult();
    OpcStatusCode opcStatusCode;

    // Set execute flag of command
    opcStatusCode = WriteNode(
        $"{cmdNode}.Execute", true);
    if (opcStatusCode != OpcStatusCode.Good)
    {
        plcResult.Result = EPlcResult.PlcCommandFailed;
        return plcResult;
    }
    // Wait for command state to become 'Done',
    // 'Canceled' or 'Error' or watchdog timer expires
    DateTime watchdogTimeExpired =
        DateTime.UtcNow.AddMilliseconds(timeoutMs);
    do
    {
        OpcValue cmdStateNode = _opcClient.ReadNode(

```

Listing 2: OPC-UA-Methodenaufrufe (Teil 2)

```

    $"{cmdNode}.State", _opcUaDefaultNamespaceIndex);
    if (cmdStateNode.Value == null)
    {
        plcResult.Result = EPlcResult.PlcCommandFailed;
        break;
    }
    // Command execution failed in PLC (State is
    // error) -> get errorID from PLC.
    if ((EPlcCmdState)(int)cmdStateNode.Value ==
        EPlcCmdState.Error)
    {
        OpcValue cmdErrorId = _opcClient.ReadNode(
            $"{cmdNode}.ErrorID",
            _opcUaDefaultNamespaceIndex);
        if (cmdErrorId.Value == null)
        {
            plcResult.Result =
                EPlcResult.PlcCommandFailed;
            return plcResult;
        }
        plcResult.Result =
            EPlcResult.TargetReturnedError;
        plcResult.ErrorId = (UInt32)cmdErrorId.Value;
        break;
    }
    // Function state changed to 'Done' -> get
    // additional optional return parameters.
    if ((EPlcCmdState)(int)cmdStateNode.Value ==
        EPlcCmdState.Done)
    {
        plcResult.Result = EPlcResult.Ok;
        break;
    }
    // Function state changed to 'Canceled'
    if ((EPlcCmdState)(int)cmdStateNode.Value ==
        EPlcCmdState.Canceled)
    {
        plcResult.Result = EPlcResult.Canceled;
        break;
    }
    } while (DateTime.UtcNow < watchdogTimeExpired);
    if (DateTime.UtcNow >= watchdogTimeExpired)
    {
        plcResult.Result =
            EPlcResult.FunStateNotDoneOrErrorWithinTime;
    }
    // Reset execute flag of command
    opcStatusCode = WriteNode($"{cmdNode}.Execute",
        false);
    if (opcStatusCode != OpcStatusCode.Good)
    {
        plcResult.Result = EPlcResult.PlcCommandFailed;
        return plcResult;
    }
    return plcResult;
}

```

Schnittstelle). Der Rückgabewert ist eine Instanz der Klasse *PlcResult*, er beschreibt mit dem Enumerator *EPlcResult*, wie der Aufruf gelaufen ist, und besitzt ein Feld *ErrorID*, das optional die *ErrorID* des SPS-Befehls enthält.

Wie man am Code sehen kann, braucht die Ausführung eines Befehls einige Roundtrips: Setzen und Zurücksetzen des *Execute*-Feldes, Lesen des *Status* und im Fehlerfall Lesen der *ErrorID*. Wenn der Befehl noch zusätzliche Parameter hat, kommen noch ein oder zwei Roundtrips dazu. Um Race Conditions zu vermeiden, muss man nach dem Zurücksetzen des *Execute*-Feldes zusätzlich den Status lesen und warten, bis der Befehl wieder *Idle* oder aber *Locked* ist.

OPC-UA-Methodenaufrufe als Alternative

OPC UA erlaubt es, auf dem Server Methoden zu definieren, die durch den Client aufgerufen werden können. Da die Ausführung von SPS-Befehlen Minuten dauern kann, blockieren die Methodenaufrufe den Client entsprechend lange. Das UI der kommerziellen OPC-UA-Viewer friert dann ein, und der Viewer läuft in einen Time-out, wenn der Aufruf zu lange dauert. Entweder nimmt man das in Kauf und implementiert jeden Aufruf auf einem eigenen Thread, oder man implementiert asynchrone Patterns mit *Begin*- und *End*-Befehlen (Listing 2).

Eigenschaften und Ereignisse

Zusätzlich zu den Methoden, um SPS-Befehle auszuführen, verfügt die Proxy-Klasse über Eigenschaften wie *CurrentPosition* und *IsGripperClosed* sowie die dazugehörigen Wertänderungsereignisse. Es gibt zwei Vorgehensweisen, um die Prozesswerte in die Proxy-Klasse zu bekommen: Polling oder Subscriptions.

Wenn das Datenvolumen und die gewünschte Aktualisierungsrate im normalen Bereich liegen, ist es durchaus praktikabel, alle Prozesswerte zyklisch in einem Roundtrip zu lesen. Zyklisches Polling ist sehr robust, und wie bereits beschrieben geht durch den OPC-UA-Protokoll-Overhead viel Bandbreite verloren, weniger durch die Nutzlast. Trotzdem wird immer dann unnötig Bandbreite verschwendet, wenn sich die Prozesswerte nicht ändern.

Mit Subscriptions (auch bekannt als PubSub-Pattern oder Werteüberwachung) wird der Bandbreitenverlust verhindert. Hierzu wird für jeden Wert auf dem OPC-UA-Server ein Monitor installiert. Dieser Monitor sendet nur dann eine Benachrichtigung, wenn sich der Wert ändert. Diese Lösung hat allerdings einen Haken: Wenn der Client ausfällt oder neu gestartet wird, greifen die serverseitigen Monitore ins Leere und müssen serverseitig sauber aufgeräumt werden. Das Pro-

blem macht sich auch bemerkbar, wenn man während der Entwicklungsphase die Ausführung des C#-Client-Codes durch beispielsweise einen Breakpoint anhält. Das Subscription-System gerät durcheinander, und eine Fortsetzung der Ausführung des Client-Codes ist nicht mehr möglich.

Es sei noch erwähnt, dass weder Polling noch Subscriptions richtige Ereignisse erzeugen. Beim zyklischen Polling liegt es auf der Hand, dass zwischenzeitliche Wertänderungen verpasst werden. Aber auch bei Subscriptions wird der Wert zyklisch ermittelt, nur diesmal lokal durch den Monitor auf dem OPC-UA-Server. Ändert sich der Wert in einem Intervall einmal hin und zurück, wird diese Änderung auch mit Subscriptions verpasst.

Fazit

Dieser Artikel hat eine mögliche Lösung vorgestellt, um die Echtzeit-Einschränkungen von Windows und C# mit einer kommerziellen SPS zu umgehen. Der SPS-Code wurde so gestaltet, dass seine OPC-UA-Schnittstelle eins zu eins durch eine C#-Proxy-Klasse abgebildet werden kann.

Der C#-Code in diesem Artikel ist stark vereinfacht und erläutert nur die Grundprinzipien: Der C#-Client für die Produktionsumgebung kapselt die Methoden, Eigenschaften und Ereignisse in generischen Klassen und lässt so den objektorientierten Ansatz der SPS-Software voll zur Geltung kommen. In der produktiven Lösung befindet sich der kommerzielle Treiber hinter einer Fassade, damit er einfach aus-

getauscht werden kann. Alle Klassen implementieren Interfaces für die Testbarkeit, und alle Zugriffe auf die Funktionalität sind Thread-safe realisiert.

Nach der Lektüre dieses Artikels werden möglicherweise einige Leserinnen und Leser zu dem Schluss kommen, dass das Realisieren einer Hybrid-Echtzeitsteuerung mit einer SPS und C# keine triviale Aufgabe ist. Dieser Artikel zeigt jedoch die notwendigen Grundlagen auf und sollte damit den Einstieg in das Thema erleichtern. ■

- [1] *Strukturierter Text (STX) bei Wikipedia*, www.dotnetpro.de/SL2503SPS1
 [2] *OPC Foundation Documents, ExtensionObject*, www.dotnetpro.de/SL2503SPS3
 [3] *QuickOPC User's Guide and Reference, Data Types in OPC-UA*, www.dotnetpro.de/SL2503SPS3



Erik Stroeken

arbeitet als Expert Software Engineer bei Noser Engineering AG in der Schweiz. Der gebürtige Niederländer ist 35 Jahre mit Herz und Seele in der Softwareentwicklung im industriellen Umfeld tätig.

erik.stroeken@nosser.com

dnpCode

A2503SPS

DEV ACADEMY

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN

**3 TAGE
REMOTE/INHOUSE**

Design Patterns in C++

- Design-Prinzipien
- Dynamische Entwurfsmuster
- Statische Patterns
- Klassische Entwurfsmuster
- Moderne Patterns

TRAINING!



Klaus Iglberger
Trainer

OPEN PLATFORM COMMUNICATION UNIFIED ARCHITECTURE

Umstieg bei der Steuerung

Mit dem Maschinenpark nach OPC UA wechseln und bei .NET bleiben

Maschinen sprechen miteinander. Sie stimmen Prozessschritte ab, nehmen Anweisungen entgegen und melden ihren Zustand. Eine ihrer Sprachen ist die Open Platform Communication. Einst war der Standard fest an DCOM und damit an Windows gekoppelt. Seit die Unified Architecture sich am Markt durchsetzt, stehen alle Freiheiten moderner .NET-Entwicklung offen. Die OPC Foundation stellt sogar eine Referenzimplementierung in C# zur Verfügung [1].

Wir müssen reden

Industriegeräte tauschen ständig Information aus. Sie fragen nach Sensorwerten oder dem Arbeitsfortschritt. Sie erteilen einander Aufträge, und sie stoppen, sobald an einer anderen Stelle im Prozess ein Fehler auftritt.

Als Beispiel dient hier ein Windpark mit zwei Turbinen, stellvertretend für jeden beliebigen Maschinenpark. Mit ihren Sensoren überwachen die Windräder sich selbst, doch allein reparieren können sie sich natürlich nicht (siehe Bild 1).

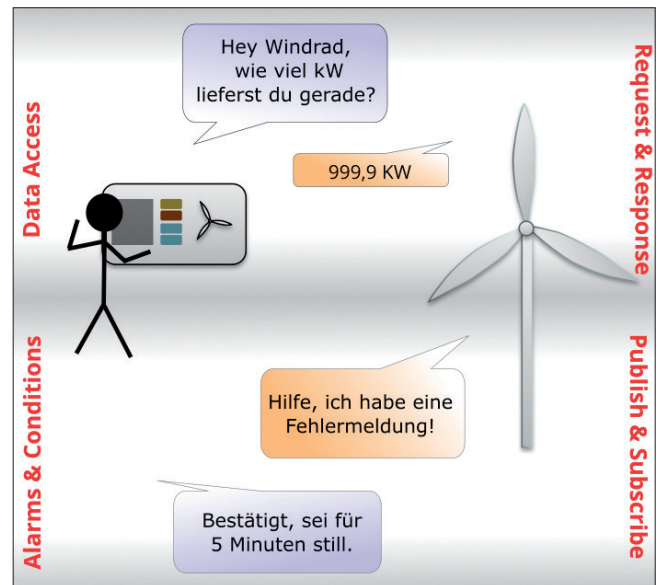
Deshalb müssen bevorstehender Wartungsbedarf und akute Schäden sofort an die Technikabteilung gemeldet werden. Letztere antwortet darauf mit Steuerbefehlen, die der Windpark umsetzen muss. Für den stabilen Netzbetrieb ist es außerdem wichtig, die aktuelle Einspeiseleistung genau zu kennen und sie gegebenenfalls herunterregeln zu können.

Die historische Verflechtung von OPC mit der Windows-Plattform

Bereits in den 1990ern gründeten einige Hersteller eine Taskforce, um die Steuerung von Geräten per Software zu vereinheitlichen. Sie nannten sich OPC Foundation und veröffentlichten einen Standard, der zunächst „OLE for Process Control“ hieß. OLE steht für „Microsoft Object Linking and Embedding“, denn der erste OPC-Standard basierte vollständig auf dem Component Object Model (COM) von Windows.

Erst als der Standard sich über Prozesssteuerungen hinaus etablierte, wurde er umbenannt in „Open Platform Communication“. Seitdem verlor Windows in hardwarenahen Systemen an Relevanz. Gleichzeitig entstanden flexiblere Netzwerkprotokolle, sodass heute niemand mehr DCOM in einer Systemarchitektur einplant.

Ursprünglich geht das COM auf Windows 3.1 zurück, wo es die Kommunikation zwischen lokalen Prozessen vereinfachte. Ein beliebiges Programm, häufig Word oder Excel, registrierte sich als COM-Komponente. Jedes andere Programm konnte daraufhin vom ersten Programm einen sogenannten COM-Server anfordern, um es per Remote Procedure Call (RPC) fernzusteuern. Mit Distributed COM (DCOM) wurde es noch komplizierter, denn in der zentralen Registry



OPC ermöglicht einzelne Aufrufe sowie Ereignisse (Bild 1)

durften sich auch Komponenten von entfernten Rechnern registrieren. Nach dem Prinzip der Ortstransparenz sollten Clients sich nicht darum kümmern, wo ihr Serverprozess läuft. Die Transparenz in Sinne „geplanter Ignoranz“ führte in der Praxis oft zu Durcheinander und vielen Sicherheitslücken.

Mittlerweile hat Microsoft sich von DCOM verabschiedet. Selbst sein Nachfolger, die Windows Communication Platform, wurde vor einigen Jahren wieder abgekündigt. Höchste Zeit also, die technologische Basis von OPC auszuwechseln.

Im Jahr 2006 veröffentlichte die OPC Foundation schließlich den Folgestandard Unified Architecture (UA). Er basiert auf TCP, wobei wahlweise ein binäres oder ein XML-Protokoll eingesetzt werden kann. Damit ist er plattformunabhängig und könnte sogar auf eingebetteten Steuergeräten laufen. Das alte OPC ist heute als „OPC Classic“ bekannt. In neuen Projekten sollte nur noch OPC UA zum Einsatz kommen.

Gründe für eine Migration von OPC Classic zu UA

Obwohl OPC UA seit mehr als 15 Jahren existiert, bauen viele produktive Systeme nach wie vor auf OPC Classic auf. Denn um die Software umzustellen, müssen alle beteiligten Geräte und Anwendungen OPC UA unterstützen. Bei der Steuerungssoftware für Maschinen, die über Jahrzehnte in der Produktion oder in Kraftwerken genutzt werden, ist kaum mit tiefgreifenden Updates zu rechnen.

Andere Geräte sind systemkritisch, sodass sie nicht für ein riskantes Upgrade mehrerer Softwarekomponenten abge-

schaltet werden können. In manchen Betrieben fehlt möglicherweise auch die Einsicht des Managements oder schlichtweg das qualifizierte Fachpersonal.

Wer zu spät mit der Umstellung beginnt, wird in Kürze jedoch vor Problemen stehen. DCOM läuft ausschließlich auf Windows-Betriebssystemen und ist anfällig für Konfigurationsfehler. Dabei sind brandneue Komponenten, die in den bestehenden Prozess eingebunden werden sollen, heute oft außen vor.

Abschied von DCOM

Zudem hat Microsoft im Jahr 2023 per automatisches Update „DCOM Hardening“ verpflichtend eingeführt. Die DCOM-Sicherheit abzuschalten, um sich nicht um deren Konfiguration kümmern zu müssen, ist seitdem nicht mehr möglich. Alte OPC-Anwendungen verlassen sich eventuell auf Letzteres und müssen schlimmstenfalls umprogrammiert werden.

Eine weitere Fehlerquelle für DCOM entsteht, wenn alte Rechner auf Windows Server 2019 aktualisiert werden. In einem Projekt beobachteten wir danach mysteriöse Timeouts

beim Lesen von Datenpunkten. Nur ein Downgrade auf Windows Server 2008 schien zu helfen.

Nach langer Analyse stellte sich heraus, dass alte OPC-Server Pakete senden, die größer als 1500 Byte sind. Unter Standardeinstellungen filtern Netzwerktreiber diese „Jumbo Packets“ aus, sofern man nicht ausdrücklich die Maximum Transfer Unit (MTU) vergrößert. So verschwand ein Teil von OPC Classic bereits auf Netzwerkebene.

In einem anderen Fall verschwand bei einem Update die registrierte Programm-ID eines OPC-Servers aus den Komponentendiensten. Die DCOM-Komponente war nur noch über ihre Class-ID ansprechbar, was eine Reihe von Workarounds in den Konfigurationen aller Clients zur Folge hatte.

Standardisiertes Sicherheitskonzept

Früher spielte Sicherheit in OPC keine große Rolle. Jedes Programm durfte den COM-Server starten und üblicherweise auch anonym alle Tags lesen. Aufmerksame Server-Hersteller forderten, dass der anonyme Client ein Passwort in ein bestimmtes Tag schrieb, bevor sie die Steuerkommandos ►

● Listing 1: Eine sichere Session aufbauen

```

UserIdentity identity = null;

if (!string.IsNullOrEmpty(user))
{
    // einfache Authentifizierung mit Name + Passwort
    identity = new UserIdentity(user, password);
}
else if (!string.IsNullOrEmpty(certificateName))
{
    // installiertes Zertifikat aus dem Store holen
    var store = new X509Store(StoreLocation.LocalMachine);
    store.Open(OpenFlags.ReadOnly);
    var certificate = var x509Certificates =
        store.Certificates
            .Find(X509FindType.FindByThumbprint,
                certificateName, false)
            .OfType<X509Certificate2>()
            .FirstOrDefault();

    if(certificate != null)
    {
        // mit diesem Zertifikat anmelden
        identity = new UserIdentity(certificate);
    }
}

// Anforderungen an den Endpunkt definieren
var config = new ApplicationConfiguration
{
    ApplicationName = "My Application",
    ApplicationType = ApplicationType.Client,

    CertificateValidator = new CertificateValidator {
        AutoAcceptUntrustedCertificates = true },
    ClientConfiguration = new ClientConfiguration {
        DefaultSessionTimeout = _options.SessionTimeout },
    SecurityConfiguration = new SecurityConfiguration {
        AutoAcceptUntrustedCertificates = true }
};

var endpointConfiguration = EndpointConfiguration.
    Create(config);

// alle Endpunkte es OPC-Servers anrufen
using var client = DiscoveryClient.Create(
    opcServerUri, endpointConfiguration);
var endpoints = client.GetEndpoints(null);

// daraus den Endpunkt wählen, der die eigenen
// Sicherheitsanforderungen erfüllt
var endpointDescription = CoreClientUtils.
    SelectEndpoint(opcServerUri, endpoints, useSecurity);
var endpoint = new ConfiguredEndpoint(null,
    endpointDescription, endpointConfiguration);

// am ausgewählten Endpunkt anmelden
var session = await Session.Create(
    config,
    endpoint,
    false,
    "My Application",
    0,
    identity,
    null);

```

zum Schreiben freigaben. Es fehlte jedoch ein einheitliches Vorgehen. Clients mussten mehrfach implementiert werden, weil die Sicherheitskonzepte von Servern unterschiedlicher Hersteller nicht ansatzweise kompatibel waren.

Am Beispiel des Windparks bedeutete klassische Sicherheit zum Beispiel, dass vor dem Stoppen einer Turbine des Herstellers Nordex die Tags *meinWindrad/username* und *meinWindrad/password* gefüllt werden mussten. Solange sie leer waren, durfte die aktuelle Session lediglich lesen. Falls der Hersteller aber Enercon war, so musste man zunächst eine Struktur aus Benutzername und zwei Zufallszahlen ins Tag *SessionRequest* schreiben und danach den erzeugten Schlüssel aus *SessionPubKey* lesen. Dann konnte man Steuerbefehle schreiben und sie am Ende freigeben, indem man den zuvor generierten Schlüssel in *SessionSubmit* schrieb.

In der Unified Architecture wurde Sicherheit von Anfang an mitgedacht. Das Netzwerkprotokoll basiert auf TCP, das heißt, TLS kann und sollte verwendet werden. Für den Aufbau einer Session muss der OPC-Client sich authentifizieren. Dafür besitzt er ein X.509-Zertifikat, das nach der Anmeldung auch zum Signieren und Verschlüsseln der gesamten Kommunikation verwendet wird. Für die Entwicklungsumgebung lässt sich die Sicherheit auch auf Benutzername und Passwort zurückstufen oder ganz abschalten. Wie das mit dem Referenz-Framework funktioniert, zeigt [Listing 1](#).

Authentifizierung ist damit Teil des Standards und für alle Gerätehersteller identisch. Jeder Client meldet sich bereits beim Verbindungsaufbau an. Ab da ist die Verbindung komplett verschlüsselt, und alle Berechtigungen stehen fest.

Flexibilität bei der Systemarchitektur

Für eine Verbindung ist nur der URL des OPC-Servers nötig. Client und Server müssen weder im gleichen Netzwerk noch auf dem gleichen Betriebssystem laufen. Auch einer Abkehr von Windows steht OPC nicht länger im Weg. Eine sorgfältige Firewall-Konfiguration vorausgesetzt, darf das Softwaresystem auf Container und Cloudservices verteilt werden.

Neu ist auch die Möglichkeit, eigene komplexe Datentypen mit Clients auszutauschen. Der Adressraum sieht einen

Zweig vor, in dem der Server alle Ereignis- und Objekttypen hinterlegen kann. Der Client holt die Definitionen nach dem Verbindungsaufbau ab und kann dann strukturierte Alarmer empfangen sowie ganze Objekte aus einem Node abrufen.

OPC UA als gemeinsame Sprache

In einem Windpark steht üblicherweise eine Technikhütte, die alle Windräder gemeinsam mit einem Datennetz verbindet. Der Windpark muss unterschiedliche Datenpunkte zur Verfügung stellen, etwa die technische Beschreibung seiner Turbinen und deren momentane Leistung. Solche Informationen werden bei Bedarf im sogenannten Request-Response-Verfahren abgefragt. Warnungen bei Schäden und Ausfällen hingegen können nicht warten, bis jemand sie abrufen. Sie werden im Publish-Subscribe-Verfahren sofort an alle interessierten Gegenstellen gesendet.

Wenn Fachkräfte entscheiden, ein Windrad aus- oder wieder einzuschalten, wird ein Rückkanal benötigt, der die Steueranweisung zielsicher an die richtige Turbine zustellt. Das heißt, jeder Knoten im Gerätenetz muss eindeutig adressierbar sein.

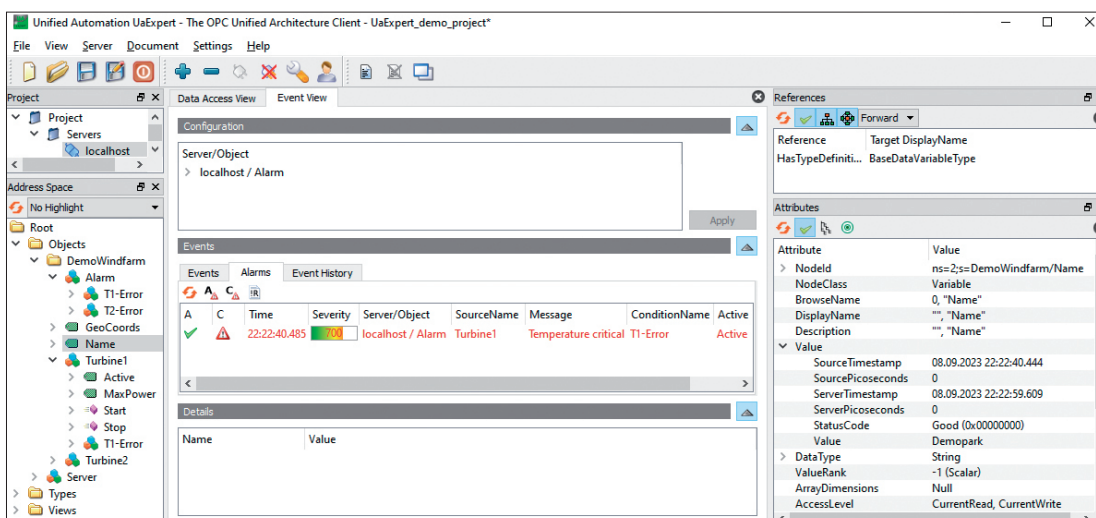
Diese Fälle werden von den OPC-Unterprotokollen Data Access (DA) beziehungsweise Alarms & Conditions (AC) abgedeckt. Es gibt zwei weitere Unterprotokolle, Historical Data Access (HA) und Programs (Prog), die weniger gebräuchlich sind und hier nicht näher betrachtet werden.

Der Adressraum präsentiert das Datenmodell eines OPC-Servers

Ein OPC-Server veröffentlicht einen eigenen Adressraum, in dem jeder noch so kleine Datenpunkt als Knoten repräsentiert ist. Meistens wird er als Baumstruktur dargestellt. In Wirklichkeit handelt sich allerdings um einen beliebigen Graphen, denn jeder Knoten darf mehrere Elternknoten besitzen.

Entsprechend seiner Funktion hat jeder Knoten einen Datentyp. So kapseln Attributknoten einen primitiven Wert, Ereignisknoten melden Alarmer, Methodenknoten lösen Aktivitäten aus und Objektknoten fassen andere Knoten zusammen.

Der Windpark lässt sich als Ordnerknoten modellieren, der Turbinen enthält, die wiederum Attribute und Ereignisse ha-



Ein einfaches Modell im Testclient UaExpert (eigenes Server-Beispiel, das Programm UaExpert kann kostenlos von Unified Automation bezogen werden) (Bild 2)

● Listing 2: Die Werte einiger Nodes lesen

```
public List<ReadResult> Read(IEnumerable<string>
    nodeIds)
{
    // Von jedem Node das Attribut "Value" anfordern
    var nodesToRead = new ReadValueIdCollection();
    nodesToRead.AddRange(nodeIds.Select(nodeId =>
        new ReadValueId
        {
            NodeId = nodeId,
            AttributeId = Attributes.Value
        }));

    session.Read(
        null,
        0,
        TimestampsToReturn.Neither,
        nodesToRead,
        out var dataValues,
        out var diagnosticInfos);

    // Die Ergebnisse mit den angefragten NodeIds
    // zusammenfügen
    return nodeIds.Zip(dataValues, (id, value) =>
        new ReadResult(id, value)).ToList();
}
```

ben. **Bild 2** zeigt, wie ein solches Modell in dem beliebten Testclient UaExpert aussieht.

Data Access für den gezielten Einzelzugriff

DA bedeutet, dass der Client einen Knoten abfragt und der Server mit dessen Inhalt antwortet. So kann der Adressraum im Request-Response-Verfahren durchsucht oder direkt ein bekannter Knoten abgefragt werden. Das löst den ersten Anwendungsfall: Der Windpark stellt auf Abruf seine Selbstbeschreibung zur Verfügung.

Um einen Knoten eindeutig zu adressieren, braucht er eine Adresse, seine sogenannte *NodeId*. Eine *NodeId* kann numerisch oder eine Zeichenkette sein. Häufig benutzt man dafür den Pfad im Adressraum, zum Beispiel *DemoWindfarm/Name* für die Variable mit dem sprechenden Namen des Parks.

Um Überschneidungen zu verhindern, gilt jede *NodeId* in genau einem Namespace. Dessen Index wird dem Knotennamen vorangestellt. So ergibt sich ein strukturierter Bezeichner für jeden Knoten, etwa *ns=2;s=DemoWindfarm/Name* oder *ns=2;i=5* für eine numerische Identität. Namenskollisionen mit vordefinierten Standardknoten oder zwischen unterschiedlichen Maschinen, die vom selben Server verwaltet werden, sind dadurch ausgeschlossen.

Über den Pfad eines Attributknotens kann der Client auch dessen Wert neu setzen. So könnte man bei *DemoWindfarm/Turbine1/MaxPower* etwa die maximale Leistung drosseln, in-

dem man den Wert von *MaxPower* von 100 auf 50 herabsetzt. Das kann notwendig sein, um die Netzfrequenz zu stabilisieren, wenn mehr Energie eingespeist als verbraucht wird.

Ein besonderes Maß an Interaktivität ermöglichen Methodenknoten. In **Bild 2** kann ein Client die vom Server veröffentlichten Methoden *Stop* und *Start* aufrufen, um das Windrad abzuschalten und später wieder zu starten. Damit löst DA auch den dritten Anwendungsfall: Der Windpark kann ferngesteuert werden.

Alarms und Conditions benachrichtigen über Ereignisse

Um Probleme sofort mitzubekommen, wird ein Publish-Subscribe-Verfahren benötigt. Dabei meldet sich der Client einmal für Ereignisse eines Knotens an, von da an wird er über jede Zustandsänderung aktiv informiert.

In **Bild 2** ist *Alarm* ein solcher Ereignisknoten. Er enthält von jeder Turbine einen Notifier. Dieselben Notifier *T1-Error* und *T2-Error* finden sich auch unterhalb der jeweiligen Turbine. Sie haben jeweils zwei Elternknoten – *Turbine* und *Windpark* –, damit Clients wahlweise nur eine Turbine oder den ganzen Park beobachten können. Beide Alarmereignisse sind normalerweise inaktiv. Nur wenn ein Fehlercode vorliegt, wird der Alarm aktiv. So löst AC den zweiten Anwendungsfall: Alle interessierten Clients werden im Problemfall sofort gewarnt. Die Unterprotokolle DA und AC sind also ausreichend, um einen Windpark fernzusteuern.

Neues bei Clients

Jedes OPC-Framework bringt eine primitive Client-Klasse mit, sodass man sich um Details wie den Aufbau einer Session kaum kümmern muss. Der größte Unterschied in der Handhabung zeigt sich erst beim Versuch, mit Classic oder UA dieselben Daten zu lesen.

In OPC Classic war es üblich, Tags (Datenpunkte, die Vorläufer von Nodes) zu gruppieren und dann die ganze Gruppe am Stück zu lesen. Genauso umfasste eine Subscription immer eine Gruppe von Tags, bei einer Datenänderung lieferten alle ihren aktuellen Wert.

Auch in OPC UA darf eine Session mehrere Subscriptions anlegen, eine reicht jedoch völlig aus. Um Änderungen zu verfolgen, wird der einen Subscription jeder zu überwachende Knoten einzeln hinzugefügt. Ein solches Monitored Item definiert eine *NodeId* als Startpunkt; dieser Node und alle seine Notifier werden verfolgt. Filter und Trigger legen fest, welche Änderungen – Qualität, Wert, Zeitstempel – dabei relevant sind.

Genauso darf jedes Attribut eines Knotens einzeln geschrieben werden. Üblicherweise betrifft das nur das Attribut *Value*, die übrigen enthalten Metadaten. Aber Vorsicht: Der Datentypen muss passen. Um den Typ eines Attributs zu ermitteln, muss es zuvor gelesen werden. Wie das mit dem kostenlosen Referenz-Framework [1] geht, zeigen **Listing 2** und **Listing 3**.

Neues bei Servern

Ein OPC-UA-Server besteht immer aus zwei Teilen: Die eigentliche Serverkomponente kümmert sich um Verbindungen und Authentifizierung, der sogenannte Node Manager ►

verwaltet den Adressraum. Die meiste Entwicklungsarbeit steckt im Node Manager, der die zu steuernden Maschinen auf ein Mesh von Knoten abbilden muss.

Beim Start baut er den Adressraum auf, das heißt, er stellt alle Datentypen, Ereignistypen, Knoten und deren Relationen bereit. Zur Laufzeit vermittelt der Node Manager zwischen Clients und Maschine. Statusänderungen von der Maschine werden in Attribute geschrieben, gegebenenfalls werden Ereignisse an die Clients gesendet. Schreibzugriffe der Clients werden entsprechend an die Maschine weitergereicht.

Das Referenz-Framework von der OPC Foundation [1] enthält ausführliche Beispiele für Node Manager und Server, auch kommerzielle Frameworks liefern Vorlagen für eigene Node Manager mit.

Überlegungen zur Auswahl eines Frameworks

Diverse Hersteller bieten kostenpflichtige Software Development Kits an, mit denen sich Clients sowie Server bequem implementieren lassen. Dazu zählen etwa die amerikanische Matrikon mit ihrem Flex SDK, die deutsche Traeger GmbH mit UaFx und auch Unified Automation.

Im praktischen Einsatz gelingt der Projektstart mit einem kommerziellen SDK zwar schneller als von Hand, beim anschließenden Test zeigen sich jedoch die Schwächen. Wenn das Framework sich unerwartet verhält, wünscht man sich einen schnellen Blick in den Quellcode, ist letzten Endes jedoch auf den langsamen Support des Herstellers angewiesen. Auch sind wichtige Klassen häufig sealed oder besitzen weder Konstruktor noch Interface, sodass sie sich nicht mit Unit-Testing-Tools „mocken“ lassen.

Wenn Abweichungen von der OPC-UA-Spezifikation auftreten oder der Standard unvollständig implementiert ist, bleibt einem nicht viel mehr übrig, als einen Bugreport einzureichen. Für die Version mit dem Bugfix gilt dann eventuell die zuvor gekaufte Lizenz nicht mehr. Bei einem langfristigen Entwicklungsprojekt, das über die gesamte Nutzungsdauer eines Maschinenparks gewartet werden muss, sollte man sich daher gut überlegen, ob man sich die Abhängigkeit von Closed-Source-Bibliotheken leisten kann.

Das Referenz-Framework

Wenn Sie ausreichend Zeit in die Entwicklung investieren können, lohnt sich ein Blick auf die Referenzimplementierung von der OPC Foundation [1]. Die Lernkurve bis zur ersten lauffähigen Anwendung ist etwas steiler, doch die mitgelieferten Samples und Quickstart-Beispiele helfen sehr. Besonders der Quickstart-Client lässt sich leicht kapseln und als handliche Client-Klasse ins eigene Projekt einbauen.

Falls Sie in der Referenzimplementierung einmal auf einen Bug stoßen, liegt es nahe, das Repository auf GitHub zu forken. Die OPC Foundation freut sich über Pull Requests aus der Community und bearbeitet sie innerhalb weniger Wochen.

Fazit

Der Umstieg von OPC Classic auf die Unified Architecture wird zusehends dringender. Allein schon um die Sicherheit zu gewährleisten, sollte man ihn nicht zu lange aufschieben.

Listing 3: Den Wert eines Nodes ändern

```
public ReadResult Write(
    string nodeId, string newValue)
{
    // Den erwarteten Datentyp ermitteln
    var before = Read(nodeId).DataValue.WrappedValue;
    var targetType = before.TypeInfo.BuiltInType;

    // Den neuen Wert konvertieren und in "Value"
    // schreiben
    var valueToWrite = new WriteValue();
    valueToWrite.NodeId = nodeId;
    valueToWrite.AttributeId = Attributes.Value;
    valueToWrite.Value.Value = ChangeType(
        newValue, targetType);
    valueToWrite.Value.StatusCode = StatusCodes.Good;

    session.Write(
        null,
        new WriteValueCollection { valueToWrite },
        out var results,
        out var diagnosticInfos);

    if (StatusCode.IsBad(results[0]))
    {
        throw new ServiceResultException(results[0]);
    }

    return Read(nodeId);
}
```

Dank der neuen Plattformunabhängigkeit kann ein OPC-Client zeitgemäß in einem Docker-Container laufen und dort mit Zugangsdaten aus dem Microsoft Key Vault versorgt werden.

Für einen schnellen Übergang bietet der Markt einiges an Werkzeugen, Middlewares und SDKs. Wer bereit ist, etwas Zeit zu investieren, kommt mit der Open-Source-Variante am Ende dennoch weiter und behält die volle Kontrolle über das eigene System. ■

[1] Referenzimplementierung der OPC Foundation in C#, www.dotnetpro.de/SL2503OPCUA1



Corinna John

arbeitet als Senior Software Engineer bei der adesso SE. Sie verfügt über mehr als 20 Jahre Berufserfahrung mit .NET. Der Schwerpunkt ihrer Tätigkeit liegt darin, alte Software in moderne Ökosysteme zu retten.

dnPCode

A2503OPCUA

SHOPIFY-PLUG-INS MIT .NET

Zusatzverkauf

Mit ShopifySharp steht eine Bibliothek zur Verfügung, die .NET-Entwicklern die Erweiterung des Vertriebssystems Shopify mit eigenen Applikationen ermöglicht.

Gedanklicher Auslöser für diesen Artikel war die Beobachtung, dass das zusätzliche Anbieten von Hardware für Softwareunternehmen verschiedenste Vorteile insbesondere in Bezug auf Kunden-Lock-in bringen kann. Das österreichische Start-up Runtastic ist – zumindest nach Ansicht des Autors – unter anderem auch deshalb so erfolgreich, weil es die hauseigene Handcomputerapplikation sehr schnell um ein Hardware-Angebot erweitert hat.

Dieser Artikel stellt deshalb die Bibliothek ShopifySharp [1] vor, die .NET-Entwicklern das Erweitern des Vertriebssystems Shopify mit eigenen Applikationen ermöglicht. Quasi nebenbei lernen wir dabei einige allgemein nützliche Werkzeuge kennen, die uns bei der lokalen Entwicklung von ASP.NET-Applikationen, die für das Internet vorgesehen sind, gute Dienste leisten.

Aufgrund der enormen Komplexität von Shopify dürfte klar sein, dass wir hier keine vollständige End-to-End-Applikation realisieren können. Ziel des Artikels ist vielmehr, die Grundlagen der Arbeit mit Shopify zu vermitteln, um die Leserinnen und Leser für eigene Experimente fit zu machen.

Exkurs: Vertrieb realer Produkte aus Applikationen

Bevor wir uns in diesem Artikel der eigentlichen Realisierung von Erweiterungen für Shopify zuwenden, sei darauf hingewiesen, dass Shopify für Android, iOS und JavaScript spezialisierte SDKs für den (hier als häufig angenommenen) Anwendungsfall anbietet, dass In-App-Stores realisiert werden sollen, über die Nutzer von Applikationen Hardware erwerben können. Die nach Ansicht des Autors effizienteste Vorgehensweise ist die Nutzung eines derartigen SDK, um entweder eine native Insel (für MAUI) oder eine Webapplikation zu realisieren und diese im Gerätebrowser oder in einer Webview zur Anzeige zu bringen.

Aus Funktionalitätssicht führt dies nicht zu einer Einschränkung: Der Vertrieb von Gütern ist sowieso nur bei bestehender Internetverbindung möglich, weshalb das Aufrufen einer Web-App keine Reduktion der Offline-Fähigkeiten darstellt. Weitere Informationen zu dieser Vorgehensweise

finden sich unter [2] – in diesem Bereich ist die Shopify-Dokumentation erfreulich detailliert.

Eine Applikation für Shopify erzeugen

Shopify-Erweiterungen liegen in Form von Plug-ins vor, die der Entwickler auf einem hauseigenen Webserver zu hosten hat. Zur Verfügung gestellt wird die Applikation durch das Erzeugen einer Manifestdatei, die man danach im Shopify-

System einpflegt. Unsere erste Aktion ist deshalb das Starten von Visual Studio 2022 – wichtig ist dazu, dass die Payload für die Webentwicklung mit den Standardeinstellungen vorinstalliert ist.

Im nächsten Schritt entscheiden wir uns für die Vorlage *ASP.NET Core Web App (Model View Controller)* und erzeugen ein neues Beispielprojekt – der Autor wird sich in den folgenden Schritten für den Namen *EMGShopifyTest1* entscheiden. Im Einstellungs-Assistenten wählen Sie als Ziel-Version *.NET 7.0* und belassen die restlichen Einstellungen so, wie sie sind. Nach der erfolgreichen Generierung des Projektskeletts folgt das Öffnen der Rubrik *NuGet-Pakete* verwalten, wo wir uns für das Paket *ShopifySharp* entscheiden – dieser Artikel wird auf der Version 6.3.3 des Produkts basieren.

Aufgrund der Einbindung durch einen öffentlichen URL ist ein Local Host Forwarder zwingend erforderlich. ngrok gilt in diesem Bereich als Goldstandard: Es

ist – siehe auch die Ankündigung unter [3] – seit einiger Zeit kostenlos nutzbar und weist kostenlosen Nutzern nicht mehr eine sich permanent verändernde Domain zu.

Unsere nächste Aktion ist deshalb das Öffnen des Frontends unter [4] – der Autor meldet sich im Interesse der Bequemlichkeit mit seinem Google-Konto an. Danach blendet das System auch schon den Link *Claim your free static domain!* ein, den Sie anklicken. Daraufhin erscheinen als Ergebnis die in **Bild 1** gezeigten Domain-Informationen.

Im nächsten Schritt laden Sie den ngrok-Client auf Ihren Rechner herunter – bei Abfassung dieses Artikels entspricht das der Datei *ngrok-v3-stable-windows-amd64.zip*. Extrahieren Sie die EXE-Datei und integrieren Sie sie, wenn gewünscht, in den *Path* der Workstation. ►





Der **Localhost Forwarder** exponiert einen Dienst Ihres Systems im globalen Internet (Bild 2)

Im nächsten Schritt gilt es einige der in der Anmeldeseite eingegebenen Informationen einzupflegen. Am wichtigsten ist die Anmeldung des Authentication-Tokens, über das sich der Dienst anmeldet. Da der Autor die Datei *ngrok.exe* nicht in den *Path* seines Rechners kopiert hat, ist beim Aufruf in der PowerShell die Zeichenkette `.\` voranzustellen:

```
PS C:\Users\tamha\Downloads\ngrok> .\ngrok config
add-authtoken
2UcdxRnVWQrGaQnkNoeuaDFJ4Sk_6AjL82PXEtMg6Qo25YCb6
Authtoken saved to configuration file:
C:\Users\tamha\AppData\Local\ngrok\ngrok.yml
```

Zu guter Letzt sollten Sie noch den eigentlichen Namen der Domain notieren – er lässt sich über einen nach folgendem Schema aufgebauten Kommandozeilenbefehl finden:

```
ngrok http
--domain=diverse-grossly-phoenix.ngrok-free.app 80
```

Im Hintergrund realisiert ngrok dabei die in Bild 2 gezeigte Systemarchitektur.

Im nächsten Schritt bringen Sie unser Programm in Visual Studio zur Ausführung. Achten Sie dabei darauf, als Kommunikationsprotokoll gewöhnliches HTTP auszuwählen. Sinn dieser Operation ist Zurücklieferung eines nach dem Schema `http://localhost:5224/` aufgebauten URL – dieser informiert uns darüber, unter welchem Port Visual Studio unsere ASP-Applikation ansprechbar macht.

Sofern sie – wie der Autor – mit Port 5224 arbeiten, lässt sich die Verbindung danach durch Eingabe des folgenden Befehls freigeben:

```
PS C:\Users\tamha\Downloads\ngrok> .\ngrok http
--domain=diverse-grossly-phoenix.ngrok-free.app 5224
```

Die Arbeitsumgebung informiert während der Ausführung wie in Bild 3 gezeigt über die Menge der übertragenen Informationen. Es ist empfehlenswert, diese Daten stets im Blick zu halten, um eventuelle Angreifer so schneller erkennen zu können. Einzelne URLs scheinen übrigens ebenfalls im Client auf, was eine grundlegende Überwachung des HTTP-Traffics ermöglicht.

Die Applikation lässt sich fortan durch Aufrufen des URL `https://diverse-grossly-phoenix.ngrok-free.app/` ansprechen. Beachten Sie, dass das System mitunter einen Warnhinweis einblendet, die weggeklickt werden muss. Zur Umgehung dieses Problems bietet sich der Umstieg auf einen monatlich abgerechneten Premiumtarif an.

Einrichtung eines Kontos bei Shopify

Die nächste Aktion ist das Aufrufen des URL `https://partners.shopify.com/organizations`, wo Sie sich abermals unter Nutzung eines Google-Kontos anmelden können. Entscheiden Sie sich im nächsten Schritt für die Option *Shopify-ID erstellen* und wählen Sie danach die Option *Neues Partnerkonto erstellen* aus. Sofern Shopify nach dem Verwendungszweck des neu zu erzeugenden Kontos fragt, entscheiden Sie sich für die Rubrik *Apps für den Shopify App Store entwickeln*. Nicken Sie die sonstigen Einstellungen dann ab und warten Sie, bis das Shopify-Backend die Konfiguration des Beispiel-Shops erfolgreich abgeschlossen hat.

Öffnen Sie danach die Rubrik *Shopify Partners | Apps* und klicken Sie auf *App erstellen*. Die Option *App manuell erstellen* erlaubt die Vergabe eines Namens, in den folgenden

```

ngrok (Ctrl+C to quit)

Take our ngrok in production survey! https://forms.gle/aXiBFwzEA36DudFn6

Session Status      online
Account             tam.hanna@gmail.com (Plan: Free)
Version             3.3.4
Region              Europe (eu)
Latency              -
Web Interface       http://127.0.0.1:4040
Forwarding           https://diverse-grossly-phoenix.ngrok-free.app -> http://localhost:5224

Connections      ttl    opn    rt1    rt5    p50    p90
                  0      0      0.00   0.00   0.00   0.00
  
```

Die Applikation ist exponiert (Bild 3)

Schritten verwenden wir *TamsEMGApp1*. Lohn der Mühen ist das Erscheinen eines Fensters mit Credentials. Diese sollten nach Maßgabe der Möglichkeiten sicher notiert werden, der Autor entschied sich für eine Klasse im Code-behind der Applikation.

Sodann folgt der Wechsel in *App Setup*, wo Sie im Feld *URL* folgende Zeile einpflegen:

```
https://diverse-grossly-phoenix.ngrok-free.app/shopify/
hallo
```

In die Textbox *Zulässige Weiterleitungs-URL(s)* wandern danach folgende URLs:

```
https://diverse-grossly-phoenix.ngrok-free.app/shopify/
authresult
https://diverse-grossly-phoenix.ngrok-free.app/shopify/
chargerresult
```

Als Nächstes folgt ein Klick auf *Vertriebsmethode auswählen*, wo wir uns für *Öffentliche Verteilung* entscheiden – die App soll jedem Store-Betreiber zur Verfügung stehen. Das Backend setzt Sie danach in einer anderen Einstellungsseite ab, weshalb es empfehlenswert ist, die Einstellungen vorher durch Anklicken des *Speichern*-Buttons zu sichern.

Implementierung von OAuth und Vertiefung der Verbindung

Die ASP.NET-Applikation wird von Microsoft mit verschiedenen Andockstellen für Erweiterungen ausgeliefert. Der erfolgreiche Verbindungsaufbau mit Shopify setzt aufseiten des Entwicklers dann ein wenig Mitarbeit voraus. Im ersten Schritt gilt es die Datei *Program.cs* zu öffnen und eine nach folgendem Schema aufgebaute Methode einzupflegen:

```
void ConfigureCookieAuthentication(
    CookieAuthenticationOptions options) {
```

Sich schnell änderndes Thema

Die praktische Erfahrung des Autors lehrt, dass sich der Aufbau der Shopify-Webseite mitunter über Nacht verändert – naturgemäß hat dies auch Auswirkungen auf APIs und Co. Eine überraschend gute und sehr preiswerte Quelle, um in Bezug auf Änderungen auf dem Laufenden zu bleiben, ist das unter [5] erwerbbar Handbuch. Das Werk wird vom Maintainer der *ShopifySharp*-Bibliothek gepflegt und erfährt bei Änderungen an der Shopify-Struktur üblicherweise binnen kürzester Zeit Updates.

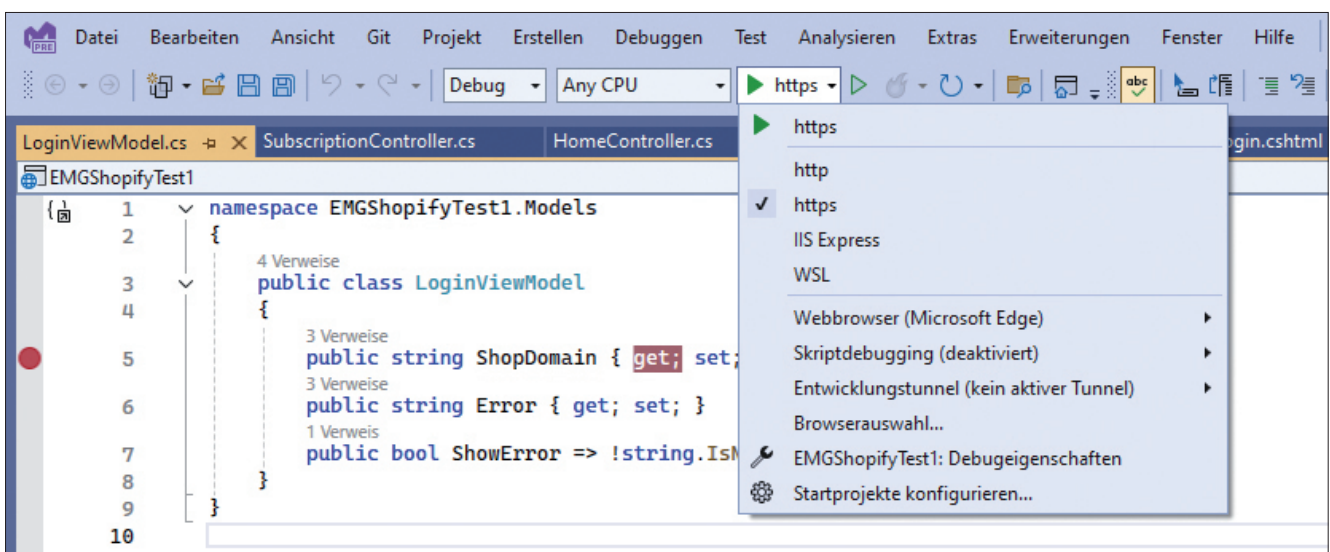
```
options.Cookie.HttpOnly = true;
options.SlidingExpiration = true;
options.ExpireTimeSpan = TimeSpan.FromDays(1);
options.LogoutPath = "/Auth/Logout";
options.LoginPath = "/Auth/Login";
options.AccessDeniedPath = "/Auth/Login";
options.Validate();
}
```

Aufgabe der Methode ist die Überprüfung der Cookie-Informationen. Das *True*-Setzen von *SlidingExpiration* sorgt dabei dafür, dass jede Nutzung der Applikation die Lebensdauer der im Cookie gespeicherten Informationen verlängert.

Wichtig sind außerdem die drei *Path*-Zuweisungen, da sie über die Weiterleitungspfade entscheiden, die den verschiedenen Zuständen zuzuordnen sind.

Im nächsten Schritt müssen die Services zum von ASP.NET angelegten Request-Container hinzugefügt werden. Nur auf diese Weise ist sichergestellt, dass die Services für die Applikationskomponenten zur Verfügung stehen.

```
builder.Services.AddControllersWithViews();
```



Visual Studio unterstützt Entwickler bei der Zielauswahl (Bild 4)


```
var authScheme =
    CookieAuthenticationDefaults.AuthenticationScheme;
builder.Services.AddAuthentication(authScheme).AddCookie(
    ConfigureCookieAuthentication);
```

Die in vielen Beispielen als separate *Configure*-Methode angelegte Konfiguration des ASP.NET-Serververbunds beginnt in von aktuellen Versionen von Visual Studio generierten Projektskeletten nach folgendem Schema mit der Festlegung der im Fehlerfall zu aktivierenden Methoden:

```
if (app.Environment.IsDevelopment()) {
    app.UseDeveloperExceptionPage();
}
else {
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}
```

Besonders interessant ist hier die Verwendung von *app.Environment.IsDevelopment()*: Der zurückgelieferte Wert erlaubt der Applikationslogik die Reflexion darüber, ob die Programmausführung derzeit im Debug- oder im Release-Modus aktiviert ist.

In der eigentlichen Applikationskonfiguration, die darunter erfolgt, ist das Hinzufügen der beiden folgenden Methoden erforderlich:

```
app.UseStatusCodePages();
app.UseAuthentication();
```

Die eigentliche Authentifikation der Benutzer soll unter Nutzung eines von Shopify vorangetriebenen OAuth-Prozesses erfolgen. Da es sich dabei um einen vor allem von der Shopify-Seite gesteuerten Prozess handelt, muss unsere Applikation lediglich die Domain des Shops und ein Flag mit Fehlerinformationen vorhalten. Im *Models*-Ordner der ASP.NET-Applikation kommt das folgende vergleichsweise einfache *LoginViewModel* unter:

```
public class LoginViewModel {
    public string ShopDomain { get; set; }
    public string Error { get; set; }
    public bool ShowError => !string.IsNullOrEmpty(Error);
}
```

Da eines der Fernziele unserer Applikation auch die Realisierung eines Vertriebs für Abonnements ist, sind zwei weitere Modelle interessant: Erstens das *SubscribeViewModel*, das darüber informiert, welchen Zustand das Abonnement des gerade aktivierten Benutzers aufweist:

```
public class SubscribeViewModel {
    public string Error { get; set; }
    public bool ShowError =>
        !string.IsNullOrEmpty(Error);
}
```

Der eigentliche Vertrieb erfolgt dann über das *SubscriptionViewModel*, dessen Konstruktor folgendermaßen beginnt:

```
public class SubscriptionViewModel {
    public SubscriptionViewModel(RecurringCharge charge) {
        ChargeName = charge.Name;
        Price = charge.Price.Value;
        TestMode = charge.Test == true;
        DateCreated = charge.CreatedAt.Value;
        TrialEndsOn = charge.TrialEndsOn;
    }
}
```

Hervorhebenswert ist dabei die Nutzung des *RecurringCharge*-Objekts: einer in *ShopifySharp* angelegten Klasse, die Entwickler:innen das Zusammenfassen aller zu einem Abonnement gehörenden Informationen ermöglicht. Der Rest des Codes ist gewöhnliche C#-Logik, die sich um das Exponieren der diversen Eigenschaften kümmert:

```
public string ChargeName { get; }
public decimal Price { get; }
public bool TestMode { get; }
public DateTimeOffset DateCreated { get; }
public DateTimeOffset? TrialEndsOn { get; }
public bool IsTrialing => TrialEndsOn.HasValue;
}
```

An dieser Stelle bietet sich der Wechsel in die Datei *Start.cshtml* an, die bei Bedarf unter *Views | Subscription* anzulegen ist:

```
@model SubscribeViewModel
@{
    ViewData["Title"] = "Subscribe";
}
<div class="text-center">
    <h2>Activate Subscription</h2>
    <hr />
</div>
<form method="post" asp-controller="Subscription"
    aspaction="HandleStartSubscription">
    @if (Model.ShowError)
    {
        <p class="red error">@Model.Error</p>
    }
    <button type="submit" class="btn btn-primary">
        Subscribe to app
```

● Auf Sicherheit achten

So nützlich ein Tunnel ist, so zweckdienlich ist er auch für angriffswillige Hacker – achten Sie darauf, Antivirenprogramme zu aktivieren, eine lokale Firewall zu nutzen und ganz allgemein die Best Practices der Computersicherheit einzuhalten.

```
</button>
</form>
```

Da wir die Klasse weiter oben um ein Convenience-Attribut für die Interaktion beziehungsweise Erkennung eines Fehlerzustands ergänzt haben, ist die Auswahl der anzuzeigenden Teile des Markups in der Datei vereinfacht.

Aus dem Vorhandensein der Markup-Datei folgt, dass logischerweise für sie auch ein ViewModel erforderlich ist. Im Fall des vorliegenden Programms lässt sich dieses beispielsweise nach folgendem Schema aufbauen:

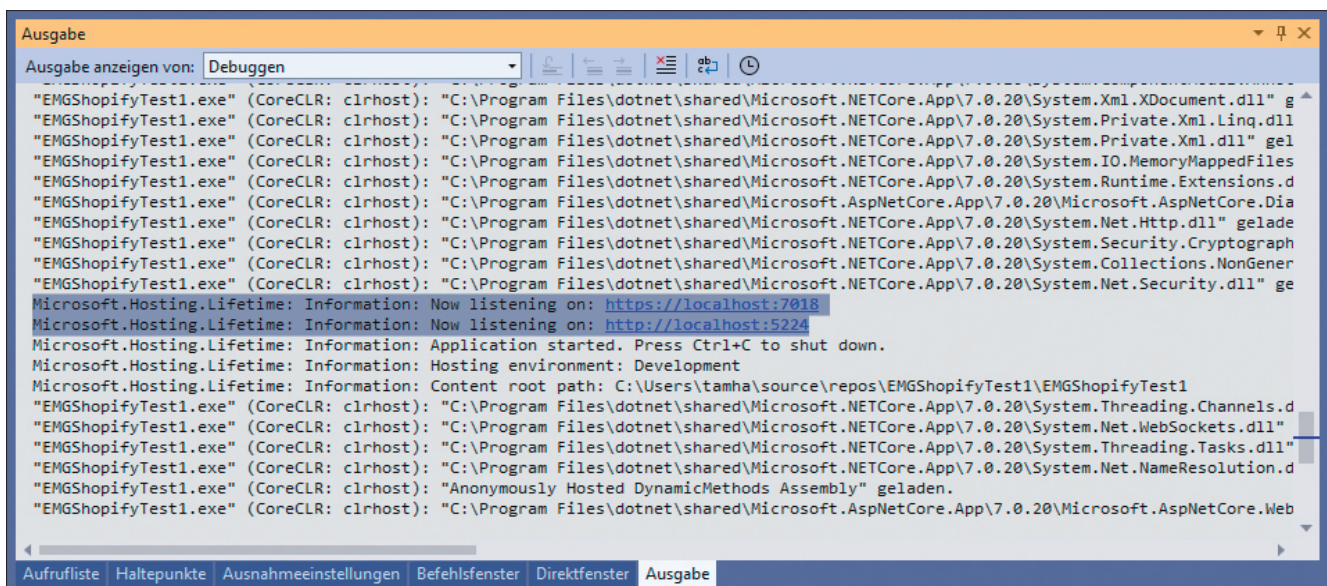
```
@model SubscriptionViewModel
@{
    ViewData["Title"] = "Your Subscription";
    var formattedPrice = "$" + Model.Price.ToString("f2");
}
```

derlich, die nach folgendem Schema mit der Inklusion der Modelldeklaration und dem Erzeugen eines Eingabefelds für den URL des anzusprechenden Shopify-Shops beginnt:

```
@model LoginViewModel

@{
    ViewData["Title"] = "Login";
}

<div class="text-center">
    <h2>Login with Shopify</h2>
    <hr />
</div>
<form method="post"
    enctype="application/x-www-form-urlencoded"
    asp-controller="Auth" asp-action="HandleLogin">
```



Die startende Visual-Studio-Instanz informiert über die für die verschiedenen Protokolle verwendeten Ports (Bild 5)

```
<div class="text-center">
    <h2>Your Subscription Details</h2>
    <h5>
        @if (Model.IsTrialing) {
            <span class="badge badge-pill badge-primary">
                Free Trial</span>
        }
    </h5>
    ...
</div>
```

Aus Platzgründen drucken wir hier nicht den vollständigen Code ab – es geht lediglich darum, die diversen im Modell enthaltenen Informationen in HTML-Markup umzuwandeln.

Für die Inbetriebnahme des Login-Prozesses kehren wir an eine bereits etwas zurückliegende Stelle zurück und erinnern uns an die Konfiguration der Cookie-Klasse. In den beiden *Path*-Attributen wurden Pfade angelegt, die nun zu beleben sind. Konkret ist die Datei `\Models\LoginViewModel.cs` erforder-

```
<div class="form-group">
    <label for="shop">Login-Domäne</label>
    <input id="shop" name="shop" class="form-control"
        type="text" value="@Model.ShopDomain" required />
</div>
```

Das Hinzufügen eines „Komfort-Flags“ im Modell erleichtert uns auch hier die Abfrage, ob es während der Programmausführung zu einem Fehler kam:

```
@if (Model.ShowError) {
    <p class="red error">@Model.Error</p>
}
<button type="submit" class="btn btn-primary">
    Einloggen
</button>
</form>
```

Auch für die Login-View ist danach ein Controller erforderlich. Er muss auf den Namen *Controllers\AuthController.cs* hören, weil wir ja weiter oben nach dem Schema */Auth/Login* aufgebaute Endstellen für die Implementierung des OAuth-Prozesses avisiert haben.

Zunächst implementieren wir den Logout-Prozess, der vor allem die *HttpContext*-Hilfsmethode zum Trennen der Verbindung aufrufen muss:

```
public class AuthController : Controller {
    [HttpGet]
    public async Task<ActionResult> Logout() {
        await HttpContext.SignOutAsync();
        return RedirectToAction("Login");
    }
}
```

Die für die Anmeldung vorgesehene Route präsentiert sich dann konzeptuell ähnlich:

```
[HttpGet]
public ActionResult Login([FromQuery] string shop =
    null) {
    return View(new LoginViewModel
    {
        ShopDomain = shop
    });
}
```

Interessant ist der Code, der bei der Aktivierung der Routine zur Ausführung gebracht wird. Seine erste Aktion besteht darin, eine Überprüfung der angelieferten Werte durchzuführen und einige Hilfselemente ins Leben zu rufen:

```
[HttpPost]
public async Task<ActionResult>
    HandleLogin([FromForm] string shop) {
    if (!await
        AuthorizationService.IsValidShopDomainAsync(
            shop)) {
        return View("Login", new LoginViewModel {
            ShopDomain = shop,
            Error = $"Parameter {shop} ist ungültig!"
        });
    }
}
```

● Cookies sind unsicher

Zu beachten ist, dass Cookies – logischerweise – auf dem Client gespeichert werden und somit für Angreifer auslesbar und manipulierbar sind. Im Interesse der Sicherheit sei hier nochmals angemerkt, dass das Ablegen von Session Tokens, kryptografischen Schlüsseln und ähnlichen sensiblen Informationen in Cookies ein sicherheitstechnisches Antipattern erster Ordnung darstellt.

```
var requiredPermissions = new[] { "read_orders" };
TamsPublic.myOAppState = new OAuthState {
    DateCreated = DateTimeOffset.Now,
    Token = Guid.NewGuid().ToString()
};
```

Der nächste Schritt ist das Zusammenstellen des OAuth-URL. Zu beachten ist, dass der Basis-URL naturgemäß von dem String abgeleitet werden muss, der in seiner Konfiguration des ngrok-Weiterleitungsdiensts gültig ist. Das Feld *TamsPublic.SpotPasskey* kümmert sich um die weiter oben angelegten kryptografischen Primitiva, während der letzte Parameter einfach eine Instanz der Klasse *TamsPublic.myOAppState* aufnimmt:

```
string myOAuthUrl =
    "https://diverse-grossly-phoenix.ngrok-free.app/" +
    "/shopify/authresult";
var oauthUrl =
    AuthorizationService.BuildAuthorizationUrl(
        requiredPermissions,
        shop,
        TamsPublic.SpotPasskey,
        myOAuthUrl,
        TamsPublic.myOAppState.Token);

return Redirect(oauthUrl.ToString());
}
```

Im Hintergrund ist *myOAppState* ein nach folgendem Schema aufgebauter Wrapper um die diversen Attribute, die für eine OAuth-Session erforderlich sind:

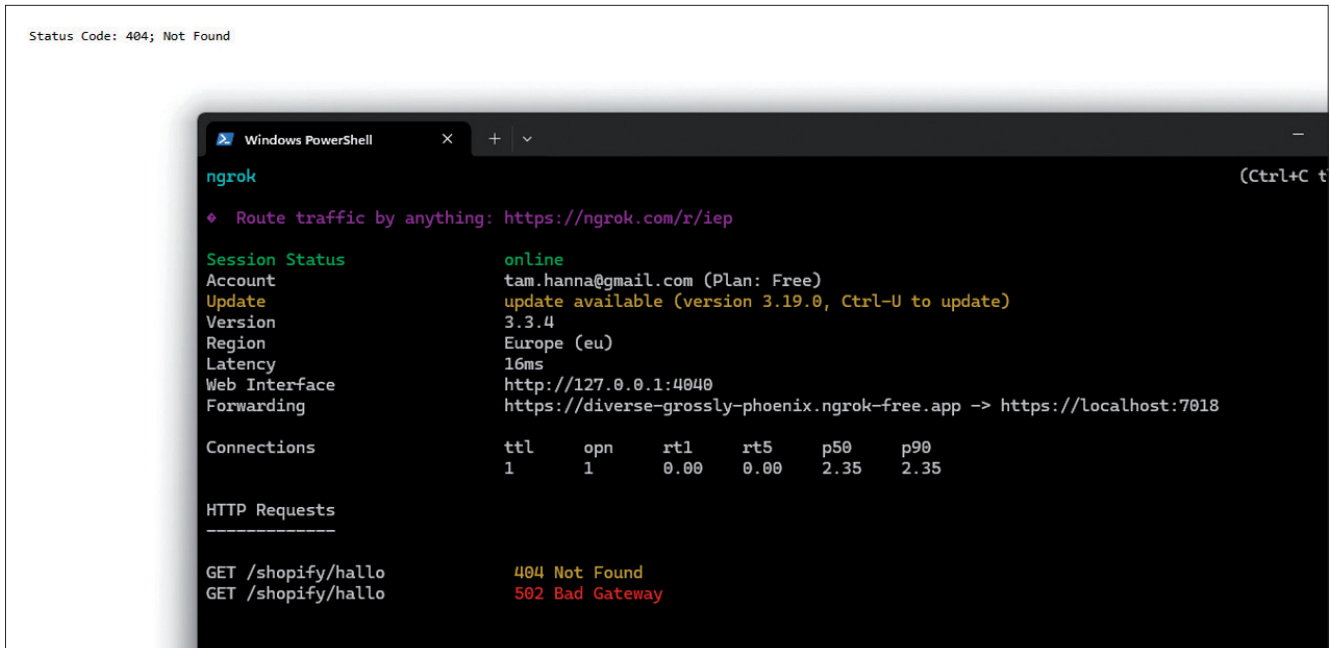
```
TamsPublic.myOAppState = new OAuthState
{
    DateCreated = DateTimeOffset.Now,
    Token = Guid.NewGuid().ToString()
};
```

An dieser Stelle ist die Applikation für eine abermalige Kompilation und einen weiteren Trockenstart bereit.

Umstellung von ngrok auf HTTPs und Durchführung weiterer Rauchtests

Der Siegeszug der sicheren Variante von HTTP ist heute nicht mehr aufzuhalten. Der Gutteil der Interaktionen zwischen dem Shopify-Backend und der lokal gehosteten Applikation erfolgt unter Nutzung des sicheren Protokolls HTTPS.

Der Weg zu einer per HTTPS ansprechbaren Variante unserer Beispielapplikation beginnt dabei im Debugger, wo Sie – wie in **Bild 4** gezeigt – das Protokoll auswählen können beziehungsweise dürfen, das Visual Studio heranzieht, um den Auslieferungsserver ansprechbar zu machen. Im nächsten Schritt bietet es sich bei gestartetem Tunnel an, einen URL nach dem Schema *https://diverse-grossly-phoenix.ngrok-free.app/* einzugeben.



Die Kommunikation zwischen dem Backend und der in Visual Studio lebenden Instanz funktioniert (Bild 6)

Der Verbindungsversuch wird daraufhin in den meisten Fällen scheitern. Ursache dafür ist, dass Visual Studio den HTTP- und den HTTPS-Server mit jeweils unterschiedlichen Ports verbindet – darüber informiert die IDE während des Programmstarts allerdings auch, wie in Bild 5 gezeigt.

Problematisch ist, dass das einfache Anpassen der an den ngrok-Client übergebenen Portnummern nicht ausreicht, um die Weiterleitung funktionsfähig zu bekommen. Vielmehr ist darüber hinaus auch eine Anpassung der Parametrierung erforderlich, die sich im Fall der vom Autor verwendeten Konfiguration folgendermaßen präsentiert:

```
.\ngrok http https://localhost:7018
--domain=diverse-grossly-phoenix.ngrok-free.app
```

Bei Verwendung des hier abgedruckten Konfigurationsstrings sollte es danach jedenfalls möglich sein, den HTTPS-URL aus einem Browser Ihrer Wahl heraus zu aktivieren und mit den Inhalten der ASP.NET-Applikation zu interagieren.

Anschließend kehren Sie in das Shopify-Partners-Portal zurück, wo Sie nun die Überblicksoption des Applikationseintrags aktivieren. Unter dem Header findet sich der Link zum Testen in einem Developer-Shop. Dieser kann nach einem Rechtsklick in einem neuen Fenster geöffnet werden.

Im Idealfall findet sich dort bereits ein *Development Shop*, der auf die Ausführung unserer Erweiterung wartet. Die im Header der Seite eingeblendete Warnung darüber, dass der Shop nach der Aktivierung der Erweiterung nicht mehr weitergegeben werden kann, ist im Allgemeinen irrelevant.

Als nächste Aktion versucht das System den Start, der zu diesem Zeitpunkt aber wie in Bild 6 gezeigt fehlschlagen wird. Die Ursache dafür ist, dass derzeit noch kein Code zur Verfügung steht, der die Initial-Parametrierung entgegennehmen könnte.

Der nächste Schritt, der zu einer funktionsfähigen Applikation erforderlich ist, wäre hier das Implementieren eines Handlers für die Route *hello* – eine Aufgabe, die der Autor schon aus Platzgründen dem Leser überlassen muss.

Fazit

Dass beim Gold Rush in den USA vor allem die Verkäufer von Schaufeln und sonstiger Support-Infrastruktur reich wurden, ist hinreichend bekannt. Der Autor hofft, dass die hier gezeigten Grundexperimente zu eigenen Versuchen mit der Technologie führen und die daraus resultierenden Shopify-Erweiterungen ähnliche Erfolge erreichen! ■

- [1] *ShopifySharp bei GitHub*,
www.dotnetpro.de/SL2503ShopifySharp1
- [2] *JavaScript Buy SDK*,
www.dotnetpro.de/SL2503ShopifySharp2
- [3] *Static domains for all ngrok users*,
www.dotnetpro.de/SL2503ShopifySharp3
- [4] *Zugang zum ngrok-Frontend*,
<https://dashboard.ngrok.com/signup>
- [5] *The Shopify Development Handbook*,
www.dotnetpro.de/SL2503ShopifySharp4



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.

dnpCode

A2503ShopifySharp

MONOGAME FÜR EINSTEIGER, TEIL 8.2

Des Einstiegs letzter Teil

Nach dem Spiel ist bekanntlich vor dem Spiel, und was das in diesem Fall bedeutet, erfahren Sie am Ende dieses Artikels. Aber vorher geht es noch in die Verlängerung ...

Um direkt einmal die Erwartungshaltung ein wenig zu dämpfen: Das Spiel, das Sie im Verlauf der bisherigen acht Artikel mit mir zusammen entwickelt haben, wird mit diesem letzten Artikel nicht final abgeschlossen sein. Auch hier gilt das Pareto-Prinzip: Die letzten 20 Prozent werden wahrscheinlich 80 Prozent des Gesamtaufwands benötigen. Was das in Zahlen bedeutet, können Sie sich ausrechnen. In Bezug auf den Gamecode bedeutet das, dass für ein wirkliches „Done“ noch einiges fehlt: ein dynamischeres NPC-Verhalten, mehr Ablenkung im Spiel, ein Game-Score, ein wenig was fürs Auge, eine Pause-Funktion vielleicht, Scripting – denn jeder liebt Scripting – und natürlich ein Restocking-Mechanismus.

Hier die gute Nachricht: Der Plan ist, all das tatsächlich noch in diesem vorerst letzten Teil zu behandeln. Und nun die schlechte: Selbst dann fehlt noch einiges – das finale Balancing, Start-Menü, Optionsmenü, Pause-Menü, Game-over-Screen mit Highscore-Anzeige, eventuell ein Online-Leaderboard und sicherlich noch mehr.

Das macht aber nichts, denn inzwischen haben Sie alles an Wissen, was erforderlich ist, um die nächsten Schritte (allein) zu gehen. Und da dieses Spiel sicher nicht bei mir in der Schublade verstauben wird, werden Sie über diesen Print-Artikel hinaus in Zukunft noch mehr darüber lesen können.

• ;tldr

Das bisher von Ihnen und mir programmierte Spiel wird mit diesem Artikel abgerundet – sowohl in Hinsicht auf die noch nicht implementierten, aber erforderlichen Funktionen als auch optisch; denn das Auge spielt bekanntlich mit. Das bedeutet: Heute geht es um die NPCs, visuelle Effekte, den Score und ein paar interessante Helfer, wie zum Beispiel die Pause-Funktion und das Scripting. Viel Spaß dabei!

Sprungmarken

Lassen Sie uns beginnen mit den schon länger versprochenen Sprungmarken. Kurz angerissen hatte ich die Idee schon: Statt eines ausschließlich linearen Ablaufs von Waypoints soll ein NPC unterschiedliche Wege laufen können – je nach Situation. In diesem Fall: Hat er sein Getränk bekommen? Dann geht er an einen Stehtisch. Und falls nicht, verlässt er direkt die Lounge. Mit der starren *WayPoint*-Klasse wäre das nicht

so ohne Weiteres möglich gewesen. Mit der neuen State Maschine aber geht das ganz leicht!

Ergänzen Sie zunächst Ihr Projekt um eine neue Klasse *NpcStateSatisfied*. Der Code ist denkbar einfach:

```
public class NpcStateSatisfied() : NpcStateBase
{
    public override void Update(Npc npc, GameTime gameTime)
    {
        npc.GoToNextState();
    }
}
```

Kopieren Sie die Klasse nun unter dem neuen Namen *NpcStateDissatisfied* und ändern Sie daran ansonsten ... nichts.

Im Grunde genommen handelt es sich bei diesen beiden Klassen in der Tat nur um „Marker“, die funktional nichts weiter tun, als zum nächsten State vorzurücken.

Damit das auch klappt, erweitern Sie als Nächstes die *Npc*-Basisklasse um eine Überladung der Methode *GoToNextState* wie folgt:

```
public void GoToNextState(Type targetState)
{
    do
    {
        currentStateIndex = (currentStateIndex + 1) %
            states.Count;
    } while ( states[currentStateIndex].GetType()
        != targetState);
}
```

Sie erkennen bereits, worauf das hinausläuft? Sehr gut. Denn nun wenden Sie sich der Klasse *NpcStateOrdering* zu und werfen einen Blick auf die Methode *TransitionToNextState*. Falls Sie die dort erforderliche Anpassung selbst vornehmen können: Go for it! Anderenfalls käme hier mein Vorschlag: Statt direkt den Eventhandler *onDrinkDelivered* aufzurufen und im Anschluss zum nächsten State zu wechseln, differenzieren Sie nun zwischen den beiden bekannten Fällen.

```
if (currentOrder == Order.Satisfied)
{
    onDrinkDelivered?.Invoke(true);
    npc.GoToNextState(typeof(NpcStateSatisfied));
}
```

```

else
{
    onDrinkDelivered?.Invoke(false);
    npc.GoToNextState(typeof(NpcStateDissatisfied));
}

```

Alles, was nun noch bleibt, ist, diese States in der Konfiguration der NPCs zu integrieren. Oder? Denken Sie einen Moment darüber nach. Denn ganz korrekt ist das nicht.

Betrachten wir zum Beispiel diesen Ablauf:

StateWalking – Zum Tresen laufen
StateOrdering – Ein Getränk bestellen
StateSatisfied – Das richtige Getränk bekommen
 StateWalking – Zum Tisch gehen
 StateIdling – Warten
 StateWalking – Die Lounge verlassen
StateDissatisfied – Das falsche oder kein Getränk bekommen
 StateWalking – Die Lounge direkt verlassen

Was fehlt? Ganz klar, oder? Wenn der NPC das richtige Getränk erhalten hat, zum Tisch gegangen ist und dann die Lounge verlässt, sollte der Prozess wieder von vorne beginnen und nicht mit *StateDissatisfied* fortgesetzt werden. Was also fehlt, ist die Möglichkeit, den Prozess zurückzusetzen.

Ergänzen Sie zu diesem Zweck einen weiteren State mit dem Namen *NpcStateReset*:

```

public class NpcStateReset : NpcStateBase
{
    public override void Update(
        Npc npc, GameTime gameTime)
    {
        npc.GotoFirstState();
    }
}

```

Und damit das funktioniert, braucht es noch die Methode *GotoFirstState* in der *Npc*-Basisklasse:

```

public void GotoFirstState()
{
    currentStateIndex = 0;
}

```

Das dürfte es auch schon gewesen sein. Nun können Sie sich gerne mal einen Moment mit Ihren NPCs beschäftigen und Ihre Lounge nach Belieben bevölkern und dabei auch unterschiedliche Wege gehen. Ein Beispiel für einen NPC finden Sie in [Listing 1](#). In dem Beispiel sehen Sie auch, dass ich den NPCs bei der Gelegenheit die Startkoordinaten direkt über den Constructor übergebe. Die erste Positionierung geschieht dabei direkt in *LoadContent*:

```

SetX(startPosition.X);
SetY(startPosition.Y);

```

Listing 1: Ein NPC-Beispiel

```

NPCs.Add(new Npc(this, GetRandomCharacter(), states:
[
    new NpcStateWalking(
        new Vector2(28, 170), reset: true),
    new NpcStateWalking(new Vector2(170, 170)),
    new NpcStateWalking(new Vector2(170, 230)),
    new NpcStateWalking(new Vector2(220, 230)), // Bar
    new NpcStateOrdering(
        patienceTime: TimeSpan.FromSeconds(5),
        unpatienceTime: TimeSpan.FromSeconds(4),
        faceDirection: Direction.Up,
        onDrinkDelivered: OnDrinkDelivered),

    new NpcStateSatisfied(),
    new NpcStateWalking(new Vector2(220, 260)),
    new NpcStateWalking(new Vector2(190, 260)),
    new NpcStateWalking(new Vector2(190, 295)),
    new NpcStateWalking(new Vector2(175, 295)),
    // Table
    new NpcStateIdling(
        idleTime: TimeSpan.FromSeconds(3),
        faceDirection: Direction.Up),
    new NpcStateWalking(new Vector2(285, 295)),
    new NpcStateWalking(new Vector2(285, 474)),
    new NpcStateIdling(
        idleTime: TimeSpan.FromSeconds(1)), // Off
    new NpcStateReset(),

    new NpcStateDissatisfied(),
    new NpcStateWalking(new Vector2(285, 230)),
    new NpcStateWalking(new Vector2(285, 474)),
    new NpcStateIdling(
        idleTime: TimeSpan.FromSeconds(3)), // Off
]));

```

More Spice

Bevor Sie sich dem Thema „Score“ widmen, hier noch eine Kleinigkeit, die eigentlich gar keine Kleinigkeit ist. Ihnen ist aufgefallen, dass Sie vermutlich gar nicht so viele NPCs im Spiel haben. Und vielleicht haben Sie auch mit der Anzahl an NPCs experimentiert, die tatsächlich an den Tresen gehen und dort etwas bestellen. Die Frage dabei ist die: Wie viele braucht es dafür eigentlich? Nun, Sie wollen auf jeden Fall erreichen, dass die Lounge gut besucht aussieht, und Sie wollen den Spieler möglichst viel von seinem Job ablenken – denn woher käme sonst der ganze Spaß?

Also muss es nach viel Bewegung aussehen. Aber auch danach, als handele es sich um viele unterschiedliche NPCs, während es eigentlich immer die gleichen sind.

In meinem Spiel habe ich genau sieben NPCs. Drei besuchen regelmäßig den Tresen, einer steht die ganze Zeit an einem Tisch, einer läuft einfach durch die Lounge, und die ►

übrigen zwei schauen immer mal wieder rein, gehen dann aber wieder. Das Ganze ist durch die IdleTimes so aufeinander abgestimmt, dass es im Ergebnis halbwegs zufällig aussieht.

Noch zufälliger sieht es aus, wenn die NPCs immer anders aussehen, wenn sie die Lounge erneut betreten. Das können Sie erreichen, indem Sie den Character beim Wechsel in *GoToNextState* zufällig neu setzen:

```
public void GoToNextState()
{
    currentStateIndex = (currentStateIndex + 1) %
        states.Count;

    if (currentStateIndex == 0)
    {
        Character = (Character)random.Next(6);
    }
}
```

Probieren Sie das aus. Funktioniert es? Nein? Richtig. Denn da fehlt noch eine Kleinigkeit: Aktuell laden Sie die Animationen aus dem SpriteSheet während des initialen Aufrufs von *LoadContent* in der Klasse *GameCharacter*. Nun aber müssen Sie die für die Animationen nötigen Ausschnitte aus dem SpriteSheet beim Character-Wechsel neu definieren.

Lagern Sie dazu die Aufrufe zu *AddAnimationCycle* in *GameCharakter.LoadContent* einfach in eine eigene Methode *CreateAnimations()* aus und sorgen Sie in der Methode *AddAnimationCycle* gleich zu Beginn dafür, dass die Animation – falls schon vorhanden – zunächst entfernt wird:

```
protected void AddAnimationCycle(SpriteSheet
    spriteSheet, string name, int[] frames, bool
    islooping = true, float frameDuration = .15f)
{
    spriteSheet.RemoveAnimationDefinition(name);
    ...
}
```

Nun können Sie abschließend den Setter der *Character-Property* ausimplementieren und dabei *CreateAnimations* aufrufen:

```
public Character Character { get { return character; }
    set {
        character = value;
        CreateAnimations();
    }
}
```

Vergessen Sie dabei übrigens nicht, die Deklaration von *SpriteSheet* aus *LoadContent* heraus in die Klasse zu heben, damit Sie die Variable später wiederverwenden können.



Penalty Time! (Bild 1)

Last, but not least – nein, versprochen, dieser Abschnitt ist nun eigentlich fertig – gäbe es noch eine weitere Kleinigkeit, die die Illusion vieler Charaktere noch mehr unterstützt. Dazu aber später.

Score and more

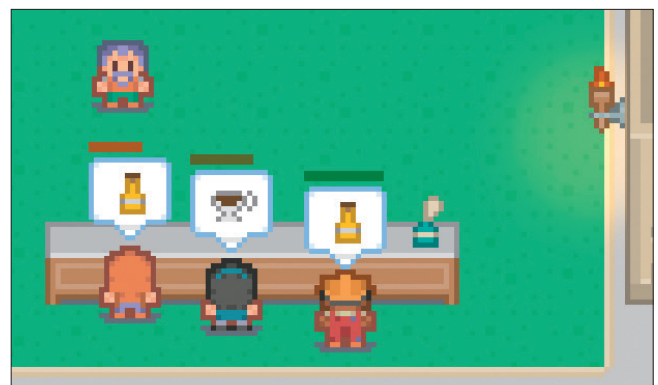
Sie erinnern sich an den Abschluss des letzten Artikels? Es ging unter anderem darum, dass es endlich Zeit wird, den Spieler für sein (oder ihr) Treiben zu belohnen (oder zu bestrafen?). In der Tat

gehört das zu den Grundzutaten für ein interessantes Spielerlebnis. Natürlich muss der Spieler Punkte bekommen für korrekt ausgelieferte Bestellungen. Sollte es mal nicht klappen, wäre es aber nicht förderlich, einmal gegebene Punkte wieder wegzunehmen.

Das bedeutet nicht, dass es keine Möglichkeit gibt, den Spieler zu „bestrafen“. Hier die Idee: Zwar läuft die Spielzeit im Sekundentakt ab, doch jedes Mal, wenn der Spieler ein Getränk korrekt ausliefert, bekommt er nicht nur Punkte dafür, sondern auch einen Zeitbonus. Klappt das mal nicht, werden ihm im Gegenzug wertvolle Sekunden Spielzeit abgezogen (Bild 1). Um den Highscore zu knacken, ist dem Spieler also daran gelegen, ein möglichst perfektes Spiel hinzulegen. Denn durch die Zeitboni kann er das Spielende hinauszögern und so mehr Punkte einheimsen.

Nicht nur das: Was halten Sie davon, dass die Anzahl der Punkte, die er erhält, sogar noch davon abhängt, wie schnell er eine Bestellung ausgeliefert hat? Auf diese Weise implementieren Sie nicht nur ein Belohnungssystem für den Spieler, das motiviert, sondern geben ihm auch ein Gefühl der Kontrolle darüber. Besser gehts nicht!

Experimentieren Sie übrigens auf jeden Fall mit der Art, wie Sie solche Dinge zur Anzeige bringen. In meinem Fall wird der Zeitbonus oder Abzug für ein paar Sekunden angezeigt, während er zugleich immer größer wird und dabei ausfadet. Die Implementierung finden Sie verteilt auf den Eventhandler *OnDrinkDelivered* in *GameMain* und die jeweiligen *Update-/Draw-Methoden*. Lust, sich das auch mal „in bewegt“ anzuschauen? Da sind zwar einige Spoiler drin enthalten, aber sehen Sie sich gerne einmal das Video unter [1] an.



Wo ist ein Pixel Shader am Arbeiten? (Bild 2)

Das Pixel hat einen Schatten?

Kommen wir zu einem faszinierenden Thema: Pixel Shader. Vereinfacht gesagt geht es darum, mittels Code einzelne Pixel zu verändern. Im Grunde können Sie ja nicht viel mehr machen, als die Farbe oder die Transluzenz zu manipulieren. „Was genau bringt das also?“, wäre die folgerichtige Frage, und „Eine ganze Menge!“ die passende Antwort. Aber wir stehen hier nicht im Wald und ich bin kein Schwertmeister. Also schauen Sie sich einmal **Bild 2** genauer an. Was darauf könnte auf die Verwendung eines Pixel Shaders hinweisen?

Falsch. Der Lichtkegel rund um die Fackel ist es nicht. Übrigens: Wundern Sie sich eigentlich auch, wie es sein kann, dass da im Konferenzzentrum eine Fackel brennt? Nun ja ...

Nein, es sind die T-Shirts der NPCs! Weiter oben in diesem Artikel sprachen wir ja bereits davon, dass es wichtig ist, für ein wenig optische Abwechslung zu sorgen. Deshalb tauschen wir die NPCs nach jedem Durchlauf immer zufällig aus. Dennoch sind es immer die gleichen – und da wir uns für einen der Charaktere als Bartender entschieden haben (finden Sie eigentlich auch, dass der Ähnlichkeit mit unserem Chef-redakteur Fernando Schneider hat?), bleiben durch das gewählte Asset Pack nur noch fünf für die NPCs.

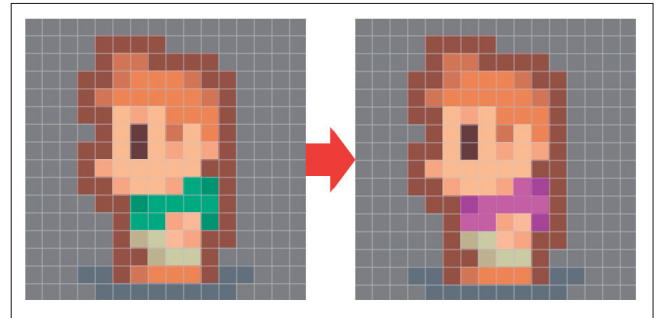
Ändern Sie das also, indem Sie die Farbe der Shirts ebenfalls zufällig anpassen. Ich schlage für den Anfang aber vor, nicht gleich mit Kanonen auf Spatzen ... – Sie wissen schon. Sondern legen Sie einfach ein paar Farben fest, die gut zur Optik des Spiels passen, und wählen daraus aus.

Wie das geht? Um mittels Pixel Shading überhaupt dazu in der Lage zu sein, die Farbe der NPC-Shirts zur Laufzeit zu verändern, müssen Sie dem Pixel Shader irgendwie eine Information darüber geben, welche Pixel eigentlich davon betroffen sind. Natürlich gibt es hierfür – wie für so ziemlich alles – verschiedene Möglichkeiten. Der Begriff „Maske“ würde Ihnen dabei sicher häufiger begegnen, sofern Sie sich im Internet über dieses Thema informieren wollten. Im Kontext dieses Artikels aber gehen wir einen viel simpleren Weg: Wir malen die Shirts in Aseprite neu an und nutzen dafür entsprechend den Originalfarben zwei verschiedene Schattierungen – allerdings in einer Farbe, die ansonsten so nicht in der bisherigen Farbpalette auftaucht: Magenta; beziehungsweise Magenta-ähnliche Farbtöne.

Für die Hauptfarbe ergänzen Sie den Farbcode `#fe00fe` zu Ihrer Palette, für die etwas dunklere Schattierung den Farbcode `#b400b4`. Danach gehen Sie in etwa so vor wie auf **Bild 3** zu sehen, um die Shirts aller NPCs und ihrer Animationssequenzen neu einzufärben.

Als Nächstes brauchen Sie einen tatsächlichen Pixel Shader. Im Grunde genommen ist das auch nur ein Code Snippet. Legen Sie dazu im *Content*-Verzeichnis ein neues Unterverzeichnis mit dem Namen *Shader* an und erstellen Sie über den MGCN einen neuen Pixel Shader darin mit dem Namen *ShirtColorShader.fx*.

Den Code können Sie mit dem Inhalt von **Listing 2** überschreiben. Wenn Sie sich den Code des Listings genauer anschauen, erkennen Sie darin neben einigen statischen Variablen auch zwei, die nicht statisch sind: *SpriteTexture* und *ShirtColor*.



Hoffentlich gibt das keinen Ärger mit ... (**Bild 3**)

SpriteTexture wird – ohne jetzt zu sehr ins Detail zu gehen – über den *Draw*-Aufruf implizit gesetzt. Die *ShirtColor* aber setzen Sie gleich selbst.

Was dann in der Methode *MainPS* passiert, ist, dass Pixel für Pixel überprüft wird, ob die Farbe entweder dem hellen Magenta oder aber dem dunklen Magenta entspricht. Wenn das der Fall ist, wird die neue Farbe gesetzt (im Fall des dunklen Magenta wird diese zuvor ebenfalls abgedunkelt). Ansonsten wird die Original-Pixelfarbe beibehalten.

Für eine kleine Liste möglicher Shirt-Farben ergänzen Sie als Nächstes eine neue Property und ein entsprechendes Array in der NPC-Basisklasse:

```
Vector4 shirtColor;
public Vector4 ShirtColor { get => shirtColor; }

readonly Vector4[] possibleColors = [
    new Vector4(0.26f, 0.64f, 0.47f, 1.0f),
    // Original Grün (#42a379)
    new Vector4(0.76f, 0.31f, 0.30f, 1.0f),
    // Original Rot (#c2504d)
    new Vector4(0.86f, 0.53f, 0.32f, 1.0f),
    // Original Orange (#dc8652)
    new Vector4(0.57f, 0.56f, 0.73f, 1.0f),
    // Original Grau (#918eb9)
    new Vector4(0.78f, 0.48f, 0.28f, 1.0f),
    // Original Braun (#c77b47)
    new Vector4(0.29f, 0.56f, 0.64f, 1.0f),
```

Color Tolerance

Lassen Sie sich nicht von der Art irritieren, wie die Farbwerte im Pixel Shader verglichen werden. Der Grund dafür, dass nicht einfach ein direkter Vergleich stattfindet, sondern stattdessen der zu vergleichende Magenta-Farbtönen von der Pixelfarbe abgezogen wird, um dann zu prüfen, ob das Ergebnis kleiner als eine vorgegebene Toleranz ist, ist der, dass die GPU mit Fließkommazahlen arbeitet und es schwer wäre, die Vergleichswerte wirklich exakt im Code zu hinterlegen. Auch einige andere Faktoren tragen dazu bei, dass es stabiler und einfacher funktioniert, auf diese Weise zu vergleichen.

● Listing 2: ShirtColorShader.fx

```

#if OPENG
    #define SV_POSITION POSITION
    #define VS_SHADERMODEL vs_3_0
    #define PS_SHADERMODEL ps_3_0
#else
    #define VS_SHADERMODEL vs_4_0_level_9_1
    #define PS_SHADERMODEL ps_4_0_level_9_1
#endif

Texture2D SpriteTexture;

// Färbung für das Shirt (z.B. Rot)
float4 ShirtColor = float4(1.0, 0.0, 0.0, 1.0); // Rot

// Faktor zum Abdunkeln der Basisfarbe für die
// Schattenbereiche
static float ShadowFactor = 0.8;
// 60% Helligkeit für die Schattenbereiche

// Heller Magenta-Ton
static float3 LightMagenta = float3(0.996, 0.0,
    0.996); // #fe00fe

// Dunkler Magenta-Ton
static float3 DarkMagenta = float3(0.706, 0.0, 0.706);
// #b400b4

// Toleranz für die Erkennung der Magenta-Farben
static float ColorTolerance = 0.01;

sampler2D SpriteTextureSampler = sampler_state
{
    Texture = <SpriteTexture>;
};

struct VertexShaderOutput
{
    float4 Position : SV_POSITION;
    float4 Color : COLOR0;
    float2 TextureCoordinates : TEXCOORD0;
};

float4 MainPS(VertexShaderOutput input) : COLOR
{
    // Lade die Farbe des Sprites an der aktuellen
    // Position
    float4 spriteColor = tex2D(SpriteTextureSampler,
        input.TextureCoordinates);

    // Prüfe, ob der Pixel die helle Magenta-Farbe hat
    if (all(abs(spriteColor.rgb - LightMagenta) <
        ColorTolerance))
    {
        // Verwende die Basis-Shirt-Farbe für helle
        // Magenta-Bereiche
        return float4(ShirtColor.rgb, spriteColor.a);
    }

    // Prüfe, ob der Pixel die dunkle Magenta-Farbe hat
    if (all(abs(spriteColor.rgb - DarkMagenta) <
        ColorTolerance))
    {
        // Verwende eine abgedunkelte Version der
        // Shirt-Farbe für Schattenbereiche
        return float4(ShirtColor.rgb * ShadowFactor,
            spriteColor.a);
    }

    // Ansonsten verwende die Originalfarbe
    return spriteColor * input.Color;
}

technique SpriteDrawing
{
    pass P0
    {
        PixelShader = compile PS_SHADERMODEL MainPS();
    }
};

// Blaugrünlich (#4a90a4)
new Vector4(0.49f, 0.37f, 0.60f, 1.0f),
// Gedämpftes Violett (#7c5e99)
new Vector4(0.82f, 0.61f, 0.61f, 1.0f)
// Staubiges Rosa (#d19c9c)
];

Random random = new();

private void SetRandomShirtColor()
{
    int randomIndex = random.Next(possibleColors.Length);
    shirtColor = possibleColors[randomIndex];
}

```

Um direkt in der Lage zu sein, jetzt und auch in Zukunft zufällige Shirt-Farben zu setzen, kommt nun noch eine entsprechende Methode hinzu, die Sie dann auch direkt in *LoadContent* aufrufen können:

Lighting

So. Jetzt aber mal Butter bei die Fische! Seien Sie ehrlich. Flüssige Kamerabewegungen, viele zufällig wirkende NPCs,

coole Musik, Soundeffekte, ablaufende Zeit, Score-System mit Belohnungseffekt, Raumwechsel ... wie viel Zeit verbringen Sie inzwischen damit, Ihren letzten Score zu übertreffen, und wie viel damit, den Code feinzuschleifen?

Unter uns: Spätestens ab jetzt werden Sie sich mehr mit Folgendem beschäftigen: more Spice für mehr visuelle Abwechslung, Bugfixing, Workflow und Polishing.

Kommen wir aber erst einmal zu „more Spice“. Denn da wären noch immer die Fackeln im Spiel. Schauen Sie noch einmal auf **Bild 2** von eben. Wie funktioniert das mit dem Lichtkegel unter der Fackel? Und was Sie auf dem Bild nicht sehen können: Erstens flackert der Lichtkegel mit der Fackel, und zweitens erhellt der nicht nur den Boden, sondern auch den Spieler, wenn er unter der Fackel entlangläuft. Schauen wir uns also an, wie das funktioniert.

Obwohl es zunächst sinnvoll anmutet, auch für diesen Effekt Pixel Shader einzusetzen, wäre das in diesem Fall wirklich wie mit Kanonen auf Spatzen zu schießen (nein, bei der Entwicklung dieses Spiels wurden keine Tiere verletzt). Denn es gibt eine viel einfachere Möglichkeit: Sie legen eine entsprechend vorbereitete Textur über alles und verknüpfen diese bitweise mit allem, was sich „darunter“ befindet. Die Textur selbst besteht dabei aus verschiedenen Abstufungen von Weiß inklusive einem Transparenzlayer. Aber schauen Sie selbst (**Bild 4**).

Warum die Textur abgeschnitten ist, hören Sie fragen? Na, schauen Sie sich mal die Fackel in der Lounge an. Die hängt an der Wand; nach rechts geht das ja nicht mehr weiter. Natürlich gäbe es an dieser Stelle auch ganz ausgefuchste Möglichkeiten, damit umzugehen; doch wir machen es uns erst einmal sehr einfach, indem wir die Textur ganz pragmatisch ... abschneiden.

Die Originaltextur finden Sie übrigens wie gewohnt in den Asset Packs von Kenney. Genauer: im Particle Pack, das Sie entweder direkt von seiner Website unter [2] herunterladen können oder im All-In-One Bundle im 2D-Assets-Verzeichnis finden. Für dieses Spiel habe ich mich für die Textur *Circle-05.png* entschieden. Kopieren Sie sie als *LightTexture.png* in Ihr *Content*-Verzeichnis (zum Beispiel in ein neues Unterverzeichnis *Fx*) und binden Sie sie wie gewohnt ein. In der Lounge (*MainScreen*) beispielsweise als neue Variable *lightTexture* vom Typ *Texture2D*. Denken Sie daran, die Textur in *LoadContent* auch zu laden.

RenderTarget

Bevor es mit der Frage weitergeht, wie Sie diese Textur verwenden, ist es wichtig zu verstehen, wie in MonoGame entsprechende Operationen überhaupt möglich sind. Dazu ist es erforderlich, etwas Neues einzuführen: das *RenderTarget*.

Ergänzen Sie ebenfalls in *MainScreen* die folgende neue Variable:

```
private RenderTarget2D renderTarget;
```

Entsprechend der aktuell gewählten Auflösung initialisieren Sie sie in *LoadContent* mit:

```
renderTarget = new(GraphicsDevice, 1600, 900);
```

In der Methode *Draw* geht es nun zur Sache. Merken Sie sich schon mal vor, dass es eine gute Idee sein kann, sich die Position der Fackel in Echtzeit aus der Map zu holen und die exakte Textur für die Verwendung konfigurierbar zu machen. Für einen schnellen Erfolg aber gehen wir händisch vor:

```
Vector2 torchWorldPosition = new(1316, 780);
```

```
Matrix cameraViewMatrix = Game.Camera.GetViewMatrix();
```

```
Vector2 torchScreenPosition = Vector2.
```

```
Transform(torchWorldPosition, Matrix.
```

```
CreateScale(1f / 5f) * cameraViewMatrix);
```

```
GraphicsDevice.
```

```
SetRenderTarget(renderTarget);
```

```
GraphicsDevice.Clear(Color.Black);
```

```
spriteBatch.Begin(SpriteSortMode.
```

```
Immediate, BlendState.AlphaBlend,
```

```
SamplerState.PointClamp);
```

```
spriteBatch.Draw(lightTexture,
```

```
torchScreenPosition, Color.Orange);
```

```
spriteBatch.End();
```

```
GraphicsDevice.SetRenderTarget(null);
```

Das alles ergänzen Sie in der *Draw*-Methode vor *enableFloorLayer*. Als Nächstes zeichnen Sie wie gewohnt den *FloorLayer*, den *DecorationLayer*, den *Player* und was auch immer er in der Hand hält.

Nun kommt der Trick: Bevor Sie den *ForegroundLayer* zeichnen – in dem sich die Fackel befindet –, legen Sie das *RenderTarget* mit dem Lichtschein über die Szene:

```
spriteBatch.Begin(
```

```
SpriteSortMode.Immediate, BlendState.Additive);
```

```
spriteBatch.Draw(
```

```
renderTarget, Vector2.Zero, Color.White);
```

```
spriteBatch.End();
```

Wenn das so weit funktioniert, bewundern Sie für einen Moment das Ergebnis. Doch so richtig gut sieht das noch nicht aus. Zunächst einmal ist der Lichtschein zu grell, und er flackert auch nicht (**Bild 5**). Ändern wir das also. Weiterhin in *MainScreen* ergänzen Sie folgende Variablen:

```
private float lightIntensity = 0.15f;
```

```
// Anfangsintensität
```

```
private double timeSinceLastFluctuation = 0;
```

Und in der *Update*-Methode steuern Sie das Flackern des Lichtscheins: ▶

```
timeSinceLastFluctuation +=
    gameTime.ElapsedGameTime.
        TotalMilliseconds;
if (timeSinceLastFluctuation
    >= 200)
{
    lightIntensity = 0.15f +
        (float)new Random().NextDouble()
            * 0.1f;
    timeSinceLastFluctuation = 0;
    // Timer zurücksetzen
}
```

Die Idee: Der Wert einer Zufallszahl zwischen 0 und 1 wird praktisch durch 10 geteilt und dann zu 0,15 hinzudiert. Ergebnis: Die Lichtintensität schwankt zufällig zwischen 0,15 und 0,25. Und das alle 200 ms.

Abschließend müssen Sie diesen Wert nur noch mit der Farbe multiplizieren, die Sie für den Lichtschein ausgewählt haben:

```
spriteBatch.Draw(lightTexture,
    torchScreenPosition,
    Color.Orange * lightIntensity);
```

Das Ergebnis sieht so weit richtig gut aus, finden Sie nicht auch (**Bild 6**)? Experimentieren Sie aber gerne mit den Werten für diesen Effekt, und denken Sie auch gerne darüber nach, wie sich das Ganze auf eine Weise programmieren ließe, die einfacher wiederver-

Abschließend zu diesem Thema vielleicht noch eine kleine Challenge: War im Storage-Room nicht auch eine Fackel ...?

Zeit für eine Pause

Stress. STRESS. Getränke ausliefern! Unglückliche Besucher.
Die Zeit läuft ab! Schon wieder daneben! Das Smartphone
klingelt... <Pause>

Tja. Spieler benötigen eine Pause-Funktion in so hektischen Spielen wie diesem. Oder? Die Wahrheit ist: Eigentlich brauchen vor allem Sie sie. Oder noch genauer: ich. Manche Screenshots lassen sich schwer anfertigen, weil das Spiel nun einmal läuft und bestimmte Situationen – die auf dem Bild zu sehen sein sollen – nur schwer einzufangen sind. Es sei denn natürlich. Sie können das Spiel pausieren.

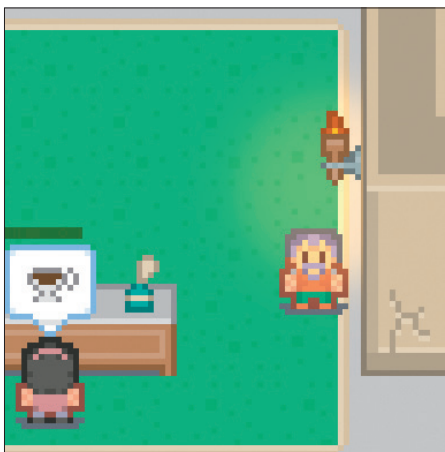
Und deshalb tun Sie das auch jetzt und ergänzen eine Pause-Funktion, die auf die Taste [P] reagiert. Beginnen Sie damit, *GameMain* um eine Variable *GamePaused* zu ergänzen:

```
public static bool GamePaused = false;
```

In *GameMain.Update* benötigen Sie nun folgende Zeilen, die Sie direkt im Anschluss an die Ermittlung des aktuellen Key-



Nicht ins Licht schauen! (Bild 5)



Wäre das kein Pixelspiel, sähe es regelrecht realistisch aus (Bild 6)

board-States und vor der Prüfung auf die [Tab]-Taste für das Debug-UI legen:

```
if (newState.IsKeyDown(Keys.P) &&
    previousState.IsKeyUp(Keys.P))
{
    GamePaused = !GamePaused;
}

if (GamePaused) return;
```

Last, but not least prüfen Sie nun noch in den *Update*-Methoden der Klassen *MainScreen*, *StorageScreen* und sonstiger Screens, die Sie eventuell bereits implementiert haben, auf den Status und springen direkt wieder raus, sofern das Spiel gerade pausiert ist:

```
if (GameMain.GamePaused) return;
```

Das war es. Mehr gibt es dazu nicht zu sagen. Bis auf das vielleicht: Es wäre gut, der Pause-Funktion auch aus Perspektive des Spielers einen Sinn zu verleihen. Dazu vielleicht später mehr.

Steht alles im Skript

Bevor wir zum letzten für den Moment noch wichtigen Thema kommen – nämlich der Tatsache, dass Getränke irgendwann ausgehen und es erforderlich sein könnte, sie wieder aufzustocken –, lassen Sie uns über etwas voll-

kommen anderes sprechen: das Scripting. Tatsache ist: Für dieses Spiel ist es ein wenig „over the top“, über das Thema überhaupt nachzudenken. Bis auf vielleicht... nun, stellen Sie doch einmal zunächst die Frage: Was ist Scripting im Kontext der Spieleentwicklung überhaupt, und wozu ist es gut?

Schön, dass Sie fragen! Zunächst einmal: Wenn Sie – so wie bisher – in der IDE mittels C# den Code für Ihr Spiel schreiben und [F5] drücken, meißelt der Compiler Ihre bis dato sehr flexiblen Ideen praktisch in Stein. Zur Laufzeit lassen sie sich schlicht nicht mehr verändern. Ist ja auch im Grunde genommen nicht gewollt – also alles gut.

Wären da nicht solche Szenarien wie beispielsweise das Verhalten der NPCs. Ihnen wird mit Sicherheit aufgefallen sein, dass Sie nicht selten das Spiel angehalten, die Initialisierung des NPC verändert, das Spiel neu gestartet und so lange gewartet haben, bis Sie das neue Verhalten des NPC zur Laufzeit validieren konnten. Manchmal mussten Sie das Spiel dazu mit Absicht spielen (menno!), um es so lange am Laufen zu halten wie nötig, um das erwartete Verhalten eines NPC überprüfen zu können.

An dieser Stelle (unter anderem, denn es gibt natürlich noch viel mehr Szenarien, fragen Sie mich nach einem Post-Mortem zu Projekt II – Rise of the Hero ... nein ernsthaft: Fra-

gen Sie!) kann Scripting helfen. Denn Skripte können zur Laufzeit übersetzt werden. Ich wäre geneigt zu sagen: im Gegensatz zu C#. Doch das stimmt heute so nicht mehr, denn es gibt auch mit C# einen von Microsoft dafür vorgesehenen Scripting-Ansatz. Doch den verfolgen wir hier nicht.

Stattdessen weise ich auf eine der bekanntesten Skriptsprachen hin, die in der Spieleentwicklung seit Jahrzehnten erfolgreich im Einsatz ist: Lua [3]. Für dieses Projekt nutzen wir mit NLua [4] ein NuGet-Paket, das .NET-Bindings für die ansonsten nativ implementierte Skriptsprache bereithält.

Mit Lua können wir relativ einfach die Initialisierung der NPCs in Skripten auslagern und dafür sorgen, dass diese zur Laufzeit – zum Beispiel bei Änderungen – neu übersetzt und direkt angewandt werden. Die Implementierung und Experimente damit überlasse ich Ihnen. Wie immer finden Sie meine konkreten Ansätze dazu im Repository. Hier in aller bescheidenen Kürze aber ein paar Teaser.

Ein solches NPC-Skript kann zum Beispiel aussehen wie in **Listing 3**. Die Solution habe ich dabei um einen ScriptManager erweitert, der unter anderem via FileSystemWatcher das Verzeichnis mit Skripten auf Veränderungen überwacht und in dem Fall das entsprechende Skript neu lädt und übersetzt. Obwohl Lua in dieser Hinsicht extrem mächtig ist, habe ich mich für dieses Spiel dazu entschieden, einen für diesen Anwendungsfall extrem simplen Weg zu gehen.

So kann Lua direkt auf C#-Methoden zugreifen und diese ausführen, und andersherum ist es auch möglich, aus C# he-

raus Lua-Funktionen auszuführen. In unserem Fall aber lässt sich das drastisch vereinfachen: Das Skript enthält die jeweiligen Typen für die States und deren Konfigurationsparameter. Der ScriptManager wiederum liest diese ein und switcht über die Typen, um dann in C# die passenden Klassen zu instanzieren und zu konfigurieren. That easy. Beispiel gefällig?

```
public List<NpcStateBase> LoadNpcStates(
    string scriptPath)
{
    List<NpcStateBase> states = [];
    using (var lua = new Lua())
    {
        lua.DoFile(scriptPath);
        var npcConfig = lua["npc"] as LuaTable;

        foreach (LuaTable stateEntry in npcConfig.Values)
        {
            string type = stateEntry["type"] as string;
            NpcStateBase npcState = null;

            switch (type)
            {
                case "NpcStateWalking":
                    float x = Convert.ToSingle(stateEntry["x"]);
                    float y = Convert.ToSingle(stateEntry["y"]);
                    bool reset = stateEntry["reset"] != null &&
                        (bool)stateEntry["reset"];
                    npcState = new NpcStateWalking(
                        new Vector2(x, y), reset);
                    break;

                    ...

            }

            if (npcState != null)
            {
                states.Add(npcState);
            }
        }
    }
    return states;
}
```

Drumherum ist natürlich noch ein wenig mehr Einsatz erforderlich. Sie müssen sich darum kümmern, NPCs auch zur Laufzeit reinitialisieren zu können und dabei beispielsweise auch Assets neu zu laden – oder die alten zu verwerfen. Dennoch stellt das Scripting eine interessante Möglichkeit dar, sogar in diesem kleinen Spiel die Designphase ein wenig zu beschleunigen. Zumindest aber etwas Neues zu lernen.

Woodstock? Nein. Restock

Da haben Sie vermutlich noch gar nicht drüber nachgedacht: Denn bevor Sie sich Gedanken darüber machen, wie Sie die Getränkebar wieder auffüllen, muss sie erst einmal leer werden. Erinnern Sie sich an den Moment, in dem es darum ►

● Listing 3: NPC.lua

```
npc = {
    { type = "NpcStateWalking", x = 28, y = 170,
      reset = true },
    { type = "NpcStateWalking", x = 170, y = 170 },
    { type = "NpcStateWalking", x = 170, y = 230 },
    { type = "NpcStateWalking", x = 220, y = 230 },
    { type = "NpcStateOrdering", patienceTime = 3,
      unpatienceTime = 2, faceDirection = "Up" },
    { type = "NpcStateSatisfied" },
    { type = "NpcStateWalking", x = 220, y = 260 },
    { type = "NpcStateWalking", x = 190, y = 260 },
    { type = "NpcStateWalking", x = 190, y = 295 },
    { type = "NpcStateWalking", x = 175, y = 295 },
    { type = "NpcStateIdling", idleTime = 1,
      faceDirection = "Up" },
    { type = "NpcStateWalking", x = 285, y = 295 },
    { type = "NpcStateWalking", x = 285, y = 474 },
    { type = "NpcStateIdling", idleTime = 1 },
    { type = "NpcStateReset" },
    { type = "NpcStateDissatisfied" },
    { type = "NpcStateWalking", x = 285, y = 230 },
    { type = "NpcStateWalking", x = 285, y = 474 },
    { type = "NpcStateIdling", idleTime = 3 }
}
```


ging, Objekten in Tiled Namen zu geben? Dies ist einer der Momente, für die das tatsächlich nützlich ist. Und auch hier gilt: Es gibt elegantere Wege, Folgendes umzusetzen. Doch wir gehen erst einmal den wirklich simplen Weg. Ob Sie den Code im Anschluss optimieren möchten, obliegt Ihrer Entscheidung. Lassen wir es also langsam angehen.

Beginnen Sie damit, dass Sie zunächst einmal Referenzen auf alle Getränke im Regal benötigen. Bei mir stehen dort drei Kaffeetassen und drei Energydrink-Flaschen. Ergänzen Sie in *GameMain* folgende Variablen:

```
TiledMapObject energy1, energy2, energy3;
TiledMapObject coffee1, coffee2, coffee3;
```

```
List<TiledMapObject> energyObjects;
List<TiledMapObject> coffeeObjects;
```

```
bool energyDrinkAvailable = true;
bool coffeeAvailable = true;
```

Und in der Methode *LoadContent* ergänzen Sie folgenden Code, um sich die Referenzen auf die jeweiligen Objekte zu holen:

```
energy1 = objectsLayer.GetObjectByName("Energy1");
energy2 = objectsLayer.GetObjectByName("Energy2");
energy3 = objectsLayer.GetObjectByName("Energy3");
```

```
energyObjects = [energy1, energy2, energy3];
```

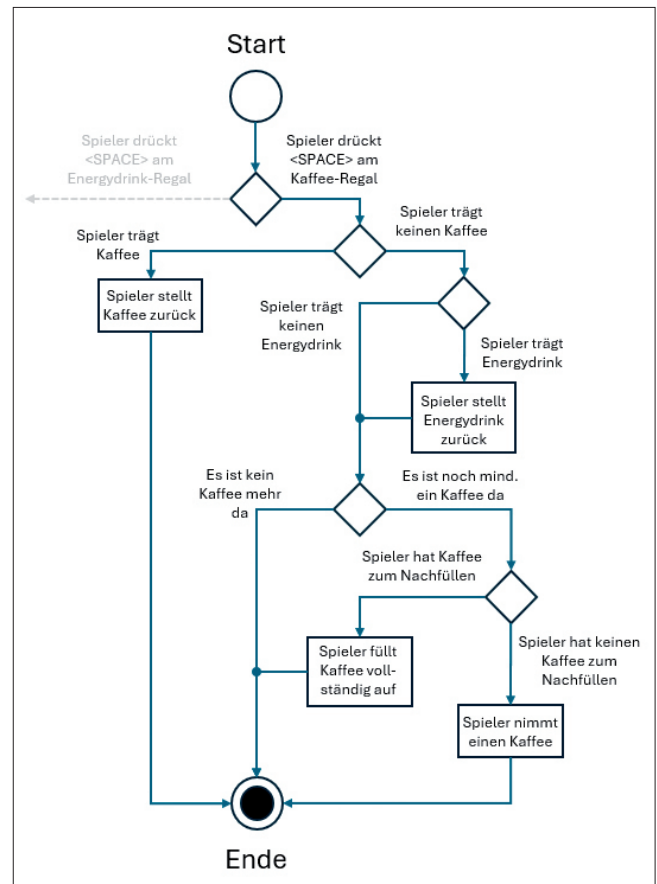
```
coffee1 = objectsLayer.GetObjectByName("Coffee1");
coffee2 = objectsLayer.GetObjectByName("Coffee2");
coffee3 = objectsLayer.GetObjectByName("Coffee3");
```

```
coffeeObjects = [coffee1, coffee2, coffee3];
```

In der *Update*-Methode geht es nun rund. Bisher haben Sie lediglich geprüft, ob Sie am Regal stehen (beziehungsweise auf welcher Seite), dann das jeweilige Sprite sichtbar geschaltet (und das andere unsichtbar) und im Wesentlichen war es das. Nun aber wird es etwas komplizierter. Vielleicht hilft für das bessere Verständnis ein erneutes Nassi-Shneiderman-Diagramm aka Struktogramm!

„Neeeeein!!!“, höre ich Sie schreien und knicke ein. Okay, dann eben ein Ablaufdiagramm. Mit Flow Design bin ich noch nicht so weit. Sehen Sie sich einmal **Bild 7** an und gehen Sie die Fälle einmal durch.

Drückt der Spieler auf der Kaffeeseite des Regals die Leerschritt-Taste, während er schon einen Kaffee in der Hand hält, dann stellt er den Kaffee zurück. Trägt er stattdessen einen Energydrink mit sich herum, stellt er diesen zurück und nimmt sich dafür einen Kaffee – falls noch welcher da ist. Ist keiner mehr da, stellt sich die Frage, ob der Spieler welchen zum Nachfüllen dabei hat. In dem Fall wird das Regal wieder voll aufgefüllt. Ansonsten: Pech gehabt. Im Code stellt sich dieser Ablauf relativ komplex dar. Und genau deshalb, weil solche Abläufe im Kopf zunächst einmal klar scheinen, sich



Der Getränke-Flow (Bild 7)

im Code aber eher komplex darstellen, hilft es tatsächlich, sie in einem ersten Schritt sinnvoll zu visualisieren.

Meinen Vorschlag zu einer möglichen Umsetzung finden Sie in *MainScreen.Update*. Der Code mag aufgrund der vielen Verzweigungen zwar kompliziert aussehen, ist im Grunde aber extrem langweilig. Deswegen ist er hier auch nicht abgedruckt. Enthalten sind auch ein paar nette Gimmicks, die steuern, wann welche Objekte vom Regal verschwinden oder wieder auftauchen. Und ein fast leeres Regal (**Bild 8**) motiviert doch direkt zum nächsten Schritt!

Nachdem die Besucher der Konferenz das Regal nahezu abgeräumt haben, wird es nun endlich Zeit, es auch wieder aufzufüllen. Das Interessante daran ist: Es fällt mir schwer, Ihnen hierzu nun noch etwas Neues zu erzählen. Erweitern Sie einfach den *StorageRoom* um ein paar Zeilen, um die Kollision mit den entsprechenden Regalen zu prüfen und darauf zu reagieren. In dem einen oder anderen Screenshot haben Sie bei mir vielleicht schon die Box unten rechts wahrgenommen. Die wäre hier ideal dazu geeignet, das „Inventar“ anzuzeigen (**Bild 9**).

Und so ergibt der komplexe Workflow aus dem Ablaufdiagramm auch noch mehr Sinn. Denn sobald Ihr Player Kaffee oder Energydrinks aus dem *StorageRoom* mit sich herumträgt, muss eine andere Form der Visualisierung her als die, bei der der Player eine Flasche oder eine Kaffeetasse trägt. Nun ist ganz klar, dass er mit den Getränken aus dem Inventar zunächst das Regal wieder auffüllen kann. Wunderbar!

Recap

Der Abschluss dieser Serie erscheint mir als geeigneter Zeitpunkt, einen kleinen Rückblick zu wagen und damit auch die Themen noch einmal Revue passieren lassen, mit denen Sie sich in den vergangenen Monaten beschäftigt haben.

Mit „Es begann mit XNA ...“ [5] haben Sie zunächst vieles über die Herkunft des MonoGame-Frameworks gelernt und zudem Ihr erstes Projekt angelegt. Für einen ersten Test haben Sie einen Pixelhaufen mittels der orthografischen 2D-Kamera aus MonoGame.Extended pixelgenau skaliert und dabei etwas über Transformationsmatrizen gelernt.

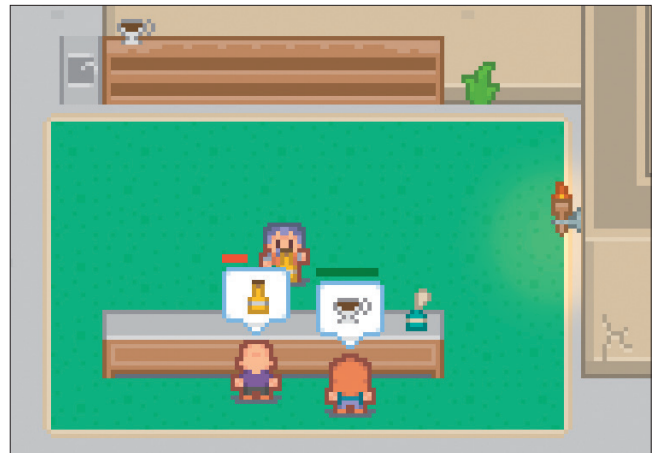
In „Da bewegt sich doch was?“ [6] sind Sie im wahrsten Sinne des Wortes einen Schritt weiter gegangen. Sie haben die Asset Packs von Kenney Vleugels kennengelernt, ein SpriteSheet in Ihr Spiel eingebunden und dieses erstmalig bewegt. Sogar mit den Cursor-Tasten.

Mit dem dritten Artikel „Ja wo läuft er denn hin?“ [7] kam das Tooling ins Spiel. Mit dem Programm Tiled haben Sie Ihre erste Map erstellt, diese in Ihrem Spiel geladen und dabei sogar einzelne Tiles animiert. Darüber hinaus haben Sie gelernt, wie die Kamera Ihrem Spieler folgen kann und wie Sie mit dem Programm Aseprite an Character-Animationen arbeiten können. Die erste Sprite-Klasse hielt in Ihr Spiel Einzug, aber auch der erste wirklich hartnäckige Bug.

Teil 4 – „Perfekte Pixel für Enthusiasten [8] – hat zunächst einmal gezeigt, wie Sie diesen Bug wieder loswerden (erinnern Sie sich noch an die Streifen?) und zugleich die Kamera besänftigt (im wahrsten Sinne des Wortes). Bei der Gelegenheit konnten Sie direkt etwas über den Rasterizer lernen. Der Player hat eine Idle-Animation spendiert bekommen, und mit MonoGame.ImGuiNet haben Sie ein Framework kennengelernt, das es Ihnen ermöglicht, zur Laufzeit Debugging-Informationen einzublenden. Den fulminanten Abschluss dieses Artikels bildete der erste Kontakt mit dem Thema Collision Detection.

„Zwischen den Ebenen“ [9] hatte es in sich. Inzwischen waren Sie schon regelmäßige Refactorings gewohnt, und in Teil 5 sind Sie konsequenterweise von Ihrer eigenen Sprite-Implementierung auf die aus MonoGame.Extended gewechselt. In dem Kontext haben Sie das Thema Player-Tile-Collision weiter ausgeführt und etwas über die Unterscheidung zwischen Full-Body- und Leg-Collision gelernt. Aber auch darüber, wie wichtig es ist, in einer Tiled-Map mit Ebenen zu arbeiten und diese in der richtigen Reihenfolge zu rendern. Abschließend haben Sie das Zeichnen der Map noch auf den sichtbaren Ausschnitt begrenzt und sich erste Gedanken zum Thema Screen-Management gemacht.

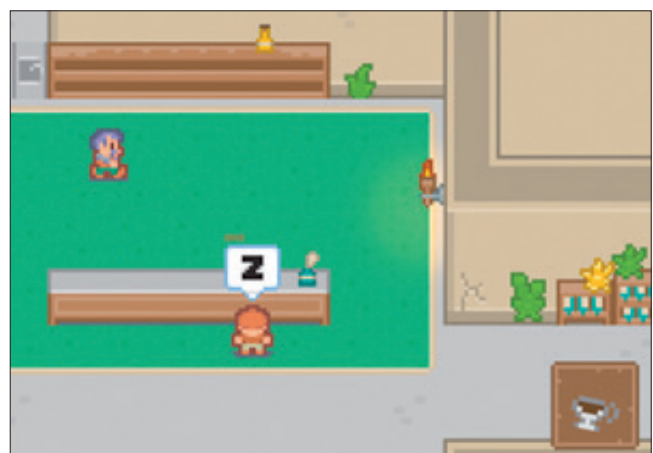
Teil 6 [10] dieser kleinen Serie hätte Sie fast dazu gebracht, mit GameComponents zu arbeiten, dann dagegen gewettert, nur um mit den GameScreens dann doch welche einzuführen. Natürlich war auch der Wechsel zwischen solchen ein Thema. Auf einmal gab es nicht mehr nur die Lounge, sondern auch schon mal so etwas wie eine Abstellkammer – den späteren StorageRoom. Außerdem haben Sie begonnen, NPCs in Ihr Spiel zu integrieren ... und Musik! Endlich! Selten allerdings gibt es neue Features ohne Probleme, und so konnten Sie direkt erfahren, dass einen Track in einer Schlei-



Nachfüllen! Jetzt! (Bild 8)

fe abzuspielen mit den integrierten Möglichkeiten von MonoGame nicht ohne Weiteres möglich ist – oder eben doch, wenn man weiß, wie. Sie wissen es jetzt ja.

„Einfach um die Ecke gebracht“ [11] war (zugegeben) so eine Art gedruckter Clickbait. Natürlich musste hier niemand das Gras von unten wachsen sehen: Es ging vielmehr darum, die Playersteuerung dergestalt zu optimieren, dass der Spieler nicht mehr an irgendwelchen Ecken hängen bleiben konnte. Das stellte sich technisch gesehen als eine recht komplexe Herausforderung dar; im Ergebnis war es die Mühe aber wert. Adaptive Collision Response haben wir diese Technik genannt. Auch rückte erstmals die Frage in den Vordergrund „Ab wann ist ein Spiel ein Spiel?“. Ihre NPCs haben Sie optisch um mehr Visualisierungsmöglichkeiten erweitert und aus einer einfachen Liste von Wegpunkten in diesem Artikel eine eigene *WayPoint*-Klasse mit noch mehr Möglichkeiten für Ihre NPCs entwickelt. In Tiled haben Sie etwas über das sogenannte Off-Grid Positioning gelernt und konnten damit Ihre Map noch interessanter gestalten. Inzwischen liefen dort bereits mehrere NPCs herum und konnten auch in unterschiedliche Richtungen schauen sowie stehen bleiben. Der Einsatz von Soundeffekten hat den Fortschritt des Spiels in diesem Artikel zudem abgerundet. ►



Ein Inventar muss her (Bild 9)

Im achten Teil [12] Ihres Einstiegs in die Welt der MonoGame-Spieleentwicklung wurde die Aussage „Das finden Sie im Repository“ zu einem geflügelten Wort. Das hat den einfachen Grund, dass das Manuskript auf 15 Seiten angewachsen war, während nur zehn geplant waren. Ganz klar, was dabei als Erstes der Schere zum Opfer fiel. Dennoch hielt dieser Artikel immer noch eine Menge Inhalte für Sie bereit! Sie haben die Waypoint-Navigation schrittweise in eine Finite State Machine umgewandelt, mit Ihrem Player tatsächlich Getränke vom Regal genommen, sie den inzwischen recht pfliffigen NPCs überreicht – die endlich richtige Bestellungen aufgeben konnten – und abschließend erste richtige Game-UI-Elemente auf den Bildschirm gezaubert. Dabei konnten Sie durch das Thema „Nine Patch“ mal wieder etwas in Ihr eigenes Glossar zu Begriffen aus der Welt der Spieleentwicklung hinzufügen.

Mit dieser Fortsetzung – Teil 8.2 sozusagen, der Verlängerung, denn eigentlich war auch geplant, diese Serie nach acht Teilen enden zu lassen – wurden Ihre NPCs dank der Sprungmarken noch etwas interessanter in ihrem Verhalten, und durch ein paar optische Tricks sah es gleich so aus, als gäbe es viel mehr davon, als da tatsächlich unterwegs sind (Ablenkung ist alles!). Dabei haben Sie auch endlich etwas über Pixel Shader gelernt, wenn auch nicht so, wie Sie vermutet hätten. Auch wurde das Spiel durch die Vergabe von Punkten so richtig interessant, und weitere visuelle Effekte wie der flackernde Lichtschein der Fackel haben das Gesamtbild abgerundet. Während die Idee mit LuaScript eher unter ferner liefen stattfand, konnten Sie das Spielkonzept selbst mit dem Restocking-Mechanismus für den Moment vollenden.

Fazit und Ausblick

Wie schon erwähnt, fehlt natürlich noch ein bisschen, um das Spiel tatsächlich fertigzustellen. So, wie es sich gerade präsentiert, ließe es sich kaum auf beispielsweise itch.io hochladen. Das fängt schon damit an, dass das Spiel nicht sinnvoll aufhört: Zeit abgelaufen, Spiel weg. Was aber das Fazit zu dieser Serie aus meiner Perspektive betrifft, bin ich überzeugt davon, dass Sie inzwischen über alles Handwerkszeug verfügen, um Ihr Spiel zu komplettieren. Zu Beginn dieses Artikels habe ich dazu schon ein paar Gedanken zusammengestellt: finales Balancing, Start-Menü, Optionsmenü, Pause-Menü, Game-Over-Screen mit Highscore-Anzeige, eventuell ein Online-Leaderboard und sicherlich noch mehr.

Innerhalb des letzten Jahres hat sich übrigens auch einiges in der MonoGame-Welt getan. Falls Sie daran interessiert sind, lesen Sie unter [13] einen schönen Blogbeitrag zu dem Thema von der MonoGame Foundation selbst. Für angehende Spieleentwickler wie Sie ist dabei sicherlich eines der Highlights ein in sich vollständiges Platformer-2D-Starter Kit, über das Sie unter [14] mehr erfahren können. Enthalten sind darin einige Themen, die wir in dieser Serie noch gar nicht angegangen sind. Dazu gehören eine GamePad-Steuerung (oder Touch), Game Physics, Cross Platform Development, aber auch simple Dinge wie das Laden von Levels und Menüsteuerungen inklusive Game-over-Screen.

Gerade zu Letzterem empfehle ich Ihnen, sich noch mal Gedanken zu machen. Um nämlich ein letztes Mal meine Fra-

ge zu stressen: „Wann ist ein Spiel ein Spiel?“ Einige Antworten haben Sie inzwischen parat. Ziel, Motivation, Ablenkung, Kontrolle. In unserem Spiel zum Beispiel hängt die Punktevergabe davon ab, wie schnell Sie Bestellungen ausgeliefert haben. Und je besser Sie insgesamt performen, desto länger läuft das Spiel. Es ist eine Frage des Balancings, wie weit Sie den Spieler das Ende der Spielzeit hinauszögern lassen wollen (und es wäre eine gute Idee, im Optionsmenü den Schwierigkeitsgrad einstellbar zu machen).

Hier aber eine noch interessantere Idee: Wie wäre es, wenn im Game-over-Screen zusätzlich die finale Punktzahl ermittelt würde? Zu dem Score, den Sie während des Spiels erreicht haben, könnten zum Beispiel weitere Punkte hinzukommen – auf Grundlage der Anzahl bedienter Kunden, oder in Gestalt von Bonuspunkten dafür, ein perfektes Spiel hingelegt zu haben (keine Fehlbedienung und alle Bestellungen ausgeliefert), et cetera. Vielleicht fällt Ihnen noch mehr ein.

Erinnern Sie sich an Spiele wie Super Mario Bros. Auch da wurde die finale Punktzahl erst nach Levelende ermittelt. Auf diese Weise kommt zu dem Gefühl der Kontrolle über das Spielgeschehen noch die Vorfreude und Überraschung hinzu, sobald die Punkte noch einmal weiter in die Höhe schnellen! Probieren Sie es im Zweifel aus, um festzustellen, ob Sie das noch mehr motiviert, das Spiel zu spielen.

Klassiker in diesem Genre speichern zudem den Highscore auch ab. So können Sie sich zumindest selbst zu übertreffen versuchen. Noch interessanter wäre es, könnten Sie sich mit anderen messen. Ob dieses Spiel dazu geeignet ist, sich viral zu verbreiten, sei einmal dahingestellt. Für Sie aber wäre es eine spannende Fingerübung, sich mit der Frage auseinanderzusetzen, wie sich ein Online-Highscore – also praktisch ein Leaderboard – implementieren ließe. Ohne das Konzept gerade zu diesem Spiel bereits vollständig angegangen zu sein, wäre mein erster Gedanke, dazu einfach einen Secret-Gist bei GitHub einzurichten und dann die erforderlichen Daten verschlüsselt per API dort hochzuladen.

Und wie geht es nun weiter? Nun, versprechen kann ich noch nichts, aber so wie es aussieht, wird nicht nur dieses Spiel im Lauf der kommenden Wochen noch eine Vollendung erfahren, es wird darüber hinaus sogar noch an einem alternativen Konzept gearbeitet.

Gestatten, ECS mein Name

Ungewöhnlich für diese Artikelserie folgt auf das Fazit noch ein letzter Zwischentitel. ECS – das steht für Entity Component System, und dabei handelt es sich um eine Art, (primär) Spiele zu entwickeln, die sich erheblich von dem unterscheidet, was wir neun Artikel lang gemacht haben. Die extrem zusammengekürzte Erklärung dürfte wohl in etwa so lauten: Neben dem üblichen Rahmenwerk eines Spiels besteht darüber hinaus alles Weitere aus Entitäten, die sich aus Komponenten zusammensetzen und mittels entsprechender Systeme gesteuert werden.

Hier wird also Form von Funktion stark entkoppelt, was das Hinzufügen von Funktionen deutlich vereinfacht – im Vergleich zu unserer bisherigen Herangehensweise. Die Entität „Player“ beispielsweise könnte sich aus Komponenten wie

einer Position, einer Geschwindigkeit, Gesundheit und mehr zusammensetzen. Beispiel gefällig?

```
public static class PlayerFactory
{
    public static Entity CreatePlayer(
        World world,
        Vector2 position,
        AnimatedSprite animatedSprite)
    {
        var entity = world.CreateEntity();
        entity.Attach(new PositionComponent(
            position.X, position.Y));
        entity.Attach(new VelocityComponent(0, 0));
        entity.Attach(new InputComponent());
        entity.Attach(new AnimatedSpriteComponent(
            animatedSprite));
        entity.Attach(new PlayerComponent());
        entity.Attach(new HealthComponent(100));
        // Maximal 100 Lebenspunkte

        return entity;
    }
}
```

Derartige Komponenten sind dabei in aller Regel hochgradig spezialisiert. Das können Sie gut zum Beispiel an der Positions-Komponente erkennen:

```
public class PositionComponent(float x, float y)
{
    public float X { get; set; } = x;
    public float Y { get; set; } = y;
}
```

Beispiele für Systeme können dabei Animationssysteme sein, oder Collision-Systeme, Input-Systeme und mehr. Über allem steht dabei die Spielewelt (World), der Sie alle diese Systeme übergeben und dann in jedem Frame aktualisieren lassen.

Die Bibliothek MonoGame.Extended enthält bereits eine ECS-Implementierung, die Sie verwenden können. Mehr darüber erfahren Sie in der kleinen Einführung in der Dokumentation von MonoGame.Extended unter [15]. Enthalten in dem Framework ist alles, was Sie zum Start benötigen würden. Allerdings – das gebe ich zu – ist das Thema Entity Component System für Einsteiger in die Spieleentwicklung nicht so einfach greifbar, da sich die Konzepte darin doch erheblich davon unterscheiden, wie Sie bisher gewohnt waren, Spiele zu entwickeln.

Aus diesem Grund plane ich aktuell eine neue Serie, die sich hinsichtlich des Spiels selbst an der Idee des Spiels aus dieser Serie orientiert, sich dabei aber mehr auf die technische Implementierung mithilfe eines ECS fokussiert. Damit das klappt und wir uns nicht in etwaigen Details verlieren, wird es für das neue Spiel wichtig sein, nicht visuell auszuweichen. Und ohne nun weiter darauf einzugehen (halten Sie einfach Ausschau danach und schreiben Sie mich an, wenn

Sie informiert sein wollen), bekommen Sie hiermit einen letzten Teaser.

Das Spiel dieser Serie hat nun einen finalen Namen: Deadline Drinks. Und das Spiel, das darauf basiert, aber technisch vollkommen anders mittels Entity Component System aufgesetzt werden wird, bekommt einen davon abgeleiteten Namen: Deadline Drinks Redux (Untertitel: Das Spiel, das Sie auf Ihrem Nokia hätten haben wollen).

Für jetzt, hier und heute aber verabschiede ich mich nach 96 Seiten Einstieg in die Welt der Spieleentwicklung mit dem MonoGame-Framework vorübergehend von Ihnen und hoffe, Sie hatten viel Spaß und konnten viel (Neues) lernen. Vielleicht arbeiten Sie ja schon an Ihrer eigenen Spiele-Idee. Probieren Sie doch auch einfach mal, an einer Game Jam oder einer der zahlreichen Game Development Challenges teilzunehmen. Ich wünsche Ihnen viel Spaß dabei! ■

- [1] Ein perfektes Spiel, <https://youtu.be/dmNW05GYrkQ>
- [2] Particle Asset Pack, <https://kenney.nl/assets/particle-pack>
- [3] Lua, <https://lua.org>
- [4] NLua, <https://github.com/NLua/NLua>
- [5] Christian Jacob, Es begann mit XNA..., dotnetpro 6/2024, Seite 118 ff., www.dotnetpro.de/A2406MonoGame
- [6] Christian Jacob, Da bewegt sich doch was?, dotnetpro 7/2024, Seite 128 ff., www.dotnetpro.de/A2407MonoGame
- [7] Christian Jacob, Ja wo läuft er denn hin?, dotnetpro 8/2024, Seite 104 ff., www.dotnetpro.de/A2408MonoGame
- [8] Christian Jacob, Perfekte Pixel für Enthusiasten, dotnetpro 9/2024, Seite 120 ff., www.dotnetpro.de/A2409MonoGame
- [9] Christian Jacob, Zwischen den Ebenen, dotnetpro 10/2024, Seite 110 ff., www.dotnetpro.de/A2410MonoGame
- [10] Christian Jacob, Wie neu geboren, und das im Loop, dotnetpro 12/2024, Seite 122 ff., www.dotnetpro.de/A2412MonoGame
- [11] Christian Jacob, Einfach um die Ecke gebracht, dotnetpro 1/2025, Seite 114 ff., www.dotnetpro.de/A2501MonoGame
- [12] Christian Jacob, Heute wird gespielt!, dotnetpro 2/2025, Seite 128 ff., www.dotnetpro.de/A2502MonoGame
- [13] MonoGame Jahresrückblick 2024, www.dotnetpro.de/SL2503MonoGame1
- [14] MonoGame Platformer 2D Starter Kit, www.dotnetpro.de/SL2503MonoGame2
- [15] MonoGame.Extended ECS, www.dotnetpro.de/SL2503MonoGame3



Christian Jacob

ist Entwickler und Co-Founder der Blue Sky Innovations GmbH. Er unterstützt Unternehmen bei der Einführung moderner Strategien sowie bei Planung und Umsetzung individueller Softwarelösungen auf Basis von Microsoft .NET.

cjacob@blueskyinnovations.de

dnpCode

A2503MonoGame

ARBEITSMARKT

TRENDS UND JOBS FÜR ENTWICKLER

Bitkom

Rund 20 000 neue IT-Jobs

Der Digitalverband Bitkom erwartet im deutschen Markt für IT und Telekommunikation (ITK) für 2025 ein Umsatzplus von 4,6 Prozent auf 232,8 Milliarden Euro. Im vergangenen Jahr hatten die ITK-Umsätze um 3,3 Prozent auf 222,6 Milliarden Euro zugelegt.

Parallel soll die Zahl der Beschäftigten im ITK-Sektor laut Bitkom im Jahresverlauf 2025 um rund 20 000 auf 1,371 Millionen wachsen. Im Jahr 2024 sind lediglich 9000 neue Arbeitsplätze entstanden.

Vor allem das Geschäft mit Software soll nochmals stark zulegen (plus 9,8 Prozent auf 51,1 Milliarden Euro). Insbesondere der anhaltende Boom bei Künstlicher Intelligenz sticht hervor: Das Geschäft mit KI-Plattformen, auf denen KI-Anwendungen entwickelt, trainiert und betrieben werden können, soll laut Bitkom-Prognose ra-

sant um 43 Prozent auf 2,3 Milliarden Euro wachsen.

Kollaborationstools zur Zusammenarbeit und zum mobilen Arbeiten in Unternehmen wachsen ebenfalls stark: um 12 Prozent auf 1,4 Milliarden Euro. Sicherheitssoftware steigt um 11 Prozent auf 5,1 Milliarden Euro an.

Zweistellige Wachstumsraten werden außerdem bei Cloud-Services vorhergesagt, die um 17 Prozent auf 20 Milliarden Euro zulegen sollen. Die Umsätze mit IT-Dienstleistungen insgesamt steigen laut Bitkom 2025 um 5,0 Prozent auf 53,8 Milliarden Euro.

Bitkom-Präsident Dr. Ralf Wintergerst: „Die Digitalwirtschaft bietet einen Lichtblick in schwierigen Zeiten, steigert den Umsatz und schafft neue Jobs. Mittlerweile ist die ITK-Branche Deutschlands größter industrieller Arbeitgeber. Die Geschäfte der meisten Unternehmen der IT- und Telekommunikationsbranche verlaufen den Prognosen zufolge 2025 positiv.“

Qlik

KI-Bestandsaufnahme

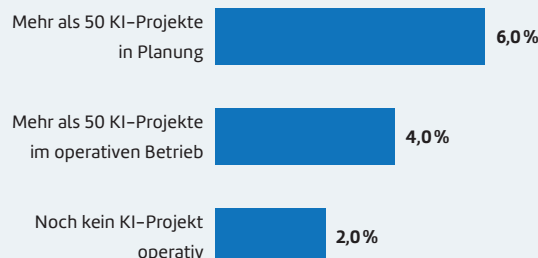
Qlik, Anbieter für Datenintegration, Analysen und Künstliche Intelligenz (KI), hat in seiner Studie „Build or Buy“ 4200 C-Suite-Führungskräfte und KI-Entscheider zu ihrer Einschätzung befragt, was den KI-Fortschritt behindert und wie diese Hindernisse überwunden werden können.

85 Prozent der Entscheider hiezulande halten den Einsatz von KI für den eigenen geschäftlichen Erfolg für absolut notwendig oder zumindest sehr wichtig. Dementsprechend findet sich auch kein einziger Befragter, in dessen Unternehmen sich nicht mindestens ein KI-Projekt in der Planung befindet, und nur bei 2 Prozent ist noch kein Projekt operativ. 6 Prozent der Führungskräfte hingegen geben an, dass sich bei ihnen bereits mehr als 50 Projekte in der Planung befinden, bei 4 Prozent sind so viele sogar schon im operativen Betrieb (Bild 1).

Auf dem Weg zum erfolgreichen Go-Live der KI-Projekte nennen die Befragten einige Hürden. Insbesondere deutsche Entscheider sehen einen Mangel an Fachkräften mit geeigneten Qualifikationen für die Entwicklung und Umsetzung von KI-Projekten (Bild 2), insbesondere für die Betreuung nach der Einführung. Als weitere Probleme genannt werden ein nicht ausreichend gutes Datenmanagement und ein Mangel an vertrauenswürdigen Daten, die für die Nutzung von KI geeignet sind. Budgetmangel ist nur bei 18 Prozent der Befragten ein relevanter Punkt.

Es zeigt sich: Deutsche Unternehmen wollen KI, sehen sich aber nicht in der Lage, entsprechende Projekte vom Reißbrett weg selbst zu entwickeln. 70 Prozent der Befragten bezeichnen extern entwickelte KI-Lösungen als gute Grundlage. Nur etwas mehr als die Hälfte aller Befragten greift überwiegend auf Eigenentwicklungen zurück.

● KI in Unternehmen

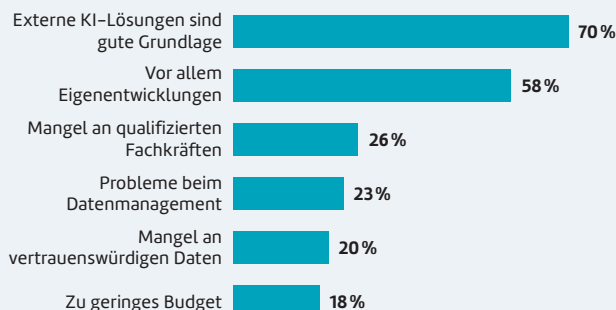


Bei der Umsetzung von KI-Projekten sind deutsche Unternehmen noch nicht sehr weit (Bild 1)

dotnetpro 3/2025

Quelle: Qlik-Studie Build or Buy, Mehrfachnennungen

● KI-Umsetzung



Die meisten deutschen Unternehmen stehen externen KI-Lösungen offen gegenüber. Etwas mehr als die Hälfte plant vor allem Eigenentwicklungen (Bild 2)

dotnetpro 3/2025

Quelle: Qlik-Studie Build or Buy, Mehrfachnennungen

Zahl des Monats

Zumindest einen Teil ihrer täglichen Aufgaben wollen **92 Prozent** der Arbeitnehmer weltweit laut einer Umfrage an KI abgeben. In Deutschland würden fast **42 Prozent** der Befragten ihren derzeitigen Arbeitsplatz gegen eine Stelle tauschen, an der KI effektiver eingesetzt wird. **30 Prozent** der Befragten gaben an, dass sie durch die Nutzung von KI bereits eine Gehaltserhöhung oder eine Beförderung erhalten haben.

Quelle: Freshworks, AI Workplace Report

Veracode

KI: Auswirkungen für Entwickler

GenAI-generierter Code wird Unternehmen mit mehr Sicherheitsschulden belasten. Chris Wysopal, Chief Security Evangelist bei Veracode: „Mit KI-generiertem Code steigt die Geschwindigkeit beim Coden. Im Gegenzug werden auch die Anzahl der Schwachstellen und die Höhe der kritischen Sicherheitsschulden zunehmen. Da immer mehr Code in rasantem Tempo erstellt wird, werden Entwickler mit Compliance-Risiken, Sicherheitswarnungen und Qualitätsproblemen überschwemmt werden. Die Suche nach einer Lösung, die hier Abhilfe schafft, ist entscheidend. Die Verwendung von GenAI zum Schreiben von Code ist der Verwendung der gleichen Technologie für die Sicherheitshärtung und -behebung derzeit zwei Jahre voraus“, so Chris Wysopal. „Aus diesem Grund ist 2025 mit einem Anstieg der Nutzung von

KI-gestützter Fehlerbehebung zu rechnen, um Schwachstellen schneller zu beseitigen und so die Sicherheit von Unternehmen zu verbessern.“

Sohail Iqbal, Chief Information Security Officer bei Veracode: „KI-Anwendungen lernen dazu und werden immer intelligenter. Damit wird die Kluft zwischen den Anforderungen der Unternehmen und den Kenntnissen der arbeitssuchenden Fachkräfte immer größer. Viele befürchten, dass wir uns in Sachen Sicherheit zu sehr auf KI verlassen. Aus diesem Grund werden GenAI-Prompting-Fähigkeiten sowohl für Entwickler als auch für Sicherheitsteams im Jahr 2025 und darüber hinaus unverzichtbar werden. Wir werden mehr Prompting-Engineering-Rollen sehen, die eine Weiterentwicklung der traditionellen System-Engineering-Rolle darstellen. Menschliche Fähigkeiten wie Geschäftssinn und Prozessmanagement werden jedoch für Sicherheits- und Entwicklerteams weiterhin un-

erlässlich sein, um erfolgreich zu arbeiten.“

IT-Jobs

KI-Wissen wird unverzichtbar

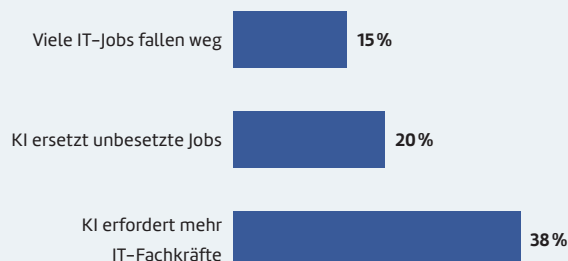
Der Branchenverband Bitkom ist der Frage nachgegangen, ob KI in der IT zu Jobverlusten führen wird. Dazu wurden 852 Unternehmen in Deutschland befragt. 15 Prozent dieser Unternehmen gehen davon aus, dass es durch KI-Einsatz zu einem Stellenabbau bei IT-Fachkräften kommt. 20 Prozent rechnen damit, dass durch Künstliche Intelligenz Stellen ersetzt werden, für die sich ohnehin keine geeigneten Bewerber finden. Umgekehrt sagen 38 Prozent der befragten Unternehmen, dass KI für einen zusätzlichen Bedarf an IT-Fachkräften im Unternehmen sorgen wird, vergleiche Bild 3.

Bitkom-Präsident Dr. Ralf Wintergerst: „Angesichts des strukturellen Fachkräftemangels und der absehbaren de-

mografischen Entwicklung ist KI keine Bedrohung für den Arbeitsmarkt in der IT. KI ist eine Chance, die Fachkräftelücke zumindest teilweise zu schließen. Dadurch können IT-Projekte und -Aufgaben in Deutschland umgesetzt und gehalten werden, für die es ansonsten hierzulande keine Kapazitäten geben würde.“

KI soll starke Auswirkungen auf IT-Berufe haben. So geht ein Drittel der Befragten davon aus, dass neue Berufsbilder entstehen, wie etwa KI-Trainer oder Prompt Engineer (Bild 4). Auf der anderen Seite werden nach Ansicht von rund einem Viertel einzelne IT-Berufe und -Berufsbilder künftig weitestgehend durch KI ersetzt und damit verschwinden. Knapp ein Fünftel sagt zudem, dass IT-Fachkräfte ohne KI-Wissen in Zukunft kaum mehr nachgefragt werden. 44 Prozent der Unternehmen gehen davon aus, dass KI die Produktivität in der IT erhöht, weil sie Beschäftigte von Routineaufgaben entlastet.

● KI und IT-Jobs

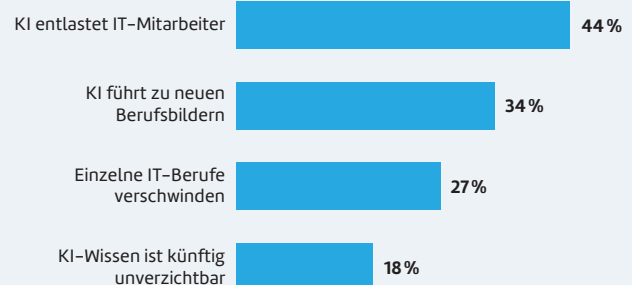


Mehr als ein Drittel der Unternehmen sagen, dass KI für einen zusätzlichen Bedarf an IT-Fachkräften sorgt (Bild 3)

dotnetpro 3/2025

Quelle: Bitkom, n=852, Mehrfachnennungen

● KI: Auswirkungen auf IT-Personal



Neue, durch KI geprägte Berufsbilder erwarten 34 Prozent der befragten Unternehmen (Bild 4)

dotnetpro 3/2025

Quelle: Bitkom, n=852, Mehrfachnennungen



developer media jobs

Der Stellenmarkt für Entwickler

**ES GEHT NICHT
UM EINEN JOB.**

**ES GEHT UM
DEINEN JOB!**

**JETZT
TRAUMJOB
FINDEN!**



jobs.developer-media.de

**Reinhold Fritsch, Tel.: +49 731 88005-8285 | Sabine Vockrodt, Tel.: +49 731 88005-8222
oder per E-Mail an: jobs@developer-media.de**

JOBS für Entwickler

Auf jobs.developer-media.de finden
Software-Entwickler und -Architekten
ständig aktuelle Job-Angebote

Ihr Job-Angebot fehlt?

Sie suchen Entwickler in Festanstellung oder projektbezogen?
Dann sollten auch Sie unser Online-Jobportal für Ihre Mitarbeitersuche nutzen.
Ihr Stellenmarkt-Team berät Sie gern zu den attraktiven Promotionmöglichkeiten.

Gerne können Sie Ihre Stellenausschreibung zu jederzeit online
selbst einbuchten unter jobs.developer-media.de



Diese und viele weitere aktuelle Job-Angebote finden Sie auf jobs.developer-media.de

DFS Deutsche Flugsicherung GmbH	Software Engineer (w/m/d) für Flugsicherungssysteme	Langen bei Frankfurt am Main
Landesbank Hessen-Thüringen	Position Application Manager SAP ERP (w/m/d)	Offenbach am Main, Erfurt
Helvetia Schweizerische Versicherungsgesellschaft AG	Netzwerkadministrator IT-Infrastruktur (m/w/d)	Frankfurt am Main
Postnova Analytics GmbH	Junior Softwareentwickler C# (m/w/d)	Landsberg am Lech
Katholisches Klinikum Bochum GmbH	IT-Leitung (m/w/d)	Bochum
Justus-Liebig Universität Giessen	IT-Administrator/in (m/w/d) für (BOS-)Funksysteme und Gefahrenmeldesysteme	Gießen
Hochschule Worms	Informationssicherheitsbeauftragte/r (m/w/d)	Worms
Landesbank Hessen-Thüringen	Team Lead IT-System Engineer z/OS Mainframe (w/m/d)	Offenbach am Main, Erfurt
PMG AG	Fachinformatiker / Systemtechniker (Voice Solutions) (m/w/d)	Inning am Ammersee

Ihre Anzeige ist noch nicht dabei? Rufen Sie uns an:
Reinhold Fritsch, Tel.: +49 731 88005-8285, E-Mail an: jobs@developer-media.de

● Impressum

**Ebner Media Group GmbH & Co. KG****Büro München**

Brienner Str. 45 a-d, 80333 München

Sitz von Redaktion, Anzeigen, Vertrieb

Brienner Str. 45 a-d, 80333 München

Telefon +49 731 88005-8000

info@dotnetpro.de

www.dotnetpro.de

Sitz des Verlags

Ebner Media Group GmbH & Co. KG

Karlstraße 3, 89073 Ulm

Geschäftsführer

Marco Parrillo

Chefredaktion

Fernando Schneider (ViSdP)

Redaktion

Tilman Börner

Freie Mitarbeiter dieser Ausgabe**Redaktion**

Ernst Altmannshofer, Sabine Block,
Bernhard Lauer

Autoren

Klaus Aschenbrenner, Elena Bochkor, Tilman Börner,
Dr. Fabian Deitelhoff, Christian Giesswein,
Tam Hanna, Christian Havel, Christian Jacob,
Corinna John, Sebastian Kleinschmager, Roland König,
Dr. Veikko Krypczyk, Bernhard Lauer, Stefan Lieser,
Robert Mühsig, Bernhard Pichler, Jan Hendrik Schreier,
Dr. Holger Schwichtenberg, Christian Sender,
Erik Stroeken, David Tielke, Helmut Vonhoegen,
Christian Wenz, Alexander Zeitler

Art Directorin

Maria-Luise Steinkühler

Gestaltung**EMG Design Unit**

Dagmar Breitenbauch, Simone Köhnke,
Petra Reichensperner

Kundenservice / Einzelheftbestellung

EBNER MEDIA GROUP,

Brienner Str. 45 a-d, 80333 München

Telefon +49 731 88005-8205

kundenservice@ebnermedia.de

(Mo.-Do. 9.00 Uhr bis 17.00 Uhr,

Fr. 9.00 Uhr bis 15.00 Uhr)

Erscheinungsweise

dotnetpro erscheint zweimonatlich.

ISSN 1610-1553. Postvertriebskennzeichen B30751

Nachdruck

© Nachdruck nur mit schriftlicher Genehmigung

Developer Events

Charlotte Frieß, Director Live Communication Developer
charlotte.friess@ebnermedia.de

Mediaberatung und Content-Marketing-Lösungen

Thomas Deck

Telefon +49 731 88005-8282

thomas.deck@ebnermedia.de

Beratung Stellenmarkt

Reinhold Fritsch

Telefon +49 731 88005-8285

reinhold.fritsch@ebnermedia.de

Client Success (Disposition)

clientsuccess@ebnermedia.de

Leitung Herstellung / Vertrieb / Kundenservice

Thomas Heydn

Anzeigentarife

Anzeigenpreisliste gültig ab 1.1.2025

digital.dotnetpro.de/mediadaten/

Druck

Silber Druck GmbH & Co. KG

Otto-Hahn-Straße 25, 34253 Lohfelden

Sonderdruck

Von allen Beiträgen dieser Ausgabe können für
Werbezwecke Sonderdrucke hergestellt werden.

Anfragen bitte an fernando.schneider@ebnermedia.de

Bezugspreise

dotnetpro ist das Profi-Magazin für Entwickler
und erscheint sechsmal im Jahr.

Das Einzelheft kostet im Inland 24,90 €.

Der Preis für das Premium-Plus-Abonnement

(6 Ausgaben Print + Digital)

beträgt im Inland 182,40 €

(Digital 152,40€ + Print 30,00 €),

Schweiz 304,00 CHF

(Digital 167,70 CHF + Print 136,30 CHF),

sonstiges Ausland 206,40 €

(Digital 152,40 € + Print 54,00 €) pro Jahr.

Abonnement-Bedingungen

Das Abonnement gilt für ein Jahr, anschließend bis
zur Kündigung zum jeweils geltenden Jahresbezugspreis.

Dein Jahresabo ist ab dem Folgejahr auch monatlich

kündbar. Bei vorzeitiger Kündigung deines bereits

abgerechneten Bezugszeitraums erhältst du den

Betrag der Restlaufzeit zurückerstattet.

Veröffentlichung gemäß Art. 8 Abs. 3 BayPrG:

Alleinige Gesellschafterin der Ebner Media Group
GmbH & Co. KG ist die Ebner Ulm MGv GmbH, Ulm.

● Inserentenverzeichnis

Ebner Media Group

www.ebnermedia.de 3-6, 7,

15, 49, 52, 65, 71, 87,

115, 142-143, 147, 148

Autor:in werden!

Helfen Sie uns,
Softwareentwickler
weiterzubilden.

Schreiben Sie
Artikel für die
dotnetpro

info@dotnetpro.de

Anbietersverzeichnis

Persönliche Beratung bei Thomas Deck | +49 731 88005-8282 | thomas.deck@ebnermedia.de

	COUNT IT Group Softwarepark 49 4232 Hagenberg/AUSTRIA T: +43 (0)7236 – 20077 – 0 F: +43 (0)7236 – 20077 – 6717 m.szoeky@countit.at www.countit.at Markus Szöky	Die COUNT IT Group ist österreichweit einer der führenden Unternehmensdienstleister und mit über 120 MitarbeiterInnen das größte Unternehmen im Softwarepark Hagenberg. Unser agiles Softwarehouse realisiert neben maßgeschneiderten ERP- und DMS-Lösungen vor allem Individualsoftware-Projekte auf Basis modernster .NET-Technologien. Die Bandbreite reicht hier von Web-Anwendungen bis Softwarelösungen für die Industrie 4.0, wobei für uns als Qualitätsdienstleister Testautomatisierung wesentlich ist. Auf uns zählen (inter) nationale Kunden aus Handel, Baubranche und Industrie. Rechnen auch Sie mit uns!
	DATEV eG Paumgartnerstr. 6 – 14 90429 Nürnberg Deutschland Telefon: +49 911 319-0 https://www.datev.de	Die DATEV eG ist der drittgrößte Anbieter für Business-Software in Deutschland und einer der großen europäischen IT-Dienstleister. Das Leistungsspektrum der DATEV reicht von mehr als 200 PC-Programmen über Cloud-Lösungen wie Online-Anwendungen, Datenverarbeitung und -archivierung im Rechenzentrum bis hin zu Outsourcing-Leistungen sowie Sicherheitsdienstleistungen, mit denen die DATEV Unternehmen, Selbstständige, Kommunen, Vereine und Institutionen unterstützt. Beratungsleistungen und Angebote zur Wissensvermittlung in Deutschland und einigen anderen europäischen Ländern runden das Angebot ab.
	Developer Express, Inc 505 N. Brand Blvd. Suite 1450 CA 91203 Glendale USA ClientServices@devexpress.com www.devexpress.com	DevExpress engineers feature-complete UI Controls, enterprise-ready reporting systems, IDE productivity tools and business application frameworks for Visual Studio. Our technologies help you build your best, see complex software with greater clarity, increase your productivity and create stunning touch-enabled applications for Windows, the web and mobile devices without limits or compromise. To learn more and see how you can create business solutions customers expect today and leverage existing skillsets to build next generation applications for tomorrow, visit us online at www.devexpress.com .
	Kutura GmbH Oberaltenallee 20a 22081 Hamburg Telefon: +49 177 216 70 80 Mail: contact@kutura.digital kutura.digital/contact	Unternehmen, in denen sich Menschen entwickeln und wachsen können, gehört die Zukunft. Das gilt besonders in Zeiten, in denen Künstliche Intelligenz, Krisen und Komplexität unsere Aufmerksamkeit erfordern und unsere mentalen Ressourcen beanspruchen. Kutura entwickelt für und mit ihren Kunden eine zukunftsfähige Führungs- und Unternehmenskultur, die drei Zielen dient: Gemeinsam Ergebnisse schaffen, mehr Motivation und weniger Konflikte, zukünftige Stärke sicherstellen. Reimagining Leadership, Inspiring Happiness.
	prodot GmbH Schifferstraße 196 47059 Duisburg T: +49 (0)203 – 396508 – 0 info@prodot.de https://prodot.de	prodot – Software für Marktführer Intelligente Software für internationale Konzerne und mittelständische Unternehmen: prodot stärkt seit über 20 Jahren namhafte Kunden im weltweiten Wettbewerb mit effizienten, stabilen und kostensenkenden Lösungen. Mit Sorgfalt und Enthusiasmus entwickeln wir hochwertige Software, die sich durch intuitives Design auszeichnet. Digitale Prozesse und innovative Technologien wie IoT, AI, Mobile, Cloud sind unser Antrieb – Fortschritt, Qualität und Kontinuität unser Anspruch. Mit rund 80 Mitarbeitern setzen wir die Anforderungen unserer Kunden in hochwertige Softwareprodukte um. ALDI SÜD, Microsoft, Siemens, TK Elevator und weitere vertrauen uns seit vielen Jahren. Gerne zeigen wir Ihnen warum! Sprechen Sie uns an.
	Space Blocks Permissions as a Service Ein Service der wemogy GmbH Graf-Bernadotte-Str. 23 47906 Kempen Deutschland https://www.spaceblocks.cloud hello@spaceblocks.cloud	Die Developer-Plattform für Rollen- und Rechtemanagement Einfach Rollen und Berechtigungen im grafischen Low-Code-Editor anlegen und Zugriffsabfragen mit wenigen Zeilen Quellcode in die eigenen Anwendungen integrieren. Inklusive RBAC, Multi-Tenancy, hierarchischer Berechtigungsvererbung und vielem mehr. Geschrieben in .NET und für .NET-Entwickler konzipiert. Entweder von uns in der Space Blocks Cloud bereitgestellt oder im Selbstbetrieb. Jetzt ausprobieren mit der kostenlosen Entwickler-Version oder eine unverbindliche, kostenfreie Demo buchen!
	Text Control GmbH Überseetor 18 28217 Bremen Deutschland Telefon: +49 421 42706710 https://www.textcontrol.com/	Text Control wurde 1991 gegründet und ist ein preisgekrönter Microsoft Visual Studio Industry Partner und führender Anbieter von Reporting- und Textverarbeitungskomponenten für Microsoft-Entwickler. Die Produkte helfen tausenden von Entwicklern, Text- und Dokumentenverarbeitung in ihre Windows, Web und Mobil-Anwendungen zu integrieren. Text Control ist zu 100% inhabergeführt und damit komplett unabhängig von Investorenentscheidungen. Kunden profitieren von jahrelanger Erfahrung, einer sehr großen Nutzerbasis und den ausgereiften und zuverlässigen Produkten. Kernprodukt ist das Reporting-Framework, welches die Stärken eines Reporting-Tools mit einer, einfach zu handhabenen, Textverarbeitung kombiniert – voll programmierbar und integrierbar in einer .NET Anwendung.
	Carl Zeiss Digital Innovation GmbH Fritz-Foerster-Platz 2 01069 Dresden Telefon: +49 351 49701 – 500 info.digitalinnovation.de@zeiss.com www.zeiss.de/digital-innovation	ZEISS Digital Innovation – Als Teil der ZEISS Gruppe entwickeln wir individuelle und hochwertige digitale Lösungen, die unter anderem auf modernen Web- und Cloud-Technologien basieren. Mit agilen Methoden unterstützen wir unsere Kunden entlang ihrer Digitalisierungsprojekte – und darüber hinaus. Dabei entstehen innovative Plattformen auf einer skalierbaren Gesamtarchitektur. So steigern wir die Innovationskraft unserer Zielgruppe in ihren Märkten. www.zeiss.de/digital-innovation

Die Ausgabe 4-5/2025 erscheint am 17. April 2025

Bild: Shutterstock / Dave Hoesk



Single Page Applications

SPAs sind aus der Web-Landschaft nicht mehr wegzudenken – nicht ohne Grund, denn sie sorgen dank dynamischer Inhaltsaktualisierung für eine bessere Performance und eine flüssigere User Experience, erlauben durch die Verwendung von Service Workern eine verbesserte Offline-Funktionalität und schaffen durch die strikte API-Fokussierung der Frontend-Backend-Kommunikation ein hohes Maß an Skalierbarkeit, Wiederverwendbarkeit und Integrationsfähigkeit.

Im Schwerpunkt der **dotnetpro 4-5/2025** geben Peter Kröner, Patrick Schnell, Christian Sender und Christian Wenz Einblicke in spannende Themen rund um die SPA-Entwicklung.

Azure Integration Services

Bei der Integration von Anwendungen, Systemen und Datenquellen helfen die Integrationsdienste für Azure.

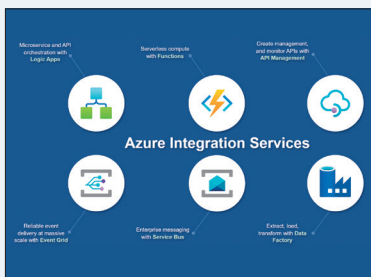
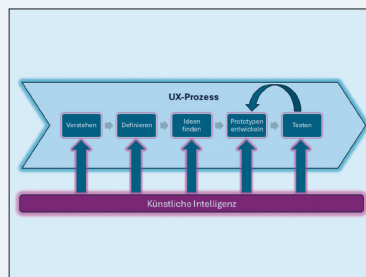


Bild: bloomreach.com

KI in der UI-Gestaltung

Künstliche Intelligenz kann die Produktivität in manchen Bereichen der UI-Gestaltung steigern. Ein Überblick.



CUPID versus SOLID

Jeder, der wartbaren Code schreiben möchte, kennt die SOLID-Prinzipien. Doch kennen Sie auch CUPID?



End of file

There are three kinds of people: those who can count and those who can't.



Unsere digitalen Angebote



Wöchentlicher Entwickler-Newsletter
www.dotnetpro.de/newsletter



dotnetpro digital lesen: Android- und iPad-App
www.dotnetpro.de/dnpapps



YouTube: Videos von den Konferenzen
www.youtube.com/@developerweek



Facebook: die News im Stream
<https://de-de.facebook.com/dotnetpro>



SAVE
THE DATE

.NET DEVELOPER CONFERENCE '25

DIE KONFERENZ FÜR .NET ENTWICKLER
24. - 27. NOVEMBER 2025 | KÖLN

Die .NET Developer Conference ist eine der führenden Plattformen für Weiterbildung in .NET-Technologien. Mit einem umfangreichen Programm aus Vorträgen und Workshops bietet die Konferenz fundiertes Wissen und praxisnahe Einblicke in zentrale Themen des .NET-Bereichs im Zeitalter von KI. Ein Muss für alle, die ihr Know-how erweitern und aktuelle Trends verfolgen möchten.

dotnet-developer-conference.de | #DDC | Find us on [f](#) [ig](#) [in](#)

Präsentiert von:



DEVELOPER WEEK '25

30. JUNI - 3. JULI 2025

M:CON CONGRESS CENTER ROSENGARTEN
MANNHEIM

**JETZT
EARLY-BIRD-
TICKET
SICHERN!***

DAS EVENT FÜR AI, CLOUD, WEB UND .NET

Software Design Principles | Software Architecture | Software Quality | Software Testing | Software Security | Doing Agile/Agile in Practice | DevOps | CI / CD
Git, Github | .NET Fundamentals | .NET Backend (Development)
.NET Frontend (Development) | .NET Doing | Web Frontend | Web Backend
Web Hybrid | Web Design | Web Languages | Cloud Development
Container | Data & Analytics | Databases | GenAI Fundamentals
GenAI Tools & Solutions | Personal Growth | From Me to We | Sustainability
Robotics | IoT & Industry 4.0

developer-week.de | #DWX | Find us on   

*Das Early-Bird-Ticket gilt vom 12.02.2025 bis 11.03.2025

Präsentiert von: 