

.NET 9

.NET, ASP.NET Core, EF Core, Blazor, MAUI, C#:

Was ist neu? Wo liegen die Vorteile?

Wie können Sie sie nutzen?

Ab Seite 6

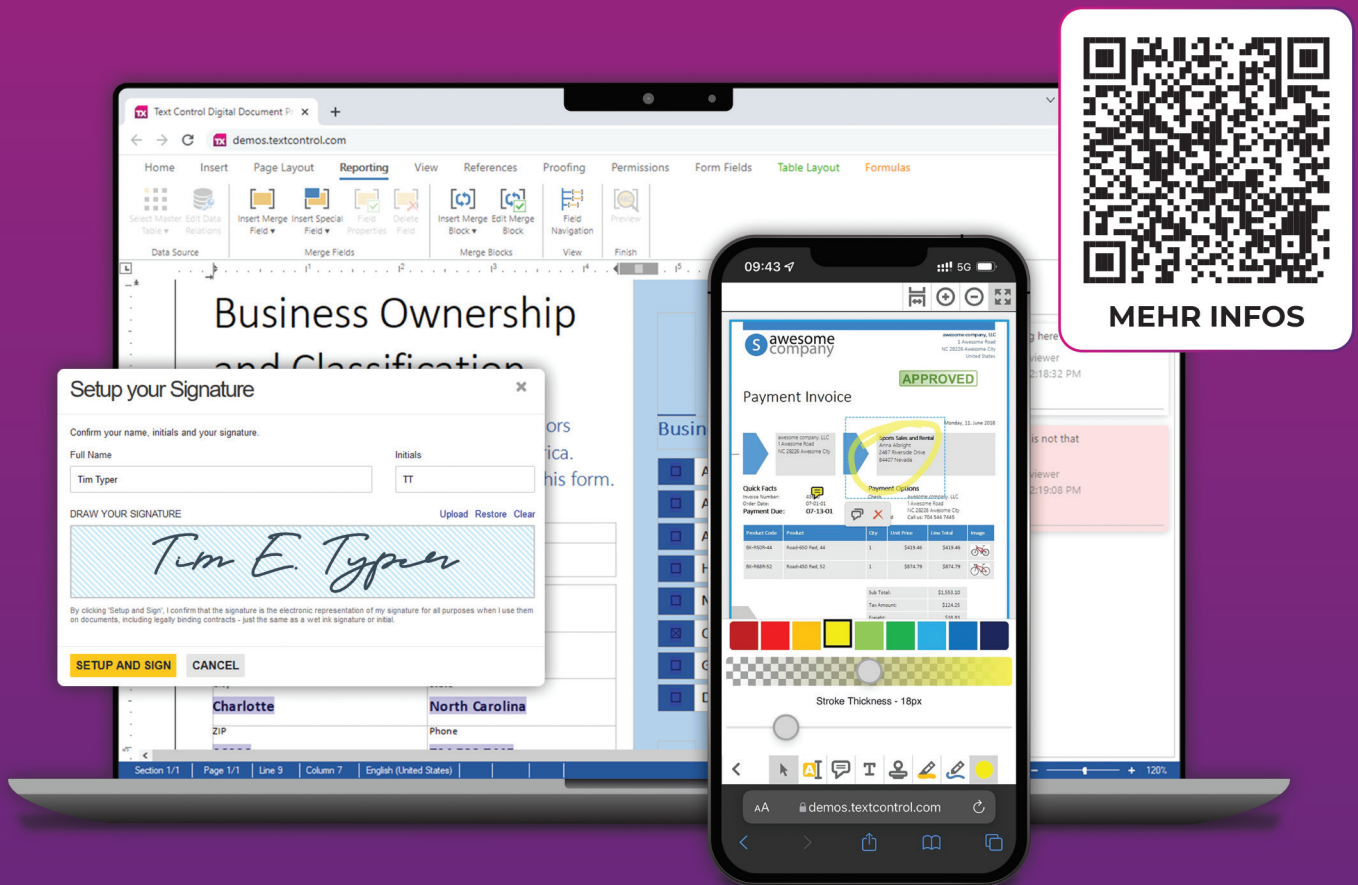


.NET DEVELOPER CONFERENCE '24

Jetzt Programm ab S.39 entdecken und auf dotnet-developer-conference.de

Wir denken Dokumente neu!

Integriere die Bearbeitung, Anzeige, Erstellung und Automatisierung digitaler Dokumente in deine .NET-Anwendungen.





Alle Jahre wieder

Wenn sich der Sommer verabschiedet und die Tage immer kürzer werden, warten Entwickler gespannt auf die neue .NET-Version.

Ein bisschen ist es ja wie Weihnachten, wenn Microsoft gegen Jahresende seine neue .NET-Version veröffentlicht. So wie die Weihnachtsmänner bei schönstem Sonnenschein plötzlich reihenweise in den Supermärkten stehen, erreicht die Entwickler im Frühsommer die Ankündigung der neuen Version. Ehe man sich dann aber versieht, liegt ein Paket unter dem Baum, und man fragt sich, ob darin auch wirklich das steckt, was man sich gewünscht hat.

Zugegeben, der Vergleich hinkt. Denn anders als bei Weihnachtsüberraschungen verrät uns Microsoft im Lauf der Wochen und Monate, womit Softwareentwickler diesmal rechnen dürfen – und womit nicht.

Bereits auf Microsofts Build-Konferenz Ende Mai in Seattle gewann man den Eindruck, dass .NET nicht unbedingt zu den Release-Highlights des Jahres aus dem Hause Microsoft zählen würde. Was irgendwie nachvollziehbar ist, wenn wir uns die dominante Rolle vor Augen halten, die das Thema „Künstliche Intelligenz“ spielt, nicht nur in Redmond.

„Anders als in den Jahren zuvor glänzt die neue .NET-Version durch zahlreiche kleine Verbesserungen.“

Trotzdem halte ich .NET 9 nicht für einen unwichtigen Release, den man ruhigen Gewissens einfach links liegen lassen kann. Richtig ist, dass es diesmal nicht das eine, alles überragende neue Feature gibt, wie es in Vorgängerversionen der Fall war. Doch wenn Sie genauer hinschauen, entdecken Sie bei .NET, ASP.NET Core, Blazor, .NET MAUI, Entity Framework Core und selbst bei WPF durchaus interessante Neuerungen.

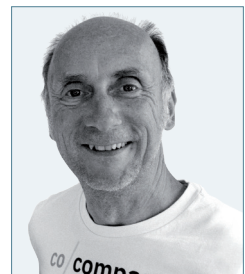
Aus diesem Grund erwarten Sie im Schwerpunkt dieser dotnetpro-Ausgabe nicht wie sonst üblich zwei oder drei, sondern gleich fünf Beiträge, in denen unsere Autoren nicht nur diesen Neuerungen nachgehen, sondern Ihnen auch verraten, was sie sich für .NET 9 gerne gewünscht hätten. Und weil das neue .NET auch in einigen der anderen Beiträge eine Rolle spielt, lege ich Ihnen diese Ausgabe besonders ans Herz.

Ich wünsche Ihnen eine schöne Zeit mit der neuen dotnetpro!

Herzlichst,

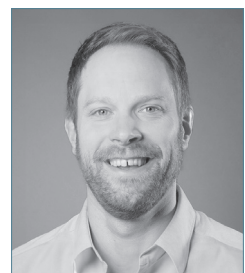
Fernando Schneider
Chefredakteur dotnetpro

P.S.: Noch mehr Wissen zum neuen .NET bietet Ihnen die .NET Developer Conference Ende November in Köln. Ich würde mich freuen, Sie dort zu begrüßen!



Ralf Westphal

präsentiert ein Architekturmuster für bessere Testbarkeit und damit bessere Erfüllung der Anforderungen (S. 46)



Sebastian Meyer

verrät, wie Sie mithilfe von Messaging Cloud-Apps mit On-Premises-Anwendungen verbinden (S. 115)



INHALT

Schwerpunkt

.NET 9 in den finalen Zügen

Neuerungen in .NET 9

Ausbesserungsarbeiten im Web

Neues in ASP.NET Core 9

Politur

Neuigkeiten in Blazor 9.0

Endlich Land in Sicht dank MAUI 9

Ein Blick auf .NET MAUI 9

Evolution statt Revolution

Neuerungen in C# 13

Planung

Time-out: Einfach mal lossegeln

Neue Chancen im Unerwarteten

Den Kopf vom Körper trennen

Sleepy-Hollow-Architektur

Dauids Deep Dive: Eine Frage der Führung

Wissen an der richtigen Stelle

Frontend

MAUI-Tools

Steuerelemente der Community

Core

NETirol: C#-Locking

Multithreading

LINQ für Events

Reaktiv programmieren, Teil 2

Grundlagen der Kryptografie

Verschlüsselungskonzepte erklärt

Gelungene Integration?

Messaging mit Wolverine und .NET, Teil 5

Frameworks: Die Kunst, Abfragen zu erzeugen

SqlKata/QueryBuilder

6

10

17

28

33

Foto: Shutterstock / Mykola Mazuruk



Ist denn heute schon Weihnachten? Was Entwickler von der neuen .NET-Version erwarten dürfen.

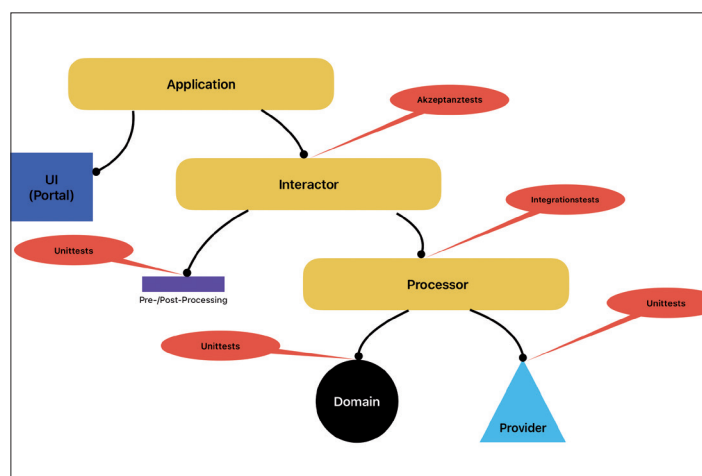
6

44

46

58

60



Kopf und Körper: Das Architekturmuster Sleepy Hollow verspricht bessere Testbarkeit und saubereren Code.

46

68

71

78

84

88

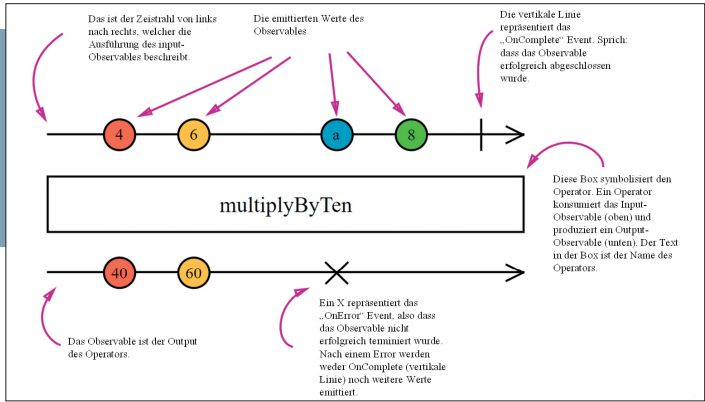
Experten in dieser Ausgabe



Christian Giesswein hebt die Bedeutung des Locking im Multithreading hervor und verrät, was sich mit C# 13 ändert. 68



Christian Havel vermittelt wichtige Begriffe, Konzepte und Zusammenhänge der Kryptografie. 78



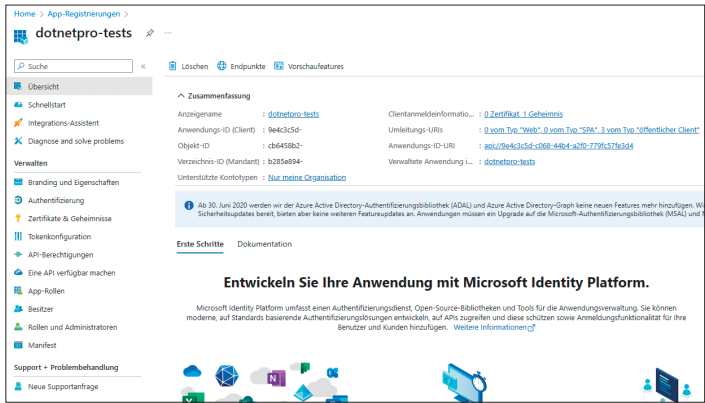
LINQ für Events: Mithilfe von Operatoren lassen sich in Rx.NET effiziente Datenverarbeitungsketten gestalten.

71



Die Kunst, Abfragen zu erzeugen: Wie Sie mit der .NET-Bibliothek QueryBuilder den Prozess für die Erstellung von SQL-Abfragen vereinfachen – und dadurch Fehler vermeiden.

88



Verbunden: Integrieren Sie MS Teams mit dem Microsoft Graph API passgenau in Ihre eigenen Anwendungen.

132

KI

Vom Hype zum Mehrwert

KI und Data

94

Was siehst du auf dem Bild?

GenAI – mit OpenAI und dem Azure-API neue Inhalte generieren

99

Backend

Data Access: AOT noch in Arbeit

Neuerungen in Entity Framework Core 9.0

108

Hybride Integration mit Azure Service Bus

Die Verbindung zwischen Cloud- und On-Premises-Anwendungen

115

Tools

Das Web nativ für die Hosentasche

Wisej.NET Hybrid

126

Workout

Teams als Kommunikationsservice

Microsoft Teams in eigene Anwendungen integrieren

132

Editorial

3

Lernstoff

124

Arbeitsmarkt

140

Impressum/Inserentenverzeichnis

144

Anbieterverzeichnis

145

Vorschau

146

Quellcode gefälltig?

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.



Foto: Shutterstock / Mykola Mazuruk

NEUERUNGEN IN .NET 9

.NET 9 in den finalen Zügen

Wie jedes Jahr präsentiert Microsoft auch im November 2024 auf der .NET Conf, was es Neues in der .NET-Welt geben wird. Ein Vorab-Blick auf .NET 9.

Stand heute (Ende August) wurde gerade die letzte Vorschauversion von .NET 9 vorgestellt, und im September und Oktober werden noch die finalen Release Candidates mit den letzten Qualitätsupdates veröffentlicht. Doch was kommt nun mit .NET 9 tatsächlich auf uns zu?

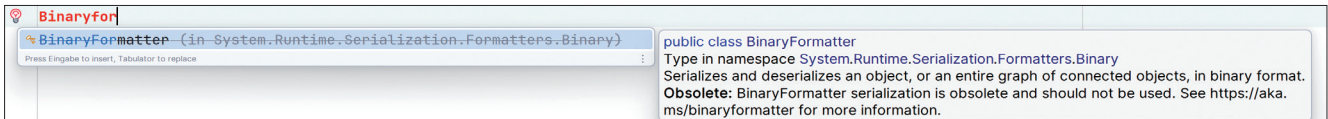
Die Kommunikation mit Microsoft hat sich geändert

Früher – vor .NET 9 – war das .NET-Blog die erste Anlaufstelle für alle Änderungen, die in die neue Version aufgenommen werden. Mit .NET 9 hat Microsoft verstärkt auf GitHub Discussions gesetzt und dort den Verlauf der Änderungen von

.NET 9 dokumentiert. Ob nun die Runtime, die Library oder auch EF Core 9 oder ASP.NET Core 9: Man wird dort mit all den Issues konfrontiert, die gelöst worden sind [1]. Auch auf der gewöhnlichen Learn-Plattform fasst Microsoft knapp und einfach die wichtigsten Punkte zusammen [2].

Der BinaryFormatter ist tot

Wie bereits seit Jahren geplant und angekündigt, ist es mit .NET 9 so weit – der BinaryFormatter ist endgültig nicht mehr vorhanden. Bei der Verwendung ist der Datentyp selbst komplett als „obsolete“ markiert (Bild 1). Dies war auch schon früher der Fall; nun ist es aber so, dass der BinaryFormatter nicht



Obsoleter BinaryFormatter (Bild 1)

nur eine Warnung ausgibt, sondern das Laufzeitverhalten ein gänzlich anderes ist.

Der BinaryFormatter ist im Typsystem noch komplett erhalten geblieben, und es scheint somit auf den ersten Blick, als sei er noch verwendbar. Nachdem man mithilfe eines Pragmas die Warnung `SYSLIB0011` deaktiviert hat, kann man den Compiler auch dazu zwingen, den `BinaryFormatter`-Datentyp zu verwenden. Wird nun aber auf eine Methode dieses Datentyps zugegriffen, resultiert dies zur Laufzeit nur noch in einer `PlatformNotSupportedException` (Bild 2).

Wer also noch immer auf den BinaryFormatter setzt, sollte sich nun langsam, aber sicher nach Alternativen umschauen. Grund für diese Änderung waren unter anderem Sicherheitslücken in der Art der Serialisierung (siehe dazu meinen Artikel

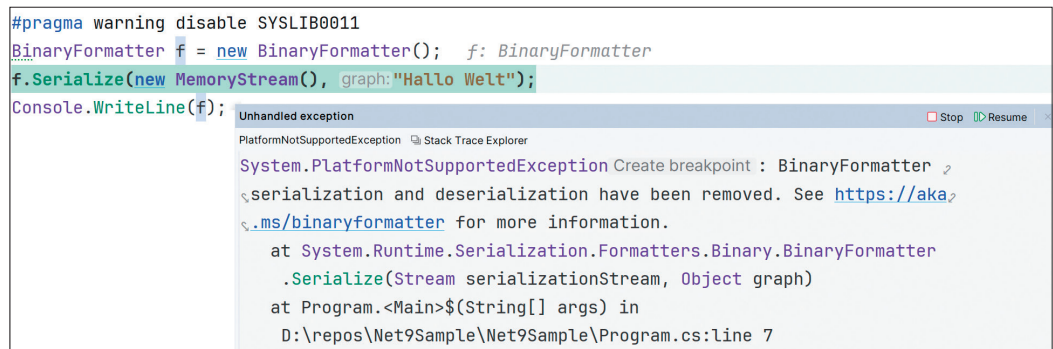
aus der dotnetpro-Ausgabe 12/2023 unter [3], aber auch [4]). Theoretisch gibt es allerdings noch eine Hintertür, die Microsoft liefert, und zwar kann der BinaryFormatter über ein spezielles NuGet-Paket [5] wieder nachgerüstet werden – damit wird die „alte“ Implementierung wieder aktiviert, und selbst unter .NET 9 können wir damit immer noch den BinaryFormatter nutzen.

Dies ist eine Änderung, die Microsoft langjährig verfolgt hat und die nun endlich am Ende angekommen ist. Fraglich ist an dieser Stelle, ob man den Datentyp mit .NET 10 dann endlich wirklich entfernt, oder ob uns die `PlatformNotSupportedException`

portedException noch die nächsten Jahre oder Jahrzehnte begleiten wird.

Die JSON-Standardweboptionen

Neu mit .NET 9 ist beim JsonSerializer eine Erweiterung um die `JsonSerializerOptions.Web`-Eigenschaft (Bild 3). Diese Eigenschaft kann beim Serialisieren sowie Deserialisieren genutzt werden. Diese Einstellungen sind deckungsgleich mit



BinaryFormatter ist nun Geschichte (Bild 2)

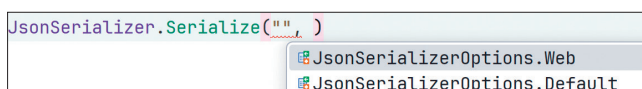
den Einstellungen, die Microsoft bei ASP.NET Core in der Vergangenheit bereits genutzt hat, und sind somit vor allem für Bibliotheken relevant, die nicht im ASP.NET-Core-Umfeld liegen – Bild 4 zeigt dabei die unterschiedlichen Werte, die man üblicherweise manuell setzen musste, doch mit .NET 9 eben komfortabel über eine zentrale Eigenschaft lösen kann.

LINQ hat neue Operationen

Unser geliebtes LINQ hat wieder neue Operationen verpasst bekommen und damit wieder neue Funktionen für uns freigegeben, die wir in speziellen Fällen nutzen können. Konkret geht es dabei um die Methoden `CountBy` und `AggregateBy`. Am Ende sind diese beiden Methoden nur Abkürzungen zu der bisherigen `GroupBy`-Variante und kürzen somit hier die Schreibweisen etwas ab.

Dabei ist der große Unterschied, dass man eben eine Menge erhält, bei der einerseits der „Gruppierungsschlüssel“ und andererseits der Wert direkt zurückgeliefert werden. Dies war mittels eines `GroupBy` gefolgt von einem `Select` auch vorher schon technisch möglich, ist damit nun aber etwas leichter und kürzer geworden (Bild 5).

Eine ebenfalls etwas kurios anmutende Umsetzung ist die neue Methode `Index`. Schon immer war es ein Problem von LINQ, dass die Indexermittlung in der Vergangenheit recht eigenartig war. Entweder man merkte sich einen separaten `integer` und erhöhte diesen bei jedem Schleifendurchlauf, ►



JSON-Standardweboptionen (Bild 3)

Webstandardwerte für JsonSerializerOptions

Die folgenden Optionen weisen unterschiedliche Standardwerte für Web-Apps auf:

- `PropertyNameCaseInsensitive` = `true`
- `PropertyNamingPolicy` = `CamelCase`
- `NumberHandling` = `AllowReadingFromString`

Die unterschiedlichen Werte im Vergleich zu „Default“ (Bild 4)

oder man wechselte auf eine gute, altbewährte *for*-Schleife (Bild 6).

Neuartig ist nun die Methode *Index*, die ein *ValueTuple* mit dem Eintrag und ebenfalls den Index des Eintrags zurückgibt. So klein diese Änderung auch sein mag – ab und an bietet sie einen gewissen Komfort.

```
int[] numbers = [1,2,3,4,5,6,7,8,9];
foreach (var countGroup:KeyValuePair<bool,int> in numbers.CountBy(x:int => x % 2 == 0))
{
    Console.WriteLine($"{countGroup.Key} {countGroup.Value}");
}
```

Ausgabe der Ergebnismenge mit Anzahl (Bild 5)

Persistenter AssemblyBuilder

Eine letzte „Kleinigkeit“ ging dem .NET-Core-Ökosystem noch ab, insbesondere wenn man vom .NET Framework migrierte oder dies noch vor sich hat – persistente Assembly-Builder (Bild 7).

Dabei kann man mittels .NET und C# bereits seit den Anfängen des .NET Frameworks zur Laufzeit IL-Code erzeugen und diesen dynamisch laden. Insbesondere dynamische Frameworks wie Entity Framework Core oder auch Castle Windsor setzen recht stark auf solche Features, um performanten Code zu ermöglichen.

Was bisher in .NET nicht möglich war, ist, dass dieser dynamische Code abgespeichert und zum Beispiel auf die Festplatte geschrieben wird. Dies wurde nun mit .NET 9 behoben, und man erhält Zugriff auf die neue Methode *AssemblyBuild-*

```
foreach ((int Index, int Item) entry in numbers.Index())
{
    // 0 - 1
    // 1 - 2
    // 2 - 3
    // ...
    Console.WriteLine($"{entry.Index} {entry.Item}");
}
```

Indexermittlung mit foreach – simpel! (Bild 6)

der.DefinePersistedAssembly, mit deren Hilfe man die Assembly persistieren kann, um sie später oder auch ein anderes Mal zu laden.

```
C# Kopieren

public void CreateAndSaveAssembly(string assemblyPath)
{
    AssemblyBuilder ab = AssemblyBuilder.DefinePersistedAssembly(
        new AssemblyName("MyAssembly"),
        typeof(object).Assembly
    );
    TypeBuilder tb = ab.DefineDynamicModule("MyModule")
        .DefineType("MyType", TypeAttributes.Public | TypeAttributes.Class);

    MethodBuilder mb = tb.DefineMethod(
        "SumMethod",
        MethodAttributes.Public | MethodAttributes.Static,
        typeof(int), [typeof(int), typeof(int)]
    );
    ILGenerator il = mb.GetILGenerator();
    il.Emit(OpCodes.Ldarg_0);
    il.Emit(OpCodes.Ldarg_1);
    il.Emit(OpCodes.Add);
    il.Emit(OpCodes.Ret);

    tb.CreateType();
    ab.Save(assemblyPath); // or could save to a Stream
}

public void UseAssembly(string assemblyPath)
{
    Assembly assembly = Assembly.LoadFrom(assemblyPath);
    Type type = assembly.GetType("MyType");
    MethodInfo method = type.GetMethod("SumMethod");
    Console.WriteLine(method.Invoke(null, [5, 10]));
}
```

Dynamische Assemblies zur Laufzeit bauen (Bild 7)

Performance

Beim Release von .NET 9 wird Microsoft wieder alles auffahren, was die Marketingabteilung so hergibt. Insbesondere wird mit Sicherheit wieder erwähnt werden, dass .NET 9 das schnellste .NET ist und um soundsoviel Prozent besser performt als .NET 8. Natürlich stimmt daran einiges, gerade beim Blick auf die Neuerungen in der Runtime selbst. Mit JIT-Optimierungen, Schleifenoptimierungen und GC-Optimierungen wird .NET 9 (je nach Situation) bestimmt schneller sein [6]. Doch eines soll an dieser Stelle herausgehoben werden: Microsoft hat Exceptions beschleunigt.

Mit .NET 9 hat Microsoft den Weg, wie Microsoft mit Exceptions umgeht, grundlegend geändert. War es bisher so, dass .NET noch die Unterstützung des „Windows structured exception handling“ einschloss, ist diese nun entfernt worden. Daraus ergibt sich, dass die Performance beim Exception-Handling mit .NET 9 ungefähr um den Faktor zwei bis vier verbessert werden konnte – ein beeindruckender Wert.

Neue Methoden bei „TimeSpan“

Eine weitere „Kleinigkeit“ hat Microsoft nun geradegezogen: Mithilfe der Factory-Methoden können auf dem Datentyp *TimeSpan* beliebige Methoden genutzt werden, um einen *TimeSpan* zu erzeugen. Bis dato war dies nur mit einem *double* möglich.

Neu ist nun mit .NET 9, dass auch mit einem *int* oder *long* gearbeitet werden kann. Hintergrund für diese Änderungen war, dass es zu Rundungsfehlern kommen konnte, wenn ein *int* in einen *double* abgeändert wird. Deswegen hat Microsoft den Datentyp dahingehend angepasst, dass er eben auch Ganzzahlen entgegennehmen kann (Bild 8 und Bild 9).

.NET Standard wird immer „unwichtiger“

Ebenfalls mit .NET 9 kommt der eine oder andere „Breaking Change“ [7]. Zu diesen zählt zum Beispiel, dass .NET Standard 1.x nicht mehr als „aktuell“ gilt.

Kompiliert man eine Bibliothek nun gegen .NET Standard 1.x mit dem .NET-9-SDK, so wird eine Compilerwarnung ausgegeben: „Warning NETSDK1215: Targeting .NET Standard prior to 2.0 is no longer recommended.“ Dies deutet auch immer mehr in die Richtung, dass mittlerweile der eigentliche Grund für .NET Standard – die Migration von .NET Framework zu .NET – aus Sicht von Microsoft langsam, aber sicher als abgeschlossen gilt.

Fazit

.NET 9 – eine bemerkenswerte Version? Wohl kaum. Gerade im Hinblick auf die Features, die in .NET enthalten sind, als auch jene in ASP.NET Core (siehe in dieser dotnetpro-Ausgabe ab Seite 10 [8]) oder EF Core (siehe ab Seite 108 [9]) sind sicherlich einige nützliche „Kleinigkeiten“ dabei, aber die großen Features und echten Innovationen sind etwas hinter

den (persönlichen) Erwartungen zurückgeblieben. Vor allem wurden Features erneut verschoben oder auch versprochen und dann verschoben – sehr zum Bedauern der Community.

Am Ende bleibt das jedoch Meckern auf hohem Niveau: Schließlich gibt es bei Microsoft schon lange das Verspre-

```
using System;
```

```
TimeSpan.FromSeconds()
```

(double value) : TimeSpan

Returns a TimeSpan that represents a specified number of seconds, where the specification is accurate to the nearest millisecond.

value: A number of seconds, accurate to the nearest millisecond.

TimeSpan-Methoden bei .NET 8 (Bild 8)

```
using System;
```

```
TimeSpan.FromSeconds()
```

(double value) : TimeSpan

✓ (long seconds) : TimeSpan

(long seconds, long milliseconds = 0L, long microseconds = 0L) : TimeSpan

Im Vergleich dazu – mit .NET 9 (Bild 9)

chen, uns in der agilen Form ein Framework zu liefern, auf das Verlass ist. Und das hält Microsoft seither auch ein. ■

- [1] GitHub Discussions, .NET 9 Release Index #9234, www.dotnetpro.de/SL2411NET9_1
- [2] Microsoft Learn, Neuerungen in .NET 9, www.dotnetpro.de/SL2411NET9_2
- [3] Christian Giesswein, [Serializable] ist tot, dotnetpro 12/2023, Seite 60 f., www.dotnetpro.de/A2312NETirol
- [4] Common Weakness Enumeration #502: Deserialization of Untrusted Data, www.dotnetpro.de/SL2411NET9_3
- [5] NuGet-Paket System.Runtime.Serialization.Formatters, www.dotnetpro.de/SL2411NET9_4
- [6] Microsoft Learn, What's new in the .NET 9 runtime, www.dotnetpro.de/SL2411NET9_5
- [7] Microsoft Learn, Breaking Changes in .NET 9, www.dotnetpro.de/SL2411NET9_6
- [8] Christian Wenz, Ausbesserungsarbeiten im Web, dotnetpro 11/2024, Seite 10 ff., www.dotnetpro.de/A2411ASPNETCore
- [9] Holger Schwichtenberg, AOT noch in Arbeit, dotnetpro 11/2024, Seite 108 ff., www.dotnetpro.de/A2411DataAccess



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

A2411NET9

NEUES IN ASP.NET CORE 9

Ausbesserungsarbeiten im Web

Was ändert sich bei der Webentwicklung mit Microsoft?

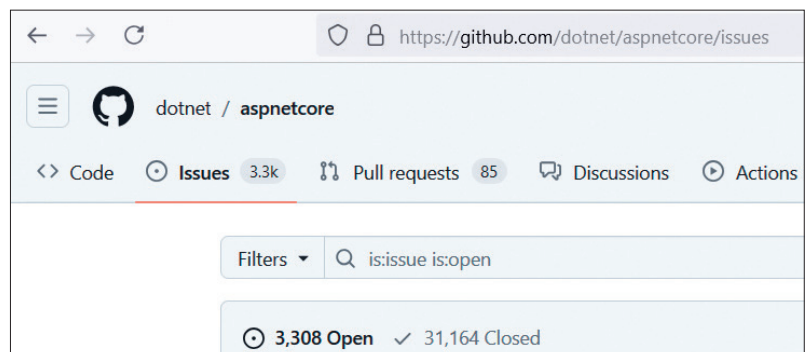
Wer erfolgsverwöhnt ist, muss mit Enttäuschungen rechnen. Das lässt sich in gewisser Hinsicht bei .NET beobachten, insbesondere bei der Webentwicklung. Galten früher neue .NET-Versionen als sehnlich erwartetes Highlight, ist heute die Begeisterung ein wenig abgeklungen. Salopp formuliert: Was soll denn noch Neues kommen? Der Web-Stack von Microsoft ist – vielleicht mit Ausnahme des jüngsten Mitglieds, Blazor, das in dieser dotnetpro-Ausgabe ab Seite 17 gesondert behandelt wird [1] – etabliert und mehr oder weniger „feature-complete“. Das zeigt sich an vielen Stellen: Der MVC-Stack wurde größtenteils noch aus dem „alten“ .NET Framework übernommen und dabei nur kosmetisch angepasst, wir reden hier also von einer erstmals 2009 eingeführten Technologie. Razor Pages sind neuer, genauer gesagt aus ASP.NET Core 2, aber das war ebenfalls bereits 2017, und die zentrale Komponente, die Razor-Syntax, stammt aus dem Jahr 2010. Bleibt noch ASP.NET Core Web API. Einst gewissermaßen ein Nebenprodukt von MVC, gab es mit Minimal APIs tatsächlich eine bemerkenswerte Neuerung, aber eher in Richtung Syntax-Zucker. Neuerungen in früheren .NET-Updates schlossen dagegen primär die Feature-Lücke zu Controller-basierten Web-APIs.

Verstehen Sie das bitte nicht falsch, die einleitenden Worte sollen keineswegs zynisch klingen. ASP.NET Core ist einfach in einem so guten Zustand, dass Neuerungen naturgemäß eher kosmetisch ausfallen. Großflächige, nicht abwärtskompatible Änderungen sind von Microsoft weder kapazitätsmäßig zu stemmen noch anhand der zum Redaktionsschluss 3300 offenen GitHub-Issues [2] (Bild 1) sinnvoll priorisierbar. Davon abgesehen würde die Kundschaft sicherlich Sturm laufen, und das mit Recht. Und so fällt das Gros der Änderungen der letzten .NET-Updates meist in eine der folgenden beiden Kategorien:

- **Viele kleinere Neuerungen**, die sinnvolle, aber nicht entscheidende Ergänzungen bieten. Nicht selten findet sehr schnell eine Gewöhnung an die neuen Möglichkeiten statt, sodass es kaum vorstellbar ist, jemals ohne sie gearbeitet zu haben. Dennoch sind es meist Kleinigkeiten. Manche eher signifikante neue Features werden gerne separat entwickelt; ein schönes Beispiel dafür sind die in der vorangegangenen dotnetpro-Ausgabe gezeigten Smart Components [3],

die in einem separaten GitHub-Repository – außerhalb der .NET-Organisation! – verfügbar gemacht wurden, zumindest bis jetzt.

- **Verbesserungen unter der Haube**, die die Performance erhöhen: schnellere Ausführung, weniger Speicherverbrauch und dergleichen. Gerne wird das mit Statistiken untermauert. Es ist ein zweischneidiges Schwert, sich damit zu brüsten, denn ein „200 Prozent schneller“ (die Zahl ist willkürlich und hat nichts mit ASP.NET Core 9 zu tun) bedeutet ja im Umkehrschluss, dass die Geschwindigkeit in der Vorgängerversion nicht gut war.



Sisyphosarbeit: Ganz schön viele offene Issues bei ASP.NET Core (Bild 1)

In vielen Fällen bedeutet eine neue .NET-Version also rein aus Feature-Sicht: Ein Update kann gemacht werden, muss aber nicht unbedingt erfolgen. Bleibt noch der Support-Aspekt: Geradzahlige .NET-Versionen erhalten drei Jahre Support, oder genauer gesagt: drei Jahre nach Veröffentlichung, und dann bis zum nächsten Patch-Dienstag, auch wenn es derselbe Tag ist. Der zweite Dienstag im Monat ist der Tag, an dem es Updates gibt; es ist also kein Zufall, dass .NET auch in den letzten Jahren immer in der zweiten Woche des Novembers erschienen ist, an einem Dienstag. Bei ungeraden .NET-Versionen läuft es ähnlich, aber der Support endet bereits nach anderthalb Jahren. Für .NET 9 bedeutet das also: Ungerade Versionsnummer, es gibt Support bis Mai 2026. .NET 8 wiederum wird als gerade Version drei Jahre unterstützt, das bedeutet bis November 2026 und damit länger.

Das führt zu einer interessanten Situation: Wenn eine neue .NET-Version kein Feature mit sich bringt, das für ein Projekt

von großem Vorteil oder gar vonnöten ist, sind viele Firmen zögerlich beim Update. In der Praxis kann das insbesondere bei komplexen Softwareprodukten festgestellt werden (Stichwort externe Abhängigkeiten). Der Support-Zeitraum für .NET ist in diesen Fällen das maßgebliche Kriterium. In der Praxis heißt das oft: Alle zwei Jahre wird zur nächsten – geradzahligen – Long-Term-Support-Version migriert, die ungeradzahligen Updates werden übersprungen.

Somit ist für ASP.NET Core 9 ein genauer Blick auf das notwendig, was sich geändert hat. Lohnt sich das Update, oder kann es sinnvoller sein, bis .NET 10 zu warten, das voraussichtlich im November 2025 erscheint und dann bis November 2028 Support genießen würde? Diese Frage lässt sich natürlich nicht allgemein beantworten, zu verschieden sind Anforderungen, Projekte und Kapazität im Entwicklungsteam.

Dieser Artikel soll aber eine Entscheidungshilfe liefern. Ausgehend von der langen Liste der – größtenteils unerheblichen – Neuerungen in ASP.NET Core 9 [4] stellen wir die subjektiv wichtigsten Änderungen vor. Wenn Ihnen gefällt, was Sie sehen, ziehen Sie ein Update in Erwägung. Wenn nicht, sollten Sie natürlich trotzdem eine Migration zur neuen Version prüfen, denn nicht sicherheitsrelevante Verbesserungen sind eher in .NET 9 zu erwarten und nicht im noch unterstützten .NET 8. Andernfalls kann aber auch das Abwarten eine sinnvolle Strategie sein. Nach dem Release von .NET 9 geht es direkt mit den Arbeiten an .NET 10 weiter. Noch ein kurzer Disclaimer zu Beginn: Grundlage dieses Artikels war der Stand von ASP.NET Core 9 zur Preview-Version 7 Mitte August 2024.

SignalR und AOT

Ein Schwerpunkt des neuen Release ist die Arbeit an der Unterstützung für natives AOT für .NET-Inhalte. Bereits mit .NET 7 wurde die Ahead-of-Time-Kompilierung als Option eingeführt, dedizierte AOT-Vorlagen als Teil von .NET inklusive. ASP.NET Core 8 hat für viele Anwendungsfälle die AOT-Unterstützung nachgerüstet, die Kompatibilitätstabelle unter [5] füllt sich sukzessive.

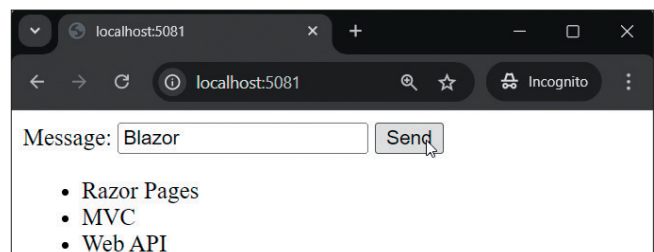
Und auch wenn seither Konsolenanwendungen, Web-APIs und gRPC-Dienste im AOT-Modus laufen können, hatte das in der Praxis zunächst wenig Auswirkungen. Hier lohnt sich ein kurzer Blick auf die Hintergründe. Mit AOT wird eine Anwendung für ein bestimmtes Zielsystem in Form eines Standalone-Binaries kompiliert. Das Ergebnis läuft dann nicht mehr als Cross-Platform-Anwendung, aber eben auf einer bestimmten Zielarchitektur, und das gut. Microsoft selbst verspricht beim Einsatz von AOT eine schnellere Start-up-Zeit und einen geringeren Speicherfußabdruck [5]. In der Praxis lässt sich das oft, aber nicht immer bestätigen. Eine viel größere Einschränkung ist jedoch, was sich mit AOT ausdrücklich nicht machen lässt: Das dynamische Nachladen von Assemblies und bestimmte Aspekte von Reflection beispielsweise widerstreben dem Ansinnen einer Vorab-Kompilierung. Durch einige Kunstgriffe lassen sich Auswege schaffen, etwa durch die Quellcodegenerierung des JSON-Serialisierers (*System.Text.Json*). Dies ist bereits bei den klassischen Projektvorlagen zu erkennen, hier ein typisches Muster:

```
[JsonSerializable(typeof(string))]
internal partial class AppJsonSerializerContext :
    JsonSerializerContext { }
```

In .NET 9 wurde weiter bei AOT nachgerüstet: SignalR unterstützt jetzt diesen Ausführungsmodus. JSON kommt als Protokoll beim SignalR-Hub zum Einsatz, auf Basis der obenstehenden Klasse:

```
builder.Services.Configure<JsonHubProtocolOptions>(o =>
{
    o.PayloadSerializerOptions.TypeInfoResolverChain
        .Insert(0, AppJsonSerializerContext.Default);
});
```

Eine möglich minimale Hallo-Welt-Anwendung lässt sich auf Basis der Web-API-Projektvorlage (mit „AOT“ im Namen!)



Echokammer: Vorne SignalR-Anwendung ... (Bild 2)

implementieren. Das Template weist nämlich unter anderem zwei Besonderheiten auf, die Sie sonst bei einer bestehenden Nicht-AOT-Anwendung nachrüsten müssten:

- Durch den Einsatz von *WebApplication.CreateSlimBuilder(args)* wird eine „schlanke“ Form der Anwendung erstellt, bei der unter anderem alles hinausfliegt, was mit AOT nicht möglich ist.
- In der Projektdatei sorgt *<PublishAot>true</PublishAot>* dafür, dass auch tatsächlich eine Veröffentlichung als AOT stattfinden kann.

Zur Implementierung: Zunächst definieren wir einen SignalR-Hub:

```
builder.Services.AddSignalR();

// ...

var app = builder.Build();

app.MapHub<EchoHub>("/echo");
```

Der Hub selbst arbeitet mit den Events *Send* und *Receive* und leitet Nachrichten direkt weiter:

```
public class EchoHub : Hub
{
```



```
public async Task Send(string message)
{
    await Clients.All.SendAsync("Receive", message);
}
}
```

Fehlt nur noch das Markup, welches das UI für die Anbindung an SignalR darstellt. Das User Interface besteht aus einem simplen Formular mit Textfeld sowie einer `<ul>`-Liste zur Ausgabe von Werten:

```
<form>
    Message: <input name="message">
```

```
form.elements['message'].value = '';
}
```

... wohingegen eingehende Nachrichten direkt in die Oberfläche eingefügt werden:

```
const messages = document.getElementById('messages');

connection.on('Receive', (message) => {
    const li = document.createElement('li');
    li.textContent = message;
    messages.appendChild(li);
});
```

```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22631.4037]
(c) Microsoft Corporation. All rights reserved.

C:\Users\cwenz\source\repos\AspNetCore9Demos\SignalRAOT>dotnet publish
Restore complete (2,2s)
You are using a preview version of .NET. See: https://aka.ms/dotnet-support-policy
SignalRAOT succeeded (15,3s) -> bin\Release\net9.0\win-x64\publish\

Build succeeded in 18,2s

C:\Users\cwenz\source\repos\AspNetCore9Demos\SignalRAOT>dir bin\Release\net9.0\win-x64\publish
Volume in drive C is OS
Volume Serial Number is 4678-D72F

Directory of C:\Users\cwenz\source\repos\AspNetCore9Demos\SignalRAOT\bin\Release\net9.0\win-x64\publish

26.08.2024  15:35    <DIR>          .
23.08.2024  19:26    <DIR>          ..
23.08.2024  01:18                127 appsettings.Development.json
23.08.2024  01:18                151 appsettings.json
23.08.2024  19:26           10.465.792 SignalRAOT.exe
23.08.2024  19:26           54.349.824 SignalRAOT.pdb
23.08.2024  19:26                40 SignalRAOT.staticwebassets.endpoints.json
                5 File(s)          64.815.934 bytes
                2 Dir(s)      401.673.281.536 bytes free

C:\Users\cwenz\source\repos\AspNetCore9Demos\SignalRAOT>
```

Einzelkämpfer: ... hinten AOT (Bild 3)

```
<input type="button" value="Send" onclick="send(
    this.form)">
</form>
```

```
<ul id="messages"></ul>
```

Der JavaScript-Code baut zunächst eine Verbindung zum Hub auf:

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl('/echo')
    .build();

connection.start();
```

Beim Formularversand wird die Eingabe verschickt ...

```
async function send(form) {
    await connection.invoke(
        'Send', form.elements['message'].value);
```

Bild 2 zeigt das UI. Diese Anwendung können wir mit *dotnet publish* als Stand-alone-AOT-Binary veröffentlichen (in **Bild 3** sehen Sie den Prozess und das Ergebnis). Das läuft dann auch tatsächlich nur auf der gewählten Architektur, aber potenziell gut. Bevor Sie begeistert in die Tasten greifen: Prüfen Sie zunächst, ob Ihr Code tatsächlich AOT-kompatibel ist, und falls ja, ob Sie tatsächlich die versprochenen Performance-Vorzüge feststellen können.

Rund um SignalR weist ASP.NET Core 9 noch weitere Neuerungen auf. Die Aktivitätsquelle *Microsoft.AspNetCore.SignalR.Server* löst Ereignisse im Hub aus; zudem können dort jetzt – durch den Einsatz von Basis-Klassen – auch polymorphe Typen verwendet werden.

OpenAPI

Im Lauf des Jahres 2024 hat sich Microsoft-Mitarbeiterin Safia Abdalla auf ihrem X-Account [6] immer häufiger zu den Themen API-Dokumentation, Swagger und OpenAPI geäußert. Besonders letzteres Thema hat es ihr angetan; dabei handelt es sich um eine anerkannte Spezifikation zur Doku-

mentation von HTTP-APIs [7]. Inzwischen ist klar, worauf sie hinauswollte: Bei ASP.NET Core 9 ist der OpenAPI-Support deutlich weiter ausgebaut. Auch in den früheren Versionen war OpenAPI mit dabei, nur der Name nicht immer prominent platziert. Wer jemals mit der Swagger-Oberfläche gearbeitet hat, die bei der Web-API-Projektvorlage automatisch mitgeneriert wird, hat auf OpenAPI gesetzt. Die Checkbox *Enable OpenAPI support* in Visual Studio hat das Projekt entsprechend eingerichtet; bei Verwendung des .NET CLI war gar ein `--no-openapi` notwendig, um die OpenAPI-Unterstützung zu entfernen.

In ASP.NET Core 9 wurden weitere Features ermöglicht und das NuGet-Paket `Microsoft.AspNetCore.OpenApi` entsprechend erweitert. Der wohl nützlichste Neuzugang ist die Generierung einer OpenAPI-Dokumentation zu einem implementierten API. Dabei ist es egal, ob Minimal APIs zum Einsatz kommen oder auf Controller gesetzt wird. Zwei Aufrufe werden in der Datei *Program.cs* erwartet:

- `builder.Services.AddOpenApi()` lädt die Middleware,
- `app.MapOpenApi()` verwendet die Middleware und legt die entsprechenden Endpunkte an.

Über den Pfad `/openapi/v1.json` erhalten Sie jetzt automatisch eine OpenAPI-Repräsentation der Dienste. Bild 4 zeigt, wie das bei der Standard-Web-API-Vorlage aussieht, wenn AOT zum Einsatz kommen soll. Dort wurde nämlich ein einfaches To-do-API implementiert.

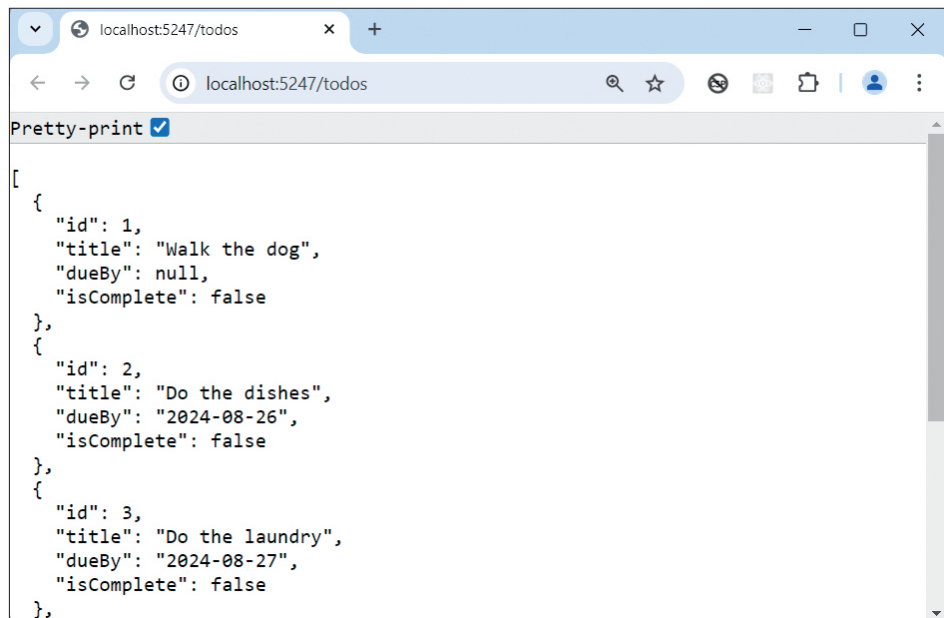
Natürlich ist es nicht sehr effizient, diese JSON-Datei jedes Mal neu zu generieren. Deswegen kann dies auch bereits schon im Build-Prozess geschehen. Dazu muss das Projekt entsprechend konfiguriert werden:

```
<OpenApiGenerateDocuments>true</OpenApiGenerateDocuments>
```

Bei der Generierung der JSON-Definition werden auch in beschränktem Maße Attribute der zugehörigen Modellklassen mitverarbeitet, weil es dafür Entsprechungen in der OpenAPI-Spezifikation gibt. Aus `[DefaultValues]` wird *default*, Modelleigenschaften mit `[Required]` werden in OpenAPI mit *required* markiert.

Für einen weiter reichenden Eingriff in die OpenAPI-Generierung kommen sogenannte Schema-Transformer ins Spiel. Diese definieren eine Blaupause für OpenAPI (Klasse: `OpenApiObject`), die dann exemplarisch in die Datei `v1.json` generiert wird.

Hier ein kleines Beispiel für das To-do-API aus der Web-API-AOT-Vorlage. Microsoft selbst stellt ebenfalls exempla-



Innenansicht: Das OpenAPI-JSON zum To-do-API (Bild 4)

rischen Code zur Verfügung, dieser war allerdings zum Redaktionsschluss so nicht lauffähig, unsere Version hingegen schon:

```
builder.Services.AddOpenApi(options =>
{
    options.AddSchemaTransformer((s
        chema, context, cancellationTokens) =>
    {
        schema.Example = new OpenApiObject()
        {
            ["id"] = new OpenApiInteger(42),
            ["title"] = new OpenApiString(
                "Bezeichnung des Todos"),
            ["dueBy"] = new OpenApiDate(DateTime.UtcNow),
            ["isComplete"] = new OpenApiBoolean(false)
        };

        return Task.CompletedTask;
    });
});
```

Das generierte OpenAPI-JSON enthält jetzt wie angegeben ein Beispielement mit den Angaben aus unserem Code. Tools wie Swagger könnten damit beispielsweise das UI befüllen, um ein Modellelement anzulegen.

```
"/todos/{id}": {
  "get": {
    "tags": [
      "OpenAPIAOT"
    ],
    "parameters": [
      {
        "name": "id",
```

```

    "in": "path",
    "required": true,
    "schema": {
      "type": "integer",
      "format": "int32",
      "example": {
        "id": 42,
        "title": "Bezeichnug des Todos",
        "dueBy": "2024-08-26",
        "isComplete": false
      }
    }
  },
  "responses": {
    "200": {
      "description": "OK"
    }
  }
}

```

Statische Assets

Auch in Zeiten von Breitband sind die Datenmenge und die Übertragungsgeschwindigkeit einer Website entscheidende Kriterien und sowohl für ein besseres Google-Ranking als auch für eine höhere Konversionsrate wichtig. Hat gerade jemand laut Blazor WehAssembly gesagt?

Die Techniken, um die Inhalte einer Website möglichst effizient zum Client zu bringen, sind seit Langem bekannt. Unter anderem können Inhalte komprimiert werden, insbesondere HTML und CSS (aber auch Grafiken ohne sichtbaren Qualitätsverlust, doch das ist ein anderes Thema). Zudem ist

Name	Header	Vorschau	Antwort	Initiator	Timing
jquery.min.js	Accept-Ranges: bytes				
bootstrap.bundle.min.js	Content-Length: 89501				
site.js?v=hRQyftXiu1ILX2P9L...	Content-Type: text/javascript				
11 Anforderungen 355 kB übertragen	Date: Fri, 23 Aug 2024 00:04:47 GMT				
	Etag: "1daf4ef15bf649d"				

Komprimiert: Erste zarte Caching-Ansätze vor ASP.NET Core 9 (Bild 5)

Eine wichtige Änderung an dieser Stelle: Die Swagger-Oberfläche zum direkten Testen des API im Browser wird nicht mehr templateseitig angelegt. Die Begründung des .NET-Teams: Es gebe so viele Alternativen (unter anderem auch den wenig bekannten Endpoints Explorer in Visual Studio), sie möchten nichts Spezifisches vorgeben.

Wenn Sie sich zu sehr an den URL-Endpunkt `/swagger` gewöhnt haben sollten: Das NuGet-Paket `Swashbuckle.AspNetCore.SwaggerUI` bringt Ihnen die alte Funktionalität zurück. Sie müssen allerdings in der Datei `Program.cs` eine entsprechende Konfiguration hinterlegen, damit die Komponente weiß, wo sich die OpenAPI-Definition des API befindet. Den URL haben wir oben schon gesehen; hier der entsprechende Code:

```

app.useSwaggerUI(options => {
    options.SwaggerEndpoint("/openapi/v1.json", "v1");
});

```

Doch kurz zurück zum Anfang der Beschreibung der neuen OpenAPI-Unterstützung. Als Beispiel wurde die AOT-Web-API-Vorlage gewählt, Stichwort To-do-API. Das war kein Versehen, ganz im Gegenteil: Auch die OpenAPI-Generierung ist ein unterstütztes Szenario für natives AOT. Auch hier erstellt `dotnet publish` eine Stand-alone-Anwendung für die gewählte – beziehungsweise, falls nicht angegeben, die aktuelle – Prozessor- und Betriebssystemarchitektur.

der Einsatz von Caching empfehlenswert, sodass der Browser beim nächsten Mal bestimmte Ressourcen gar nicht laden muss, sondern sie direkt aus dem Cache bezieht. Allerdings gilt es zu beachten, dass bei Änderungen an den Ressourcen möglichst nicht die zwischengespeicherte Kopie, sondern die aktualisierte Version vom Server verwendet wird.

Mit ASP.NET Core 9 hält eine neue Middleware Einzug, die einige der vorgenannten Techniken einsetzt:

```
app.MapStaticAssets()
```

Sie dient insbesondere als Ersatz für die sonst übliche Middleware für statische Inhalte, `app.UseStaticFiles()`. Dies allerdings nur dann, wenn die Anwendung alle statischen Ressourcen kennt (das schließt insbesondere eingebettete Dateien aus). Die Dateien werden dann zunächst komprimiert, denn Webbrowser unterstützen das automatische Dekomprieren bestimmter Inhalte schon seit Jahrzehnten. Das verringert zunächst die Datenmenge beim ersten Aufrufen der Anwendung. Dieses Packen von Dateien findet bereits beim Build-Prozess statt, verlangsamt also nur diesen Erstellsschritt, aber nicht die Anwendung selbst.

Für noch weniger Bytes beim wiederholten Besuch der Website wird das Caching durch den Browser erleichtert. Die Middleware fügt einen ETag-Header hinzu. Der Wert ist aber nicht zufällig, sondern ist im Wesentlichen der SHA256-Hash des Inhalts der Ressource. Ändert sich diese, ist der ETag-

Wert ein anderer und der Browser sollte mit der neuesten Version arbeiten. Gleichzeitig hängt ASP.NET Core 9 mit der neuen Middleware einen Fingerabdruck an den Dateinamen der Ressource an, der automatisch erzeugt wird. Ändert

Name	Header	Vorschau	Antwort	Initiator	Timing
jquery.min.dd6z7egasc.js	Content-Length: 89501				
bootstrap.bundle.min.25iw1...	Content-Type: text/javascript				
site.xtor3hu2r.js	Date: Fri, 23 Aug 2024 00:05:35 GMT				
11 Anforderungen 358 kB übertragen	Etag: "h++Eg5XgyMHxAmFglZYVUv52tqhBptqbK6YwUx2LNKY="				
	Etag: W"/xUj+3OUJ5vExlq6GSYGSHk7tPXikynS7oqEvDej/m4="				

Komprimierter: Mehr Caching bei ASP.NET Core 9 (Bild 6)

sich dann beispielsweise eine JavaScript-Datei, würde sich ein neuer Fingerabdruck ergeben und im HTML, das auf den JavaScript-Code verweist, der zugehörige Dateiname. So landet also kein veralteter Code im Browser. Gibt es keine Veränderung, kann der Browser dank Caching-Header mit der bereits vorhandenen Datei arbeiten.

Bild 5 zeigt einige der HTTP-Requests bei Verwendung von *UseStaticFiles()*, **Bild 6** die entsprechenden Aufrufe bei Verwendung von *MapStaticAssets()*. Bei der JavaScript-Bibliothek (jQuery) hängt jetzt ein Fingerabdruck am Dateinamen. Die selbst erstellte JavaScript-Datei (*site.js*) trug schon vor der neuen Middleware eine Art von Fingerabdruck im URL; das lag am Einbau mit Tag-Helper *asp-append-version*:

```
<script src="~/js/site.js" asp-append-version="true"></script>
```

Mit *MapStaticAssets()* ist der eindeutige Code nun Teil des Dateinamens.

Etwas irritierend ist noch die Verwendung von zwei ETags (siehe **Bild 6**). Der kurze entspricht dem, der bereits in alten Versionen von ASP.NET Core zum Einsatz kam; der neue ist wie bereits erwähnt der Hash des Inhalts und damit diejenige Variante, bei der eine Kollision (gleicher Hash) zweier unterschiedlicher Codeversionen quasi ausgeschlossen ist. Möglich, dass sich dies in der finalen Version noch ändert. Der Hauptvorteil des Features liegt darin, dass die Performance-Optimierungen automatisch durch das Framework erledigt werden und so nicht auf dem Webserver konfiguriert werden müssen. Gerade dieser Automatismus kann aber natürlich theoretisch zu Seiteneffekten führen, eben etwa geänderten Dateinamen.

Hybrider Cache

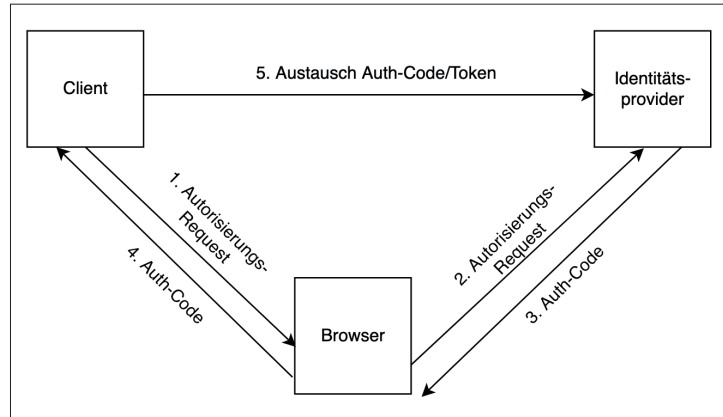
Apropos Caching: Für das Zwischenspeichern im Client haben wir mit *MapStaticAssets()* eine Lösung. Aber auch auf dem Server gibt es hier in ASP.NET Core 9 Neues. Klar, *IDistributedCache* ist ein bestehender, verbreiteter Ansatz (und, in Teilen, auch *IMemoryCache*), allerdings nicht besonders effizient und vor allem ohne Unterstützung für Serialisierung. Im Wesentlichen schreiben wir mühsam Bytes in den Cache und lesen diese Bytes wieder aus. Für 2024 entspricht das nicht der erwarteten Entwicklungs-Experience.

Das neue NuGet-Paket *Microsoft.Extensions.Caching.Hybrid* bietet eine elegante Alternative. Nach der Registrierung durch *builder.Services.AddHybridCache()* steht *HybridCache* per Dependency Injection zur Verfügung und bietet über die Methode *GetOrCreateAsync()* quasi alles, was benötigt wird: Entweder steht ein Wert schon im Cache zur Verfügung, oder er wird neu erzeugt. Microsoft ist von diesem (überfälligen!) Ansatz so überzeugt, dass er auch in älteren .NET-Versionen eingesetzt werden kann, bis zu .NET Standard 2.0 und sogar .NET Framework 4.7.2 (die NuGet-Seite [8] spricht sogar von 4.6.2, aber wir wollen es nicht gleich übertreiben)!

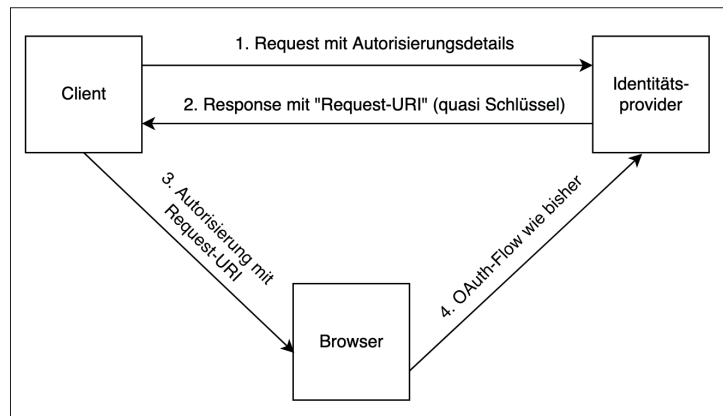
PAR

Auch im Bereich Autorisierung und Authentifizierung gibt es bei ASP.NET Core etwas Neues, nämlich die Unterstützung von PAR. Interessanter als das Feature selbst ist jedoch die Geschichte dahinter.

Zunächst zur Theorie: PAR steht für OAuth 2.0 Pushed Authorization Requests. Das hat von der IETF (Internet Engi-



Vornerum: Typischer Ablauf bei einem Identitätsprovider (**Bild 7**)



Hinterum: Bei PAR tauschen die Server direkt Informationen aus (**Bild 8**)

neering Task Force) die RFC-Nummer 9126 zugewiesen bekommen [9]. Hiermit soll einer der größten Nachteile vieler OAuth- und OIDC-Flows beseitigt werden, nämlich die Weitergabe von Informationen über URLs. Den üblichen groben Ablauf bei Einsatz eines Identitätsproviders zeigt **Bild 7** (hier anhand des Flows „Authorization Code with Proof Key for Code Exchange“ in vereinfachter Darstellung): Der Browser möchte auf eine geschützte Ressource zugreifen, wird zum Identitätsprovider weitergeleitet, und von dort geht es zurück zum Client. Je nach Fluss werden aber bei diesem „Rücksprung“ Parameter mitgegeben – etwa der Authentifizierungscode, oder ein Token, oder Scopes, oder weitere Informationen. Es gibt diverse Szenarien, bei denen diese Informationen abgefangen werden könnten (etwa: Cross-Site Scripting), was die Sicherheit des Mechanismus untergraben würde. Abhängig von den beteiligten Systemen haben ►

URLs eine Maximallänge, sodass es ein Limit bezüglich der Informationen gibt, die so mitgegeben werden können.

Beim Einsatz von PAR läuft es in etwa wie in **Bild 8**. Die zuvor über den Client – gerne „Front-Channel“ genannt – ausgetauschten Informationen werden jetzt im „Back-Channel“ transferiert. Ein clientseitiges Abfangen ist dann nicht mehr möglich, und auch ein durch die URL-Größe vorgegebenes Limit existiert dann nicht mehr. Viele Identitätsprovider un-

Features erweitert. Beispielsweise werden im Bereich „Routing“ auch Endpunkt-Metadaten mit aufgeführt (**Bild 9**), und das Schriftbild (Stichwort Umbrüche) wurde verbessert.

Auch in Visual Studio gab es Verschönerungen, nämlich eine neue Ansicht beim Debuggen von Dictionaries, die jetzt wie eine Liste von Name-Wert-Paaren angezeigt werden. Wieso das als Feature von ASP.NET Core 9 gilt? Das hat sich der Autor dieser Zeilen auch gefragt, aber Microsoft liefert als

Begründung, dass es im Web-Stack von .NET viele solcher Dictionaries gibt, beispielsweise bei allem rund um HTTP (Header, Cookies, GET- und POST-Daten) sowie auch Routen-Infos, um nur einige zu nennen. Das lassen wir jetzt mal als Begründung so stehen. Hieran sieht man schon: Zwar gibt es viele kleinere Neuerungen, aber eine große Linie oder Vision lässt sich daraus nicht ablesen. Möglicherweise gibt es das dann für .NET 10 wieder, dann auch mit drei Jahren Long-Term-Support. ■

Name	Value
Display Name	ExceptionPage.Controllers.HomeController.Crash (ExceptionPage)
Route Pattern	{controller=Home}/{action=Index}/{id?}
Route Order	1

Type	Detail
NullableContextAttribute	System.Runtime.CompilerServices.NullableContextAttribute
NullableAttribute	System.Runtime.CompilerServices.NullableAttribute
ControllerAttribute	Microsoft.AspNetCore.Mvc.ControllerAttribute
ControllerActionDescriptor	Microsoft.AspNetCore.Mvc.Controllers.ControllerActionDescriptor
DataTokensMetadata	DataTokens

Mehr Spaß beim Entwickeln: Die aktualisierte Fehlerseite (**Bild 9**)

terstützen PAR bereits – und mit ASP.NET Core 9 auch .NET, es ist sogar standardmäßig aktiviert.

Die eigentliche Überraschung ist die Quelle des Codes: Es ist ein Community-Beitrag, und zwar von dem Team des populären Identitätsproviders IdentityServer [10] (genauer gesagt: einer von dessen Entwicklern, Joe DeCock). Geradezu rührend, wenn man bedenkt wie vergleichsweise wenig Unterstützung IdentityServer selbst als Open-Source-Projekt erhalten hat, was dann schließlich zur Umwandlung in ein kommerzielles Projekt unter dem Dach von Duende Software geführt hat. Diese kleine Firma unterstützt jetzt den großen Redmonder Konzern bei ihren Open-Source-Bemühungen.

Sehr lesenswert ist die Diskussion rund um den API-Vorschlag von DeCock, der dann letztendlich zur Implementierung geführt hat [11].

Dev-Fehlerseite und mehr

Der letzte Punkt, der an dieser Stelle thematisiert werden soll, ist die Verbesserung aus Entwicklungssicht, auch wenn es sich nur um Kleinigkeiten handelt.

Die allseits beliebige Fehlerseite bei nicht abgefangenen Exceptions (die dann auch keinesfalls auf dem Produktsystem ausgegeben werden sollte) wurde um ein paar wenige

[1] Holger Schwichtenberg, *Politur*, dotnetpro 11/2024, Seite 17 ff., www.dotnetpro.de/A2411Blazor

[2] GitHub-Issues zu ASP.NET Core – Stand 26.08.2024 waren es 3308, www.dotnetpro.de/SL2411ASPNETCore1

[3] Christian Wenz, *Kluge Klemmbausteine*, dotnetpro 10/2024, Seite 78 ff., www.dotnetpro.de/A2410KIControls

[4] Microsoft Learn, *Neuerungen in ASP.NET Core 9*, www.dotnetpro.de/SL2411ASPNETCore2

[5] Aktueller Status der Unterstützung von nativem AOT, www.dotnetpro.de/SL2411ASPNETCore3

[6] Safia Abdalla bei X, ehemals Twitter, <https://x.com/captainsafia>

[7] Homepage der OpenAPI-Initiative, www.openapis.org

[8] Das NuGet-Paket Microsoft.Extensions.Caching.Hybrid, www.dotnetpro.de/SL2411ASPNETCore4

[9] OAuth 2.0 Pushed Authorization Requests, www.dotnetpro.de/SL2411ASPNETCore5

[10] IdentityServer, www.dotnetpro.de/SL2411ASPNETCore6

[11] So kam PAR in ASP.NET Core 9, www.dotnetpro.de/SL2411ASPNETCore7



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

A2411ASPNETCore

NEUIGKEITEN IN BLAZOR 9.0

Politur

Während es in .NET 8.0 für Blazor ganz neue Render-Modelle und eine generalüberholte Projektvorlage gab, glänzt Blazor 9.0 nur durch Polituren an einigen Ecken und Kanten.

Zum Redaktionsschluss für diesen Beitrag Ende Juli 2024 sind von .NET 9.0 schon sechs Preview-Versionen erschienen. Dieser Beitrag stellt die dort enthaltenen Neuerungen für Blazor vor. Die fertige Version soll im November 2024 erscheinen. Vorher wird es noch eine weitere Preview-Version im August und zwei Release-Candidate-Versionen im September und Oktober geben.

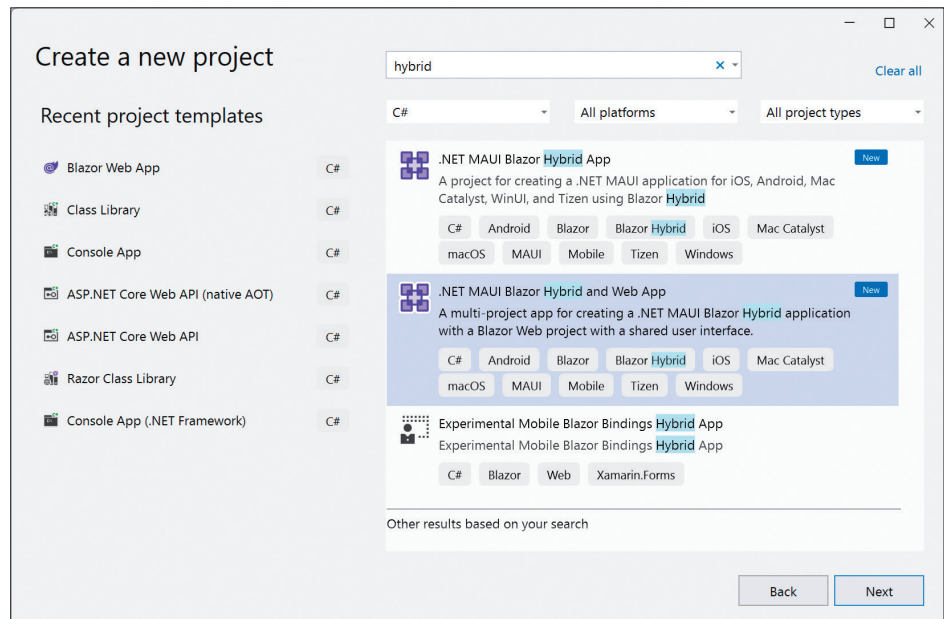
Bezugsquelle und Voraussetzungen

Blazor 9.0 ist im .NET 9.0 SDK enthalten [1]. Eine Koexistenz mit stabilen .NET-Versionen wie .NET 6.0 und .NET 8.0 ist möglich. .NET 7.0 ist im Mai 2024 aus dem Support gelaufen und sollte daher nicht mehr verwendet werden.

Zum Entwickeln braucht man Visual Studio 2022 in der Version 17.11 oder höher. Alternativ kann man mit Visual Studio Code und dem aktuellen C# Dev Kit [2] arbeiten.

Neue Projektvorlage für Cross-Platform-Apps

In .NET 8.0 gab es drei Projektvorlagen für Blazor:

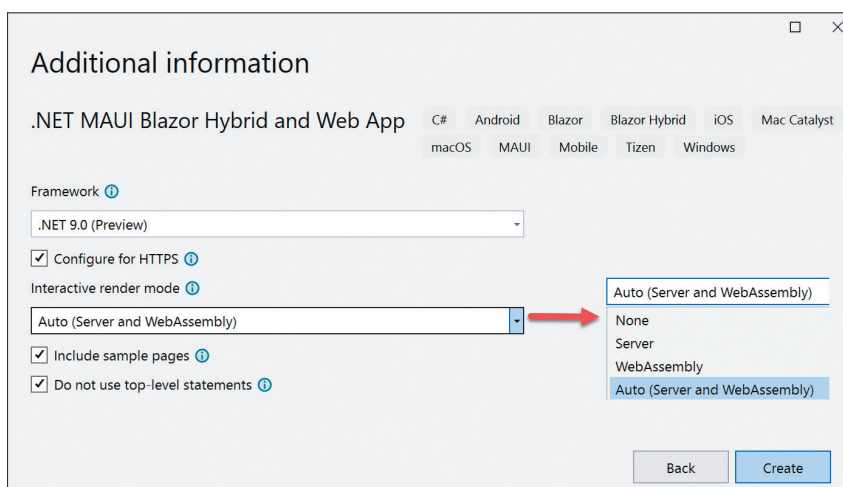


Neue Projektvorlage „.NET MAUI Blazor Hybrid and Web App“ für Cross-Platform-Apps (Bild 1)

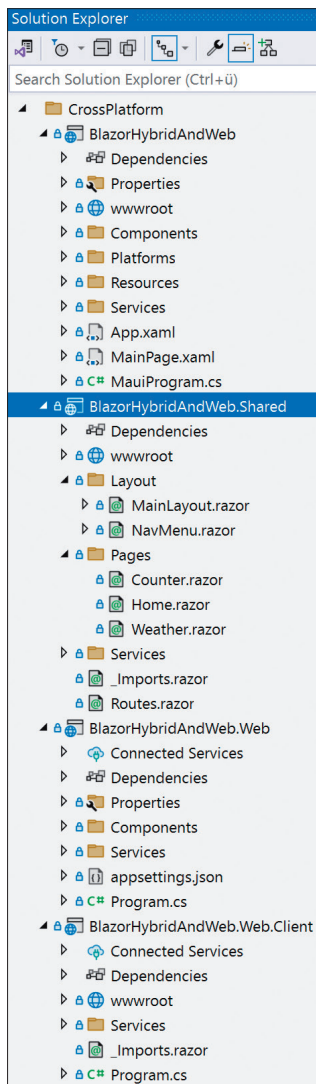
- **Blazor Web App:** Die universelle Projektvorlage für alle Blazor-Arten außer Blazor Hybrid. Sie bietet die Wahl zwischen statischem Server-Side-Rendering, interaktivem Server-Side-Rendering (alias Blazor Server), Client-Side-Rendering (alias Blazor WebAssembly) und dem Auto-Rendering-Modus mit automatischer Umschaltung zwischen den drei vorgenannten Render-Modi.

- **Blazor WebAssembly Standalone App:** Eine Webanwendung mit Blazor WebAssembly, die nicht auf einem ASP.NET-Core-fähigen Webserver gehostet werden muss.
- **.NET MAUI Blazor Hybrid App:** Eine hybride Blazor-Anwendung innerhalb einer .NET-MAUI-Rahmenanwendung, die nativ auf Windows, macOS, iOS, Android und Tizen laufen kann.

Seit .NET 9.0 Preview 5 liefert Microsoft eine weitere, neue Projektvorlage mit der Bezeichnung *.NET MAUI Blazor Hybrid and Web App* (siehe Bild 1). An der Kommandozeile heißt diese Vorlage *maui-blazor-web*. ▶

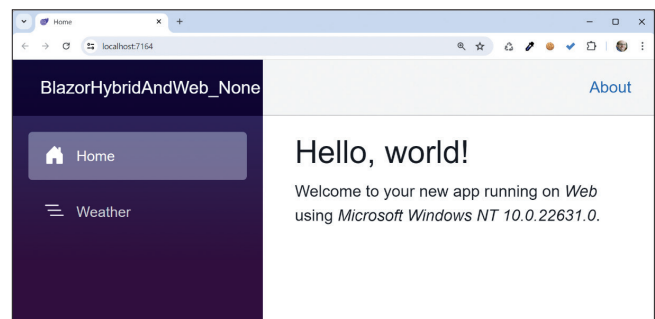
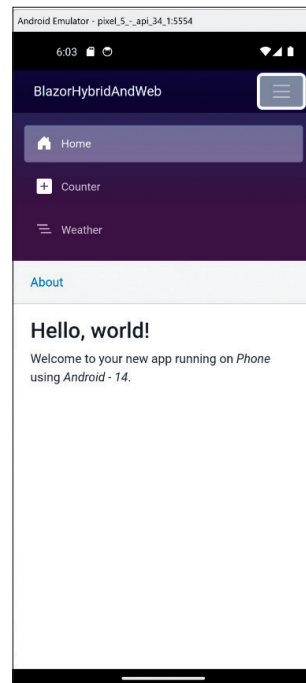


Auswahl des Render-Modus (Bild 2)



Vier Projekte in der neuen Projektvorlage „.NET MAUI Blazor Hybrid and Web App“ (Bild 3)

Cross-Platform-App, hier als Hybrid-App mit .NET MAUI auf Android (Render-Modus: Auto) (Bild 4)



Cross-Platform-App, hier als Hybrid-App mit .NET MAUI auf Android (Render-Modus: None, daher kein Counter-Beispiel) (Bild 5)

Nur jeweils drei Projekte entstehen bei den Render-Modi *None* und *Server* (MAUI ohne Zusatz, *.Web* und *.Shared*), da es hier jeweils keine Trennung zwischen Webserver und Webbrowser gibt, denn der Code läuft nur auf dem Server. Diese Aussage bezieht sich nur auf die Browseranwendung. Bei der Hybridanwendung gibt es keinen Webserver. Der Host ist im Hybrid-Fall der .NET-Client!

Beim Start der Cross-Platform-Anwendung sieht man die in Bild 4 beziehungsweise Bild 5 gezeigte Startseite. Gegenüber der Standardstartseite von Blazor-Anwendungen neu ist dort die Anzeige *running on...*

Die Anzeige *running on...* ist wie folgt realisiert: Im Projekt *Shared* gibt es eine Schnittstelle *IFormFactor*:

```
namespace BlazorHybridAndWeb.Shared.Services;

public interface IFormFactor
{
    public string GetFormFactor();
    public string GetPlatform();
}
```

Diese neue Vorlage erstellt sowohl eine Blazor-Webanwendung (wobei der Render-Modus zwischen *Static*, *Server*, *WebAssembly* und *Auto* wählbar ist, siehe Bild 2) als auch eine Blazor-MAUI-Anwendung.

Damit ist diese neue Projektvorlage ein guter Ausgangspunkt für eine Cross-Platform-Anwendung, die nicht nur im Browser, sondern auch nativ auf Windows, macOS, iOS, Android und Tizen laufen soll. Solch eine Cross-Platform-Anwendung konnte man auch vor .NET 9.0 schon selbst erschaffen, aber Microsoft macht es Entwicklerinnen und Entwicklern nun einfacher.

Bei der Auswahl *Auto* entstehen insgesamt vier Projekte (siehe Bild 3):

- ein Startprojekt (ohne Zusatz hinter einem Punkt) für Blazor MAUI mit XAML-Dateien und für den Betrieb als hybride App auf Windows, Android, iOS und macOS,
- ein Projekt für Webserver (*.Web*),
- ein Projekt für den Webbrowser (*.Web.Client*),
- ein „Shared“-Projekt mit den gemeinsamen Razor Components in dem Ordner */Pages* und gemeinsamer Rahmenlayout-Komponente mit Menü im Ordner */Layout* sowie gemeinsamer Routing-Komponente *Routes.razor*.

● Listing 1: FormFactor.cs im MAUI-Projekt

```
using BlazorHybridAndWeb.Shared.Services;

namespace BlazorHybridAndWeb.Services;
public class FormFactor : IFormFactor
{
    public string GetFormFactor()
    {
        return DeviceInfo.Idiom.ToString();
    }

    public string GetPlatform()
    {
        return DeviceInfo.Platform.ToString() +
            " - " + DeviceInfo.VersionString;
    }
}
```

● Listing 2: FormFactor.cs im Projekt „Web“

```
using BlazorHybridAndWeb.Shared.Services;

namespace BlazorHybridAndWeb.Web.Services;
public class FormFactor : IFormFactor
{
    public string GetFormFactor()
    {
        return "Web";
    }

    public string GetPlatform()
    {
        return Environment.OSVersion.ToString();
    }
}
```

Im Startprojekt für .NET MAUI findet sich die in Listing 1 gezeigte Implementierung von *IFormFactor*, die die Klasse *Microsoft.Maui.Devices.DeviceInfo* aus der Assembly *Microsoft.Maui.Essentials.dll* verwendet. Die in Listing 2 gezeigte Implementierung von *IFormFactor* aus dem Startprojekt für den Webserver liefert als Formfaktor statisch *Web* zurück. Und im Startprojekt für den Webclient gibt es die Implementierung von *IFormFactor* aus Listing 3, die als Formfaktor statisch *WebAssembly* zurückliefert.

Die Implementierungen von *IFormFactor* werden im Startcode der Anwendungen (*Program.cs* bei Webclient und Webserver, *MauiProgram.cs* bei .NET MAUI) im Dependency-Injection-Container registriert:

```
builder.Services.AddSingleton<IFormFactor,
    FormFactor>();
```

Die Seite *Page.razor* im *Shared*-Projekt bezieht dann die entsprechende Instanz:

● Listing 3: FormFactor.cs im Projekt „Web.Client“

```
using BlazorHybridAndWeb.Shared.Services;

namespace BlazorHybridAndWeb.Web.Client.Services;
public class FormFactor : IFormFactor
{
    public string GetFormFactor()
    {
        return "WebAssembly";
    }

    public string GetPlatform()
    {
        return Environment.OSVersion.ToString();
    }
}
```

```
@inject IFormFactor FormFactor
```

Sie sehen in *IFormFactor* und seinen Implementierungen ein Muster dafür, wie Sie verschiedenen Programmcode ausführen können, abhängig davon, ob die Anwendung als Webanwendung oder hybride Anwendung läuft.

Statische SSR-Inseln in Single-Page-Apps

In Blazor 8.0 hatte Microsoft die Render-Modi stark erweitert. Neben Blazor Server (das seit .NET 8.0 auch Interactive Server-Side-Rendering heißt) und Blazor WebAssembly (seitdem auch Client-Side-Rendering) gibt es einen beide Verfahren kombinierenden Auto-Render-Mode (siehe [3]). Alle drei Render-Modi produzieren Single-Page-Web-Apps (SPA). Daneben gibt es seit .NET 8.0 auch ein statisches Server-Side-Rendering für klassische Multi-Page-Web-Applications (MPA).

Bisher konnte man Inseln von Single-Page-Apps mit Blazor Server, Blazor WebAssembly und Auto-Render-Modus in eine statisch auf dem Server gerenderte Webanwendung ►

● Listing 4: App.razor einer Blazor-Server-Anwendung mit statisch auf dem Server gerenderten Einzelkomponenten

```
<!DOCTYPE html>
<html>
<head>
    ... other head content here ...
    <HeadOutlet @rendermode="InteractiveServer" />
</head>
<body>
    <Routes @rendermode="@InteractiveServer" />
    <script src="_framework/blazor.web.js"></script>
</body>
</html>

@code {
    [CascadingParameter]
    private HttpContext HttpContext { get; set; } =
        default!;

    private IComponentRenderMode? PageRenderMode
        => HttpContext.AcceptsInteractiveRouting() ?
            InteractiveServer : null;
}
```

Quelle: Microsoft [4]

einbauen. Solche SPA-Inseln in einer MPA können eine komplette Seite oder ein Teil einer Seite sein (siehe Bild 6 und [3]).

In .NET 9.0 können Entwicklerinnen und Entwickler nun auch umgekehrt in eine Single-Page-App einzelne statische, komplett auf dem Server gerenderte Webseiten einbauen. Dazu annotieren Entwicklerinnen und Entwickler eine Razor Komponente (.razor-Datei) mit:

```
@page "/xyz"
@attribute [
    ExcludeFromInteractiveRouting]
...
```

Wenn dann eine Navigation zu /xyz im Browser erfolgt, wird diese Seite nicht im Client gerendert oder per Websockets-Verbindung vom Server angefordert, sondern mit einem vollständigen HTTP-Request vom Webserver geholt.

In der *App.razor*-Datei muss dieses Verhalten aktiviert werden (siehe Listing 4 und [4]).

Ermitteln des aktuellen Render-Modus

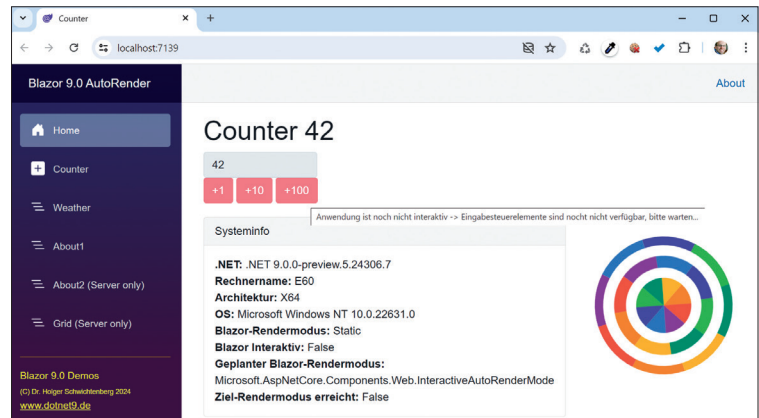
Seit .NET 8.0 gibt es bei Blazor-basierten Browseranwendungen nicht nur die Render-Modi Blazor Server und Blazor WebAssembly, sondern auch Static Server-Side-Rendern und den Auto-Modus, der von Static Server-Side-Rendern zu Blazor Server und Blazor Server zu Blazor WebAssembly umschalten kann. Bisher war es allerdings für Entwicklerinnen und Entwickler nur sehr umständlich möglich, zu ermitteln, in welchem Modus Blazor gerade rendert.

In .NET 9.0 bietet Microsoft nun in der Basisklasse *ComponentBase* für alle Razor Components die Eigenschaft *RenderInfo* (in Preview 5 noch *Platform* genannt) mit den Werten *Name* (string) und *IsInteractive* (bool) an. Mögliche Zeichenketten bei der Eigenschaft *Name* sind:

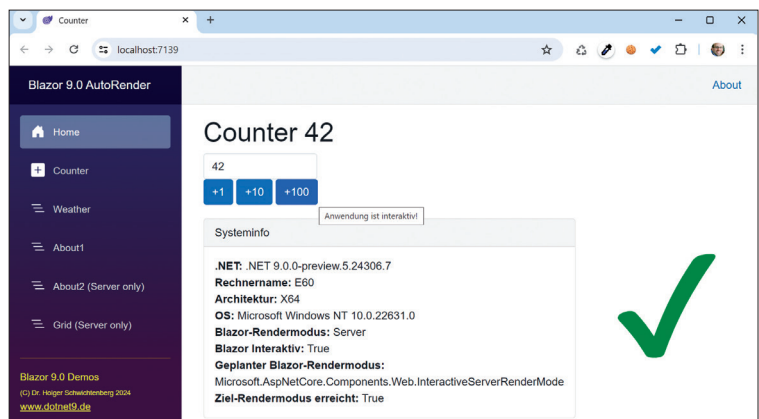
- *Static* für Static Server-Side-Rendern,
- *Server* für Blazor Server,
- *WebAssembly* für Blazor WebAssembly,
- *WebView* für hybride Blazor-Apps in WPF, Windows Forms und .NET MAUI.

Zudem können Entwicklerinnen und Entwickler über die Eigenschaft *AssignedRenderMode* innerhalb einer Komponente auslesen, welcher Render-Modus als nächster folgen wird.

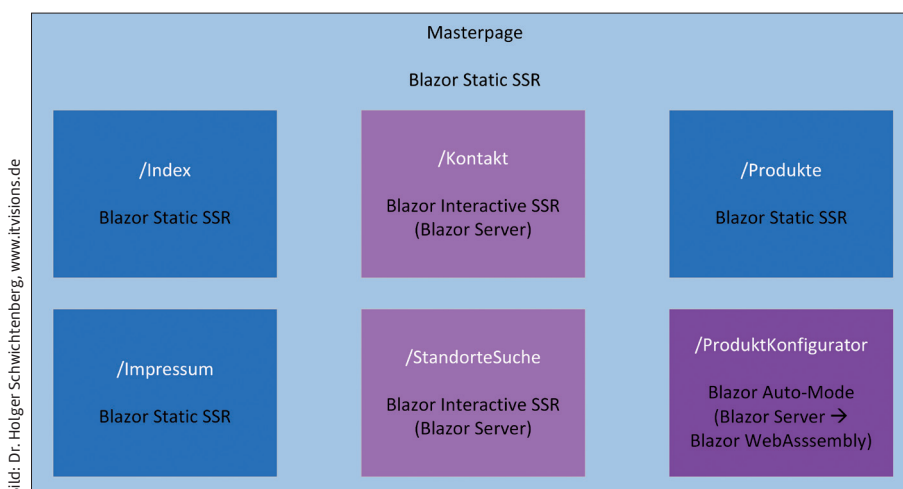
Das folgende Beispiel besteht aus drei Komponenten: *Counter.razor*, *SysInfo.razor* und *Animation.razor*. Die Counter-Komponente (Listing 5) ist eine erweiterte Version des Standardbeispiels aus der Projektvorlage mit einer zusätzlichen Zahleingabe und zwei zusätzlichen Schaltflächen. Alle Eingabesteuerelemente sind deaktiviert, wenn *IsInteractive* den Wert *false* hat. *SysInfo.razor* liefert Informationen über die .NET-Versionen, den Rechnernamen, die Systemarchitektur und



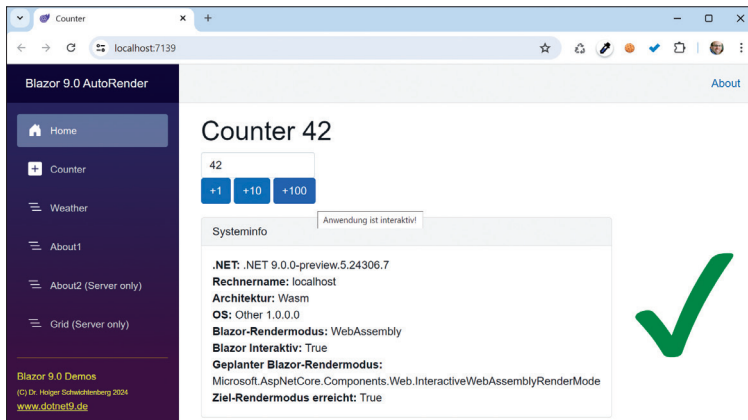
Diese Seite wurde statisch auf dem Webserver gerendert. Eingestellt ist aber der Auto-Render-Modus, das bedeutet, ein Übergang zu Blazor Server oder direkt zu Blazor WebAssembly ist in Arbeit (Bild 7)



Aktuell wird diese Seite mit Blazor Server gerendert, und das soll auch so bleiben, zumindest solange die Seite nicht neu aufgerufen wird (Bild 8)



Interaktive Blazor-Inseln in statisch gerenderten Webseiten gibt es seit .NET 8.0 (Bild 6)



Diese Seite wird mit Blazor WebAssembly im Webbrowser gerendert (Bild 9)

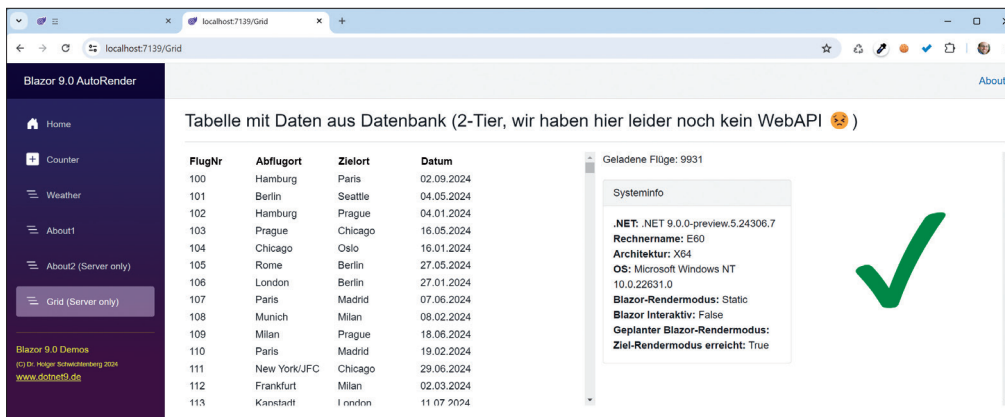
den aktuellen Render-Modus (Listing 6 und Bild 7, Bild 8, Bild 9 sowie Bild 10) Falls der aktuelle Render-Modus noch nicht der angestrebte Render-Modus ist (ermittelt wird dies in der Hilfsfunktion *Render-ModusZielErreicht()*), so sieht man eine Animation, deren Quellcode hier nicht abgedruckt ist.

Optimierungen für statische Webressourcen

Microsoft hat in Blazor 9.0 das Ausliefern von statischen Webressourcen wie CSS-Dateien, JavaScript-Dateien, Grafiken und Videos optimiert. In der Blazor-Projektvorlage *Blazor Web App* hat Microsoft dafür den bisherigen Aufruf in der Startdatei *Program.cs* *app.UseStaticFiles()*; ersetzt durch *app.MapStaticAssets()*;

Die neue Konfigurationsmethode *MapStaticAssets()* anstelle des bisherigen *UseStaticFiles()* sorgt für drei Dinge:

- eine Komprimierung (siehe Bild 11),
- das Einfügen von E-Tags mit SHA-256-Hash zur Versionierung von Ressourcen [5],



Diese Seite wird statisch auf dem Webserver gerendert und ist via `@attribute [Microsoft.AspNetCore.Components.ExcludeFromInteractiveRouting]` so eingestellt, dass sie nicht zu einer der interaktiven Blazor-Varianten übergehen wird (Bild 10)

• Listing 5: Counter.razor

```
@page "/counter"
@page "/"

<PageTitle>Counter</PageTitle>

<h1>Counter @currentCount</h1>

<div title="@((this.RendererInfo.IsInteractive ?
    "Anwendung ist interaktiv!" : "Anwendung ist noch
    nicht interaktiv -> Eingabesteuerelemente sind noch
    nicht verfügbar, bitte warten..."))">

    <input type="number" @bind="currentCount"
        class="form-control" style="width:165px"
        disabled="@(!this.RendererInfo.IsInteractive)" />

    <button class="btn @(this.RendererInfo.IsInteractive
        ? "btn-primary" : "btn-danger")" @onclick=
        "()"=>IncrementCount(1)" disabled=
        "@(!this.RendererInfo.IsInteractive)">+1
    </button>

    <button class="btn @(this.RendererInfo.IsInteractive
        ? "btn-primary" : "btn-danger")" @onclick=
        "()"=>IncrementCount(10)" disabled=
        "@(!this.RendererInfo.IsInteractive)">+10</button>

    <button class="btn @(this.RendererInfo.IsInteractive
        ? "btn-primary" : "btn-danger")" @onclick=
        "()"=>IncrementCount(100)" disabled=
        "@(!this.RendererInfo.IsInteractive)">+100</button>
</div>

@code {
    private int currentCount = 42;

    private void IncrementCount(int x)
    {
        currentCount += x;
    }
}

<SysInfo></SysInfo>
```

- die Adressierung von statischen Ressourcen über einen an den Dateinamen angehängten Hashwert (Fingerprint).

Die letztgenannten beiden Neuerungen dienen dazu, dass ein Browser erkennen kann, ob eine Datei in seinem Cache veraltet ist (siehe **Bild 12**).

Während in ASP.NET Model-View-Controller und ASP.NET Razor Pages die integrierten Tag Helper `<link>`, `<script>` und `<image>` automatisch einen Fingerprint-Hashwert an alle Pfade anhängen, muss man in Blazor eine spezielle Syntax mit der Seitendirektive `@Assets` nutzen, etwa:

```
<link rel="stylesheet" href=
  "@Assets["bootstrap/bootstrap.min.css"]" />
<link rel="stylesheet" href="@Assets["app.css"]" />
<link rel="stylesheet"
  href="@Assets["Anwendungsname.styles.css"]" />
```

Daraus wird dann zur Laufzeit zum Beispiel:

```
<link rel="stylesheet" href=
  "bootstrap/bootstrap.min.bpk8xqwxhs.css" />
<link rel="stylesheet" href="app.da95v2qkru.css" />
<link rel="stylesheet"
  href="Anwendungsname.wthn6682r1.styles.css" />
```

Damit auch die Import-Maps für JavaScript-Dateien funktionieren, muss man in Blazor in der Anwendungsrahmen-HTML-Datei *App.razor* das Tag `<ImportMap />` setzen.

Man sieht in der Registerkarte *Netzwerk* der Entwicklerwerkzeuge des Webbrowsers, dass die oben genannten CSS-Dateien über den Fingerprint-Hashwert geladen werden (**Bild 13**).

Das Laden ohne Fingerprint-Hashwert funktioniert aber weiterhin auch, also:

● Listing 6: SysInfo.razor

```
<div class="row">
  <div class="col">
    <div class="card" style="margin-top:20px">
      <div class="card-header">
        Systeminfo
      </div>
      <div class="card-body">
        <div>
          <b>.NET:</b> @System.Runtime.InteropServices
            .RuntimeInformation.FrameworkDescription
        </div>
        <div>
          <b>Rechnername:</b> @System.Environment
            .MachineName<br />
          <b>Architektur:</b> @System.Runtime
            .InteropServices.RuntimeInformation
            .ProcessArchitecture<br />
          <b>OS:</b> @System.Environment.OSVersion
        </div>
        <div>
          <b>Blazor-Rendermodus:</b>
            @this.RendererInfo.Name<br />
          <b>Blazor Interaktiv:</b>
            @this.RendererInfo.IsInteractive<br />
          <b>Geplanter Blazor-Rendermodus:</b>
            @this.AssignedRenderMode?.ToString()<br />
          <b>Ziel-Rendermodus erreicht:</b>
            @this.RenderModusZielErreicht
        </div>
      </div>
    </div>
  </div>
  <div class="col" align="center">
    @if (this.RenderModusZielErreicht)
    {
      // Grünes Häkchen
      <span style="color:green;font-size:200px;align-
        content:center">&#10004;</span>
    }
    else
    {
      // Arbeits-Animation
      <br />
      <br />
      <Animation></Animation>
    }
  </div>
  @code
  {
    bool RenderModusZielErreicht =>
      this.AssignedRenderMode switch
      {
        Microsoft.AspNetCore.Components.Web
          .InteractiveServerRenderMode x =>
            (this.RendererInfo.Name == "Server" &&
              this.RendererInfo.IsInteractive),
        Microsoft.AspNetCore.Components.Web
          .InteractiveWebAssemblyRenderMode y => (
            this.RendererInfo.Name == "WebAssembly"),
        Microsoft.AspNetCore.Components.Web
          .InteractiveAutoRenderMode z =>
            this.RendererInfo.Name == "WebAssembly", => true
      };
  }
```



```
<link rel="stylesheet"
href="bootstrap/
bootstrap.min.css" />
<link rel="stylesheet"
href="app.css" />
<link rel="stylesheet"
href="BlazorServer9
.styles.css" />
```

File	Original (KB)	Compressed (KB)	% Reduction
bootstrap.min.css	163	17.5	89.26%
jquery.js	89.6	28	68.75%
bootstrap.min.js	78.5	20	74.52%
Total	331.1	65.5	80.22%

Verringerung der Netzwerknutzung durch die Komprimierung anhand der Microsoft-Projektvorlage „Blazor Web App“ (Bild 11)

Der folgende Test mit einer Grafikdatei mit und ohne Fingerprint-Hashwert

```

<br />

```

ergab Stand Preview 6 Folgendes:

- Die Grafik wird in beiden Fällen geladen.
- In der Registerkarte *Netzwerk* der Entwicklerwerkzeuge des Webbrowsers sieht man, dass die Datei beide Male ohne den Hashwert angefordert wurde (Bild 14).

Dependency Injection im Konstruktor von Razor Components

Blazor unterstützt bisher schon Dependency Injection innerhalb und außerhalb von Razor Components. Während normale .NET-Klassen Dependency Injection per Konstruktor unterstützen, konnte Dependency Injection in Razor Components vor Blazor 9.0 nicht per Konstruktor, sondern nur durch *@inject* im Razor-Template oder durch die Annotation *[Inject]*

auf einer Eigenschaft im Code erfolgen.

In Webanwendungen auf Basis von Blazor ist es ab .NET 9.0 möglich, Dependency Injection auch über den Konstruktor in einer Code-behind-Datei in der Razor Component durchzuführen. Wie Listing 7 zeigt, können auch die bereits in C# 12.0 eingeführ-

ten Primärkonstruktoren dafür verwendet werden. Listing 8 und Listing 9 zeigen Im Vergleich dazu die bisherigen Möglichkeiten mit *@inject* im Razor-Template oder durch die Annotation *[Inject]* auf einer Eigenschaft.

Verbesserungen beim Wiederherstellen von Blazor-Server-Verbindungen

Einen größeren Fortschritt gibt es in Blazor 9.0 für Blazor Server beim Wiederherstellen von Verbindungen.

Blazor Server ist eine Variante von Blazor, bei der die Benutzeroberfläche auf dem Webserver erzeugt und mit einem kontinuierlichen Datenaustausch per Websockets so zum Webbrowser übertragen wird, dass der Benutzer das Erlebnis einer Single-Page-App erhält – mit Ausnahme der Offline-Fähigkeit, die es bei Blazor Server nicht geben kann.

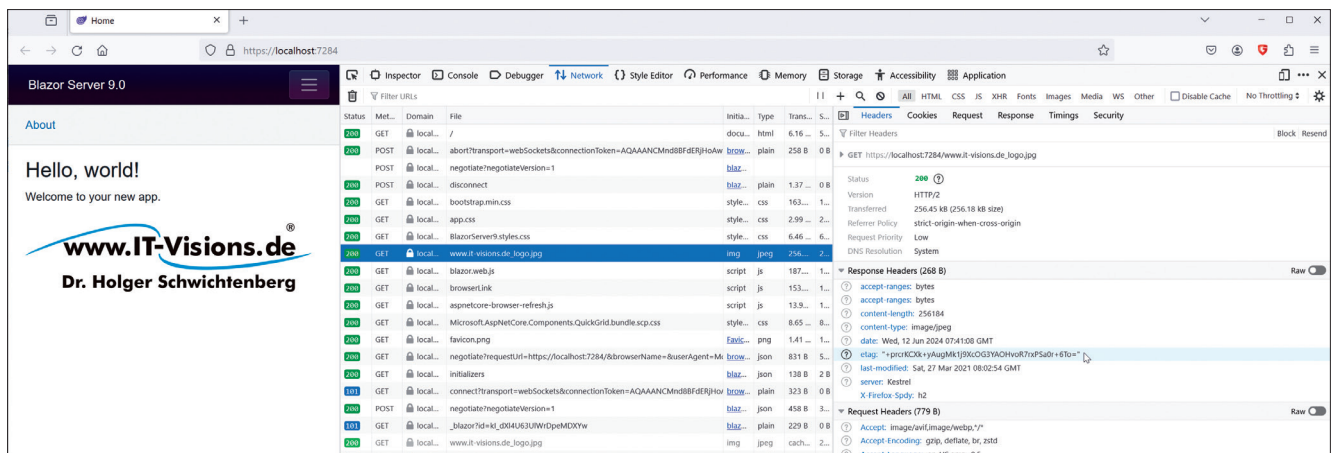
Microsoft bietet in .NET 9.0 jetzt nicht nur eine Überarbeitung der Anzeige für den Endbenutzer beim Verbindungsverlust (siehe Bild 15), sondern auch Verbesserungen beim Wiederherstellen der Verbindung. Beispielsweise wird eine Blazor-Server-Verbindung nun automatisch wieder aufgenommen, wenn der Benutzer zu einem inaktiven Browser-Tab zurücknavigiert. ►

• Listing 7: Konstruktorinjektion in einer Code-behind-Datei einer Blazor-Komponente

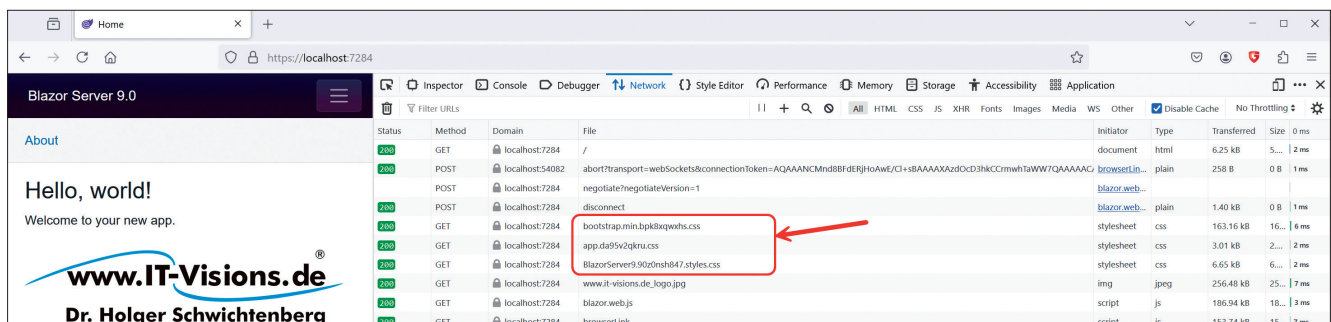
```
public partial class Counter(NavigationManager
    navigationManager, IJSRuntime js)
{
    private void NavigateHome() =>
        navigationManager.NavigateTo("/");
    private string GetBlazorType
    {
        get
        {
            if (js.GetType().Name.Contains("Remote"))
                return "Blazor Server";
            if (js.GetType().Name.Contains("WebAssembly"))
                return "Blazor WebAssembly";
            return "Blazor Hybrid " + js.GetType().Name;
        }
    }
}
```

• Listing 8: Dependency Injection per @inject im Razor-Template

```
@* DI in Razor Components. Bisherige Variante 1*@
@inject NavigationManager nav
@inject IJSRuntime js
@page "/counter"
...
```



Einsatz von E-Tags für eine Grafik (hier in einer Blazor-Server-Anwendung) (Bild 12)



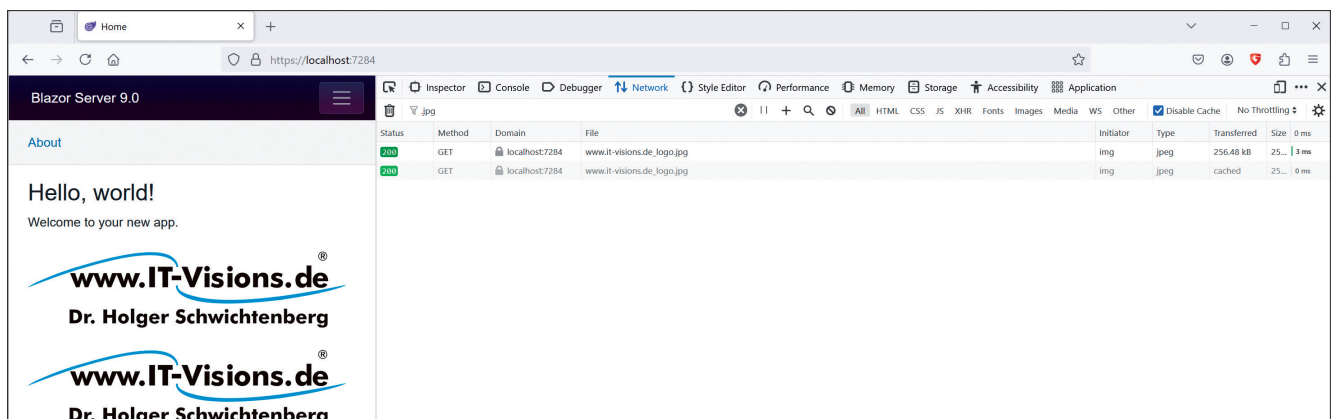
Laden von CSS-Dateien via Seitendirektive @Assets (Bild 13)

Zudem verwendet Microsoft für das Wiederherstellen einer Verbindung nicht mehr Versuche mit linearen, sondern exponentiellen Intervallen. Dies ist konfigurierbar per JavaScript-Code in der Funktion *Blazor.start()*, siehe Listing 10.

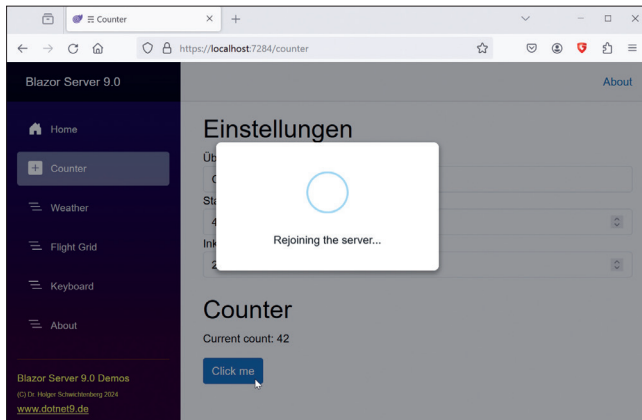
Falls eine Verbindung nicht wiederhergestellt werden kann, weil Blazor Server den Zustand auf dem Webserver bereits verworfen hat oder der Webserverprozess neu gestartet wurde (zum Beispiel beim Austausch des Programmcodes), führt der Browser automatisch ein komplettes Neuladen der Anwendung aus. Dieses Neuladen musste der Benutzer bisher manuell anstoßen.

Wenn ein Neuladen der Seiten erfolgen muss, gehen weiterhin der Zustand der Anwendung und alle Formulareingaben des Benutzers verloren. Microsoft präsentierte aber auf der Build-Konferenz im Mai beim Vortrag „Modern Full-Stack Web Development with ASP.NET Core & Blazor“ [6] einen bisher nicht veröffentlichten Prototyp eines neuen Tags *<StatefulReconnection>* in der *App.razor*-Datei, mit der bei Blazor Server zumindest die aktuellen Formulareingaben des Benutzers beim Neuladen erhalten bleiben.

Auf dem Server gespeicherte Zustände gehen aber weiterhin verloren. Diese ursprünglich für .NET 9.0 geplante Funk-



Laden einer Grafikdatei ohne und mit Seitendirektive @Assets (Bild 14)



Verschönerte modale Anzeige der Neuverbindung bei Blazor Server ab .NET 9.0 (Bild 15)

tion hat Microsoft leider inzwischen auf .NET 10.0 verschoben [7].

OverScan beim QuickGrid

Beim *QuickGrid*-Steuerelement für das Rendern von Datentabellen, welches es seit .NET 7.0 experimentell und seit .NET 8.0 stabil in Blazor gibt, dürfen Entwicklerinnen und Entwickler seit .NET 9.0 im Virtualisierungsmodus einen „OverScan“ explizit angeben. Damit legt man fest, wie viele Zeilen das *QuickGrid*-Steuerelement vor oder nach dem aktuell sichtbaren Bereich noch auf Vorrat rendern soll. Stan-

dardmäßig sind dies drei Zeilen. Entwicklerinnen und Entwickler können dies nun per Einstellung *OverScanCount* im Tag *<QuickGrid>* ändern:

```
<QuickGrid ItemsProvider="itemsProvider"
  Virtualize="true" OverScanCount="10">
  ...
</QuickGrid>
```

Das *QuickGrid*-Steuerelement unterstützt auch in .NET 9.0 keinen Eingabemodus, sondern nur Datendarstellung. Man kann aber mit etwas Eigenarbeit das *QuickGrid* auch zur Dateneingabe verwenden, siehe [8].

Zusammengesetzte Eingaben

In Blazor 9.0 hat Microsoft die Klasse *KeyboardEventArgs* um die Eigenschaft *IsComposing* erweitert, mit der man abfragen kann, ob der Tastaturanschlag Teil einer zusammengesetzten Tastatureingabe ist (siehe Listing 11). Die Klasse *KeyboardEventArgs* wird in folgenden Blazor-Ereignissen verwendet:

- *onkeydown*: Dieses Ereignis wird ausgelöst, wenn eine Taste auf der Tastatur gedrückt wird.
- *onkeyup*: Dieses Ereignis wird ausgelöst, wenn eine gedrückte Taste auf der Tastatur losgelassen wird.
- *onkeypress*: Dieses Ereignis wird ausgelöst, wenn eine Taste gedrückt wird und ein Zeichen erzeugt wird. Zu beachten ist, dass *onkeypress* veraltet ist und in neueren Browsern möglicherweise nicht mehr unterstützt wird. ▶

● Listing 9: Dependency Injection per [Inject] in der Code-behind-Datei einer Razor Component

```
public partial class Counter()
{
    // DI in Razor Components. Bisherige Variante 2
    [Inject]
    NavigationManager nav { get; set; }
    [Inject]
    IJSRuntime js { get; set; }

    private void NavigateHome() => nav.NavigateTo("/");

    private string GetBlazorType
    {
        get
        {
            if (js.GetType().Name.Contains("Remote"))
                return "Blazor Server";
            if (js.GetType().Name.Contains("WebAssembly"))
                return "Blazor WebAssembly";
            return "Blazor Hybrid " + js.GetType().Name;
        }
    }
}
```

● Listing 10: Konfiguration der Intervalle für die Verbindungsversuche bei Blazor Server

```
Blazor.start({
    circuit: {
        reconnectionOptions: {
            retryIntervalMilliseconds: (
                previousAttempts, maxRetries) =>
                previousAttempts >= maxRetries ? null :
                previousAttempts * 1000,
        },
    },
});
```

Quelle: Microsoft

Ein zusammengesetzter Tastaturanschlag bedeutet dabei nicht die Eingabe von Akzenten (´ und á für à), sondern die Eingabe von Zeichen über einen Input Method Editor (IME), etwa den Microsoft Pinyin IME für chinesische Schriftzeichen (Bild 16).

Konfiguration der Websocket-Komprimierung und Content Security Policy (CSP)

Blazor Server verwendet im Standard die Komprimierung für die Websockets-Verbindung. Laut Microsoft [9] ist dies aber mit einem Sicherheitsrisiko verbunden. Daher lässt sich in Blazor 9.0 die Komprimierung deaktivieren:

```
app.MapRazorComponents<App>()
    .AddInteractiveServerRenderMode(o =>
        o.DisableWebSocketCompression = true);
```

Alternativ kann man die Komprimierung aktiv lassen und das Einbetten der Blazor-Server-Anwendung per `<iframe>` anbieten:

```
app.MapRazorComponents<App>()
    .AddInteractiveServerRenderMode(o =>
        o.ContentSecurityFrameAncestorsPolicy = "none");
```

Übergabe des Authentifizierungszustands

Für Blazor-basierte Anwendungen in .NET 9.0 existiert eine einfachere Möglichkeit, den serverseitigen Authentifizierungszustand an Blazor WebAssembly im Browser zu übergeben. Dafür gibt es die neue Methode `AddAuthenticationStateSerialization()` für den Server und `AddAuthentication-`

`StateDeserialization()` für den Client, die jeweils im Startcode in der `Program.cs` aufzurufen sind. Im Standard werden dabei der Name und die Rollen des authentifizierten Benutzers per HTML-Kommentar (`<!--Blazor-WebAssembly-Component-State:eyJfX2ludGVybmFsX19BdXRhZl9BdXNhdGlvb... -->`) vom Server zum Browser übergeben.

Optional können Entwicklerinnen und Entwickler auch alle Claims so übergeben:

```
builder.Services.AddRazorComponents()
    .AddInteractiveWebAssemblyComponents()
    .AddAuthenticationStateSerialization(options =>
        options.SerializeAllClaims = true);
```

Verschobene Features

Leider hat Microsoft im Juni und Juli 2024 die Roadmap für Blazor 9.0 zusammengestrichen. Das Blazor-Team kommentierte dies mit Aussagen wie „Unfortunately, we no longer believe we have the time and resources available to deliver this feature for .NET 9“ [10]. Auf .NET Version 10.0 verlagert wurden unter anderem:

- In Blazor Server sollen nach einer erneuten Aufnahme der Verbindung nach einem Verbindungsabbruch die vorausgegangenen Formulareingaben des Benutzers erhalten bleiben („Stateful Reconnection“) [7].
- Den Transfer von Zuständen vom Webserver zum Webclient will Microsoft in Blazor-Anwendungen vereinfachen, auch mit einem deklarativen Modell, siehe [11] und [12].
- Microsoft will die Authentifizierung mit Microsoft Entra (ehemals Azure Active Directory) wieder besser in Blazor

• Listing 11: Beispiel für Ausgabe, wann ein Tastaturanschlag zusammengesetzt ist

```
@page "/Keyboard"
<PageTitle>Home</PageTitle>
<h1>Neu in Blazor 9.0: isComposing</h1>
<hr />
<h3>Keyboard Input Example</h3>
<input @onkeydown="HandleKeyDown"
    @onkeypress="HandleKeyPress"
    @onkeyup="HandleKeyUp"
    placeholder="Press any key" />
<p>@((MarkupString)log)</p>
@code {
    private string log = "";

    private void HandleKeyDown(KeyboardEventArgs e)
    {
        log += "Key Down: " + e.Key +
            " (Zusammengesetzter Tastaturanschlag in einem
            IME-Editor: " + e.IsComposing + ")<br>";
    }

    private void HandleKeyUp(KeyboardEventArgs e)
    {
        log += "Key Up: " + e.Key +
            " (Zusammengesetzter Tastaturanschlag in einem
            IME-Editor: " + e.IsComposing + ")<br>";
    }

    private void HandleKeyPress(KeyboardEventArgs e)
    {
        log += "Key Pressed: " + e.Key +
            " (Zusammengesetzter Tastaturanschlag in einem
            IME-Editor: " + e.IsComposing + ")<br>";
    }
}
```


integrieren. Im Zuge der neuen, verallgemeinerten Projektvorlage *Blazor Web App* war die Option für Microsoft Entra in Blazor 8.0 entfallen. Microsoft Entra soll in Blazor 9.0 nun nicht nur in die Projektvorlage zurückkehren, sondern es soll auch eine Funktion in Visual Studio geben, um die Authentifizierung mit Microsoft Entra zu bereits bestehenden Blazor-Projekten hinzuzufügen [13].

- Metriken für Blazor [14].
- Verbesserungen für das bisher leider immer noch häufig nicht korrekt funktionierende Hot Reloading [15].

Fazit

In Summe fällt die Anzahl und Bedeutung der Verbesserungen in Blazor 9.0 deutlich bescheidener aus als in .NET 8.0. Diese Feststellung zum Umfang der Neuerungen in Blazor 9.0 möchte der Autor dieses Beitrags aber nicht so verstanden wissen, dass Microsoft schon wieder das Interesse an Blazor verliert, denn es gibt immer noch und immer wieder Entwicklerinnen und Entwickler, die kolportieren, dass es Blazor wie Silberlicht ergehen wird. Blazor ist auch in .NET 9.0 eines der Gebiete mit den meisten Neuerungen. Es gibt heutzutage in der IT für kaum eine Technik eine Bestandsgarantie, aber derzeit sieht es wirklich nicht so aus, als würde Blazor ein baldiges Ende drohen. ■

[1] .NET 9.0 SDK, www.dotnetpro.de/SL2411Blazor1

[2] C# Dev Kit, www.dotnetpro.de/SL2411Blazor2

[3] Holger Schwichtenberg, *Auf der Insel und im Auto*, [dotnetpro 6/2024, Seite 46 ff.,
www.dotnetpro.de/A2406BlazorUnited](https://dotnetpro.de/2024/06/04/Seite-46-ff.-www-dotnetpro.de/A2406BlazorUnited)

[4] Microsoft, *Add static SSR pages to a globally-interactive Blazor Web app*, www.dotnetpro.de/SL2411Blazor3

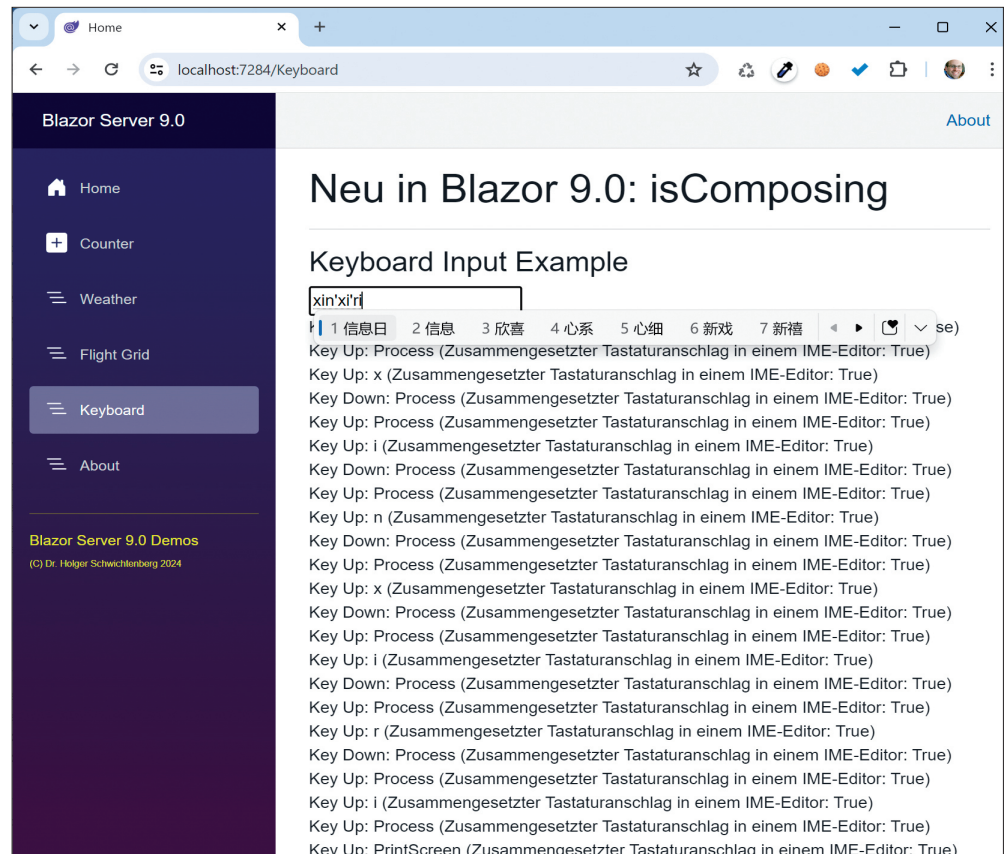
[5] MDN Web Docs, *Etag*, www.dotnetpro.de/SL2411Blazor4

[6] *Modern Full-Stack Web Development with ASP.NET Core & Blazor*, www.dotnetpro.de/SL2411Blazor5

[7] *GitHub Discussions, APIs for components to persist state across circuits manually #30344*, www.dotnetpro.de/SL2411Blazor6

[8] *StackOverflow, Making a Cell Editable in Blazor Quick Grid*, www.dotnetpro.de/SL2411Blazor7

[9] *Microsoft Learn, WebSocket compression for Interactive Server components*, www.dotnetpro.de/SL2411Blazor8



Ausgabe des Beispiels bei der Eingabe chinesischer Schriftzeichen mit Microsoft Pinyin IME (Bild 16)

[10] *Kommentar von Daniel Roth zu GitHub Discussions, APIs for components to persist state across circuits manually #30344*, www.dotnetpro.de/SL2411Blazor9

[11] *GitHub Discussions, Support persistent component state across enhanced page navigations #51584*, www.dotnetpro.de/SL2411Blazor10

[12] *GitHub Discussions, Declarative model to serialize state on prerendering with Blazor and restore it on the client #26794*, www.dotnetpro.de/SL2411Blazor11

[13] *GitHub Discussions, Add Microsoft Identity Platform auth option to the Blazor Web App template #51202*, www.dotnetpro.de/SL2411Blazor12

[14] *GitHub Discussions, Add Metrics for Blazor #53613*, www.dotnetpro.de/SL2411Blazor13

[15] *GitHub Discussions, Revisit hot reload browser refresh script injection #45213*, www.dotnetpro.de/SL2411Blazor14



Dr. Holger Schwichtenberg

zählt zu den bekanntesten .NET- und Webexperten in Deutschland. Microsoft zeichnet ihn seit 20 Jahren als MVP aus. Mit dem 53-köpfigen Expertenteam bei www.IT-Visions.de bietet er Schulungen und Softwareentwicklung an.

www.dotnetdoktor.de

dnpCode

A2411Blazor

EIN BLICK AUF .NET MAUI 9

Endlich Land in Sicht dank MAUI 9

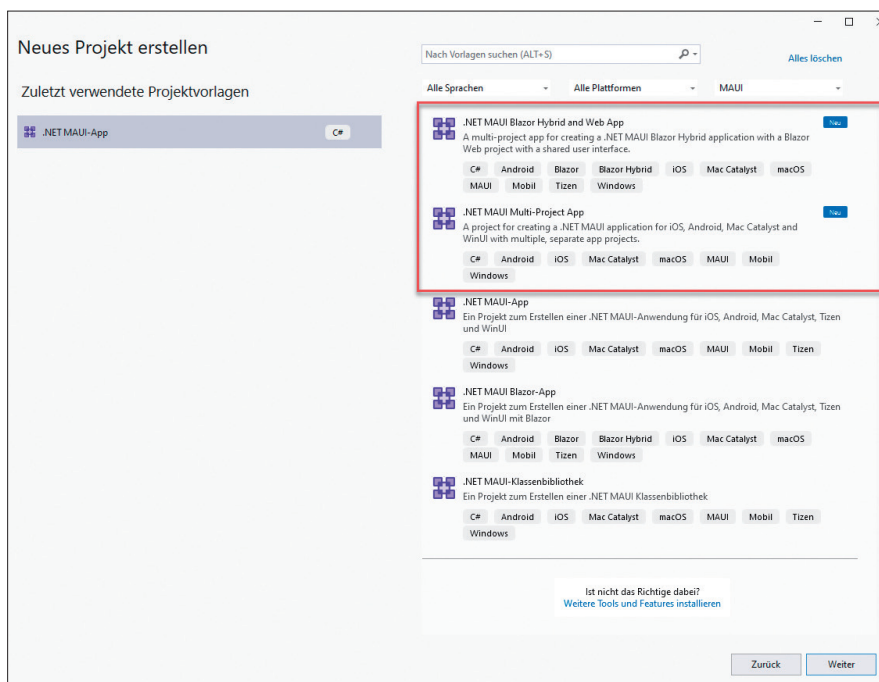
.NET MAUI 9 bringt kaum Neues, dafür aber die Stabilität, die Xamarin.Forms-Entwickler zur Migration ihrer bestehenden Apps dringend benötigen.

Nach mehrjähriger Entwicklungsarbeit präsentierte Microsoft im Mai 2022 das Cross-Plattform-Framework .NET MAUI 6 als evolutionären Xamarin.Forms-Nachfolger. Trotz vielversprechender Neuerungen wie dem Single-Project-Ansatz und einer neuen, lose gekoppelten Steuerelementarchitektur empfanden viele Entwickler .NET MAUI aufgrund erheblicher Stabilitätsprobleme nicht als Fortschritt, sondern eher als Rückschritt.

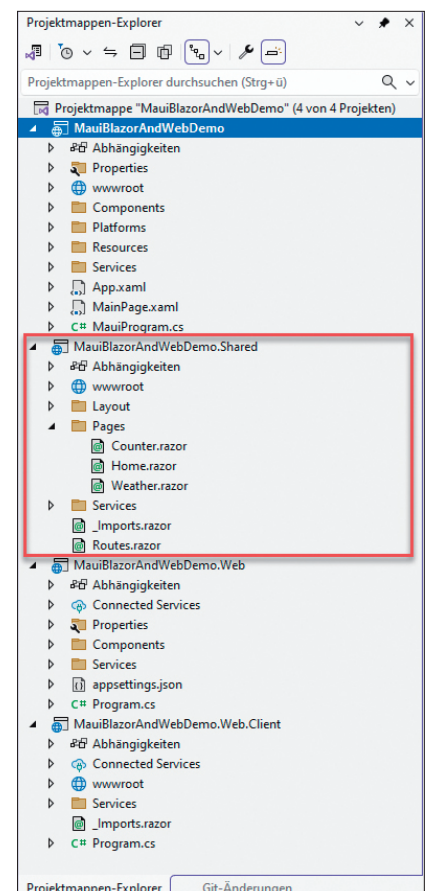
Als Reaktion auf diese Wahrnehmung setzte Microsoft bei den beiden Nachfolgeversionen 7 und 8 den Fokus der Entwicklung vorrangig auf die Stabilisierung des Produkts und führte nur marginale Neuerungen ein. Diese Strategie zahlte sich aus: Ab Version 8 lässt sich von einer stabilen Entwicklererfahrung sprechen. Leider gibt es trotz der vielen Stabilitätsverbesserungen immer noch kritische Fehler. Außerdem fehlen einige Funktionen unter .NET MAUI, die unter Xamarin.Forms vorhanden waren.

Um dies zu adressieren, setzt Microsoft mit .NET MAUI 9 die Strategien der Versionen 7 und 8 fort, indem der Fokus weiterhin auf Fehlerbehebung, verbesserte Testautomatisierung und das Ergänzen fehlender Funktionen aus Xamarin.Forms gelegt wird. Für .NET MAUI 9 bedeutet dies trotz über 1000 Commits in mehr als 3500 Dateien vornehmlich Bugfixes und neue automatisierte Tests, wodurch kaum neue Funktionen eingeführt werden.

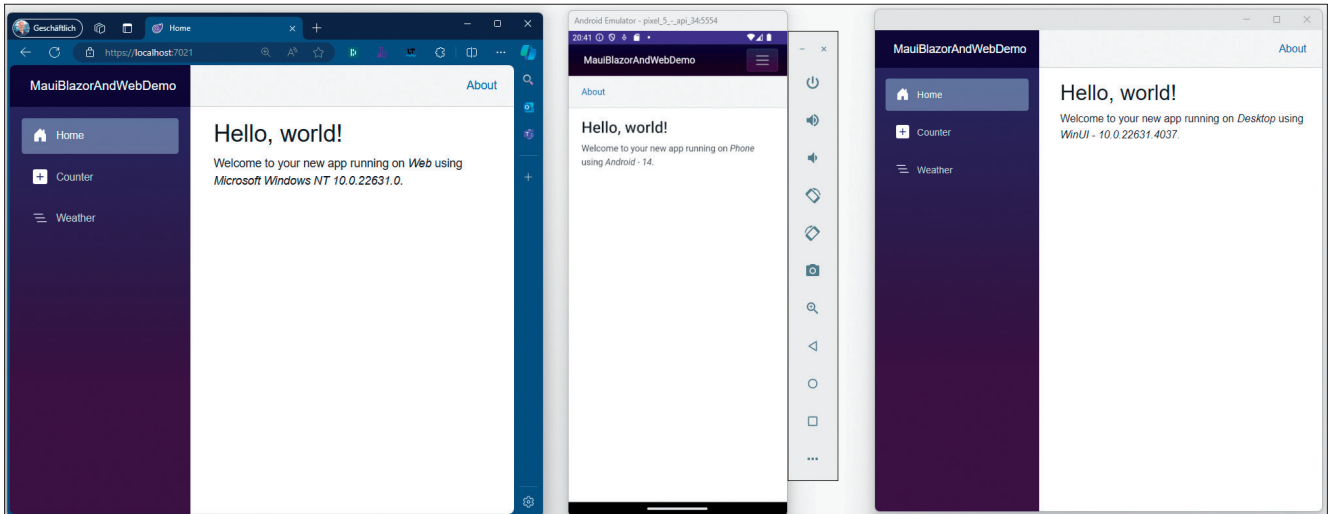
Die Entscheidung des .NET-MAUI-Teams, für .NET MAUI 9 den Schwerpunkt erneut auf Fehlerbehebung statt auf die Einführung neuer Funktionen zu legen, ist eine nachvollziehbare Strategie, insbesondere mit Blick auf das Support-Ende von



Der Dialog zum Erstellen eines neuen Projekts bietet ab .NET MAUI 9 zwei neue Projektvorlagen (Bild 1)



Die Projektstruktur eines Projekts vom Typ .NET MAUI Blazor Hybrid and Web App (Bild 2)



Eine Blazor-Anwendung im Web sowie unter Android und Windows als MAUI-Anwendung (Bild 3)

Xamarin.Forms vom 1. Mai 2024. Seit diesem Datum sind viele Xamarin.Forms-Entwickler gezwungen, ihre Apps kurzfristig auf .NET MAUI zu migrieren. Der weitere Fokus auf Qualität statt Funktionsvielfalt bei .NET MAUI 9 stellt eine solide Basis für neue Apps und Xamarin.Forms-Migrationen sicher. In der Praxis bedeutet dies, dass sich Entwickler auf eine robustere Plattform verlassen können, was letztendlich die Qualität und Zuverlässigkeit der Endprodukte verbessert.

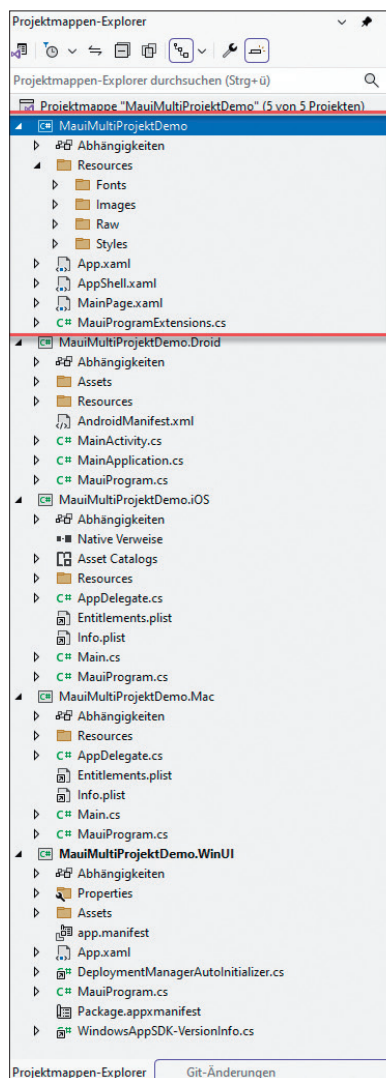
Die Konzentration auf Stabilität und Fehlerbehebung bedeutet jedoch nicht, dass es keine neuen Funktionen in .NET MAUI 9 gibt. Einige wenige Neuerungen haben es in die neue Version geschafft. Hierzu gehören neue Projektvorlagen und zwei neue Steuerelemente.

Neue Projektvorlagen

Die auffälligsten Neuerungen von .NET MAUI 9 sind die beiden neuen Projektvorlagen *.NET MAUI Blazor Hybrid and Web App* und *.NET MAUI Multi-Project App*, die bei der Anlage eines neuen Projekts zur Auswahl stehen (siehe Bild 1).

Die neue Projektvorlage *.NET MAUI Blazor Hybrid and Web App* richtet sich primär an Blazor-Entwickler, die ihren Quellcode im Web und in einer MAUI-App wiederverwenden möchten. Bild 2 zeigt die Struktur eines mit der neuen Projektvorlage erstellten Projekts. Wie Sie dem Screenshot entnehmen können,

Die Projektstruktur einer .NET-MAUI-Multi-Project-Projektmappe (Bild 4)



wird von der Projektanlage neben einem .NET-MAUI- und einem Blazor-Projekt eine Razor-Klassenbibliothek mit der Endung *.Shared* angelegt. In diesem Projekt befinden sich die geteilten Bildschirmmasken, die sowohl im Web als auch in der App angezeigt werden können. Sowohl das .NET-MAUI-Projekt als auch das Blazor-Projekt verweisen auf die Razor-Klassenbibliothek und können somit die dort angelegten Seiten *Counter.razor*, *Home.razor* und *Weather.razor* darstellen. Bild 3 zeigt die laufende Anwendung von links nach rechts im Webbrowser, unter Android und als .NET-MAUI-Windows-Desktop-Anwendung.

Die zweite neue Projektvorlage, *.NET MAUI Multi-Project App*, wirkt auf den ersten Blick wie ein Schritt zurück in die Vergangenheit. Ein mit dieser Vorlage angelegtes Projekt verfügt über ein Projekt für den geteilten Quellcode und ein weiteres je Zielplattform (siehe Bild 4). Dies erinnert an Xamarin.Forms, denn auch dort gab es ein Projekt je Zielplattform sowie ein geteiltes Projekt für den plattformübergreifenden Quellcode. Diese Vorlage richtet sich vor allem an Xamarin.Forms-Entwickler, die mit der traditionellen Projektstruktur vertraut sind als mit dem neueren Single-Project-Ansatz. Zudem erleichtert diese Struktur die manuelle Migration bestehender Xamarin.Forms-Anwendungen.

.NET MAUI liebt JavaScript

Neben den neuen Projektvorlagen und zahlreichen Fehlerkorrekturen haben es auch zwei neue Steuerelemente in ►

● Listing 1: Einsatz des neuen HybridWebView-Steuerelements

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/
    dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/
        winfx/2009/xaml"
    x:Class="HybridWebViewDemo.MainPage">
    <HybridWebView
        x:Name="myHybridWebView"
        RawMessageReceived=
            "myHybridWebView_RawMessageReceived"/>
</ContentPage>
```

.NET MAUI 9 geschafft, das HybridWebView-Control und das TitleBar-Steuerelement.

Das HybridWebView-Steuerelement, das zuvor zwei Jahre lang als experimentelles Steuerelement im GitHub-Repository von Microsoft-Mitarbeiter Eilon Lipton (siehe [1]) entwickelt wurde, ermöglicht die Ausführung von HTML, JavaScript und CSS in .NET-MAUI-Apps. Zusätzlich bietet es eine Schnittstelle für die Kommunikation zwischen JavaScript

und C#. Damit ist es nun möglich, einen Teil der Anwendung oder sogar die gesamte Anwendung mit Webtechnologien zu entwickeln. Diese Technik war zwar bereits seit .NET MAUI 6 mit Blazor möglich, doch mit dem HybridWebView-Control können nun auch Anwendungen, die in anderen Webframeworks wie Angular, React, Svelte oder Vue geschrieben wurden, integriert werden. Dazu wird einfach ein neuer Ordner `wwwroot` im Verzeichnis `Resources\raw` angelegt (siehe Bild 5)

● Listing 2: Aussehen und Verhalten der Titelleiste unter Windows mit dem TitleBar-Steuerelement anpassen

```
protected override void OnAppearing()
{
    base.OnAppearing();

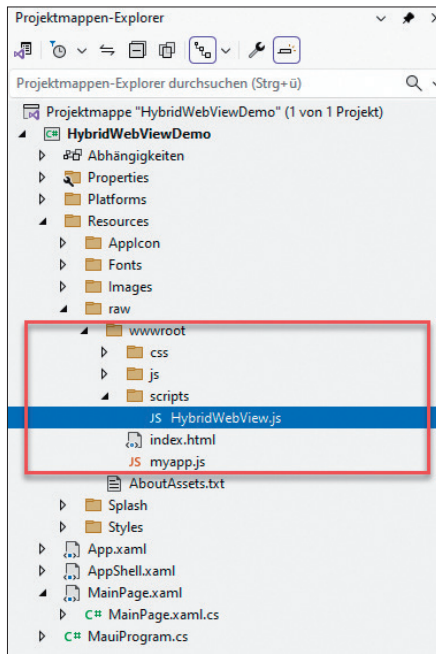
    // Aussehen der Titelleiste anpassen
    Window.TitleBar = new TitleBar
    {
        // Menü-Icon verändern
        LeadingContent = new ImageButton
        {
            Source = "menu.png"
        },
        // App-Icon
        Icon = "bolt.png",
        // Titel der App
        Title = "App Title",
        // Untertitel der App
        Subtitle = "App Subtitle",
        // Hauptinhalt der Titelleiste
        Content = new SearchBar
        {
            Placeholder = "Suche",
            MaximumWidthRequest = 300,
            Margin = 4,
            FontSize = 16,
            HeightRequest = 32,
            BackgroundColor = Colors.White
        },
        // Sekundärer Inhalt der Titelleiste
        TrailingContent =

        new HorizontalStackLayout
        {
            Spacing = 8,
            Children =
            {
                new Label
                {
                    Text = "Wilhelm Brause",
                    VerticalOptions = LayoutOptions.Center,
                },
                new Image
                {
                    Source = "favorites.png",
                    MaximumHeightRequest = 24,
                    MaximumWidthRequest = 24,
                    GestureRecognizers =
                    {
                        new TapGestureRecognizer
                        {
                            Command = new Command(() =>
                                DisplayAlert("Alert",
                                    "You tapped the favorites icon", "OK"))
                        }
                    }
                }
            }
        };
    };
}
```

und die bestehende Webanwendung dorthin kopiert. Die eingebettete JavaScript-Anwendung kann anschließend über ein *HybridWebViewElement* in eine XAML-Seite integriert werden (siehe Listing 1), ohne dass eine Internetverbindung für die Ausführung der webbasierten App benötigt wird. Weitere Informationen zur Integration des HybridWebView-Controls finden Sie unter [2].

Neue Titelleiste vorerst nur unter Windows

Das zweite neue Steuerelement ist das *TitleBar*-Control, mit dem Sie das Aussehen der Titelleiste Ihrer App unter Windows anpassen können. Wie Sie in Bild 6 sehen, können Sie das Menü-Icon verändern, ein App-Icon vergeben und den Titel sowie den Untertitel konfigurieren. Außerdem haben Sie die Möglichkeit, über die beiden Inhaltsbereiche *Content* und *TrailingContent* interaktive Elemente wie zum Beispiel Eingabefelder, Suchleisten, Buttons oder anklickbare Bilder hinzuzufügen, wie man es von zum Beispiel Microsoft Teams oder den modernen Microsoft-Office-Programmen kennt.



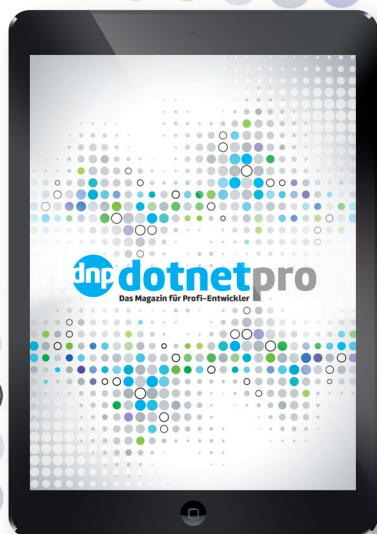
.NET MAUI 9 ermöglicht es, beliebige JavaScript-Webanwendungen mit Ihrer App auszuliefern (Bild 5)

Die Anpassung der Titelleiste können Sie entweder im XAML-Code oder per C#-Implementierung vornehmen. Listing 2 enthält die C#-Variante, deren Ergebnis Bild 6 zeigt. Zum jetzigen Zeitpunkt ist das neue Steuerelement nur unter Windows verfügbar. Zukünftige Updates sollen die Unterstützung auf Mac Catalyst ausweiten, wodurch eine plattformübergreifende Konsistenz gewährleistet wird.

Weitere Neuerungen in .NET MAUI 9

.NET MAUI 9 liefert neben den bereits erwähnten neuen Steuerelementen und Projektvorlagen noch einige kleinere Neuerungen. Dazu gehören zwei neue optionale Handler für iOS und Mac Catalyst, die die Performance der *CollectionView* und der *CarouselView* verbessern. Listing 3 zeigt, wie Sie die beiden Handler in der Datei *MauiProgram.cs* registrieren können. ▶

Jetzt kostenlos testen!



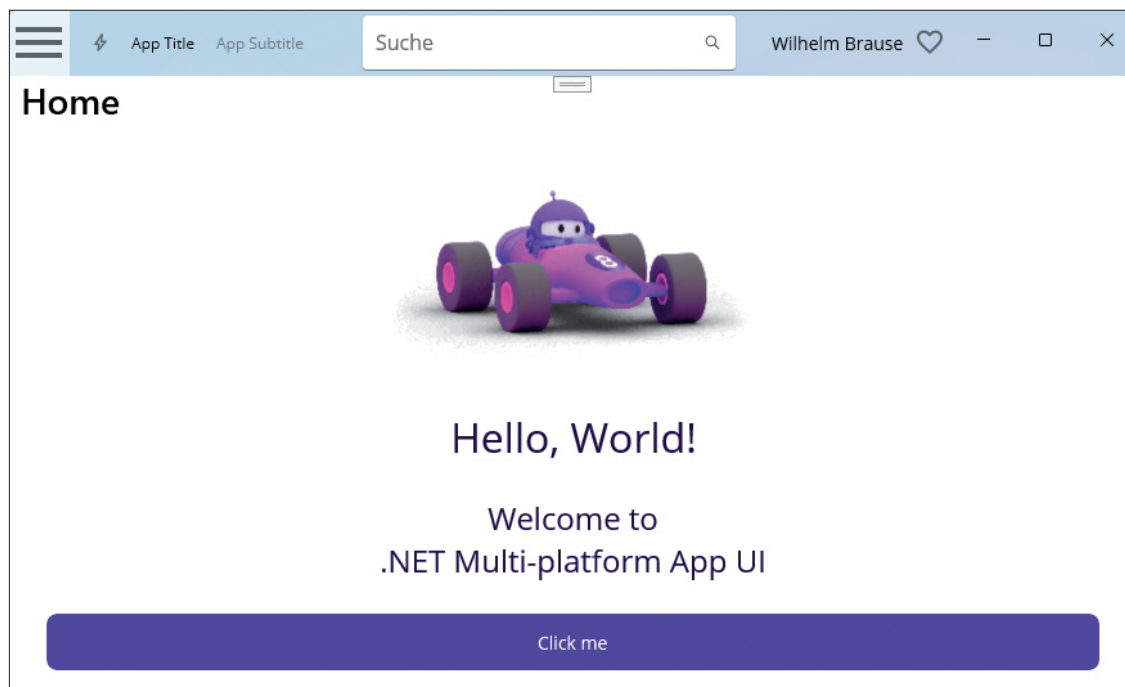
2x
gratis!

Das Fachmagazin für .NET-Entwickler

Testen Sie jetzt 2 kostenlose Ausgaben und erhalten Sie unseren exklusiven Newsletter gratis dazu.

www.dotnetpro.de/dnpabo

Die neue Titel-
leiste unter
Windows (Bild 6)



Eine weitere Neuerung betrifft Eingabefelder, für die .NET MAUI 9 eine verbesserte Eingabeunterstützung für Datums- und Uhrzeitwerte sowie Passwörter durch die virtuelle Bildschirmtastatur bereitstellt.

Zudem erweitert die neueste Version die Unterstützung auf Android 15 (API Level 35) und führt Android-Asset-Packs ein. Diese sind besonders für Spiele- und App-Entwickler relevant, deren Produkte die Standardpaketgröße von Google Play überschreiten. Mit Asset Packs können nun Pakete bis zu 2 GByte statt der üblichen 200 MByte hochgeladen werden, was die Flexibilität in der App-Entwicklung und -Distribution signifikant steigert.

Fazit

.NET MAUI 9 unterscheidet sich auf den ersten Blick kaum von seinem Vorgänger .NET MAUI 8. Schließlich gibt es nur wenige offensichtliche Neuerungen. Die wirklichen Vorteile liegen jedoch, wie eingangs erwähnt, unter der Haube. Mit der Version 9 folgt Microsoft einmal mehr dem Ruf der Entwicklergemeinschaft und legt den Fokus auf die Stabilität des Produkts, anstatt eine Vielzahl neuer Funktionen zu implementieren. Davon profitieren nicht nur Entwickler, die neue

Apps mit .NET MAUI entwickeln, sondern vor allem auch diejenigen, die ihre bestehende App von Xamarin.Forms auf .NET MAUI migrieren. Eine Liste sämtlicher Neuerungen von .NET MAUI 9 finden Sie unter [3]. ■

[1] *.NET MAUI HybridWebView experiment auf GitHub*,
www.dotnetpro.de/SL2411MAUI1

[2] *Microsoft Learn, HybridWebView in .NET 9*,
www.dotnetpro.de/SL2411MAUI2

[3] *Microsoft Learn, What's new in .NET MAUI for .NET 9*,
www.dotnetpro.de/SL2411MAUI3



André Krämer

ist CEO der Quality Bytes GmbH. Für seine Kunden entwickelt der mehrfache Microsoft MVP mit seinem Team .NET-MAUI-Apps und migriert bestehende Xamarin.Forms-Apps. Sie erreichen ihn über LinkedIn, via qualitybytes.de oder unter andrekraemer.de.

dnpCode

A2411MAUI

● Listing 3: Registrierung der iOS- und Mac-Catalyst-Handler `CollectionViewHandler2` und `CarouselHandler2`

```
#if IOS || MACCATALYST
builder.ConfigureMauiHandlers(handlers =>
{
    handlers.AddHandler<Microsoft.Maui.Controls
        .CollectionView, Microsoft.Maui.Controls
        .Handlers.Items2.CollectionViewHandler2>();

    handlers.AddHandler<Microsoft.Maui.Controls
        .CarouselView, Microsoft.Maui.Controls.Handlers
        .Items2.CarouselHandler2>());
});
#endif
```


NEUERUNGEN IN C# 13

Evolution statt Revolution

Welche Sprachfeatures von C# sich mit der neuen Version ändern werden.

Alle Jahre wieder ist es so weit, das haben zumindest die vergangenen fünf Jahre gezeigt: Eine neue Version von C# steht vor der Tür. Zusammen mit .NET 9 wird auch C# 13 zur .NET Conf im November 2024 verfügbar sein, und so ist es an der Zeit für einen Blick ins Detail: Was bringt die neue Version mit sich, wie werden neue Sprachfeatures unseren Umgang mit der Sprache in nächster Zeit ändern?

Es gibt eine ordentliche Zahl von Neuerungen zu beschreiben – etwa 11 bis 13, je nach Zählweise –, und es hätten noch mehr sein können, wenn auch die Extensions noch den Schritt geschafft hätten! Dazu später noch etwas mehr, zum Einstieg sei festgestellt: Keine Sorge, mit C# 13 wird nicht „alles anders“. Die neuen Features und Erweiterungen sind solcherart, dass Sie sich graduell an sie gewöhnen oder gar nichts davon merken, wenn Sie nicht besonders nachschauen.

„params“ jetzt ohne Array

Das Schlüsselwort *params* kennen Sie in C# seit der ersten Stunde. Damit kann eine Methode eine beliebige Anzahl von Werten entgegennehmen, die bisher immer in einem Array gesammelt wurden:

```
void AcceptSeveralIntsOldStyle(params int[] values) {
    //
}
```

Natürlich könnten Sie ein Array von *ints* auch ohne *params* übergeben, aber mit dem Schlüsselwort wird der Aufruf ohne explizit erzeugtes Array möglich:

```
AcceptSeveralIntsOldStyle(1, 2, 3, 4, 5);
```

Neu in C# 13 ist nun, dass diese Syntax außer einem Array auch jeden anderen Typ verwenden darf, der mit einer Collection Expression erzeugt werden kann. Sie erinnern sich daran aus C# 12: Ein *CollectionBuilderAttribute* definiert für einen Typ, dass beziehungsweise wie er als Collection initialisiert werden kann. Dieses Attribut findet sich in .NET bereits bei allen relevanten Typen, aber sie können es auch auf eigene Typen anwenden. Zusätzlich gilt, dass mit *params* auch kompatible Interfaces verwendet werden dürfen. Alle folgenden Varianten sind also von jetzt an möglich, und viele mehr:

```
private static void TakeThis<T>(params IEnumerable<T>
    values) {
}

private static void TakeThis<T>(params ReadOnlySpan<T>
```

```
    values) {
}

private static void TakeThis<T>(params List<T> values) {
}
```

Wie in anderen Bereichen auch legt Microsoft viel Wert auf die Nutzbarkeit der Typen *Span<T>* und *ReadOnlySpan<T>*, da diese hinter vielen Performance-Verbesserungen im .NET der letzten Jahre stecken. Wenn Sie also Spans weiterverarbeiten, können Sie diese auch direkt entgegennehmen. Dasselbe gilt allerdings zum Beispiel auch für *IEnumerable<T>*: Möchten Sie damit weiterarbeiten, etwa durch Aufrufe an LINQ-Funktionen, gibt es nun keinen Zwischenschritt mehr.

Eventuell wollen Sie nun einwenden: Mit dem *int[]* kann auch LINQ aufgerufen werden! Das ist wahr, aber Sie sollten im Kopf behalten, dass eine Funktion, die mit *params* deklariert ist, noch immer auch mit einem „fertigen“ Typ aufgerufen werden kann.

```
// Lange Syntax der Klarheit halber
TakeThis(new List<int> { 1, 2, 3 });
```

Der Aufruf mit einer bestehenden Liste wäre etwa bei Deklaration der Funktion mit *int[]* nicht möglich.

Mit einem Cast können Sie auch beeinflussen, welcher Overload aufgerufen wird:

```
TakeThis((IEnumerable<int>) [1, 2, 3])
```

Von hinten initialisiert

Eine recht spezifische Rolle spielt ein neues Feature, durch das die Unterstützung von Indexen bei der Initialisierung verbessert wird. Sie erinnern sich an Indexe, wie sie zu Zeiten von C# 8 mit den Range-Features eingeführt wurden.

```
int[] numbers = new int[5];
Index i = ^1;
numbers[i] = 5;
numbers[^2] = 4;
```

Die Verwendung des Index erlaubt es, einen Wert vom Ende des Arrays aus zu adressieren. Manchmal ist es auch möglich, einen Wert für ein gesamtes Array in einer Initialisierungszuweisung zu setzen:

```
public class CalculationMatrix {
    public int[] MagicValues { get; set; } = new
```

```
int[10];
}
...

static void Main() {
    CalculationMatrix x = new() {
        MagicValues = {
            [1] = 42,
            [^1] = 52
        }
    }
}
```

Die Verwendung der Indexsyntax in der Initialisierung ist seit C# 13 nun ebenfalls möglich. So wird im Beispiel *MagicValues[1]* auf den Wert 42 gesetzt und *MagicValues[9]* auf 52. In älteren C#-Versionen wäre dazu ein zusätzlicher Schritt nach der Initialisierung notwendig gewesen.

Fett und rot jetzt einfacher

Die Konsolenanwendung lebt! So denkt anscheinend auch Microsoft – durchaus zu Recht, denn Apps für die Kommandozeile, rein textuell oder mit TUI semigrafisch aufgebaut, gibt es heute mehr denn je. Seit jeher schrieb eine Fachperson womöglich Code wie diesen, um ein bestimmtes Format auf der Konsole zu erzeugen:

```
Console.WriteLine("This is \u001b[1mbold\u001b[0m text");
```

Klar, es gibt auch fertige Libraries zu solchen Zwecken, aber hier und da muss man so etwas vielleicht mal selbst ausführen! Nun wissen Sie bestimmt: *[1m* ist der Teil, der den Text fett macht, *[0m* schaltet die Formatierung wieder ab, und an beiden Stellen muss das Präfix *\u001b* verwendet werden. Das ist Unicode, daher *\u*, und dann der Wert des Zeichens, also ein mal sechzehn plus B, also 11, das macht 27. Escape ist das, schon seit uralten ASCII-Zeiten!

Jetzt kommt C# 13, und Escape ist viel kürzer, nämlich so:

```
Console.WriteLine(
    "This is \e[31m\e[1mbold red\e[0m text");
```

Anstelle von *\u001b* darf es jetzt also *\e* heißen. Das spart Zeit und Bytes. Vielleicht nicht das wichtigste neue Feature für manche Apps, aber trotzdem ein interessanter Exkurs.

Properties und Indexer jetzt teilweise

Lang ist es her, und die meisten von Ihnen erinnern sich vermutlich nicht, dass in C# 3 die Partial Methods (und Classes) eingeführt wurden. Sie waren dazu gedacht, Programmierern die Verwendung automatisch generierten Codes einfacher zu machen. Ein Tool konnte somit einen Teil einer Klasse erzeugen, während ein menschlicher Programmierer einen anderen Teil baute. Der Compiler konnte aus diesen Teilen ein fertiges Konstrukt zusammenfügen und gleichzeitig Optimierungen anbringen, wenn zum Beispiel eine Methode gar nicht benötigt wurde.

```
// Von einem Tool generierter Code
public partial class MyGreeter {
    public void Go() {
        SayHello();
    }

    partial void SayHello();
}

// Manuell erstellter Code
public partial class MyGreeter {
    partial void SayHello() {
        Console.WriteLine("Hi there!");
    }
}
```

Gewisse Regeln galt es zu beachten: Im Beispiel ist etwa die Methode *SayHello* an der Stelle ihrer Deklaration absichtlich nicht *public* oder auch nur *private* deklariert, denn nur Methoden ganz ohne explizite Sichtbarkeit dürfen ausdrücklich auch nicht implementiert werden (und werden dann vom Compiler entfernt).

Neu in C# 13 ist nun, dass auch Properties und Indexer *partial* deklariert werden dürfen. Das sieht so aus:

```
// Von einem Tool generierter Code
public partial class MyGreeter {
    public void Go() {
        SayHello();
    }

    partial void SayHello();

    public static partial string Greeting { get; }
    public partial int this[int index] { get; }
}

// Manuell erstellter Code
public partial class MyGreeter {
    partial void SayHello() {
        Console.WriteLine("Hi there!");
    }

    public static partial string Greeting => "Everybody";
    public partial int this[int index] => 42;
}
```

Natürlich müssen Deklaration und Implementation übereinstimmen, was die Typen sowie die Details angeht: In dem obigen Code gibt es der Kürze halber nur Getter und keine Setter; das muss an beiden Stellen gleichermaßen eingehalten werden.

GeneratedRegex nun als Property

Das neue Feature der *partial*-Properties hat sich Microsoft umgehend zunutze gemacht, um das Source-Generator-Feature zur Erzeugung von ausdruckspezifischen RegEx-Ma-

schinen auch auf Properties anwendbar zu machen. Bereits seit C# 11 konnten Sie das Attribut `[GeneratedRegex]` auf *partial*-Methoden anwenden. Nun ist das auch mit Properties möglich:

```
// Seit C# 11
[GeneratedRegex(@"[abc]+")]
private static partial Regex MyRegex();

// Seit C# 13
[GeneratedRegex(@"[abc]+")]
private static partial Regex MyRegexProperty { get; }
```

Diese Neuerung mögen Sie als Kleinigkeit betrachten, und vom technischen Gesichtspunkt aus betrachtet ist sie auch tatsächlich kein großer Schritt. Von der Verwendbarkeit her ist es jedoch wesentlich besser lesbar, einen regulären Ausdruck beim Namen einer Property zu referenzieren statt durch einen Methodenaufruf:

```
// Man könnte denken, die Methode gebe etwas Dynamisches
// zurück
int count1 = MyRegex().Count(testString);

// Wesentlich schöner und nicht verwirrend
int count2 = MyRegexProperty.Count(testString);
```

Kleine Verbesserungen und Unsicherheit als Tugend

Microsoft berichtet, dass C# 13 gewisse Verbesserungen im Bereich der „natürlichen Typen von Methodengruppen“ bringt. Diese Aussage ist derzeit nicht mit besonderen Beispielen belegt worden, weshalb wir das für den Moment einfach hinnehmen dürfen.

Solche Veränderungen im Lauf der Zeit sind durchaus nicht ungewöhnlich, denn im Compiler wird immer wieder einmal ein Fehlverhalten bereinigt oder die algorithmische Logik erweitert (wie in diesem Fall), um besondere Randfälle korrekt zu handhaben.

Im Zusammenhang mit Verbesserungen bei Co- und Contravariance schrieb ich seinerzeit, dass diese letztlich erreichen sollen, dass wir als Programmierer weniger vom Verhalten des Compilers überrascht werden. Oder umgekehrt formuliert: Der Compiler soll sich weitgehend so verhalten, wie wir es logisch erwarten. Genau das gilt auch für die natürlichen Typen von Methodengruppen, und die Details sind nur selten wichtig.

Ähnlich sind einige kleine Veränderungen einzuschätzen, die unter dem Titel „*ref* und *unsafe* in *async*-Methoden und Iteratoren“ zusammengefasst werden. Dabei war es bisher nicht möglich, *ref*-Werte und *unsafe*-Blöcke in den genannten Kontexten von *async*-Methoden und Iteratoren zu verwenden, und in C# 13 wurden diese Beschränkungen fallen gelassen. Tatsächlich ist den Diskussionen zu den Feature-Vorschlägen, die Sie bei GitHub nachlesen können, zu entnehmen, dass die Einschränkungen im Lauf der Zeit ganz einfach als zu restriktiv erkannt wurden. Nun kam der Ge-

danke auf, dass statt eines pauschalen Verbots dieser Kombinationen explizit geregelt werden könnte, wann oder wo *ref* und *unsafe* ausdrücklich nicht benutzt werden dürfen.

Die Regel ist nun diese: in *async*-Methoden und Iteratoren können *ref* und *unsafe* verwendet werden, solange dies nicht ausgerechnet in den „speziellen“ Konstrukten dieser besonderen Methoden geschieht. Also: *ref* darf nicht als Teil einer *await*- oder *yield return*-Anweisung vorkommen, und weder *await* noch *yield return* dürfen in einem *unsafe*-Block stehen.

Auf dieser neuen Basis kann nun exemplarisch ein *unsafe*-Block durchaus in einem Iterator vorkommen, wie im folgenden Beispiel zu sehen:

```
class Program {
    static void Main(string[] args) {
        byte[] array = [1, 2, 3, 4, 5];
        GCHandle handle = GCHandle.Alloc(array,
            GCHandleType.Pinned);
        IntPtr ptr = handle.AddrOfPinnedObject();

        // Dies gibt 1, 2, 3, 4, 5 aus
        foreach (byte b in ReadMemory(ptr, array.Length)) {
            Console.WriteLine(b);
        }

        handle.Free();
    }

    public static IEnumerable<byte> ReadMemory(
        IntPtr startAddress, int length) {
        for (int i = 0; i < length; i++) {
            byte value;
            unsafe {
                // Ein wenig gepflegte Pointerarithmetik,
                // nur zur Illustration
                byte* valuePtr = (byte*)startAddress + i;
                value = *valuePtr;
            }

            // yield return muss außerhalb von unsafe stehen
            yield return value;
        }
    }
}
```

Es lebe der `Span<T>` und seine `ref-struct`-Freunde

Wie bereits oben erwähnt verwendet Microsoft den *Span<T>* und den *ReadOnlySpan<T>* in .NET heute an vielen Stellen. Eingeführt in C# 7.2, liegt das Feature *ref struct* diesen Typen zugrunde. Ein *ref struct* ist ein Stack-basierter Datentyp, der sehr effizient arbeitet und gegen versehentliche Fehlbehandlung geschützt ist, wie sie etwa durch Boxing entstehen könnte. Auch Speicherlecks werden durch die Ablage auf dem Stack nebenbei verhindert.

In C# 13 gibt es zwei Neuerungen, die eine immer weiter fortschreitende Verwendung von *ref struct* fördern dürften. ►

Die erste Neuerung ist, dass ein *ref struct* nun auch Interfaces implementieren kann. Dabei gibt es allerdings wesentliche Einschränkungen, um zu gewährleisten, dass die Regeln von *ref struct* eingehalten werden! Generell sieht die Implementation ganz „normal“ aus:

```
public interface IMovable {
    public int DistanceFromZero { get; set; }
    void Advance();
}

public ref struct Turtle : IMovable {
    public int DistanceFromZero { get; set; }

    public void Advance() {
        // Turtle moves slowly
        DistanceFromZero++;
    }
}

public ref struct Person : IMovable {
    public int DistanceFromZero { get; set; }

    public void Advance() {
        // Person moves much more quickly than a turtle
        DistanceFromZero += 10;
    }
}
```

Es ist zu beachten, dass ein *ref struct* immer alle Interface-Member implementieren muss, auch solche, die eine Default-Implementation haben.

Nach der Instanziierung findet sich die nächste Einschränkung schnell: Ein Cast zum Interface-Typ ist nicht möglich.

```
Turtle turtle = new();

// Fehler!
IMovable m = turtle;

Person person = new();

// Fehler!
IMovable[] movables = [turtle, person];
```

Ein *ref struct* darf nicht direkt, im Sinne eines C#-Casts, in den Interface-Typ „konvertiert“ werden. Auch eine Übergabe an eine Methode, die den Interface-Typ als Parameter akzeptiert, ist nicht erlaubt.

```
private static void Move(IMovable movable) {
    movable.Advance();
}

...

// Fehler!
Move(turtle);
```

Nun fragen Sie sich womöglich, wozu denn die Interface-Implementation überhaupt nutzbar ist, wenn eine Instanz nicht zum Interface-Typ gecastet werden kann.

Die Antwort hängt mit dem Hintergrund dieser Erweiterung zusammen. Es war dem C#-Team nämlich nicht recht, dass während der Arbeit am *Span<T>* immer wieder Implementationen neu gebaut werden mussten, die es für *IEnumerable<T>* bereits gab.

Um dieses Problem zu lösen, sind aber keine direkten Casts notwendig oder vorgesehen, sondern nur die Übergabe als generischer Parametertyp!

So können die Werte aus dem vorherigen Beispiel nun per Interface referenziert werden:

```
private static void Move<T>(ref T movable)
    where T : IMovable, allows ref struct {
    movable.Advance();
}

private static void Output<T>(T movable)
    where T : IMovable, allows ref struct {
    Console.WriteLine($"Distance from zero: {
        movable.DistanceFromZero}");
}

...

Move(ref turtle);
Move(ref person);

Output(turtle);
Output(person);
```

Im Gegensatz zum Cast zieht diese Methode der Übergabe kein Boxing nach sich, sodass gegen die Regeln von *ref struct* nicht verstoßen wird. Nebenbei gibt es im Beispiel zwei interessante Punkte:

- Erinnern Sie sich: *ref struct* wird nicht automatisch als *ref* (also als Referenz) übergeben, ebenso wenig wie ein „normales“ *struct*. Daher ist bei der Methode *Move<T>* der Parameter als *ref* deklariert und wird auch so übergeben.
- Im *where*-Teil der Methodendeklarationen findet sich ein weiteres neues Feature in C# 13: *allows ref struct*.

In C# vor Version 13 war es so, dass *ref struct*-Typen ausdrücklich nicht als generische Typparameter fungieren durften. Folgendes ist zum Beispiel nicht möglich:

```
private static void TakeThis<T>(T thing) {
    Console.WriteLine(
        $"Took a thing, its type is {thing.GetType()}");
}

...

// Fehler!
TakeThis(Span<int>.Empty);
```

Mithilfe des neuen *allows ref struct* können Sie ab C# 13 angeben, dass der generische Parameter außer allen anderen für ihn gültigen Typen auch *ref struct*-Typen annehmen darf.

```
private static void TakeThis<T>(T thing)
    where T: allows ref struct {
    // GetType() kann nicht für ref structs aufgerufen
    // werden, da das eine Object-Methode ist.
}
```

allows ref struct wird übrigens nicht vererbt. In diesem Beispiel kann für *T* ein *ref struct* verwendet werden, aber nicht für *X*:

```
void TakeThese<T, X>(T one, X other)
    where T: allows ref struct,
           X: T {
    ...
}
```

Zum Abschluss noch einmal zurück zur Interface-Implementation mit *ref struct*, denn es gibt abseits all der Einschränkungen auch einige frohe Nachrichten dazu:

- Implementationen von *IDisposable* und *IAsyncDisposable* in *ref struct*-Typen werden von *using* beziehungsweise *await using* erkannt.
- *foreach* und *await foreach* erkennen Implementationen von *IEnumerable<T>* und *IAsyncEnumerable<T>*.

Overload Resolution Priority

Wer APIs erzeugt, die von anderen Programmierern verwendet werden, ist sicher mit dem *ObsoleteAttribute* vertraut. Damit können Sie anzeigen, dass ein Element nicht mehr verwendet werden soll, wahlweise mit einer Warnung oder einer Fehlermeldung.

Im folgenden Beispiel gibt es eine Methode *Handle*, die hypothetisch in einer alten Form eines API erzeugt wurde und mit einem einfachen *int[]* arbeitete. Diese Methode ist daher mit *Obsolete* markiert, allerdings nur mit einer Warnung – das wird oft so gemacht, damit ein Nutzer des API nicht gleich mit zahlreichen Compilerfehlern konfrontiert wird. Bei manchen APIs, die extern von unbekannten anderen Programmierern genutzt werden, ist diese Markierung mit *Obsolete* und einer Warnung eventuell der letzte und einzige Schritt, der guten Gewissens getan werden kann.

Im Beispiel sehen Sie ebenfalls eine neue Methode, die ganz modern mit *Span<int>* anstelle des *int[]* arbeitet. Wenn Sie dieses Beispiel ausprobieren, werden Sie allerdings feststellen: Der Overload-Resolution-Mechanismus des Compilers, also die Technologie, die nach der „richtigen“ Methode suchen soll, interessiert sich nicht für das *Obsolete*-Attribut! Bemerkte man als Programmierer also nicht selbst die Warnung und leitet man keine entsprechenden Schritte ein, wird immer die falsche Methode verwendet.

```
private static void Main() {
    int[] values = [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

```
// Ausgabe: Array Handler
Handle(values);
}
```

```
[Obsolete("Use the Span<int> overload instead")]
private static void Handle(int[] items) {
    Console.WriteLine("Array Handler");
}
```

```
private static void Handle(Span<int> items) {
    Console.WriteLine("Span<T> Handler");
}
```

Um dieses Problem zu lösen, hat Microsoft nun ein neues Attribut eingeführt: das *OverloadResolutionPriorityAttribute*. Die Priorität, die hier gesetzt werden kann, hat den Standardwert 0 (Null).

Wenn Sie, wie im Fall des Beispiels sinnvoll, diese Priorität ändern wollen, sollten Sie Werte größer 0 verwenden – hier genügt also 1.

```
private static void Main() {
    int[] values = [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

```
// Ausgabe jetzt: Span<T> Handler
Handle(values);
}
```

```
[Obsolete("Use the Span<int> overload instead")]
private static void Handle(int[] items) {
    Console.WriteLine("Array Handler");
}
```

```
[OverloadResolutionPriority(1)]
private static void Handle(Span<int> items) {
    Console.WriteLine("Span<T> Handler");
}
```

Mit diesem neuen Attribut wird den API-Entwicklern zusätzlich zum bekannten *Obsolete* eine wichtige neue Technik an die Hand gegeben, die es ihnen ermöglicht, ihre Anwender während der Evolution des API in die richtige Richtung zu lotsen.

System.Threading.Lock

In .NET 9 wurde ein neues Objekt namens *System.Threading.Lock* eingeführt, das für Locks gedacht ist.

Nun gab es in C# seit jeher ein Schlüsselwort *lock*, das mit beliebigen Objekten arbeitete:

```
private static readonly object objectLock = new();
```

```
private static void DoSomething() {
    lock (objectLock) {
        Console.WriteLine("Doing locked stuff");
    }
}
```


Diese Implementation von *lock* generiert Code wie den folgenden, der mit der Klasse *System.Threading.Monitor* arbeitet:

```
bool lockTaken = false;
try {
    Monitor.Enter(objectLock, ref lockTaken);
    Console.WriteLine("Doing locked stuff");
}
finally {
    if (lockTaken) {
        Monitor.Exit(objectLock);
    }
}
```

In C# 13 wird nun das Schlüsselwort *lock* mit einem neuen Verständnis für den speziellen Typ *System.Threading.Lock* ausgestattet. So können C#-Programmierer weiterhin mit dem spracheigenen *lock* arbeiten und brauchen lediglich den neuen Typ anstelle eines *object* einzusetzen, wenn Sie von den potenziellen Performance-Vorteilen profitieren wollen. Der Code sieht fast genauso aus wie zuvor:

```
private static readonly System.Threading.Lock
    improvedLock = new();

private static void DoSomethingElse() {
    lock (improvedLock) {
        Console.WriteLine("Doing other locked stuff");
    }
}
```

Der Compiler erkennt aber, was geschieht, und erzeugt nun folgenden Code:

```
using (improvedLock.EnterScope()) {
    Console.WriteLine("Doing other locked stuff");
}
```

Was nicht mehr geht

In aller Kürze sollen zwei Punkte erwähnt werden, die es auf die offizielle Breaking-Changes-Liste bei Microsoft geschafft haben. Manchmal bringen Neuerungen auch negative Nebeneffekte mit sich! Das ist etwa im Fall der neuen, präziseren Regelung von *unsafe* im Zusammenhang mit Iteratoren geschehen. So fehlte es nämlich im Roslyn-Compiler auf Versionsstand C# 12 noch an einem im Spezifikationsdokument versprochenen sicheren Kontext, sodass der nachfolgende Code ausdrücklich nicht zu einer Compiler-Fehlermeldung führte. In C# 13 wiederum arbeitet der Compiler nun wie vorgesehen und meldet den Fehler.

```
// Ein äußerer unsafe-Kontext
unsafe class MyList {
    IEnumerable<int> InnerIterator() {
        yield return 1;
        local();
    }
}
```

```
void innerInnerFunction() {
    int* p = null; // erlaubt in C# 12;
                  // Fehler in C# 13
}
}
```

Übrigens kann der Code auch in C# 13 verwendet werden, wenn die *innerInnerFunction* explizit als *unsafe* deklariert wird.

Der zweite Punkt soll nur der Vollständigkeit halber erwähnt werden: Das *InlineArrayAttribute* ist nicht mehr mit *record struct*-Typen verwendbar.

```
// Fehler CS9259 in C# 13
[System.Runtime.CompilerServices.InlineArray(10)]
record struct Buffer1() {
    private int _element0;
}
```

Wie geht es weiter?

Die Arbeit an der Sprache C# wird kontinuierlich vorangetrieben, und wer in GitHub die Diskussionen und die Notizen der „LDMs“ (Language Design Meetings) verfolgt oder gar selbst zu den Meetings geht, weiß jederzeit in großer Detailtiefe Bescheid. Ein besonders interessantes Feature, das beinahe den Weg nach C# 13 geschafft hätte, steht hoffentlich für C# 14 auf dem Programm: die „Extensions“, im Design ursprünglich auch einmal als „Rollen“ bezeichnet.

Sie kennen Extension-Methoden, wie sie seit C# 3 existieren – oberflächlich betrachtet sind die neuen geplanten Extensions ähnlich, zumindest in dem Sinne, dass sie die alten Extension-Methoden ablösen könnten (es soll aber wohl beides geben, der Kompatibilität halber). Allerdings gehen die Ideen wesentlich weiter, von mächtigen Alias-Mechanismen mit Unterstützung für Generics über Typergänzung (Augmentation) und aspektorientierte Programmierung bis hin zu Typklassen.

Gerade wegen dieser umfassenden Bedeutsamkeit der Pläne stellte sich leider heraus, dass die bereits existierende Implementation des Features aus Sicht des CLR-Teams bei Microsoft gravierende Probleme mit sich brachte. Daher wurde und wird noch immer auf eine neue Implementation mancher Teile umgesattelt. Das braucht seine Zeit, sodass für C# 13 wohl nichts mehr daraus wird. Aber immerhin können wir uns schon mal sehr auf C# 14 freuen! ■



Oliver Sturm

ist Training Director bei DevExpress. In den vergangenen 30 Jahren hat er weitreichende Erfahrungen als Softwareentwickler und -architekt, Berater, Trainer, Sprecher und Autor gesammelt.

oliver@oliversturm.com

dnpCode

A2411CSharp

PROGRAMM

DDC



.NET DEVELOPER CONFERENCE '24

DIE KONFERENZ FÜR .NET ENTWICKLER
25. - 28.11.2024 | KÖLN

dotnet-developer-conference.de | #DDC24 | Find us on [f](#) [i](#) [x](#) [in](#)

Präsentiert von:



MONTAG, 25. NOVEMBER 2024: WORKSHOPS

UHRZEIT	WORKSHOP 1	WORKSHOP 2	WORKSHOP 3	WORKSHOP 4
09.00 - 17.00	 Die perfekte .NET-Architektur David Tielke Die Welt der Architekturen für die Softwareentwicklung ist groß, der Überblick geht dabei schnell verloren. Architekturerperte David Tielke zeigt Ihnen in diesem Workshop, wie Sie mit wenig Aufwand die perfekte Architektur für Ihr .NET Projekt entwickeln, kommunizieren und kontrollieren.	 ASP.NET von 0 auf 100 Christian Giesswein ASP.NET Core ist ein leistungsstarkes, plattformübergreifendes Framework zur Erstellung moderner Webanwendungen. In diesem Workshop lernt Ihr die Grundlagen und fortgeschrittenen Techniken von ASP.NET Core kennen. Schritt für Schritt führt Euch Christian Giesswein durch den gesamten Entwicklungsprozess --- von der Einrichtung Ihrer Umgebung bis zur Bereitstellung einer voll funktionsfähigen Webanwendung.	 Dein Weg zum Cross-Plattform-App-Entwickler mit .NET MAUI und .NET 9 André Krämer André Krämer zeigt in diesem Workshop, wie .NET-Entwickler zu Cross-Plattform-Entwicklern werden können. Ausgehend von einer einfachen Bleistiftskizze entwickelt er live eine App für Android, iOS, macOS und Windows mit .NET MAUI. Ihr erhaltet eine detaillierte Einführung in die Entwicklung von Dashboards, die Gestaltung von Eingabefeldern, die lokale Datenhaltung, die Validierung von Eingaben und den Zugriff auf Gerätefunktionen. Dabei bietet der Workshop nicht nur tiefgehende Einblicke, sondern auch praktische Tipps, um die Herausforderungen der Cross-Plattform-Entwicklung zu meistern und beeindruckende Apps zu erstellen.	 Single-Page-Apps mit Blazor 9.0 und C# 13.0 Dr. Holger Schwichtenberg Microsoft hat die Einsatzszenarien von Blazor durch Blazor Static-Server-Rendern und den Auto-Modus mit Umschaltung zwischen Blazor Server und Blazor WebAssembly erheblich erweitert. In diesem DDC-Workshop präsentiert Dotnet-Doktor Holger Schwichtenberg zunächst den Stand von Blazor in Version 9.0 und stellt die verschiedenen Blazor-Architekturmodelle mit ihren Vor- und Nachteilen gegenüber und leitet die Teilnehmer*innen anschließend an, selbst eine Blazor 9.0-Anwendung zu entwickeln, die serverseitig vorrendert, dann Blazor Server nutzt und im Hintergrund automatisch die Dateien für Blazor WebAssembly herunterlädt (Progressive Enhancements).
	 From Zero to Frontend-Hero Peter Kröner Ihr sprecht fließend Java, habt aber mit JavaScript zu kämpfen? Euer C# ist verhandlungssicher, doch bei CSS versteht Ihr nur Bahnhof? Ihr beherrscht XML in Wort und Schrift, doch Euer HTML-Vokabular lässt zu wünschen übrig? Dann seid Ihr hier genau richtig! Entwickler*innen jeder Couleur haben heutzutage zwar ständig mit dem Web zu tun. Doch wer sich primär mit Datenbanken, Cloud-Architektur und Backend-Sprachen befasst, kann sich nicht jeden Hack und jede Library aus dem Web-Frontend draufschaffen. Dafür gibt es diesen Workshop. Neben einem Überblick über den aktuellen Stand der wichtigsten Webstandards bietet er Antworten auf alle Eure Frontend-Fragen. Lernt von Web-Tech-Basics über aktuelle Tools bis hin zu neuesten Standards alles, was den modernen Frontend-Stack zusammenhält.	 AI-driven development mit GPT und Co. Jörg Neumann LLMs eröffnen eine völlig neue Welt. Mussten wir bisher prozeduralen Code schreiben, um Software etwas beizubringen, reicht nun die menschliche Sprache. Doch auch für die klassische Entwicklung bieten LLMs wie ChatGPT, GitHub Copilot & Co. Erleichterungen. Fest steht schon jetzt: LLMs werden die Anwendungsentwicklung grundlegend verändern. Jörg Neumann zeigt, wie LLMs funktionieren und wie man sie in Projekte integriert. Das Spektrum reicht von einfachen Anpassungen der Modelle per Few-Shot-Learning, dem Trainieren tiefen Wissens über Fine Tuning bis hin zu semantischer Suche mittels Retrieval Augmented Generation. Darüber zeigt er, wie sich LLMs in Microsoft Teams integrieren sowie Datenbanken und APIs innerhalb einer Konversation abfrage lassen.	 Cloud aspirations: building cloud-native applications with .NET Aspire Loek Duys, Alex Thissen  Starting by examining typical distributed applications and their architecture styles in the cloud, leveraging PaaS services from cloud providers with their resilience and scalability characteristics, we will introduce common concerns for distributed applications, such as observability, service discovery and configuration, and how .NET Aspire can help. You learn how to build your apps with .NET Aspire, by making proper use of the orchestration, components and toolings. We also cover the .NET Aspire service discovery and connection management, discover the programming model and the developer workflow. In the hands-on labs you experience and practice all learnings, enhancing an application by composing it using the .NET Aspire primitives of projects, containers and cloud resources.	 Agile Discovery: Wertvolle Produkte von Anfang an richtig gestalten Ina Einemann In der Softwareentwicklung sind Flexibilität, Schnelligkeit und kontinuierliche Anpassung entscheidend, um sich ständig ändernden Anforderungen des Marktes gerecht zu werden. Oft setzt man auf agile Methoden wie Scrum. Doch während viele Projekte schnell in die Delivery-Phase starten, wird die Discovery-Phase meist vernachlässigt - eine entscheidende Etappe, um marktfähige Produkte zu entwickeln, Mehrwert zu schaffen und kostspielige Fehler zu vermeiden. Daher tauchen wir in diesem Workshop in die agile Discovery ein. Wir erkunden für effektives Anforderungsmanagement wichtige Verfahren und Praktiken. Ihr lernt, wie Ihr eine klare Produktvision erstellt, Personas entwickelt und Story Mapping anwendet, um Projekte auf Kurs zu bringen. Am Ende steht ein fundiertes Verständnis agiler Discovery.

DIENSTAG, 26. NOVEMBER 2024: KONFERENZ

UHRZEIT	SOFTWARE ARCHITECTURE, TESTING, AND QUALITY <i>David Tielke</i>	FRONTEND DEVELOPMENT <i>Fabian Gosebrink</i>	ARTIFICIAL INTELLIGENCE <i>Jörg Neumann</i>	DATA & DATABASES <i>Thorsten Kansy</i>
09.00 - 10.00	Nicht mehr das .NET Deiner Kindheit <i>Christian Giesswein</i>			
10.00 - 10.30	Kaffeepause und Besuch der Fachmesse			
10.30 - 11.30	Architekturen testen mit NDepend <i>David Tielke</i>	Let's turn your website into a hybrid .NET MAUI app <i>Mark Allibone</i>	AI-driven Software Development mit Azure AI <i>Jörg Neumann</i>	Die versteckten EF Core Features <i>Christian Giesswein</i>
RAUMWECHSEL				
11.45 - 12.45	Im Notfall Scheibe ein- schlagen: Strategien zum richtigen Behandeln von Fehlern <i>Florian Bader</i>	Migration: WinForms und WPF goes Web <i>Dr. Veikko Krypczyk</i>	Azure OpenAI und Semantic Kernel für Softwareentwickler <i>Dr. Damir Dobric</i>	Deep Dive: Entity Framework Core-Performance <i>Dr. Holger Schwichtenberg</i>
12.45 - 14.15	Mittagspause und Besuch der Fachmesse			
14.15 - 15.15	Stop starting with Micro- services - Der modulare Monolith in .NET <i>Florian Lenz</i>	Migrating from Xamarin.Forms to .NET MAUI <i>Iris Classon</i>	DDC-Panel: Chancen und Risiken der KI in der .NET- Entwicklung <i>Dr. Damir Dobric, Ina Einemann, Jörg Neumann u.a. (Host: Tilman Börner)</i>	GraphQL on .NET with steroids! <i>Mauro Bennici</i>
RAUMWECHSEL				
15.30 - 16.30	Taktisches Refactoring von .NET Legacy Code <i>Gregor Biswanger</i>	Enterprise Apps mit Blazor Server und EF Core entwickeln <i>Thomas Claudius Huber</i>	Big Brother is watching you - Lokales GenAI in Anwendungen <i>Thomas Tomow</i>	Azure Datenanalyse: Wie Licht ins Dunkel kommt <i>Florian Bader</i>
16.30 - 17.00	Kaffeepause und Besuch der Fachmesse			
17.00 - 18.00	Beyond Clean Code: Zeitgemäße Ansätze für hochqualitatives Software- engineering <i>Alexander Ramm</i>	Webanwendungen und Cross-Platform-Apps mit Blazor: Soll ich's machen, oder lass ich's lieber sein? <i>Dr. Holger Schwichtenberg</i>	„Talk to your systems“: Strukturierte Daten & Tool- Integration <i>Christian Weyer</i>	Visual Studio als Entwicklungstool für SQL Server basierte Datenbanken <i>Thorsten Kansy</i>
RAUMWECHSEL				
18.15 - 19.15	Wir testen. Aber testen wir auch gut genug? <i>Patrick Drechsler</i>	Echtes Cross-Plattform mit Blazor, WPF und MAUI - ein Praxisbericht <i>André Krämer</i>	Geschüttelt, nicht gerührt: KI-Agenten im Dev-Team <i>Alex Netsch, Markus Weiss</i>	SQL Server - eine sichere Sache <i>Thorsten Kansy</i>
ab 19.15	ABENDVERANSTALTUNG			
19.30 - 20.30	AI Ninja Moves: Wie kann ich AI in der Entwicklung konkret nutzen? <i>Jörg Neumann und Oliver Sturm</i>			
20.30 - 23.00	GET TOGETHER			

MITTWOCH, 27. NOVEMBER 2024: KONFERENZ

UHRZEIT	.NET LANGUAGES, FUNDAMENTALS, BEST PRACTICES <i>Christian Giesswein</i>	APPROACHES <i>Michael Kaufmann</i>	BACKEND TECHNOLOGIES <i>Thomas Claudius Huber</i>	CLOUD DEVELOPMENT <i>Marc Müller</i>
09.00 - 10.00	(How to destroy) Happiness at work <i>Kathleen Janz und Stefan Mintert</i>			
10.00 - 10.30	Kaffeepause und Besuch der Fachmesse			
10.30 - 11.30	.NET-Whatsup: Was gibt es Neues in .NET 9? <i>Christian Giesswein</i>	DevOps 3 Ways - Die Philosophie hinter DevOps <i>Thomas Schissler</i>	API FOMO: REST, gRPC or GraphQL? <i>Roland Guijt</i>	Azure Container Apps: The Future of Microservices in Azure <i>Geert van der Cruisen</i>
	RAUMWECHSEL			
11.45 - 12.45	Be Sharp with New Features in C# <i>Roland Guijt</i>	Event Storming: Kreativität trifft auf Struktur <i>Ina Einemann</i>	Entwicklung von ASP.NET Core Microservices mit Dapr: Eine praktische Anleitung <i>Marc Müller</i>	Azure Functions Deep Dive: Building Serverless Solutions for Scalable Applications <i>Michiel Hamers</i>
12.45 - 14.15	Mittagspause und Besuch der Fachmesse			
14.15 - 15.15	Writing (in)efficient C# <i>Callum Whyte</i>	Breaking Free: Kubernetes Unleashed for Small .NET Teams <i>Marius Boden</i>	Aus der Praxis: ASP.NET Core 8.0 WebAPI <i>Thorsten Kansy</i>	Typsicheres, polyglottes Messaging und Eventing: Stream/Queue/Topic of T <i>Clemens Vasters</i>
	RAUMWECHSEL			
15.30 - 16.30	Say goodbye to IConfiguration and embrace the power of the IOption pattern <i>Pieter Samyn</i>	Agil, aber richtig: Kundenzentrierung als Erfolgsfaktor für Teams <i>Miriam Hunger</i>	I followed the Microsoft docs, so my app is secure right? <i>Bart Lannoeye</i>	Versioniert, automatisiert und sicher - Optimale Bereitstellung von Azure-Ressourcen <i>Marc Müller</i>
16.30 - 17.00	Kaffeepause und Besuch der Fachmesse			
17.00 - 18.00	From C# to Rust <i>Sebastian Nilsson</i>	Looks GREAT To Me: Getting Past Bare Minimum Code Reviews <i>Adrienne Braganza Tacke</i>	To the Core: Migrating to ASP.NET Core Microservices <i>Iris Classon</i>	Container und WebAssembly: Verteilte Applikationen mit .NET Aspire orchestrieren <i>Thorsten Hans</i>
	RAUMWECHSEL			
18.15 - 19.15	Von Blazor zu WASI: Praktische Einblicke in WebAssembly mit .NET und C# <i>Oliver Sturm</i>	Maximale Effizienz: Automatisierte Releases in einer Multi-Team-Umgebung <i>Michael Kaufmann</i>	Containerisierung von .NET-Applikationen <i>Roland König</i>	Scaling your .NET app with Azure <i>Callum Whyte</i>
19.30 - 20.30	Being a Bad(ass) Developer <i>Iris Classon</i>			

DONNERSTAG, 28. NOVEMBER 2024: DEVSESSIONS

UHRZEIT	DEVSESSION 1	DEVSESSION 2	DEVSESSION 3	DEVSESSION 4
09.00 - 13.00	 Architecting and Development of Custom Copilots <i>Dr. Damir Dobric</i> Die DevSession richtet sich an professionelle Softwarearchitekten oder -entwickler, die gefordert sind, Anwendungen zu entwickeln, welche Large-Language-Modelle (LLM) nutzen. Dazu erfahren sie, wie LLMs und GenAI unter der Haube funktionieren. Wir beginnen mit den Grundlagen und befassen uns mit Token-Generierung, Vector-Similarity, Embeddings, verschiedene Modelltypen, Plugins, Agents usw. Sie lernen Microsofts Semantic Kernel kennen und wie Sie AI-Agents erstellen können, die intelligenten Code in jeder Anwendung über zahlreiche Demos ausführen, sowie Anwendungen und mächtige Backends, die AI in die Azure-Plattform integrieren.	 Testautomatisierung mit Playwright <i>Nico Orschel, Harald Binkle</i> „Testautomatisierung mit Playwright“ bietet eine umfassende Einführung in die Verwendung des Playwright-Frameworks zur Testautomatisierung, das die Entwicklung zuverlässiger und skalierbarer Testskripte für Webanwendungen ermöglicht. Die Teilnehmer lernen das Open-Source-Tool zu installieren, konfigurieren und Testskripte zu schreiben und auszuführen. Außerdem werden Best Practices für die Organisation und Pflege von Testfällen diskutiert. Am Ende werden die Teilnehmer in der Lage sein, Playwright effektiv einzusetzen, um automatisierte Tests für Webanwendungen zu erstellen und die Qualität ihrer Software zu verbessern.	 Mit .NET Aspire verteilte Anwendungen entwickeln <i>Christian Giesswein</i> In der modernen Softwareentwicklung ist die Entwicklung verteilter Anwendungen ein entscheidender Erfolgsfaktor. Daher tauchen wir in dieser DevSession tief in die Möglichkeiten ein, die .NET Aspire bietet, um robuste, skalierbare und performante verteilte Anwendungen zu erstellen. Sie erhalten Einblick in die hierfür notwendigen wichtigsten Konzepte und Werkzeuge. Dazu gehören unter anderem Architektur und Design verteilter Anwendungen, Kommunikationsprotokolle und -methoden sowie Datenmanagement und -synchronisation. Durch praktische Beispiele und Demos wird das theoretische Wissen direkt anwendbar gemacht.	 Hochqualitative Softwareentwicklung <i>David Tielke</i> Eine Software zu entwickeln, ist nicht besonders schwer. Eine hochqualitative Software zu entwickeln, erfordert allerdings einiges mehr. Wir schauen uns in dieser DevSession an, worauf Sie in Ihrer Softwareentwicklung achten müssen, was es in den Bereichen Anforderungen, Architektur, Entwicklung, Testen und Bereitstellung zu beachten gilt und wie auch aus Ihrer Entwicklung eine hochqualitative Software geschmiedet werden kann.
14.00 - 18.00	 Kleinteilig & Großartig - Micro Frontends mit Blazor <i>Dr. Florian Rapp</i> In den letzten Jahren hat sich das Architekturmuster der Micro Frontends zur Skalierung von Webprojekten etabliert. In dieser DevSession wollen wir gemeinsam eine Micro Frontend Lösung exklusiv in Blazor aufbauen - mit serverseitiger Zusammenstellung der Micro Frontends. Gezielt werden wir uns im Detail anschauen wie die Lösung aufgebaut ist und welche Entscheidungen dafür notwendig waren. Dabei wird der vom Monolith gewohnte Entwicklungskomfort (DX) einfach in die verteilte Welt übertragen - ohne die UX z. B. im Bereich der Performance zu verringern. Das Ziel der DevSession ist, dass einzelne Teams in eigenen Repositories verteilt an der vorgestellten Lösung weiterarbeiten und jegliche Entscheidung bzw. Deployments auch autonom vorgenommen werden können.	 Vom Monolith zum Modulith - Eine Fallstudie aus der Halbleiterindustrie <i>Hendrik Lösch</i> Monolithische Architekturansätze sind nicht sehr beliebt, da sie als komplex und träge gelten. Wir nehmen in der DevSession daher ein reales Projekt unter die Lupe, um zu sehen, welche Herausforderungen lauern, wenn ein solcher Monolith umstrukturiert wird, und wie das Architekturmuster des Moduliths helfen kann. Dazu werden wir uns das Team-Setup, die Migrationsstrategie und wichtige Detailentscheidungen ansehen. Hierbei klären wir das Vorgehen bei der Konzeptionierung, Grundlagen aus dem Domain Driven Design und die Besonderheiten des Productline Software Engineerings, das dabei hilft, gleichartige Softwareprodukte plattformgestützt weiter zu entwickeln. Ziel ist es, Muster und Verfahren kennenzulernen und zu bewerten und sich darüber auszutauschen.	 Azure Messaging - Streams, Queues und Pub Sub <i>Clemens Vasters</i> Azure Messaging Services - einschließlich Azure Service Bus, Event Grid, Event Hubs, Relay, Stream Analytics und Fabric Event Streams - bieten einige Tools für eine zuverlässige Nachrichtenübermittlung und ereignisgesteuerte Architektur. Die DevSession vermittelt ein klares Verständnis der Fähigkeiten der Dienste und wie sie integriert werden, um skalierbare, robuste Anwendungen zu erstellen. Dazu diskutieren wir den Auswahlprozess für das richtige Messaging-Tool auf Grundlage der Anforderungen Ihrer Anwendung anhand von Beispielen aus der Praxis. Der Fokus liegt auf praktischen Anleitungen und Best Practices für die Implementierung einer kohärenten Messaging-Strategie auf Azure. Zudem werden wir uns mit den neuesten Entwicklungen der einzelnen Produkte befassen.	 Deep dive in git <i>Michael Kaufmann</i> In dieser DevSession behandeln wir alle fortgeschrittenen Themen von Git, u.a.: Wie funktioniert Git? Was sind DAG, SHA, Patch und Commit? Was versteht man unter Zweigen und Tags? Wie löst man Merge-Konflikte? Wie manipulieren Sie Ihre Historie mit Commit amend, reset, rebase und cherry-pick? Wie arbeiten Sie mit dem Stash? Wie sucht man in Git? Zahlreiche praktische Tipps helfen bei der Arbeit mit Git: die richtige Konfiguration, Autokorrektur von Befehlen, Aufzeichnung von Merges und vieles anderes mehr. Der Workshop eignet sich für alle, die bereits über Grundkenntnisse in Git verfügen, und diejenigen, die neu in Git sind und nur Erfahrung mit anderen Versionskontrollsystemen haben. Voraussetzung sind eine lokale Git-Installation, ein Texteditor und ein kostenloses GitHub-Konto.

TIME-OUT

Einfach mal lossegeln

Immer alles nach Plan? Warum nicht mal dem Zufall mehr Raum geben, auch Unerwartetes zulassen – und bereit sein, neue Chancen zu sehen und zu ergreifen.



Time-out Start

Ich liebe es, Segeln zu gehen. Man bewegt sich an der frischen Luft fort, ohne dafür etwas tun zu müssen, liegt abends in einer einsamen Bucht oder mitten im Altstadthafen, und trotz aller Entspannung bleiben genügend Möglichkeiten, sich selbst mit Technik und (Crew-)Management beschäftigt zu halten.

Destination unknown

Für mich ist aber das Schönste daran, dass man sich ganz auf den Zufall einlassen kann. Am Morgen ist noch unklar, wo man am Abend sein wird. Die Planung ist wetterabhängig, und so sind es die Ziele. Sie ändern sich. Und weil die eigenen Erwartungen nicht maßgeblich sind, weil der innere Druck, genau dies oder jenes erleben zu müssen, nicht da ist, ist die Freude über das unerwartet Schöne besonders groß.

Das Ziel auf dem Plotter

So wie es Mac- und Windows-Fans gibt, so gibt es Segler und Motorbootfahrer. Segel oder Motor – das sind nicht nur zwei Antriebsarten, das sind zwei Lager, die nicht vereinbar sind. Es sind zwei Philosophien, zwei Lebenseinstellungen. Anders als der Segler überlässt der Motorbootfahrer nichts dem Zufall: Zu Urlaubsbeginn werden die Highlights aus dem Reiseführer markiert, mit Fullspeed nacheinander abgeklappert und am besten bereits Wochen vorher schon der schönste Tisch im Restaurant reserviert.

Klar – das sind Stereotype, aber die Art der Fortbewegung ist manchmal Ausdruck eines Mindsets: Will ich planen und Ziele erreichen – oder lasse ich mich auf das ein, was kommt, und freue mich (trotzdem) über alles, was ich erlebe.

Beides hat sein Für und Wider. Der eine erlebt die Highlights, ist bereits am Ziel, während die anderen noch unter-

wegs sind. Er ist aber auch massiv enttäuscht, wenn das Restaurant aus einem unvorhergesehenen Grund die besten Cevapcici Kroatiens gerade nicht vorrätig hat. Diese Enttäuschung kann dann alle bisherigen Freuden überlagern.

Ideale Arbeitsbedingungen

Warum ich das schreibe? Nun, ich erlebe bei manchen Mitarbeitern, dass sie eine genaue Vorstellung haben, wie das ideale Arbeiten auszusehen hat, und dann enttäuscht sind, wenn es mal ein paar Tage anders läuft. Wenn man nicht das macht, worauf man gerade am meisten Bock hat, sondern auch mal „beißen muss“. Wenn man nicht sofort am Ziel ist, sondern der Weg mal etwas länger erscheint. Oder wenn man nicht nach jeder Einzeltat den Dopaminstoß bekommt, den man sich bei TikTok anezogen hat.

Widewidewitt

Die Enttäuschung darüber, dass es nicht so läuft wie erwartet, wird genährt durch die Vorstellung, dass man sich seine eigene Welt schaffen kann.

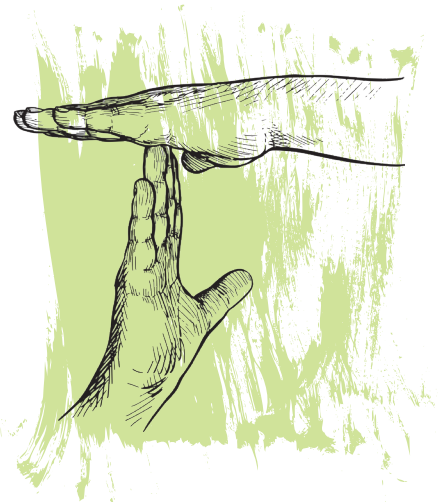
Für mich steckt dahinter die Idee, dass positives Denken das eigene Handeln und die Wirklichkeit um einen herum verändern kann. Dass man seines Glückes Schmied sei, dass ei-

nem alle (!) Möglichkeiten offenstehen, man muss nur den Motor anwerfen und die Route eingeben. Scheitern kommt nicht vor, genauso wenig wie Unwetter.

Am Ende entsteht Frust, weil sich der Plan nicht so hat verwirklichen lassen, wie man es sich vorgestellt hat.

Unabhängigkeit ist der Schlüssel

Werfen wir einen kleinen Blick zurück in die Geschichte: In den letzten Jahrhunderten waren die Möglichkeiten des Menschen sehr eingeschränkt und seine Handlungsoptionen meist



durch wirtschaftliche, kirchliche oder gesellschaftliche Rahmenbedingungen begrenzt. Die Wirklichkeit war vorgegeben, und man hatte sie anzunehmen. Das gilt heute noch für viele Bereiche der Erde. Die „Emanzipation“ der Frau konnte nur deshalb gelingen, weil sie in den westlichen Ländern und den letzten Jahrzehnten wirtschaftlich unabhängig wurde.

Selbstverwirklichung ist in vielen Teilen der Welt aber nach wie vor ein Fremdwort. Bei uns konnte sie da entstehen, wo Jugendliche nicht unbedingt gezwungen waren, sofort für ihren Lebensunterhalt aufzukommen. Das ist alles gut und wichtig so, nicht dass wir uns hier falsch verstehen. Aber es hat Auswirkungen auf die Erwartungshaltung.

Ende der 80er-Jahre kam über die Naturwissenschaften und die Psychologie ein weiterer Aspekt hinzu: Die Welt kann nicht nur gestaltet werden – sie wird in unserem Kopf durch unsere Wahrnehmung ganz individuell erfahren. Jeder lebt letzten Endes in seiner eigenen Welt. Die Welt ist so, wie wir sie wahrnehmen. Seit meiner Studienzeit nenne ich dies das „Zeitalter des Konstruktivismus“. Wikipedia schreibt dazu:

Konstruktivismus ist eine Theorie ... , die besagt, dass Menschen ihre eigene Realität durch individuelle Erfahrungen und Interpretationen der Welt konstruieren.

Astrid Lindgren lässt es ihre Pippi Langstrumpf auf den Punkt bringen: „Ich mach mir die Welt, wie sie mir gefällt.“

Kommen wir wieder auf Segler und Motorbootfahrer zurück, dann lassen sich die zwei Mindsets – stark vereinfacht – in etwa so beschreiben wie folgt:

Wirklichkeit folgt dem Plan

Beim Motorbootfahrer ist zuerst der Plan da. Die Wirklichkeit folgt dem Plan. Und sollte es einen Konflikt zwischen der erwarteten Wirklichkeit und der echten (oder erlebten) Wirklichkeit geben, ist die Enttäuschung groß. Da die Freude in der Planung bereits wie ein Aktienkurs eingepreist ist, der Mensch aber auf Verlust intensiver reagiert als auf Gewinn, ist ein großer Teil der Enttäuschung und Unzufriedenheit bereits im System vorprogrammiert.

Achtsamkeit als Gegenkonzept

Beim Segler entsteht die Wirklichkeit durch Wahrnehmung. Er muss Wetter und Wind beobachten. Man könnte Neudeutsch von Achtsamkeit sprechen. Achtsamkeit ist das gelebte Gegenteil von Konstruktivismus, sein Korrektiv, und vielleicht gerade deshalb so hype.

Der Segler lässt Spielraum für Unerwartetes. Er ist bereit für den Zufall, findet entweder einen Liegeplatz, der an Schönheit die Prospekte weit übertrifft, oder muss mit einem Restaurant in der dritten Reihe zufrieden sein.

Planung und Zufall unter einem Hut

Wie lassen sich die beiden Welten versöhnen? Wie kann man Planung und Zufall unter einen Hut bringen? Wie Frustr vermeiden und sich trotzdem keinem Fatalismus hingeben, sondern die Zukunft gestaltend angehen? Manager sprechen davon, dass die Steuerung des Erwartungshorizonts wichtiger

sei als die konkrete Performance. Wer beispielsweise nicht erwartet, dass er in seiner Beziehung jeden Tag bis über beide Ohren verliebt ist, wird vermutlich länger eine glückliche Beziehung haben.

Was wir vom Konstruktivismus lernen können: Das Glück entsteht nicht durch die reale Welt. Es entsteht durch die Art und Weise, wie wir die Welt wahrnehmen. Da reicht auch weniger aus, wenn wir glücklich sein wollen! Das ansteckende Lächeln am Morgen, eine nette Geste für einen Kollegen – das lässt auch manche Durststrecke besser überstehen.

Wir machen den Erfolg nicht alleine

Wir alle wissen, dass der Erfolg auch eine riesige Portion Zufall und Glück enthält. Auf der anderen Seite reicht der Zufall nicht aus. Wenn plötzlich eine Stelle frei wird, die mein Traumjob wäre, ich aber die Fortbildung dazu noch nicht abgeschlossen habe, dann wird meine Bewerbung ignoriert.

Wir sollten also unsere „Bewerbungsunterlagen“ beim Schicksal immer parat haben. Man muss Chancen sehen und sie ergreifen – vorbereitet sein für das Glück. Ohne es erzwingen zu wollen.

Als Segler beobachte ich den Wind. Und wenn sich eine Gelegenheit ergibt, wird das Segel gesetzt. Alles dazu ist vorbereitet. Und wenn es nicht so kommt, habe ich meinen Plan B. Mein Glück hängt aber nicht davon ab, wo ich lande. Sondern wie ich unterwegs bin. Wie ich den Weg bis dahin erlebe. Mein Glück entsteht, weil ich es will.

Ja, das ist vielleicht die positive Seite des Konstruktivismus: Die Einsicht, dass unsere Wahrnehmung das Glück macht, nicht die Wirklichkeit. Wir müssen Wirklichkeit nicht immer gestalten. Wir können auch mal lossegeln.

In diesem Sinne wünsche ich Ihnen allen eine gute Vorbereitung, viel Gelassenheit mit dem, was passiert, die Bereitschaft, die Chancen zu erkennen und dann auch etwas dafür zu tun. Und die Dankbarkeit über all das, was man bereits erlebt hat.

Mit diesem Mindset gelingt Happy Coding. ■



Time-out End



Dipl.-Theol. Bernhard Pichler

war 20 Jahre lang 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

A2411Timeout

SLEEPY-HOLLOW-ARCHITEKTUR

Den Kopf vom Körper trennen

Als Architekturmuster für Testbarkeit schließt Sleepy Hollow an die Anforderungsanalyse mit Slicing an und ist eine spezielle Interpretation der IODA-Architektur.

Automatisiertes Testen ist der Treiber für sauberen Code. Ja, es geht auch um Wandelbarkeit und Produktionseffizienz [1]. Aber wenn ich es ganz spitz auf einen Punkt bringen soll, dann sage ich: Es geht um Testbarkeit, Testbarkeit, Testbarkeit, also den Wert Korrektheit von Clean Code Development [1]. Alles andere folgt ganz natürlich aus der Testbarkeit, etwa Verständlichkeit und Modularisierung.

Und darüber liegt heute, dass KI der Treiber für automatisiertes Testen ist. KI ist also die Killerapplikation für Clean Code Development. Warum? Weil Sie ohne automatisierte Tests der KI nichts glauben dürfen. Code, den KI produziert, müssen Sie automatisiert testen, um zu beurteilen, ob die KI mit ihm die Spezifikationen in Ihren Prompts erfüllt.

```
1  const numbers = Deno.args.map(arg => {
2    |   return Number(arg);
3  });
4
5  const sum = numbers.reduce((acc, curr) => acc + curr, 0);
6  const average = sum / numbers.length;
7
8  console.log(`Durchschnitt: ${average.toFixed(2)}`);
```

Pure Logik für die Berechnung des Durchschnitts mehrerer Zahlen (**Bild 1**)

Bei Menschen als Zulieferern konnten wir uns noch darauf herausreden, dass die sicher ihren Code irgendwie getestet haben werden, bevor sie ihn uns gaben. Oder sie übergaben ihn uns in einem Dialog, und wir glaubten einfach, dass das schon alles korrekt sein würde. Das war immer schon sehr dünnes Eis, auf dem wir uns bewegt haben, doch es fühlte sich noch plausibel an.

Mit der KI ziehen diese „Ausreden“ nicht mehr als Argument, auf automatisierte Tests zu verzichten. Weder hat die KI für sich den Code, den sie uns präsentiert, in irgendeiner Weise überprüft; noch übergibt sie ihn uns in einem Gespräch voller vertrauensheischender Formulierungen. Das müssten wir erst im Chat anstoßen, und es würde Mühe machen. Genauso macht es Mühe, den KI-Code durch Augenscheinnahme zu überprüfen, ganz davon zu schweigen, dass das unzureichend wäre.

Nein, KI-generierter Code muss automatisiert überprüft werden. Dafür braucht es Testbarkeit. Und Testbarkeit ist Teil von und führt zu Clean Code Development.

Testbarkeit

Testbarkeit, also das Potenzial, automatisierte Tests möglichst einfach überhaupt an Code anlegen zu können, setzt Schnittstellen voraus. Das, was getestet werden soll, ist Logik [2]. Logik sind die Anweisungen im Code, die selbst Verhalten herstellen: Ausdrücke, Kontrollflusssteuerungen, API-Zugriffe. **Bild 1** zeigt beispielhaft die Logik für die Berechnung des Durchschnitts von Zahlen, die einer Anwendung bei deren Start auf der Kommandozeile mitgegeben werden.

Funktioniert diese Logik, ist sie korrekt? Wie kann ich das feststellen? Ich kann sie ausführen und schauen, ob die Ausgabe mit meinen Erwartungen übereinstimmt. Doch das skaliert nicht; auf diese Weise kann ich nicht systematisch viele Tests wiederholt verlässlich ausführen. Außerdem fehlt jeder Nachweis, dass und was ich überhaupt getestet habe.

Solchen Code kann ich nicht wirklich als testbar bezeichnen. Ihm fehlt eine Schnittstelle, auf die ich automatisierte Tests ansetzen kann. Im einfachsten Fall ist das eine Funktion wie in **Bild 2** gezeigt. Diese Funktion sollte so viel Logik wie irgend möglich enthalten beziehungsweise als Testansatzpunkt repräsentieren. Im Beispiel habe ich mich deshalb dafür entschieden, sogar die Konvertierung der Zeichenketten hinter die Testfassade zu stellen.

Level 1 sauberer Programmierung ist erreicht: formale Testbarkeit durch einen „Testpunkt“ (im Vergleich zu einem Service-Endpunkt). Aber was ist mit den nächsten Levels?

Wandelbarkeit

Warum zieht Testbarkeit andere saubere Eigenschaften von Code nach sich? Wie fördert Testbarkeit die Wandelbarkeit?

Für Testbarkeit braucht es erstens Schnittstellen, auf die Tests überhaupt angesetzt werden können. Schnittstellen werden von Modulen angeboten. Sie fassen Logik in Funktionen zusammen, von denen sie manche in ihrer Schnittstelle veröffentlichen und manche nicht. Veröffentlichte Funktionen sind direkt testbar, private Funktionen nur indirekt via veröffentlichte. `parseArgs` in **Bild 3** ist eine solche schlecht testbare, weil private Funktion.

```

_> tests > TS average_test.ts > TypeScript > Deno.test("CalcAverage - Normaler Fall mit negativen und positiven Zahlen", () => {
1  import { assertEquals } from "https://deno.land/std@0.97.0/testing/asserts.ts";
2  import { CalcAverage } from "../average.ts"; // Pfad anpassen, falls nötig
3
4  ▶ Run Test | Debug
5  Deno.test("CalcAverage - Normaler Fall mit positiven Zahlen", () => {
6    const result = CalcAverage(["1", "2", "3", "4", "5"]);
7    assertEquals(result, 3);
8  });
9
10 ▶ Run Test | Debug
11 Deno.test("CalcAverage - Normaler Fall mit negativen und positiven Zahlen", () => {
12   ⚡const result = CalcAverage(["-1", "0", "1"]);
13   assertEquals(result, 0);
14 });
15
16 ▶ Run Test | Debug
17 Deno.test("CalcAverage - Einzelne Zahl", () => {
18   const result = CalcAverage(["10"]);
19   assertEquals(result, 10);
20 });
21
22 ▶ Run Test | Debug
23 Deno.test("CalcAverage - Dezimalzahlen", () => {
24   const result = CalcAverage(["1.5", "2.5", "3.5"]);
25 });

```

Testbare Logik mit
automatisierten
Tests (Bild 2)

zu simulieren, um zu verstehen, was sie tut (beziehungsweise tun soll), lesen Sie das aus ihrem Namenspfad ab.

Dazu kommt, dass Logik viel weniger (oder gar nicht mehr) voneinander abhängig ist. Ohne Tiefenstaffelung über funktionale Abhängigkeiten liegt zu simulierende Logik komplett vor Ihnen, wenn Sie eine Funktion anschauen. Kein Debugging mit *step-into* ist mehr nötig, keine Sprünge in immer größere Tiefen des Quellcodes in der IDE sind erforderlich, um zu verfolgen, was passiert, wenn eine Funktion aufgerufen wird. Auf diese Weise wird auch das Prinzip des Single Level of Abstraction (SLA) quasi automatisch bedient.

Zuallererst jedoch sehe ich die Verständlichkeit durch einen veränderten Umgang mit Anforderungen gefördert: Wenn die Testbarkeit im Vordergrund steht, geht es um Anforderungen. Tests sind formalisierte Anforderungen, deren Erfüllung sie automatisch überprüfen (siehe rechts unten in Bild 2). Für diese formalisierten (oder auch informelle) Anforderungen stellt sich die Frage: Welchen Scope können Sie eigentlich klar und lückenlos verstehen, umsetzen und mit Tests abdecken?

Wenn Sie KI für die Codegenerierung benutzen, werden Sie schnell merken, dass Anforderungen sehr überschaubar sind, die diese Eigenschaften haben. Sie geben der KI keine zehn Seiten Anforderungsdefinition und warten entspannt auf Code, der sie erfüllt und dem Sie einfach vertrauen. Nein, Sie werden schon intuitiv die Anforderungen recht fein schneiden und sich die Generierung von überschaubaren Funktionen und Modulen wünschen – für die dann auch die Tests überschaubar sind.

Der Fokus auf solide Anforderungen, der durch das Primat der Testbarkeit entsteht, führt zu kleineren Scopes für kleinere Funktionseinheiten, die dann auch verständlicher sind.

Level 3 sauberer Programmierung ist erreicht. Konzeptionell jedenfalls. Denn es stellt sich die Frage: Wie sieht das praktisch aus?

Sleepy Hollow – ein Architekturmuster für Testbarkeit

SoC, SRP, SLA, DIP ... all diese Prinzipien mögen überwältigend klingen. Wie sollen Sie ihnen gerecht werden? Wann ist „sauberer Code“ mit seinen Vorteilen für die langfristige Produktivität erreicht?

Genau genommen ist das wirklich nicht so einfach. Wer es auf Perfektion abgesehen hat, wird schnell frustriert und/oder gerät in Konflikte mit anderen, die es anders sehen.

Um das zu vermeiden, möchte ich Ihnen das Sleepy-Hollow-Architekturmuster vorstellen. Es schließt an die Anforderungsanalyse mit Slicing (siehe die dotnetpro-Artikel [4] bis [5]) an und ist eine spezielle Interpretation der IODA-Architektur [6]. Inspiriert ist die Bezeichnung „Sleepy Hollow“ durch den gleichnamigen Spielfilm von Tim Burton. In dem

```
1 export function CalcAverage(args:string[]):number {
2   const numbers = parseArgs(args);
3
4   const sum = numbers.reduce((acc, curr) => acc + curr, 0);
5   const average = sum / numbers.length;
6
7   return average;
8 }
9
10 function parseArgs(args:string[]):number[] {
11   return args.map(arg => Number(arg));
12 }
13
14 console.log(`Durchschnitt: ${CalcAverage(Deno.args).toFixed(2)}`);
```

Nur öffentliche Funktionen sind direkt testbar (Bild 3)

bringt ein kopfloser Reiter Unheil über ein kleines Dorf an der Ostküste der USA im Jahr 1799. Anders als sonst spielt der Kopf eines Protagonisten nur eine geringe Rolle, sein Körper hingegen eine umso größere. Er treibt die Handlung; wo er ist, ist Action. Kopf und Körper gehören zusammen; doch das Verhalten steckt im Körper.

Das ist aus meiner Sicht eine zu suchende Grundstruktur für Software: Kopf und Körper sind zu trennen. Das, was das Verhalten erzeugt, ist von dem, wo die Interaktion stattfindet, zu separieren.

Bild 4 zeigt diese Struktur in ihrer einfachsten Form. Der Body ist der Bereich, in dem „die Musik spielt“. Der Body umfasst den testwürdigen und vor allem den testbaren Code. Testbar ist Code für mich, wenn die Logik von keiner User-Interface-Technologie getrübt wird.

Es gibt zwar Werkzeuge, mit denen man das UI und durch das UI testen kann, doch die finde ich vergleichsweise unständig, um damit wirklich mehr als die Bedienung zu testen. Für Integrationstests taugen sie nach meinem Dafürhalten nicht.

Logik, die von Persistenz- oder Netzwerktechnologie abhängig ist, ist zwar auch nicht ganz leicht zu testen, doch die Automatisierung fällt hier deutlich leichter als im User Interface. Die Schnittstellen sind stabiler.

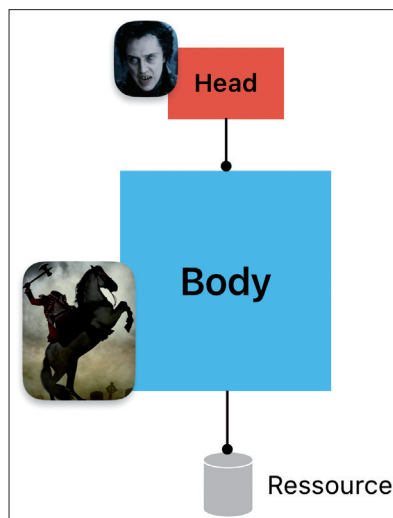
Wenn Testbarkeit der Treiber für sauberen Code ist, dann ist diese Unterteilung des Codes der erste und fundamentale Schritt hin zu mehr Sauberkeit. Mit ihm wird eine klare Grenze gezogen, wo Tests angelegt werden sollen und können: am Body.

Die dortigen Tests sind notwendig Integrationstests. Mit ihnen wird das Zusammenspiel von Domäne und Providern überprüft. Sie sind für mich Akzeptanztests auf sehr hohem Niveau.

In Bild 5 sehen Sie, wie typische Module eines Softwaresystems auf Head und

Body verteilt werden:

- Der Body umfasst die Logik für die Fachlichkeit (Domäne), den Zugriff auf Ressourcen (Provider) und potenziell weitere, orthogonale Aspekte (zum Beispiel Logging).



Das Sleepy-Hollow-Architekturmuster (Bild 4)

- Außerdem ist im Body auch UI-nahe Logik zu finden, die nicht von UI-Technologien abhängig ist.
- Der Head versammelt die Logik, die eng mit User-Interface-Technologien verzahnt ist.

Im Grunde ist das schon alles Wesentliche von Sleepy Hollow. Auf diese Weise wird ein Maximum an Logik relativ einfach automatisierbar testbar. Nur eine Schnittstelle ist nötig: die des Bodys. Durch sie laufen alle High-Level-Integrationstests: die Akzeptanztests.

Akzeptanztests nenne ich sie, weil sie aus der Perspektive von Benutzern formuliert sind. Das einzige Zugeständnis an die Testbarkeit ist, dass sie nicht auf dem Frontend ausgeführt werden. Doch ansonsten simulieren sie, wie Anwender das Softwaresystem nutzen würden. Es sind also keine technischen Tests, und es sind auch keine Tests von Details.

Auf dem Niveau von Akzeptanztests ist keine hochprozentige Pfadabdeckung der Logik zu erwarten. Ihre Aufgabe ist nicht, alles bis ins Kleinste zu prüfen. Vielmehr sollen sie „nur“ sicherstellen, dass der verhaltensrelevante Teil des Softwaresystems korrekt zusammengesetzt ist und grundsätzlich seinen Job erledigen kann, vom UI bis zurück zum UI und möglichst End-to-End.

Bitte beachten Sie die Terminologie von Sleepy Hollow! Ich spreche nicht von Frontend und Backend, und auch nicht von Client und Server. Es geht nicht um Verteilung über Prozessgrenzen hinweg. Diese Verteilung ist unabhängig von der Head-Body-Struktur.

Bild 6 macht das deutlich. Sie sehen dort sogar eine geschachtelte Anwendung des Architekturmusters.

- Einerseits sind Teile des Clients und der gesamte Server in einem Body zusammengefasst.
- Andererseits existieren getrennte Bodies für Client und Server.

Das Muster ist: Wo immer ein User Interface auftaucht – in **Bild 6** ist das sowohl im Client als auch im Server der Fall –, ist das „vom Rest“ zu trennen, um „den Rest“ testbar zu machen.

Die Schnittstelle des Bodys

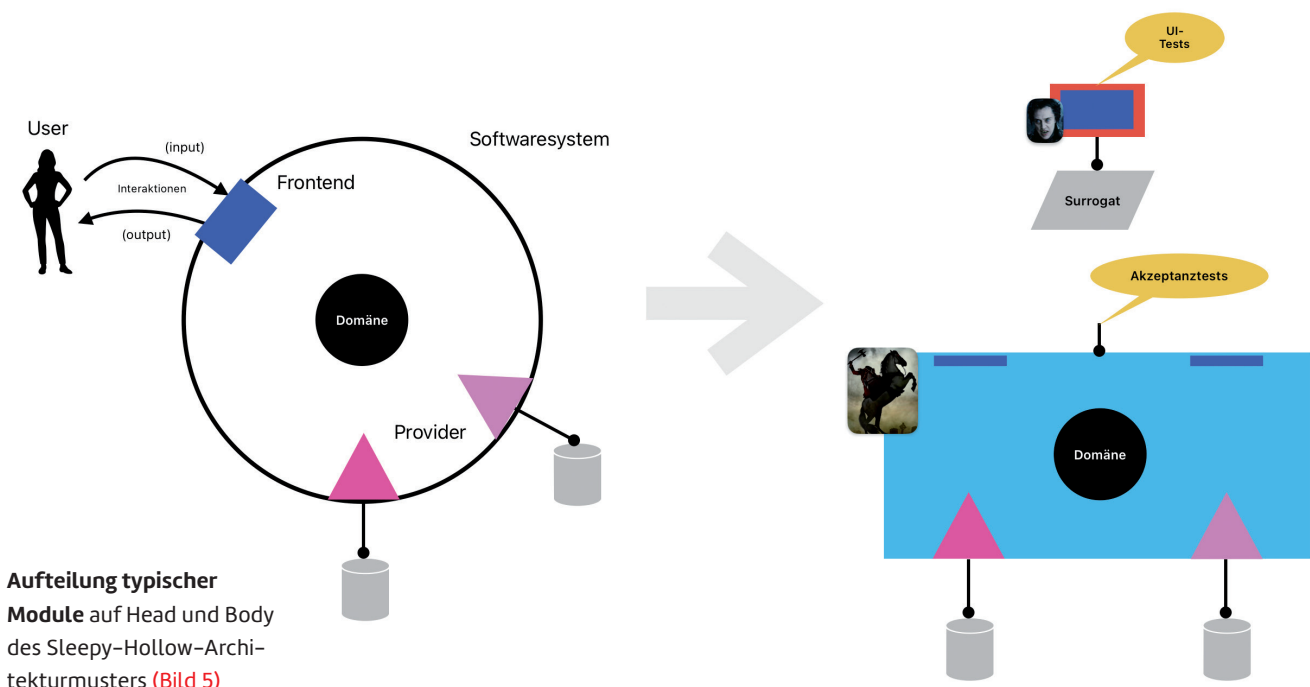
Wie funktioniert die Trennung zwischen Head und Body? Mit einem Modul mit klarer Schnittstelle. An dieser Schnittstelle veröffentlicht der Body seine Leistungsangebote, die über ein User Interface angesteuert werden können.

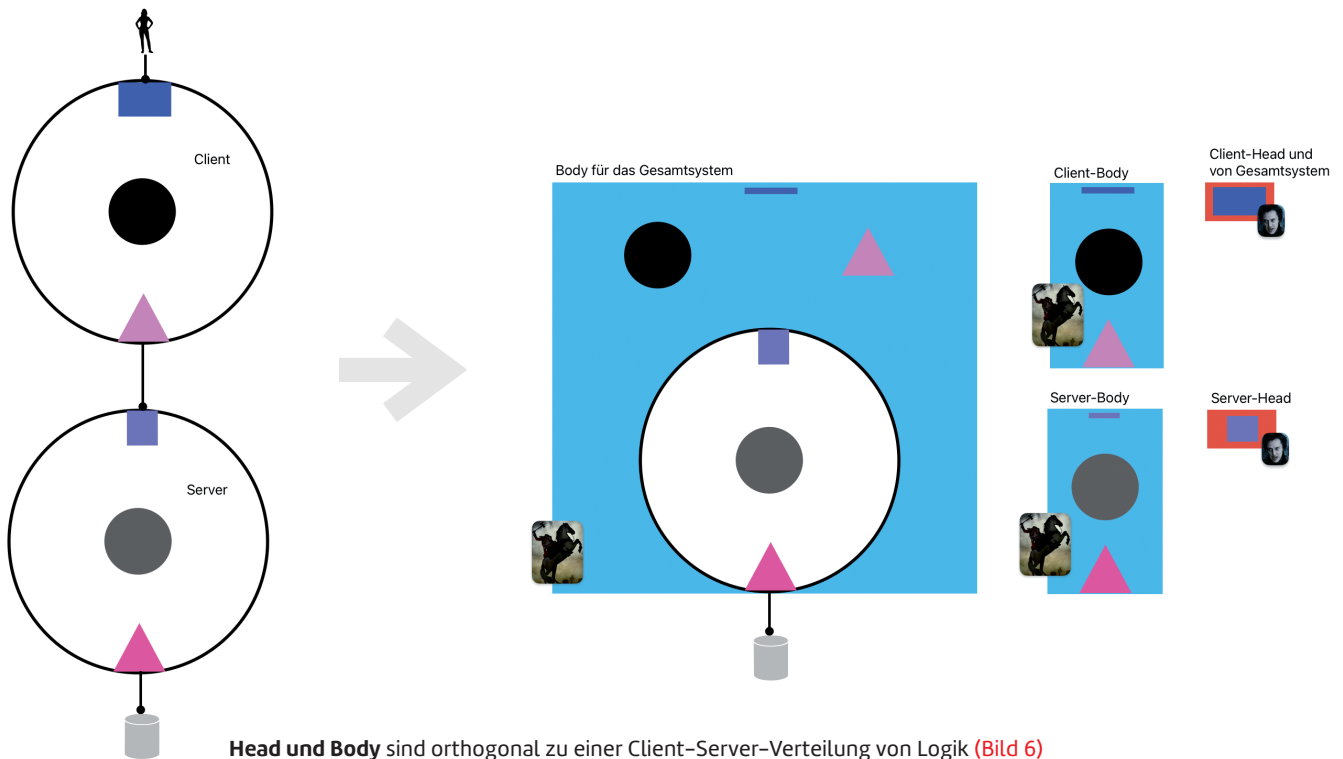
Der Body ist für jedes User Interface eine Blackbox mit einer Fassade. Vor diese Fassade können ganz verschiedene User Interfaces gesetzt werden. Die Blackbox ist UI-Technologie-agnostisch. Darin liegt der Sinn dieser Trennung: alle Abhängigkeiten von UI-Technologie diesseits der Fassade. Jenseits der Fassade gibt es zwar weitere Technologieabhängigkeiten, doch die sind leichter in automatisierten Tests zu beherrschen.

Woher kommen die Funktionen an der Schnittstelle eines Bodys? Denn Funktionen sind Leistungsangebote, weil sie Logik enthalten oder Logik in anderen Funktionen aufrufen, die das Verhalten herstellt, das die Leistung eines Bodys darstellt. Bodies sind Dienstleister für User vermittelt User Interfaces.

Für mich gibt es nur einen Weg zur Schnittstelle des Bodys: eine systematische Anforderungsanalyse mit Slicing (siehe [4] bis [5]), deren Ergebnis eine Liste von Entry-Point-Funktionen ist. Sie werden durch Interaktionen von Anwendern mit dem Softwaresystem über das User Interface getriggert. Das User Interface ist dazu da, Anwendern den Aufruf dieser Funktionen möglich zu machen.

User Interface steht dabei nicht nur für eine WPF-Oberfläche oder eine HTML-Seite, sondern auch zum Beispiel für HTTP-Endpunkte. User-Interface-Logik ist die Logik, die ▶





„weiß“, welche Technologie für die Interaktionen eingesetzt wird. Sie ist eng verzahnt etwa mit WPF oder ASP.NET Core MVC. Sie ist schwer testbar – und deshalb ist sie in einem Head zu konzentrieren und von weiterer Logik zu trennen, dem Body. Auch das ist eine Anwendung des Prinzips SoC.

Als triviales Beispiel soll noch einmal die Berechnung des Durchschnitts einer Reihe von Zahlen dienen. Welche Inter-

aktionen gibt es bei der Anwendung? Nur eine: den Programmstart. Bei dem werden die zu verrechnenden Werte auf der Kommandozeile gleich mitgegeben. Das Ergebnis erscheint als Ausgabe auf der Konsole.

Als UI-Technologien kommen Zugriff auf Kommandozeile und Zugriff auf Konsole zum Einsatz. Beides gehört in den Head. „Der Rest“ wird im Body zusammengefasst. Dessen Schnittstelle hat nur eine Funktion, weil es auch nur eine Interaktion gibt. In Bild 7 trenne ich beide Aspekte und entkopple auch noch durch ein Interface, um den Head selbst etwas testbarer zu machen.

Technisch ist diese Trennung wahrlich kein Hexenwerk. Der Trick liegt im Anlass für sie: die konsequente Entflechtung von User-Interface-Technologie und „dem Rest“ zum Zweck der Testbarkeit auf sehr hohem Abstraktionsniveau.

Diese Trennung können Sie selbst dann vornehmen, wenn Ihnen sonst keine Module einfallen. Domäne, Model, Provider, Adapter oder was sonst noch so an Begriffen in Architekturmodellen herumschwirren mag, können Ihnen (zunächst) egal sein. Tun Sie sich den Gefallen und ziehen Sie wenigstens Head-Logik und Body-Logik auseinander.

In Bild 7 habe ich rot eingefärbt, wo Logik von User-Interface-Technologie abgängig ist. Grün hinterlegt ist der Aufruf des Bodys.

Diskussionswürdig könnte sein, dass ich die Umwandlung der Zahlen auf der Kommandozeile in Zeile 5 komplett im Head gelassen habe. Wäre nicht auch das Mapping testwürdig und sollte deshalb in den Body wandern? Dafür spräche das Mehr an gut testbarer Logik. Dagegen spricht, dass Zahlen in Form eines String-Arrays eigentlich zur Eigenart des Konsole-User-Interface gehören. Wäre das User Interface ein HTTP-Controller, dessen Endpunkt die Zahlen als JSON-Ob-

```

1  class Head {
2    constructor(private body: IBody) {}
3
4    run() {
5      const numbers = Deno.args.map(arg => Number(arg));
6
7      const average = this.body.calculate(numbers);
8
9      console.log(`Durchschnitt: ${average.toFixed(2)}`);
10   }
11 }
12
13 1 implementation
14  interface IBody {
15    calculate(numbers: number[]): number;
16  }
17
18  class AverageCalculationBody implements IBody {
19    calculate(numbers: number[]): number {
20      const sum = numbers.reduce((acc, curr) => acc + curr, 0);
21      const average = sum / numbers.length;
22
23      return average;
24    }
25  }
26
27  const body = new AverageCalculationBody();
28  const head = new Head(body);
29  head.run();

```

Logik sauber in Head und Body getrennt (Bild 7)

jekt übergeben werden, würde die Umwandlung anders aussehen. Das zeigt **Bild 8** anhand eines *val.town*-Controller-Skripts, das über einen URL direkt mittels *POST* und ein JSON-Array mit Zahlen aufgerufen werden kann.

Die Schnittstelle des Bodys ist unverändert. Die Sammlung der zu verrechnenden Zahlen erfolgt allerdings anders als beim Konsolenprogramm – wie natürlich auch die Projektion des Ergebnisses für den Anwender.

Sie spüren jetzt hoffentlich schon etwas von dem Vorteil, den das Sleepy-Hollow-Architekturmuster bringt: Es entkoppelt sauber zwei sehr unterschiedliche Aspekte von Software. Vor der Fassade kann das UI wechseln, hinter der Fassade bleibt der Code unverändert. Nicht nur die Testbarkeit wird gefördert, sondern auch die Wandelbarkeit.

Und wie in **Bild 6** gezeigt, könnte jetzt sogar der *val.town*-Server mit dem Endpunkt aus **Bild 8** in einen neuen Body wandern, zu dem weiterhin ein Console-Head gehört (**Bild 9**).

Der Body ist jetzt zwar nur ein Adapter, aber das Muster wurde eingehalten. Diese Ausprägung des Bodys ist gut testbar, selbst wenn das bedeutet, einen Server aufzurufen. In Integrationstests müssen nicht immer alle Ressourcen „abgeklemmt“ und durch Surrogate ersetzt werden.

Die User-Interface-Pipeline

Die Schnittstelle des Bodys macht unabhängig von Frontend-Technologien und abstrahiert die technologischen Details innerhalb des Bodys:

```
1 export default async function(req: Request): Promise<Response> {
2   const numbers = await req.json();
3
4   const body: IBody = new AverageCalculationBody();
5   const average = body.calculate(numbers);
6
7   return new Response(JSON.stringify({ average: average }), {
8     status: 200,
9     headers: { "Content-Type": "application/json" },
10  });
11 }
```

Ein **HTTP-Controller** als alternatives User Interface erhält die zu verrechnenden Zahlen als JSON-Array (**Bild 8**)

- Im Body gibt es zum Beispiel keine Kenntnis darüber, woher die zu verrechnenden Zahlen kommen.
- Im Head gibt es zum Beispiel keine Kenntnis darüber, ob die Durchschnittsberechnung lokal oder verteilt stattfindet.

In **Bild 4** habe ich dieses Verhältnis von Head und Body mit einer Abhängigkeit eingeführt; dafür steht die Linie mit dem Punkt am Ende (statt eines sonst üblichen Pfeils für Abhängigkeiten). In **Bild 10** sehen Sie, wie diese Abhängigkeit im Code des Console-Head und des Controller-Head aussieht: Sie führt zum

```
1 class Head {
2   constructor(private body: IBody){}
3
4   async run() {
5     const numbers = Deno.args.map(arg => Number(arg));
6
7     const average = await this.body.calculate(numbers);
8
9     console.log(`Durchschnitt: ${average.toFixed(2)}`);
10  }
11 }
12
13 1 implementation
14 interface IBody {
15   calculate(numbers: number[]): Promise<number>;
16 }
17
18 class HTTPBody implements IBody {
19   private readonly url = "https://r[REDACTED].web.val.run";
20
21   async calculate(numbers: number[]): Promise<number> {
22     const response = await fetch(this.url, {
23       method: 'POST',
24       headers: {'Content-Type': 'application/json'},
25       body: JSON.stringify(numbers),
26     });
27
28     const result = await response.json();
29
30     return result.average;
31   }
32 }
33
34 const body = new HTTPBody();
35 const head = new Head(body);
36 await head.run();
```

Ein alternativer Body, der die Verarbeitung an den *val.town*-Controller delegiert (**Bild 9**)

Aufruf einer Funktion in der Schnittstelle des Bodys (grün). Trotz unterschiedlicher UI-Technologien ist das in beiden Head-Implementationen gleich.

Ebenfalls gleich ist, was vor und nach diesem Aufruf passiert. Es ist nicht derselbe Code, doch derselbe Zweck wird für beide Implementationen erfüllt. Vor und nach dem Aufruf des Bodys steht Code, der abhängig von der UI-Technologie ist und eine Klammerfunktion hat. *run()* (links) und der HTTP-Controller-Endpunkt (rechts) werden im Rahmen einer Interaktion getriggert. Diese bekommt Input und liefert Output. Das macht die Reaktion auf einen „Reiz“ aus (Programmaufruf auf der Kommandozeile beziehungsweise HTTP-Request an einen Server). *run()* und die Controller-Funktion sind Eventhandler der Interaktion; mit ihnen tritt die Kontrolle ins Softwaresystem ein.

Wie Eventhandler für Interaktionen aussehen, ist abhängig von der User-Interface-Technologie, wie **Bild 10** zeigt. Was ►

```

class Head {
  constructor(private body: IBody){}

  run() {
    const numbers = Deno.args.map(arg => Number(arg));
    const average = this.body.calculate(numbers);
    console.log(`Durchschnitt: ${average.toFixed(2)}`);
  }
}

```

```

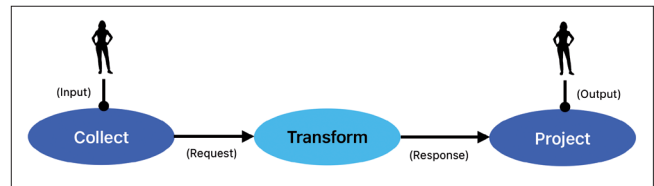
1 export default async function(req: Request): Promise<Response> {
2   const numbers = await req.json();
3
4   const body: IBody = new AverageCalculationBody();
5   const average = body.calculate(numbers);
6
7   return new Response(JSON.stringify({ average: average }), {
8     status: 200,
9     headers: { "Content-Type": "application/json" },
10  });
11 }

```

Zwei Head-Implementationen für denselben Body im Vergleich (**Bild 10**)

in Eventhandlern passiert, ist jedoch immer gleich – nur wie es genau passiert, ist wiederum abhängig von der UI-Technologie. Für jede Interaktion sieht der Prozess so aus:

- Zuerst wird der Input des Anwenders gesammelt; das nenne ich die Collection-Phase.
- Dann wird der Body aufgerufen.
- Schließlich wird das Ergebnis des Bodys in Output für den Anwender verwandelt; das nenne ich die Projection-Phase.



Die grundlegende User-Interface-Pipeline (Bild 11)

Sammeln, transformieren, projizieren: Das ist die grundlegende Pipeline, die Input bei jeder Interaktion durchläuft (Bild 11). Sie entspricht dem uralten EVA-Prinzip: Eingabe–Verarbeitung–Ausgabe.

Collection und Projection sind technologiespezifisch. Sie müssen deshalb im Head verbleiben. Sie sind von Head zu Head verschieden.

Der Benutzer liefert „irgendwie“ Daten beim Head an, zum Beispiel auf der Kommandozeile oder in einem Web-Formular oder per HTTP-Aufruf. Wie das geschieht, hängt von der UI-Technologie ab. Damit der Body damit etwas anfangen kann, müssen die Daten „gesammelt“ werden, zum Beispiel durch ein Mapping von Kommandozeilenparametern oder Parsing einer HTTP-JSON-Payload. Das Ergebnis ist ein Request für den Body, der den Input von allen Details des User Interface befreit erhält.

Auf dem Rückweg vom Body ist dasselbe mit dessen Response zu tun. „Projektion“ nenne ich die Transformation in UI-spezifischen Output, weil unter Umständen damit eine mehr oder weniger grafische Darstellung einhergeht. Aber auch die Umwandlung in eine HTTP-Response wie in Bild 10 rechts unten ist eine Projektion.

Von außen gesehen ist damit alles getan, um den Body testbar zu machen. Sleepy Hollow besteht nicht nur aus einer modularen Trennung von Logik (Head versus Body), sondern auch aus einer funktionalen im Sinne eines Datenfluss-Prozesses: sammeln, transformieren, projizieren.

Doch damit nicht genug ...

Die Body-Pipeline

In Bild 5 sehen Sie, dass ich einen Teil des User Interface in den Body übernommen habe. Warum das? Natürlich der Testbarkeit wegen.

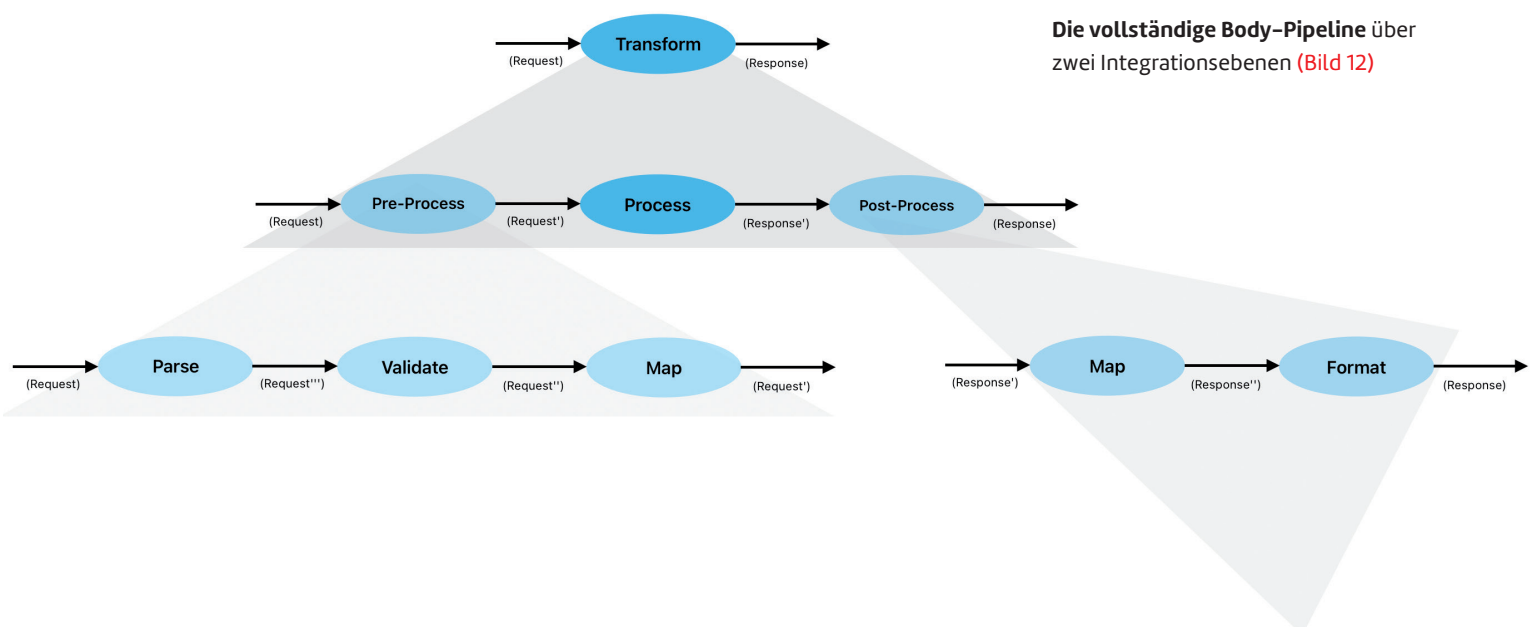
Sammlung und Projektion sind die Anteile des User Interface, die schlecht zu testen sind, weil sie mit UI-Technologie verzahnt sind.

Doch bevor Domäne oder Provider das für eine Interaktion gewünschte Verhalten herstellen können, mag es nötig sein, den eingehenden Request noch etwas „zu kneten“. Und um die Arbeit von Domäne und Providern wirklich unabhängig von der Umwelt zu machen, das heißt sogar von spezifischen Interaktionen, kann es nötig sein, eine Response von dort am Ende auch noch etwas „zu kneten“.

Für mich gibt es daher innerhalb des Bodys um die eigentliche Arbeit herum noch ein Pre-Processing und Post-Processing (Bild 12):

- Pre-Processing vor der Verhaltensgenerierung
 - Parsing
 - Validation
 - Mapping
- Post-Processing nach der Verhaltensgenerierung
 - Mapping
 - Formatting

Keiner dieser Schritte muss stattfinden, aber alle könnten stattfinden. Es hängt einfach davon ab, wie abstrakt Sie Domäne und Provider in Ihren Schnittstellen auslegen.



Die vollständige Body-Pipeline über zwei Integrationsebenen (Bild 12)

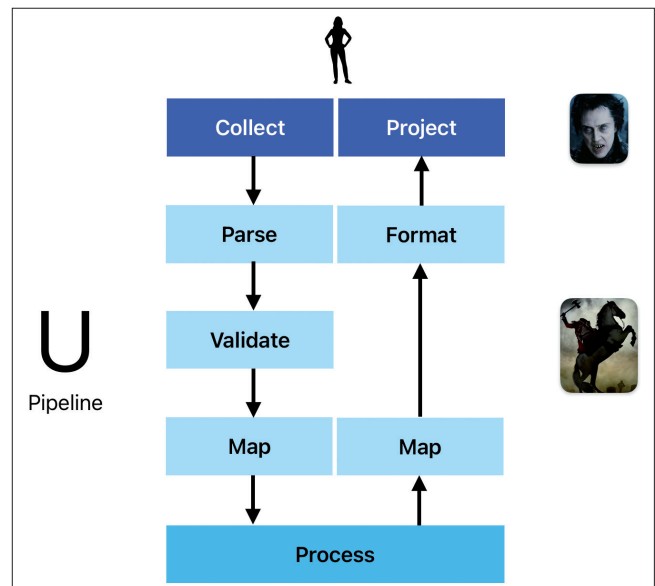
Pre- und Post-Processing sind im Body, weil sie auf keiner UI-Technologie basieren. Damit sind sie grundsätzlich gut testbar.

Das, was in Bild 11 noch ein Prozessschritt war – die Transformation von Request zu Response durch den Body –, ist in Bild 12 aufgebrochen und verfeinert. Die Transformation ist nicht monolithisch, sondern wiederum ein Prozess in Form eines Datenflusses. Die Reihenfolge der Verarbeitungsschritte liegt auf der Hand: Sie läuft vom rohen Input über dessen Zuordnung für die Domäne über die zentrale Verarbeitung weiter zur Aufbereitung des Ergebnisses in Form eines ansprechenden Outputs. In Bild 13 sehen Sie alle Schritte auf der untersten Ebene ausgehend vom Benutzer und wieder zurück zum Benutzer: Die Daten aus der Umwelt fließen in einem „U“ durch das Softwaresystem; dabei wird Input in Output verwandelt.

Jeder Verarbeitungsschritt des Bodys in diesem Datenfluss kann für sich gut getestet werden, und für alle zusammen gilt das ebenfalls.

- Kein Verarbeitungsschritt im Body ist von UI-Technologie abhängig.
- Kein Verarbeitungsschritt ist von einem anderen abhängig.

Auch wenn in Bild 13 die Schritte übereinander angeordnet sind, um anzudeuten, dass hier die Verarbeitung unter die Oberfläche der Software geht, sind die Schritte mit Pfeilen verbunden. Das bedeutet, es fließen Daten zwischen ihnen, ohne dass die Schritte sich kennen müssten. Diese „Verschaltung“ folgt dem Principle of Mutual Oblivion (PoMO) [7], ausführlicher entwickelt in [8]. Deshalb kann jeder Schritt für

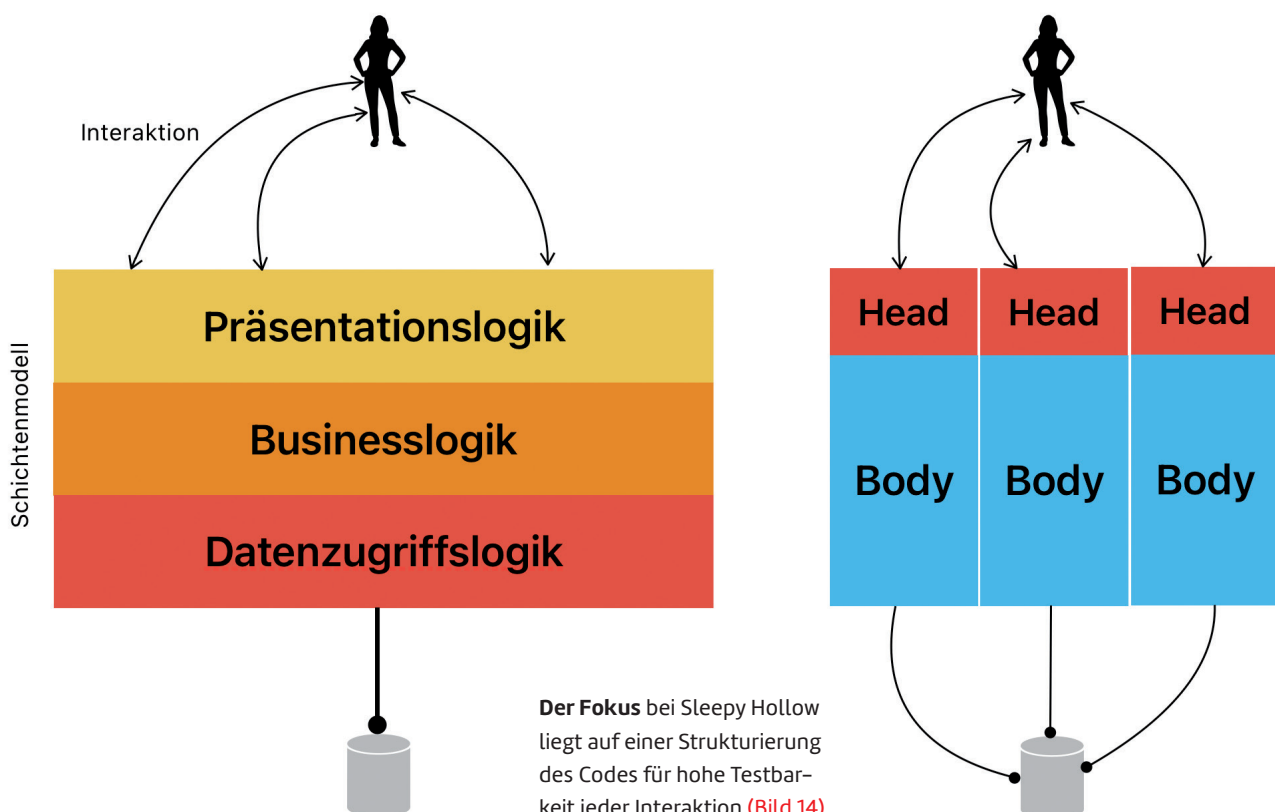


Der Datenfluss vom Benutzer durch das Softwaresystem zurück zum Benutzer (Bild 13)

sich allein getestet werden – oder alle zusammen unter dem Dach von Transform der Interaktion.

Vertical Slice Architecture

In Bild 4 mag das Sleepy-Hollow-Architekturmuster noch wie eine einfachere Form der Schichtenarchitektur aussehen. Sind dort nicht nur Businesslogik und Datenzugriff im Body zusammengefasst, und die Präsentationslogik ist zum Kopf ►



Der Fokus bei Sleepy Hollow liegt auf einer Strukturierung des Codes für hohe Testbarkeit jeder Interaktion (Bild 14)

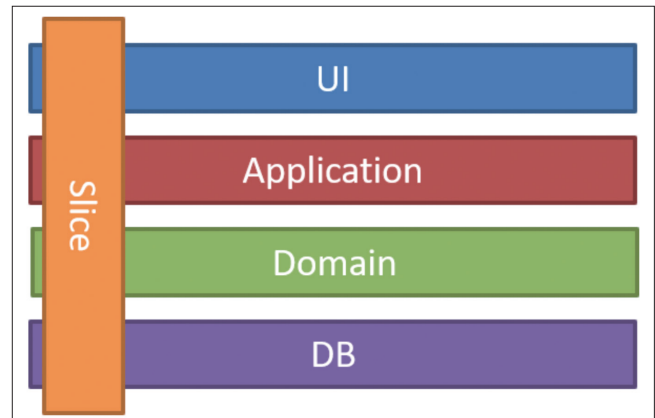
ernannt? Nein, das wäre aus mehreren Gründen ein Missverständnis:

- Zum einen sind Teile dessen, was im Schichtenmodell Präsentationslogik genannt wird, im Body (Pre- und Post-Processing).
- Zum anderen ist das Motiv für Sleepy Hollow ein anderes als das hinter dem Schichtenmodell. Das Schichtenmodell war getrieben vom Wunsch nach Wandelbarkeit und Wiederverwendbarkeit. Bei Sleepy Hollow geht es zunächst um Testbarkeit.
- Und schließlich liegt der Fokus bei Sleepy Hollow auf einzelnen Interaktionen (Bild 14).

Ich sehe aus diesem Grund Sleepy Hollow auch als einen Vertreter der sogenannten Vertical Slice Architecture [9][10] (Bild 15). Deren Motivation ist:

„Vertical Slice Architecture was born from the pain of working with layered architectures. They force you to make changes in many different layers to implement a feature.“ [10]

Anforderungen des Kunden beziehen sich nie auf Schichten oder Schalen (Onion Architecture, Clean Architecture), das heißt Module. Ihr Kontext sind stets eine oder mehrere Interaktionen. Deshalb empfehle ich auch eine Anforderungsanalyse, die Interaktionen herausarbeitet und sofort unmissverständlich mit Entry Points in Form von Funktionen im Code verbindet (siehe [4] bis [5]).



Vertical Slice Architecture nach Jimmy Bogard [9] (Bild 15)

Jedes Slice ist also zunächst „nur“ ein Datenfluss, der auf mehrere Ebenen aufgeteilt ist. Bild 12 zeigt nur drei Ebenen, weil genau diese relevant im Zusammenhang mit dem Head sind. Letztlich gibt es jedoch beliebig viele Ebenen pro Slice, die jeweils durch Prozessschritte in einem Datenfluss repräsentiert sind, in die Sie „hineinzoomen“ können. In Bild 12 sind das sichtbar Transform, Pre- und Post-Process: Die grauen Dreiecke zeigen an, was in/hinter diesen Schritten im Detail steckt.

Technisch ist die Umsetzung trivial. In Bild 17 sehen Sie jeden Pipeline-Schritt aus den Bildern 11 und 12 als Funktion umgesetzt.

Ja, Sie sehen richtig: Sleepy Hollow konzentriert sich zunächst auf Funktionen. Denn da „spielt die Musik“. Funktionen tun etwas; sie erzeugen Verhalten. Wie Verhalten erzeugt wird als Reaktion auf einen Reiz des Anwenders im Rahmen einer Interaktion, bestimmt die Logik [2], die in den Blättern der Datenflusshierarchie steckt (Integration Operation Segregation Principle (IOSP) [11], näher entwickelt in [8]).

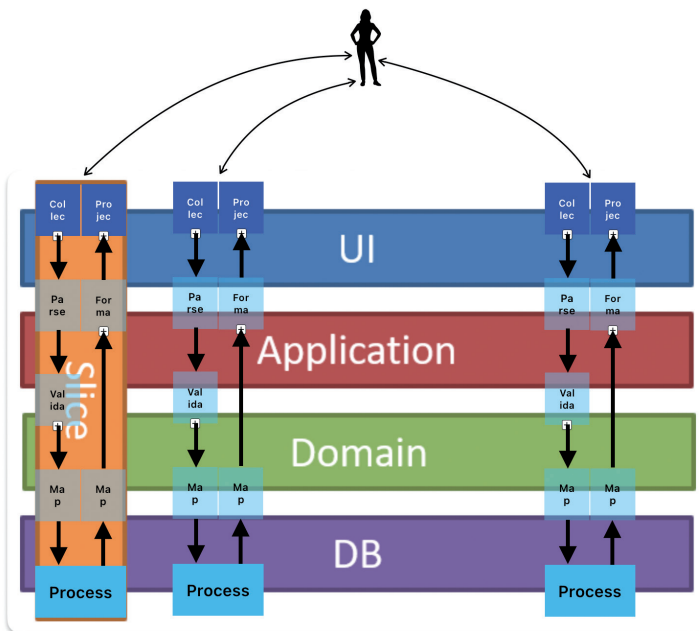
Anforderungsanalyse mit Slicing und Sleepy Hollow fokussieren auf die Verhaltenserzeugung mit Datenflüssen für einzelne Interaktionen. Umgesetzt werden diese Datenflüsse mit Funktionen – die natürlich alle eine Heimat in einem Modul haben.

IODA-Architektur

Alle Funktionen in Bild 17 können Sie sehr leicht verstehen. Das liegt daran, dass sie kurz sind und keine Logik enthalten. Es sind reine Integrationen [11]. So sehen ganz natürlich mit einer imperativen Programmiersprache implementierte Datenflüsse aus.

Integrationen sind Funktionen, die Funktionalität auf höherer Abstraktionsebene aus Funktionen niedrigerer Abstraktionsebene „komponieren“; Integrationen sind Compositions (Komposite).

Diese Idee der Unterscheidung zwischen Integration und Operation – Funktionen, die nur Logik enthalten – lässt sich nun auch auf Klassen (oder Dateien wie in Typescript oder Python) ausdehnen: Wenn ein Modul vor allem Funktionen enthält, die integrieren, dann ist es ein integrierendes Modul.



Sleepy-Hollow-Interpretation der Vertical Slice Architecture (Bild 16)

Diese Eintrittspunkte des Slicing liegen auf der Schnittstelle des Bodys. Alles darüber und darum gehört zum Head.

Anders als in der „klassischen“ Vertical Slice Architecture besteht ein Slice für mich nicht aus dünner geschnittenen Schichten, wie noch Bild 15 andeutet. Vielmehr sind Slices vor allem U-Pipelines, von denen so viele „nebeneinander liegen“, wie es Interaktionen in einem Softwaresystem gibt (Bild 16).

Wenn es hingegen vor allem Funktionen enthält, die operieren, also tatsächlich Verhalten mittels Logik herstellen, dann ist es ein operierendes Modul.

- Integrationen und Operationen abstrahieren durch Komposition: Sie fassen Verschiedenartiges zu etwas Neuem zusammen.
- Module hingegen abstrahieren durch Aggregation: Sie fassen Ähnliches unter einem Oberbegriff zusammen. Ähnlich ist, was sich demselben Thema widmet; konkret drückt sich Ähnlichkeit beziehungsweise Zusammengehörigkeit dadurch aus, dass Zustand gemeinsam genutzt wird.

Basierend auf dem IOSP definiert die IODA-Architektur deshalb eine modulare Grundstruktur für Software, die ohne funktionale Abhängigkeiten [3] auskommt.

In **Bild 18** sehen Sie, dass die eigentlichen „Arbeitspferde“ einer Software – Domäne (Core) und Adapter für die Kommunikation mit der Umwelt (Portale und Provider) – einander nicht kennen. Sie sind anders als in der Schichtenarchitektur oder auch der Clean Architecture nicht voneinander abhängig, weder direkt noch indirekt. Höchstens teilen sie die Kenntnis von Datenstrukturen.

Eine weitere Verantwortlichkeit, die Integration, ist eingeführt, um diese „Arbeitspferde“ explizit zusammenzuschreiben. Diese Verantwortlichkeit ist auf der linken Seite in **Bild 18** nicht visualisiert, aber mitgedacht; deshalb fehlen dort Beziehungspfeile zwischen den „Arbeitspferden“.

In anderen Architekturen sind die Abhängigkeiten zwischen Schichten beziehungsweise Schalen genauestens eingezeichnet (meistens mit Pfeilen) oder durch räumliche Nähe von Modulen suggeriert (zum Beispiel Schichtung ohne Zwischenräume). Die IODA-Architektur kennt solche Abhängigkeiten nicht, deshalb sind links in **Bild 18** keine Verbindungslinien. Verbindungen gibt es erst rechts und stets ausschließlich von einer Integration nach untenweisend.

Ein integrierendes Modul in der IODA-Architektur hat den Zweck, andere Module zu integrieren. Im integrierenden Modul findet sich keine Logik (beziehungsweise nur in Ausnahmefällen). Integrationen in IODA definieren Datenflüsse.

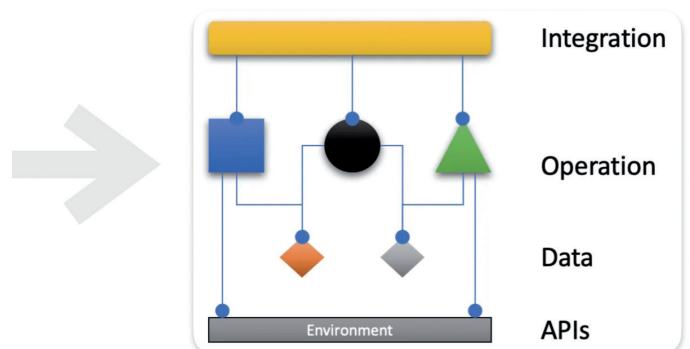
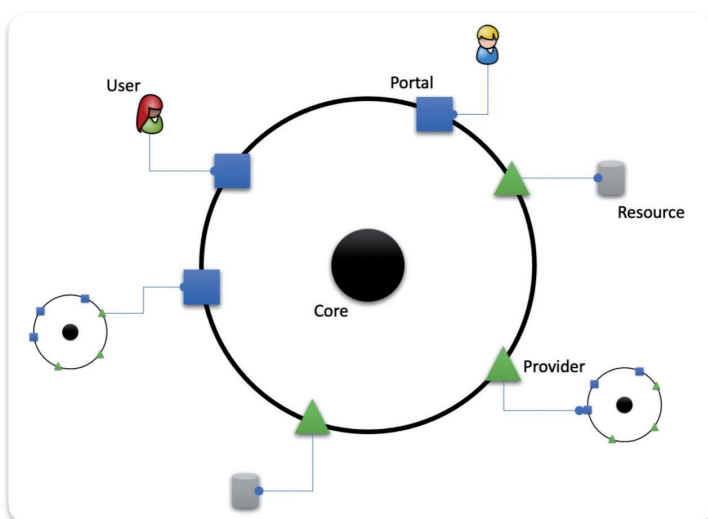
```

1  function interaction() {
2      const request = collect();
3      const response = transform(request);
4      project(response);
5  }
6
7      function transform(request:any):any {
8          const request1 = preProcess(request);
9          const response1 = process(request1);
10         return postProcess(response1);
11     }
12
13         function preProcess(request:any):any {
14             const parsed = parse(request);
15             const validated = validate(parsed);
16             return map(validated);
17         }
18
19         function postProcess(response:any):any {
20             const mapped = map(response);
21             return format(mapped);
22         }
23
24         function process(request:any):any {
25             //...
26         }
    
```

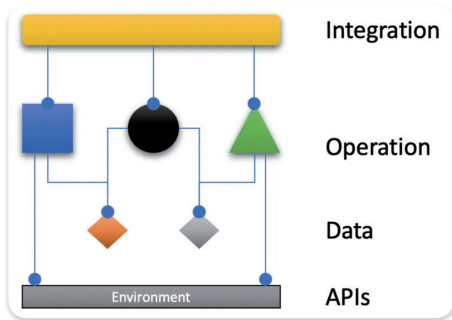
Triviale modellhafte Umsetzung der Sleepy Hollow Pipelines (Bild 17)

Das Sleepy-Hollow-Architekturmuster differenziert nun die Integration der IODA-Architektur in **Bild 18** rechts. Ganz konkret definiert Sleepy Hollow drei Ebenen der Integration (**Bild 19**):

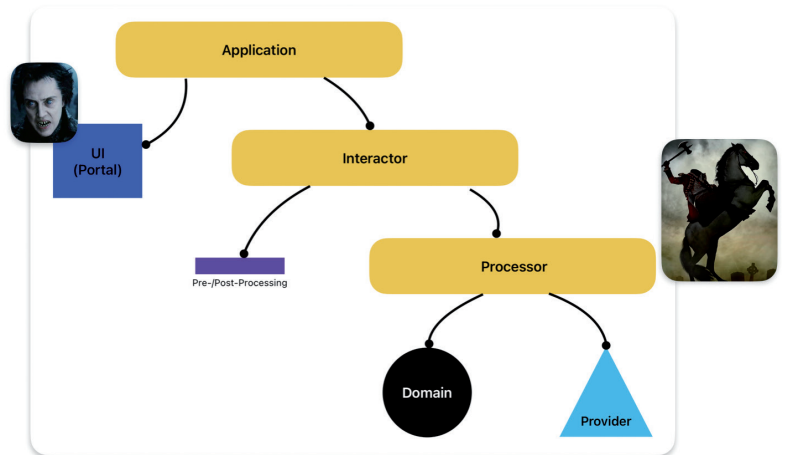
- **Eine Application** integriert Sammlung und Projektion des User Interface mit der Transformation. Die Application stellt den Datenfluss wie in **Bild 11** her. Transform ist mithin Teil der nächsten Integration. Eine Application veröffentlicht als Einstiegspunkt zum Beispiel eine `run()`-Funktion.
- **Interactor-Integrationen** integrieren innerhalb von Transformationen das Pre-/Post-Processing aus einem darauf spezialisierten Modul mit der eigentlichen Verarbeitung von ►



Die Grundstruktur von Software nach der IODA-Architektur [4] (Bild 18)



Die Ebenen der Integration im Sleepy-Hollow-Architekturmuster (Bild 19)



Requests. „Interactor“ nenne ich diese Integrationen, weil sie die Reaktionen auf die „Reize“ des Benutzers implementieren. Sie repräsentieren die Interaktionen mit der „treibenden“ Umwelt. Dass die UI-Technologie hier abwesend ist, tut dem keinen Abbruch; sie ist nur ein Detail. Interactors veröffentlichen eine oder mehrere *transform()*-Funktionen.

- **Die Processor-Integrationen** schließlich veröffentlichen eine oder mehrere *process()*-Funktionen, die an der Spitze der Integration von Domänenfunktionalität mit Providern steht. Hier wird das eigentliche Verhalten der Software produziert. „Hier spielt die Musik.“

Diese Modularisierung bietet schon auf dieser allgemeinen Ebene einige Ansatzpunkten für automatisierte Tests (Bild 20):

- Akzeptanztests auf Interactors,
- Integrationstests auf Processors,

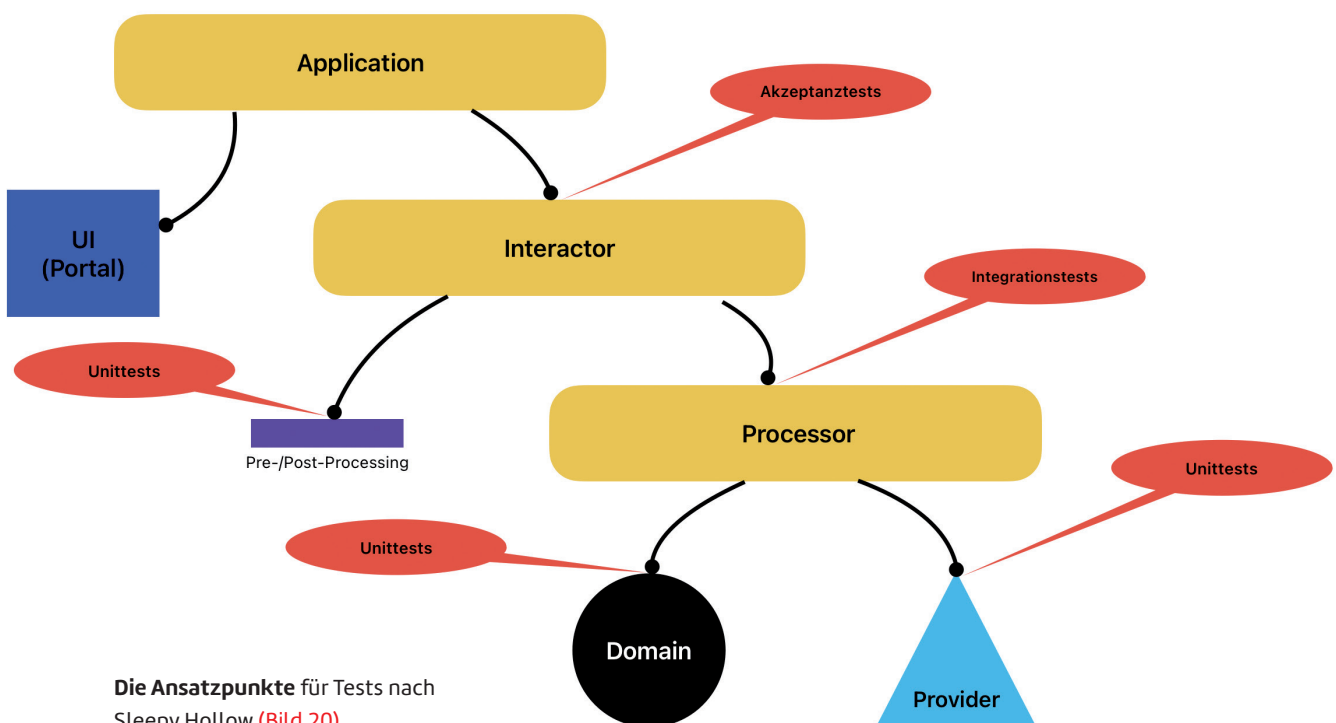
- Unit-Tests auf Domäne, Providern und Pre-/Post-Processing. Unit-Tests, weil diese Module (zumindest auf der „Flughöhe“ von Bild 19) keine Abhängigkeiten mehr aufweisen.

Selbstverständlich gilt für alle hier gezeigten Module, dass sie weiter differenziert werden können. Ich will nicht suggerieren, dass Sie zum Beispiel Ihre ganze Domänenlogik in eine Klasse „Domain“ verpacken. Sie können, sollen und müssen alle Aspekte geeignet auf mehrere Module aufteilen.

Das Architekturmuster macht nur einen allgemeinen Vorschlag für die grundsätzliche und minimale Differenzierung inklusive Definition der Abhängigkeitsverhältnisse.

Fazit

Testbarkeit ist der entscheidende Treiber für die Modularisierung. Ausgehend von dieser Überzeugung habe ich das



Die Ansatzpunkte für Tests nach Sleepy Hollow (Bild 20)

Sleepy-Hollow-Architekturmuster entwickelt. Noch vor jeder anderen Architekturempfehlung legt es Ihnen nahe, zumindest Head-Logik, die abhängig ist von UI-Technologie, und Body-Logik, die davon unabhängig ist, durch eine Schnittstelle zu trennen.

An dieser Schnittstelle können Sie Akzeptanztests ansetzen. Hier manifestieren sich die Anforderungen des Product Owners (PO) formal in Form von Testfällen, die Sie automatisiert jederzeit durchspielen können.

Sleepy Hollow dient damit auch der Dokumentation Ihrer Software. Was ist möglich? Welche Requests führen zu welchen Responses?

Mit einem vom Head getrennten Body können Sie gezielt Experimente anstellen: Wie reagiert Ihre Kreation in unterschiedlichen Situationen? (Vielleicht wäre auch „Frankenstein Architecture“ ein passender Name gewesen?)

In meiner Erfahrung macht die Anforderungsanalyse mit Slicing mit dem Ziel einer Sleepy-Hollow-Modularisierung die Gespräche mit POs geradliniger und konkreter. Sie wissen erstens genau, worauf alles hinauslaufen muss. Sie wissen zweitens, dass Sie die Erfüllung der Anforderungen jederzeit selbst überprüfen können. ■

- [1] Clean Code Developer Wiki, www.dotnetpro.de/SL2411SleepyHollow1
- [2] Ralf Westphal, Logic Makes the Software Turn Around, www.dotnetpro.de/SL2411SleepyHollow2
- [3] Ralf Westphal, Functional Dependencies, www.dotnetpro.de/SL2411SleepyHollow3
- [4] Ralf Westphal, Strukturiert zerlegen, dotnetpro 4/2024, Seite 42 ff., www.dotnetpro.de/A2404Slicing
- [5] Ralf Westphal, Inkremente innerhalb von Interaktionen finden, dotnetpro 9/2024, Seite 32 ff., www.dotnetpro.de/A2409Slicing
- [6] Ralf Westphal, IODA Architecture, www.dotnetpro.de/SL2411SleepyHollow4
- [7] Ralf Westphal, Principle of Mutual Oblivion (PoMO), www.dotnetpro.de/SL2411SleepyHollow5
- [8] Ralf Westphal, Radical Object-Orientation, www.dotnetpro.de/SL2411SleepyHollow6
- [9] Jimmy Bogard, Vertical Slice Architecture, www.dotnetpro.de/SL2411SleepyHollow7
- [10] Milan Jovanovic, Vertical Slice Architecture, www.dotnetpro.de/SL2411SleepyHollow8
- [11] Ralf Westphal, Integration Operation Segregation Principle, www.dotnetpro.de/SL2411SleepyHollow9



Ralf Westphal

ist Trainer, Berater und Mitgründer der Clean Code Developer Initiative (<https://clean-code-developer.de>). Seine Schwerpunkte sind dauerhaft hohe Produktivität für die Softwareentwicklung und zukunftsfähige Teamorganisation.

<https://ralfw.de>

dnpCode

A2411SleepyHollow



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

TRAININGS



Softwarequalität
beurteilen und verbessern
Trainer: David Tielke



Async and Await
in der Praxis
Trainer: Christian Giesswein



Neuerungen bei C#,
.NET & SQL-Server & CoPilot
Trainer: Thorsten Kansy



Einstieg in Angular für
.NET-Entwickler
Trainer: Patrick Schnell



Testen mit Playwright
Trainer: David Pinezich

NEU!



.NET MAUI -
Cross-Plattformentwicklung
leicht gemacht
Trainer: Sebastian Seidel



How to make teams awesome -
Gruppen zu Teams machen
Trainer: Janosch Felde

REMOTE
TRAININGS MÖGLICH!

developer-academy.de

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

WISSEN AN DER RICHTIGEN STELLE

Eine Frage der Führung

Oft werden Probleme voreilig auf unzureichende Arbeit von Entwicklerinnen und Entwicklern zurückgeführt. Meist liegt die Ursache aber an einer Stelle, auf die keiner gerne zeigt.

Als Berater habe ich in meiner Arbeit viele Projekte mit verschiedenen Graden an Problemen untersucht, von bedenklich bis katastrophal war alles dabei. Immer wieder überrascht mich dann, wie eindimensional die Suche nach Ursachen und Schuldigen abläuft. Im Rampenlicht stehen oft wir Softwareentwicklerinnen und -entwickler. Ganz gleich was schief läuft – meist macht man uns dafür verantwortlich, auch weil wir uns nicht wehren. In Projekten führt dies nicht selten dazu, dass Probleme eine Ewigkeit lang existieren und oft nicht abgestellt werden.

Der Klassiker

Natürlich gibt es sehr unterschiedliche Softwareunternehmen auf der Welt, aber einen bestimmten Stereotyp erkenne ich immer wieder, der ein Problem vieler Softwareunternehmen auf ganz hervorragende Art und Weise aufzeigt.

Eine Idee für ein Softwareprodukt zu generieren, mit dem Umsatz und Gewinn zu machen sind, ist auch im Jahr 2024 kein Hexenwerk. Hat der Ideenfinder auch noch Programmiererfahrungen, kann es direkt losgehen: Meist mit nur rudimentären Kenntnissen in der Softwareentwicklung ausgestattet wird das Projekt gestartet und bald schon Umsatz erzielt. Zu Beginn muss man sich selbst um alles kümmern: Marketing, Vertrieb, Buchhaltung, Geschäftsleitung und schon bald um die Führung von Mitarbeitern. Obwohl solche Geschäftsführer selbst aus der Entwicklung stammen, hat sich bei ihnen kein grundlegendes Verständnis für gute Anforderungen, Architektur, Softwaredesign, das Testen oder andere Bereiche professioneller Softwareentwicklung entwickelt. Wo Softwareentwickler im Lauf der Jahre durch zahlreiche Fehler einen oft langwierigen Lernprozess durchlaufen und dadurch nützliche Erfahrungen und wertvolles Wissen aufbauen, ist dies bei solchen ehemals entwickelnden Geschäftsführern nicht der Fall. Ihre Argumentation: Anforderungen kann doch der Entwickler aufnehmen, Architektur kostet nur und bringt nichts, testen kann der Kunde, und all die neumodischen Tools hat man damals auch nicht benötigt.

Diese Geschäftsführer bauen also ihr Unternehmen auf, und irgendwann muss eine Leitungsstelle für die Softwareentwicklung besetzt werden. Dazu werden aber nicht immer diejenigen Personen herangezogen, die das meiste Wissen im Bereich professioneller Softwareentwicklung haben, sondern die, die das Mindset der Geschäftsführung teilen: Funktionen

bringen Umsatz, Qualität ist zweitrangig, und außer ein wenig Programmieren muss ein Entwickler nichts können. Solche Entwicklungsleiter haben also oft nicht nur keine Ahnung von guter Softwareentwicklung, sondern erhalten von der Geschäftsleitung auch keine Anreize, diese Kompetenzen aufzubauen – weder bei ihren Mitarbeitern noch bei sich selbst.

Mangel an qualifizierter Mitarbeiterführung

Aus dieser Konstellation und Entwicklung resultiert eine Führungsstruktur, in der Softwareentwickler alles andere als optimal arbeiten. Nicht nur werden neue Funktionen über alles andere priorisiert, sondern es entwickelt sich auch keinerlei Verständnis für die tatsächlichen Bedürfnisse der Softwareentwicklung. Weiterbildung geschieht zu selten, Zeit für Refactorings wird nicht genehmigt, Probleme werden in Ermangelung der Fähigkeit, sie zu lösen, liegen gelassen. In der Folge führt dies zu einem schlechten Arbeitsklima und unzufriedenen Mitarbeitern in der Softwareentwicklung. Das, was auch ältere, aber insbesondere jüngere Softwareentwickler benötigen, findet nicht statt: die fachlich und methodisch qualifizierte Führung der Mitarbeiter.

Auch wenn die hier beschriebenen Umstände bei Weitem nicht auf alle Unternehmen zutreffen, wird sich der ein oder andere Leser in dieser oder einer ähnlichen Situation wiederfinden und die Qualen nachvollziehen können.

Ein Entwickler sagte im Scherz einmal: „Zeig mir deine Führungskräfte, und ich sage dir, wie gut deine Software ist.“ Obwohl nur lustig gemeint, erkenne ich bis heute in vielen Projekten, wie viel Wahrheit in diesem Satz steckt.



Programmierung versus Entwicklung

Ein schon oft thematisierter, grundlegender Fehler solcher Unternehmen ist der nicht erkannte Unterschied zwischen Programmierung und Softwareentwicklung. Dazu müssen wir uns erneut die Unterschiede zwischen funktionalen und nichtfunktionalen Anforderungen in der Softwareentwicklung in Erinnerung rufen, auf die ich bereits näher eingegangen bin [1]. Sie unterscheiden sich nicht nur in ihrer Wirkung auf das Budget des Unternehmens, sondern auch in ihrem Einfluss auf die Qualität des entwickelten Softwareprodukts.

Funktionale Anforderungen bestimmen das Was, also welche Funktionen das entwickelte Softwareprodukt besitzen soll. Da diese Funktionen oftmals von Endkunden gefordert

werden, adressiert diese Art der Anforderungsdefinition hauptsächlich die Umsätze in der Softwareentwicklung.

Die nichtfunktionalen Anforderungen betreffen dagegen das Wie, also wie bedienbar, wie schnell, wie effizient und wie erweiterbar und wartbar die Software sein muss. Solche und viele weitere nichtfunktionale Eigenschaften können wir auch unter dem Begriff der Nachhaltigkeit von Softwareentwicklung zusammenfassen. Sie sind nicht unbedingt direkter Bestandteil der Wünsche von Kunden, sondern spiegeln eher deren allgemeine Erwartungshaltung wider.

Während also die funktionalen Anforderungen für uns garantieren, dass über die Kundenwünsche zeitnah Umsätze mit dem Softwareprodukt erzielt werden, sind die nichtfunktionalen Anforderungen dafür zuständig, dass wir das Produkt lange und gut entwickeln können und damit langfristig Umsatz erzielen können. Das Verständnis dieses Gegensatzpaars ist eine der Lektionen, die ein Softwareentwickler im Lauf seiner Karriere (schmerzhaft) lernen muss.

Die schnell vom Entwickler zu einem Geschäftsführer mutierten Führungskräfte haben diese Erfahrung jedoch oft nicht gemacht, und demnach mündet ihre Marschrichtung nicht selten nur in „Funktionen, Funktionen, Funktionen!“ ...

Probleme und ihre Folgen

Auf diese Weise nicht nachhaltig entwickelte Software generiert mit zunehmender Projektlaufzeit immer mehr und größere Probleme. Die Geschwindigkeit des Entwicklungsteams sinkt, die Anzahl der Bugs nimmt zu, ständig hört man „geht nicht“, oder die Zeitschätzungen für neue Anforderungen sind astronomisch hoch. Genau in dieser Zeit entwickelt sich dann ein tiefer Graben zwischen Führungskräften und Entwicklern. Die einen sehen nur mögliche Probleme und müssen diese Tag für Tag ertragen. Die andere Seite hingegen hat überhaupt kein Verständnis für die Probleme und deren Ursachen. In der Folge gelten Führungskräfte oft als inkompetent oder überfordert, Entwickler als faul oder unzuverlässig. Diesen Graben zwischen Führungskräften und Entwicklern wieder mit Vertrauen aufzufüllen ist meiner Erfahrung nach meist eine Mammutaufgabe.

Natürlich muss an dieser Stelle erwähnt werden, dass Problemsituationen oftmals sehr komplex sind und das hier beschriebene Problem bei Weitem nicht das einzige im Unternehmen ist. Dennoch scheint es mir in mindestens jedem zweiten Projekt eine wesentliche Rolle zu spielen.

Die Wurzel des Problems

In vielen Unternehmen besteht die Ansicht, dass allgemeines Wissen über die Softwareentwicklung da angesiedelt sein muss, wo sie auch praktiziert wird: in der Abteilung oder dem Team der Softwareentwicklung. Auch wenn das Wissen hier zweifelsohne vorhanden sein sollte, ist das nur die halbe Wahrheit. Um Softwareentwicklung in einem Unternehmen richtig zu betreiben, benötigt ein Team von außen bestimmte Vorgaben: ausgereifte Prozesse, vernünftige Richtlinien, eine angemessene Architektur, gute Anforderungen und natürlich auch eine entsprechende Vision für das Produkt und das Unternehmen.

Um diese Artefakte erstellen zu können, benötigen aber die entsprechenden Rollen das Wissen dazu, wie Softwareentwicklung grundsätzlich abläuft und welche Gefahren und Risiken es dabei gibt. Sind diese nicht bekannt, fehlen auch die benötigten Vorgaben ganz oder teilweise. Ich wiederhole in meinen Projekten immer wieder einen Satz: „So funktioniert Softwareentwicklung nun einmal nicht!“ Um ein Softwareprodukt zu programmieren, benötigen Sie nicht unbedingt Experten aus der Softwareentwicklung. Aber dieses „Programmieren“ ist dann nicht nachhaltig. Wir wollen ein Produkt über lange Zeit entwickeln und es nicht nur irgendwie zusammenschustern, um damit Geld zu verdienen.

Wissen an den richtigen Stellen

In Unternehmen macht der Geschäftsführer dem Bereichsleiter Zielvorgaben für dessen Bereich, der Bereichsleiter seinem Abteilungsleiter, der Abteilungsleiter seinem Teamleiter und dieser dann seinen Entwicklern. Auf dieser gesamten Linie im Organigramm sollte zwingend ein allgemeines Wissen über die professionelle Softwareentwicklung existieren. Auf der obersten Ebene ist es natürlich nur sehr abstrakt und allgemein, aber mit jeder weiteren Stufe sollte es umso genauer und spezifischer sein. Ohne dieses Wissen kann eine Führungskraft nicht die richtigen Vorgaben machen und diese bei ihrer Weitergabe noch schlechter formulieren. Am Ende kommen bei den Entwicklern dann nicht selten nur noch die erwähnten „Funktionen, Funktionen, Funktionen!“ an. Damit ist das gesamte Produkt und oftmals das Unternehmen auf dem Weg, mit Vollgas vor die Wand zu fahren.

Hier sind Sie als Führungskraft gefragt. Prüfen Sie Ihr eigenes Wissen und das Ihrer Vorgesetzten und korrigieren Sie es entsprechend. In der Vergangenheit hatte ich immer wieder das Gefühl, dass Vorgesetzte entweder keine Notwendigkeit sehen, solches Wissen aufzubauen, oder es eine gewisse Scham gibt, weil sie es eigentlich haben müssten.

Wenn Sie ein Entwickler sind, der von dem hier beschriebenen oder einem vergleichbaren Problem betroffen ist: Sprechen Sie mit Ihren Vorgesetzten und weisen Sie auf das Problem hin. Auch wenn es sicherlich einfachere Gespräche gibt: Je länger die Betroffenen die Augen vor diesem Problem verschließen, desto größer wird das Problem am Ende. Und das kann nun wirklich niemand gebrauchen.

Ich wünsche Ihnen viel Erfolg dabei! ■

[1] David Tielke, *Der Fisch stinkt vom Kopf*, dotnetpro 8/2024, Seite 40 f., www.dotnetpro.de/A2408DDD



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

A2411DDD

STEUERELEMENTE DER COMMUNITY

MAUI-Tools

Nützliche Hilfen für besondere Aufgaben.

Es gibt kaum eine Microsoft-Technologie, für die die Community keine Bibliothek mit nützlichen Tools bereitstellt. Auch für MAUI [1] gibt es eine Community-Toolkit-Erweiterung, über die zusätzliche Komponenten zur Verfügung stehen [2][3]. Die Community verpackt darin verschiedene Steuerelemente und Helfer, die fortgeschrittene Aufgaben erleichtern.

Das Toolkit einrichten

Im Bereich der Kompatibilität leistet die Community gute Arbeit. Die folgende Aufzählung zeigt die von den jeweiligen Betriebssystemen erwarteten Mindestversionen – lobenswert ist, dass auch Samsungs Smart-TV-Betriebssystem Tizen unterstützt wird, seitdem es mit MAUI-Kompatibilität ausgestattet wurde:

- Android ab Version 5.0,
- iOS ab Version 10,
- macOS ab Version 10.15 unter Nutzung von Catalyst,
- Windows 10 ab Version 1809 mit WinUI 3,
- Tizen ab Version 7.0.

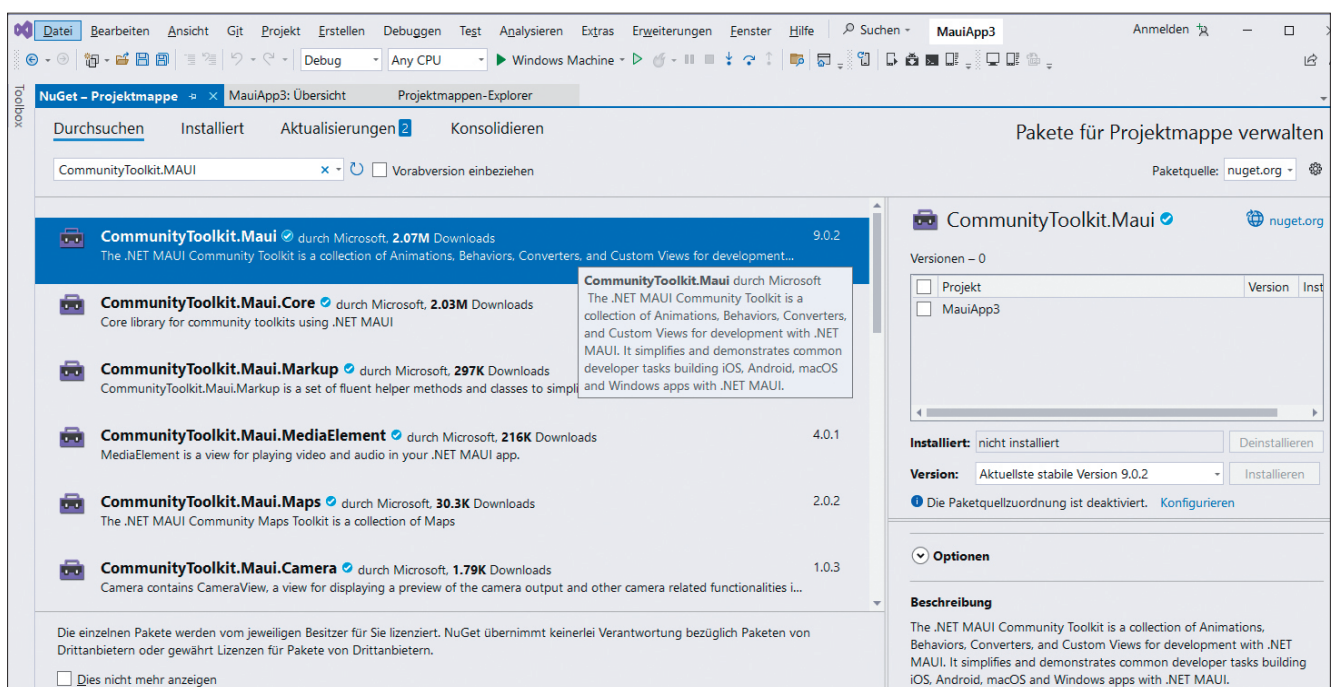
Als Experimentierplattform wird in den folgenden Schritten eine unter Windows 11 laufende Workstation zum Einsatz kommen. Als integrierte Entwicklungsumgebung wird Visual

Studio 2022 in einer Preview-Variante genutzt. Unsere erste Aktion ist das Erzeugen eines neuen Projekts auf Basis der Vorlage *.NET MAUI-App*, für das der Autor den Namen *NMGMAuiTKDemo1* vergeben hat.

Im Bereich der Auswahl des Frameworks gilt es darauf zu achten, sich für die Version 8.0 zu entscheiden. Der Grund: Schon im März hat die Community ihren Entwicklungsaufwand auf die neueste Variante des Frameworks umgeleitet, weshalb ältere Versionen keine Updates mehr bekommen.

Die eigentliche Auslieferung der zum Toolkit gehörenden Komponenten erfolgt dann über die NuGet-Konsole. Suchen Sie dort nach dem String *CommunityToolkit.Maui*, woraufhin die in **Bild 1** gezeigten Komponenten eingeblendet werden. Für die hier durchgeführten Experimente ist nur das Paket *CommunityToolkit.Maui* erforderlich – wer fortgeschrittene Funktionen nutzen möchte, muss zusätzliche Pakete laden.

Nach dem erfolgreichen Deployen bietet sich eine Neukompilation der Solution an. In manchen Fällen führt diese zu einem Fehler dieser Art: „Ein Downgrade des Pakets Microsoft.Maui.Controls von 8.0.14 auf 8.0.7 wurde festgestellt ...“ In diesem Fall kehren Sie zur NuGet-Konsole zurück, wechseln nun aber in den Tab *Aktualisierungen*. Visual Studio 2022 informiert dort über veraltete Pakete in der Solution und bietet Updates an.



Für die Beispiele im Artikel genügt das NuGet-Paket *CommunityToolkit.Maui* (**Bild 1**)

Zu beachten ist, dass Sie die in der Fehlermeldung genannte Variante des Pakets auswählen sollten – beim Schreiben des Artikels offerierte Microsoft manche Pakete als Vorabversionen der Version 9.0, die für unsere Experimente nicht geeignet sind.

Zum Finalisieren der Integration ist es dann noch erforderlich, im Rahmen der Errichtung der Applikationsstruktur die Methode `UseMauiCommunityToolkit` aufzurufen. Der nachfolgend gezeigte Code realisiert eine „Baseline“ – wir werden in diesem Artikel an späterer Stelle noch feststellen, dass die Nutzung fortgeschrittener Funktionen das Übergeben von Parametern oder das Aufrufen weiterer Hilfsmethoden voraussetzt:

```
public static class MauiProgram {
    public static MauiApp CreateMauiApp() {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .UseMauiCommunityToolkit()
            .ConfigureFonts(fonts => ...

```

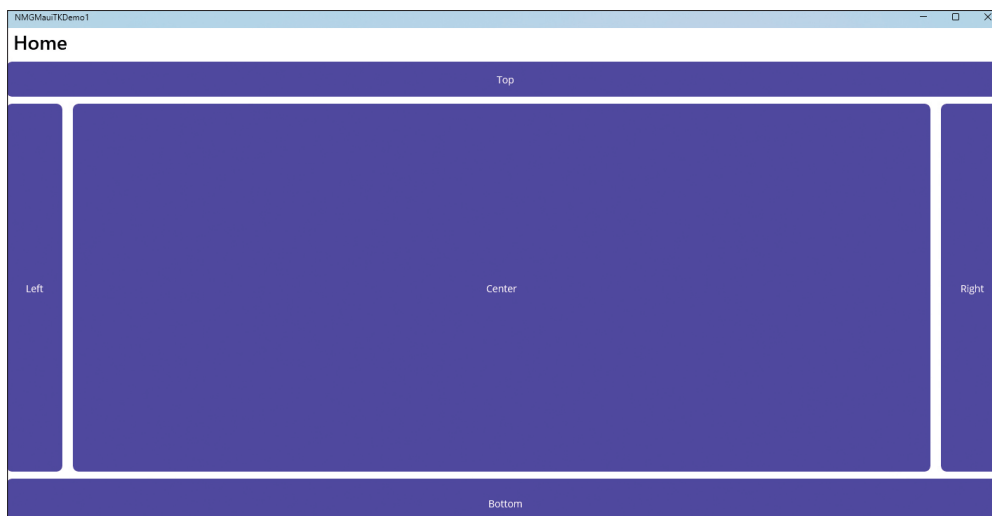
Komponenten per DockLayout organisieren

In praktischen Applikationen wünscht man sich immer wieder ein Layout, das aus Header, Footer sowie links und rechts statisch eingeblendeten Bereichen besteht. Das Community Toolkit entspricht diesen Vorstellungen durch das `DockLayout`, das seine Steuerelemente nach dem in **Bild 2** gezeigten Schema auf dem Bildschirm angeordnet.

Um das Layout nutzen zu können, wechseln Sie im ersten Schritt zur XAML-Datei des Projekts, zu der folgende Namespace-Deklaration für die im Toolkit enthaltenen Komponenten hinzugefügt wird:

```
<ContentPage ...
    xmlns:toolkit="http://schemas.microsoft.com/
        dotnet/2022/maui/toolkit"
    x:Class="NMGMAuiTKDemo1.MainPage">

```



Das DockLayout: `VerticalSpacing` und `HorizontalSpacing` sorgen für größere Abstände (Bild 2)

Im nächsten Schritt wird das vom Projektgenerator angelegte `ScrollView`-Control durch folgendes Markup ersetzt:

```
<toolkit:DockLayout>
    <Button toolkit:DockLayout.DockPosition="Top"
        Text="Top" HeightRequest="50" />
    <Button toolkit:DockLayout.DockPosition="Bottom"
        Text="Bottom" HeightRequest="70" />
    <Button toolkit:DockLayout.DockPosition="Left"
        Text="Left" WidthRequest="80" />
    <Button toolkit:DockLayout.DockPosition="Right"
        Text="Right" WidthRequest="90" />
    <Button Text="Center" />
</toolkit:DockLayout>

```

Fürs Erste soll das von Microsoft vorgegebene und aus fünf Buttons bestehende Beispiel genügen, weil es die Anordnung der in einem `DockLayout` enthaltenen Steuerelemente am besten visualisiert. Relevant ist hier vor allem das Attribut `toolkit:DockLayout.DockPosition` – es legt fest, wo das jeweilige Steuerelement unterzubringen ist. Ohne dieses Attribut ausgestattete Steuerelemente erscheinen in der Mitte.

Unbefriedigend ist an der Standardeinstellung der sehr geringe Abstand zwischen den einzelnen Elementen. Für das Beispiel sind die Attribute `VerticalSpacing="10"` und `HorizontalSpacing="15"` erforderlich, die zur in **Bild 2** gezeigten Fensterdarstellung führen:

```
<toolkit:DockLayout
    VerticalSpacing="10"
    HorizontalSpacing="15">
    <Button toolkit:DockLayout.DockPosition="Top"
        Text="Top" HeightRequest="50" />

```

Angemerkt sei, dass das `DockLayout` auch noch einige andere Attribute anbietet. Außerdem ist das Verwenden von Buttons nicht unbedingt erforderlich – es ist erlaubt, andere Steuerelement-Typen oder ganze Unterlayouts in die Layout-Engine einzubinden.

Alerts informieren den Benutzer

Als Palm mit OS 4 den Notification Manager einführte, schrieb der PDA-Hersteller auf vielerlei Art und Weise Geschichte. Das System war zur Zentralisierung der Ereignisse befähigt und ersetzte die bisher hauseigenen Alert-Dialoge durch eine vom Betriebssystem bereitgestellte und optisch aus einem Guss erscheinende Variante.

Seither bieten so gut wie alle Betriebssysteme ►

ähnliche Komponenten an. Auch das Community Toolkit offeriert ein Interface zur Vereinheitlichung der Ansprache. Bei seiner Benutzung ist zu beachten, dass es sich dabei um eine Komponente handelt, die – je nach Betriebssystem – unterschiedliche Anpassungen an der Struktur des MAUI-Projektskeletts voraussetzt. Im Allgemeinen ist es vor dem Einbinden einer derartigen Komponente empfehlenswert, die Dokumentation zurate zu ziehen – im Fall des Controls *Snackbar* präsentiert sich die Situation wie in **Bild 3** gezeigt.

Beim Schreiben dieses Artikels kam es zu Problemen beim Aktivieren der Tabs. Eine Lösung brachte die Anleitung unter [4]. Unter Windows ist folgende Anpassung erforderlich:

```
var builder = MauiApp.CreateBuilder();
builder.UseMauiApp<App>()
    .UseMauiCommunityToolkit(options => {
        options.SetShouldEnableSnackbarOnWindows(true);
    }).ConfigureFonts(fonts => ...
```

Außerdem ist die Deklaration einer Permission erforderlich: Aus Platzgründen werden die (umfangreichen, aber simplen) Anpassungen an der Manifest-Datei hier nicht abgedruckt – Sie finden sie jedoch innerhalb des Quellcodes in den Downloads zum Artikel. Zu beachten ist zudem, dass das von Microsoft online zur Verfügung gestellte Sample nicht vollständig ist. Das Attribut *com:ExeServer* muss angepasst werden – wer den vom Autor vorgegebenen Projektnamen übernimmt, kann dabei so vorgehen:

```
<!-- Register COM CLSID -->
<com:Extension Category="windows.comServer">
    <com:ComServer>
        <com:ExeServer Executable=
```

Platform specific initialization

To access the **Snackbar** functionality, the following platform specific setup is required.

Android iOS/Mac Catalyst Windows Tizen

When using **Snackbar** it is essential to perform the following two steps:

1. Enable the snackbar usage with the MauiAppBuilder

When using the **UseMauiCommunityToolkit** make use of the **options** parameter to enable the snackbar usage on Windows as follows:

```
C#
var builder = MauiApp.CreateBuilder();
builder.UseMauiCommunityToolkit(options => {
    {
        options.SetShouldEnableSnackbarOnWindows(true);
    }
});
```

The above will automatically register the required handlers by configuring lifecycle events (**OnLaunched** and **OnClosed**).

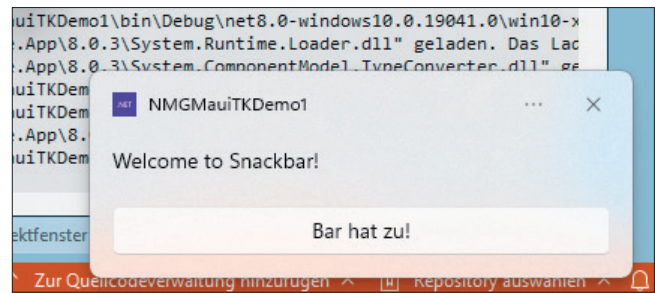
2. Include ToastNotification registrations in your Package.appxmanifest file

To handle the snackbar actions you will need to modify the **Platform\Windows\Package.appxmanifest** file as follows:

1. In **Package.appxmanifest**, in the opening **<Package>** tag, add the following XML Namespaces:

```
XML
xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
```

Snackbar: Dokumentation zum plattformspezifischen Initialisieren des Controls (**Bild 3**)



Unter Windows integriert sich die Snackbar in den Notification-Start des Betriebssystems (**Bild 4**)

```
"NMGMauITKDemo1\NMGMauITKDemo1.exe"
DisplayName="$targetnametoken$"
Arguments="----AppNotificationActivated:">
<!-- Example path to executable:
    NMGMauITKDemo1\NMGMauITKDemo1.exe -->
<com:Class Id="6e919706-2634-4d97-a93c-
    2213b2acc334" />
</com:ExeServer>
```

Nach den erfolgreichen Anpassungen an der Manifest-Datei erhält einer der Buttons einen Eventhandler, der nach folgendem Schema mit dem Erzeugen der in der Snackbar anzuzeigenden Elemente beginnt:

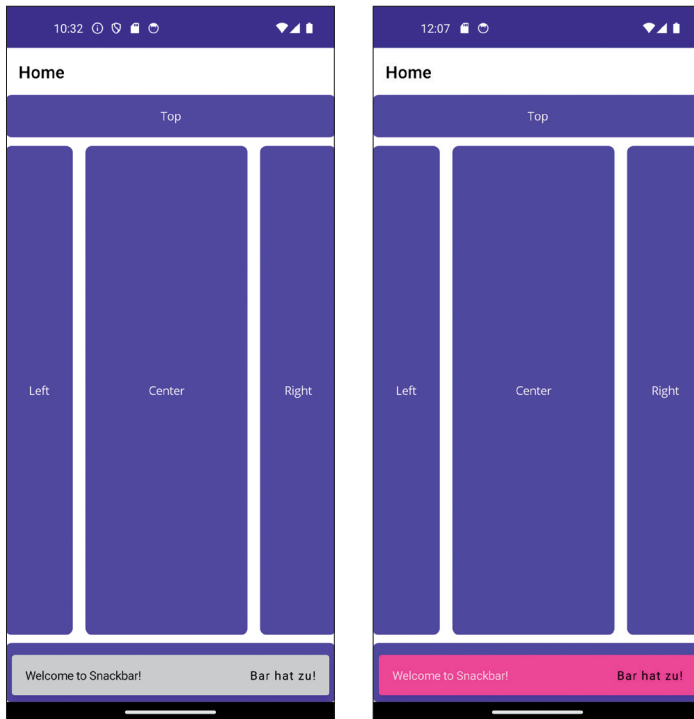
```
private void Button_Clicked(object sender, EventArgs e)
{
    CancellationTokenSource cancellationTokenSource =
        new CancellationTokenSource();
    string text = "Welcome to Snackbar!";
    string actionButtonText = "Bar hat zu!";
    Action action = async () =>
    {
        await DisplayAlert("ActionButton Tapped",
            "ActionButton!", "OK");
        TimeSpan duration = TimeSpan.FromSeconds(30);
```

Die erste Methode hat die Aufgabe, das als Attributsammlung vorliegende *Snackbar*-Steuerelement in eine für das Toolkit bekömmliche Form zu bringen. Der Aufruf *DisplayAlert* kümmert sich dann um die eigentliche Anzeige, die asynchron erfolgt – wäre der Eventhandler als asynchron deklariert, so könnten wir das Verschwinden und/oder Abblenden durch *await* abwarten.

Das Beispiel ignoriert das *SnackbarOptions*-Objekt fürs Erste – es würde das Festlegen von Farben und sonstigen Attributen erlauben. Die eigentliche Aktivierung erfolgt dann so:

```
var snackbar = Snackbar.Make(
    text, action, actionButtonText, duration);
snackbar.Show(cancellationTokenSource.Token);
}
```

Die Ergebnisse präsentiert Windows so, wie es in **Bild 4** zu sehen ist. Interessant ist, dass unter Android keine Betriebssystemkomponenten zum Einsatz kommen. Die Programm-



Unter Android kommt zur Snackbar-Anzeige haus eigenes Markup zum Einsatz (Bild 5)

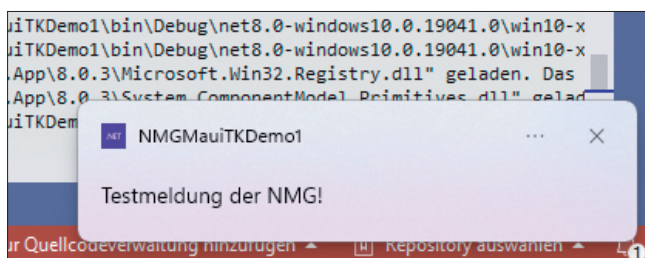
Microsofts Rendering-Logik berücksichtigt die angelieferten Attribute unter Android (Bild 6)

ausführung führt stattdessen zu dem im Bild 5 gezeigten Ergebnis. Als nächstes Experiment bietet sich die Verwendung der Klasse *SnackbarOptions* an, die unter anderem das Festlegen verschiedener Anzeigefarben erlaubt. Im Beispiel lässt sie sich nach folgendem Schema integrieren:

```
var snackbarOptions = new SnackbarOptions {
    BackgroundColor = Colors.DeepPink,
    TextColor = Colors.AliceBlue
};

var snackbar = Snackbar.Make(text, action,
    actionButtonText, duration, snackbarOptions);
```

Unter Windows bleibt das Ergebnis der Programmausführung unverändert, die erzeugte Nachricht sieht nach wie vor wie in Bild 4 aus. Unter Android ist die Änderung jedoch deutlich sichtbar, siehe Bild 6.



Unter Windows wird der Toast zu einer normalen Notification ... (Bild 7)

Toasts

Die Android-spezifische Variante der soeben vorgeführten Notification ist der Toast: ein auf der Unterseite des Bildschirms aufpoppendes Fenster mit Text-Informationen. Analog zur Realisierung einer plattformunabhängigen Schnittstelle für die Snackbar setzt Microsoft beim Toast Ähnliches um: Unter Android und teilweise unter Tizen kommt das native API zum Einsatz, während iOS, Mac und Windows „selbst geschriebene“ Anzeigen verwenden.

Wie diese Systeme funktionieren, wird im nächsten Schritt untersucht. Kehren Sie dafür zur XAML-Datei zurück und erweitern Sie das Markup um einem weiteren Button mit dem folgenden Code:

```
private void ButtonToast_Clicked(
    object sender, EventArgs e)
{
    CancellationTokenSource cancellationTokenSource =
        new CancellationTokenSource();

    string text = "Testmeldung der NMG!";
    ToastDuration duration = ToastDuration.Long;
    double fontSize = 20;

    var toast =
        Toast.Make(text, duration, fontSize);
    toast.Show(cancellationTokenSource.Token);
}
```

Der Aufbau dieser Methode bietet Bekanntes: Der Code beginnt mit der Auflistung der Eigenschaften, die danach kombiniert zur Anzeige gebracht werden.

Gut erkennbar ist, dass sich das Verhalten des Programms unter Windows und Android – wie in Bild 7 und Bild 8 gezeigt – deutlich unterscheidet.

Badge: bequemes Einblenden von Mitteilungen im Programmstarter

Ein interessantes Feature, das unter Palm OS einst von Drittanbieter-Programmstartern eingeführt wurde, war die Notification Bubble. Das war eine Anzeige, die neben dem Symbol im Programmstarter eingeblendet wurde und den Nutzer darüber informiert hat, wie viele Ereignisse in der Applikation auf ihn warten.

Problematisch ist im Fall von Android, dass es sehr viele unterschiedliche Implementierungen gibt – da Google im Betriebssystem keine gemeinsame Schnittstelle avisiert, implementiert jeder Anbieter, was ihm geeignet erscheint.

Mit der *Badge*-Klasse versucht das .NET Community Toolkit, bis zu einem gewissen Grad Abhilfe zu schaffen; quasi nebenbei gibt es auch Unterstützung für die in iOS und Tizen enthaltenen Betriebssystemdienste.

Aus der Logik der engen Betriebssystemintegration folgt, dass im ersten Schritt Anpassungen an der Applikationsstruktur erforderlich sind. Im Fall von Android gilt, dass für jedes System beziehungsweise für jeden Programmstarter eine spezifische Implementierung erforderlich ist. ►

Hier werden schematisch die Möglichkeiten für den Samsung-Launcher durchdekliniert. Im ersten Schritt ist es erforderlich, im Rahmen der *CreateMauiApp*-Funktion einen Service zu schreiben:

```
public static MauiApp CreateMauiApp()
{
    ...
    builder.Services.AddSingleton<
        IBadge>(Badge.Default);
    return builder.Build();
}
```

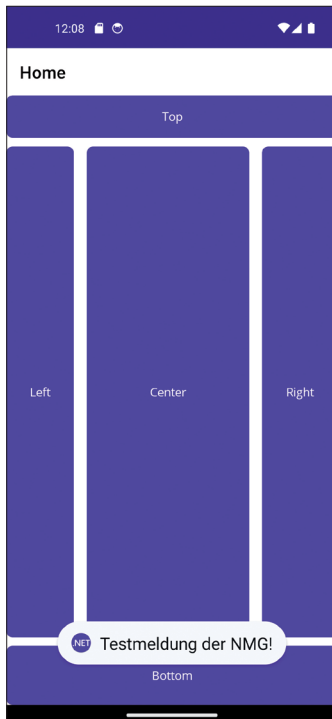
Für die Interaktion mit dem Badge beziehungsweise das Festlegen der intern anzuzeigenden Anzahl ist Dependency Injection erforderlich – eine Beispiel-Implementierung sieht wie folgt aus:

```
public partial class MainPage :
    ContentPage
{
    private readonly IBadge badge;
    public MainPage(IBadge badge) {
        InitializeComponent();
        this.badge = badge;
    }
    public void SetCount(uint value) {
        badge.SetCount(value);
    }
}
```

Zur Inbetriebnahme wird eine Instanz der Klasse *IBadgeProvider* benötigt, die sich um das Exponieren der Werte kümmert. Eine Beispiel-Implementierung für Samsung findet sich unter [5]. Zudem sind Anpassungen an der Manifest-Datei und am Einsprungpunkt der Applikation erforderlich. Da Microsoft selbst in der Dokumentation auf die Komplexität dieses Features hinweist (Bild 9), sollte man Vorsicht walten lassen – implementieren Sie dieses API also nur, wenn Sie unbedingt müssen.

Eine Datei per *DrawingView* speichern

Ein Nebeneffekt der stringenter werdenden Sicherheitsanforderungen im Bereich der Handcomputer ist, dass der Zu-



... während er unter Android die Betriebssystemkomponente nutzt (Bild 8)

griff auf das Dateisystem von Telefon und Co. bei den meisten Betriebssystemen nicht zulässig ist. Möchte man eine Datei in ein für den Benutzer zugängliches Verzeichnis speichern, artet dies in Arbeit aus.

Mit der *FileSaver*-Klasse stellt das MAUI Community Toolkit einen Workaround zur Verfügung, der diese lästige Aufgabe zu erleichtern sucht. Für einen Test wird ein neues Projekt erzeugt – die Konfiguration erfolgt gemäß den weiter oben besprochenen Anweisungen.

Da das Speichern arbiträrer Testtexte nur wenig interessant ist, soll zur Illustration der Möglichkeiten das Steuerelement *DrawingView* eingebunden werden, das Microsoft eigentlich für das Erfassen von Unterschriften vorgesehen hat. Im Beispielprogramm wird damit eine Art Notizblock realisiert, in dem sich der Benutzer mit Stift, Finger oder Maus austoben darf. Vorteilhaft ist, dass Microsofts Entwicklerteam einen bequemen Weg zum Übernehmen der gezeichneten Grafiken zur Verfügung stellt.

Nach dem Einbinden der NuGet-Pakete ist es wichtig, zur Manifestdatei zu wechseln und die folgenden Erlaubnisse (*Permissions*)

einzufragen. Wer außer Android und Windows auch andere Betriebssysteme ansprechen möchte, findet die nötigen Modifikationen – wie immer – in der Dokumentation des Community Toolkits:

```
<uses-permission android:name=
    "android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:
    name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Im nächsten Schritt wird eines der in der MAUI-Bibliothek enthaltenen Steuerelemente aktiviert. Dazu wechseln Sie zur XAML-Datei, die zunächst die von weiter oben bekannten Namespace-Erweiterungen aufnimmt. Nach dem Anpassen der Namespace-Deklarationen ergänzen Sie das Formular um die Deklaration einer neuen *DrawingView*:

```
<ContentPage ...
    xmlns:toolkit=
        "http://schemas.microsoft.com/
        dotnet/2022/maui/toolkit"
    x:Class="NMGMauiTKDemo2.MainPage">

    <VerticalStackLayout>
        <toolkit:DrawingView
            Lines="{Binding MyLines}"
            LineColor="Red"
            LineWidth="5"
            DrawingLineCompleted=
                "DrawingView_DrawingLineCompleted">
```

Android

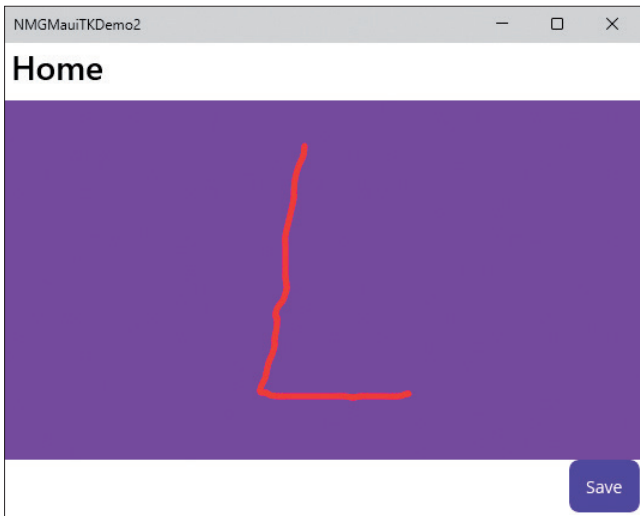
Warning

Due to the diverse landscape of Android device manufacturers and launchers, please be aware there may be unpredictable variations in how app badge counts are displayed or not displayed on different devices.

In the world of Android, one size rarely fits all. This principle holds true when it comes to setting application badge counts, due to the lack of a standardized API provided by Android system for this functionality.

Different Android launchers have chosen to implement badge counts in their unique way. Most launchers including popular ones like Nova Launcher, Microsoft Launcher, etc., have their specific methods to handle this feature.

Das Entwicklerteam warnt vor seiner eigenen Funktion (Bild 9)



Die **DrawingView** zeigt nur die zuletzt gezeichnete Linie (Bild 10)

```
BackgroundColor="BlueViolet"
MinimumWidthRequest="90"
MinimumHeightRequest="300"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" />
```

```
<Button x:Name="CmdSave" Text="Save"
        HorizontalOptions="End"
        IsEnabled="False" />
</VerticalStackLayout>
```

Hingewiesen sei darauf, dass das Entwicklerteam die Benennung der Eigenschaften verändert hat. Im Codebeispiel wird die Eigenschaft *DrawingLineCompleted* gerne noch unter dem alten Namen *OnDrawingLineCompleted* geführt – wer die für .NET 8.0 vorgesehene Version des Toolkits verwendet, muss die neue Benennung auswählen.

Dass der Button zunächst abgeschaltet ist (*IsEnabled = "False"*) ist insofern sinnvoll, als diese Funktion erst nach dem Einpflegen von Grafikinformationen in der *DrawingView* freigegeben werden soll. Hierzu reicht es aus, den Eventhandler nach folgendem Schema zu adaptieren:

```
private void DrawingView_DrawingLineCompleted(
    object sender, CommunityToolkit.Maui.Core.
    DrawingLineCompletedEventArgs e)
{
    CmdSave.IsEnabled = true;
}
```

An dieser Stelle ist das Programm für eine erste Ausführung bereit. Problematisch ist im gegenwärtigen Zustand noch, dass das Schreiben von mehreren Buchstaben nicht funktioniert – Bild 10 zeigt, dass in der derzeitigen Konfiguration immer nur die zuletzt gezeichnete Linie in der *DrawingView* sichtbar ist.

Dieses Problem ist jedoch leicht zu beseitigen: Wir setzen einfach die Eigenschaft *IsMultiLineModeEnabled* auf *True*:

```
<toolkit:DrawingView
...
    DrawingLineCompleted=
        "DrawingView_DrawingLineCompleted"
    BackgroundColor="BlueViolet"
    IsMultiLineModeEnabled="True"
```

Im nächsten Schritt geht es darum, den Eventhandler anzupassen, der nach dem Anklicken des *Save*-Buttons ausgeführt wird:

```
private async void CmdSave_Clicked(...)
{
    var cts = new CancellationTokSource();
    var stream = await DrawingView.GetImageStream(
        DrawView.Lines, new Size(DrawView.Height,
        DrawView.Width), Colors.Gray.AsPaint(), cts.Token);
```

Die erste Aktion des Codes ist das Umwandeln der in der *DrawingView* angelegten Bildinformationen in einen Stream. Zu beachten ist, dass die Dokumentation in diesem Bereich veraltet ist – es ist unbedingt erforderlich, Höhe und Breite über ein *Size*-Objekt anzuliefern. Der *Colors*-Parameter kümmert sich um die Hintergrundfarbe.

Zu beachten ist außerdem, dass die in der Klasse vorliegenden Informationen als „Linienzüge“ betrachtet werden und die auf dem Bildschirm angezeigten Inhalte nicht als Bitmap zur Verfügung stehen. Stattdessen ist immer die Methode *GetImageStream* erforderlich.

Im nächsten Schritt folgt der von Microsoft bereitgestellte Code, der sich um das Speichern der Ergebnisse im Dateisystem und die Analyse der aufgetretenen Ereignisse kümmert:

```
var fileSaverResult =
    await FileSaver.Default.SaveAsync(
        "", "test.bmp", stream, cts.Token);
if (fileSaverResult.IsSuccessful)
{
    await Toast.Make(
        $"The file was saved successfully to location:
        {fileSaverResult.FilePath}").Show(cts.Token);
}
else
{
    await Toast.Make($"The file was not saved successfully
        with error: {fileSaverResult.Exception.Message}")
        .Show(cts.Token);
}
```

Zu beachten ist, dass sich der *FileSaver* auch um das Heraus-schreiben der Informationen kümmert. Vor dem Aufruf der Methode *SaveAsync* ist es erforderlich, die im Dateisystem abzulegenden Informationen in Form eines .NET-Streams bereitzuhalten.

Bevor sich das Beispielprogramm ausführen lässt, muss dem Steuerelement im XAML-Code noch ein Name zugewiesen werden, falls das noch nicht geschehen ist: ►

```
<toolkit:DrawingView
...
x:Name="DrawView" />
```

Danach lässt sich das Programmbeispiel unter Windows ausführen. Es erscheint der bekannte Dialog, der den Benutzer zum Auswählen eines Speicherorts auffordert. Das manuelle Öffnen der Datei im Preview-System führt dann zum in **Bild 11** gezeigten Ergebnis.

Interessanter ist die Frage, wie sich das Programm unter Android verhält. Visual Studio blendet im Bezug auf die Methode *SaveAsync* eine Warnung ein, die als Mindest-API-Level 26 avisiert – die zu Beginn ausgesprochene Feststellung, dass das Toolkit Android ab Version 5 unterstützt, ist also nur unter Vorbehalt gültig. Immerhin präsentiert ein unter Android 14 laufender Emulator nach dem Anklicken das in **Bild 12** abgedruckte Ergebnis.

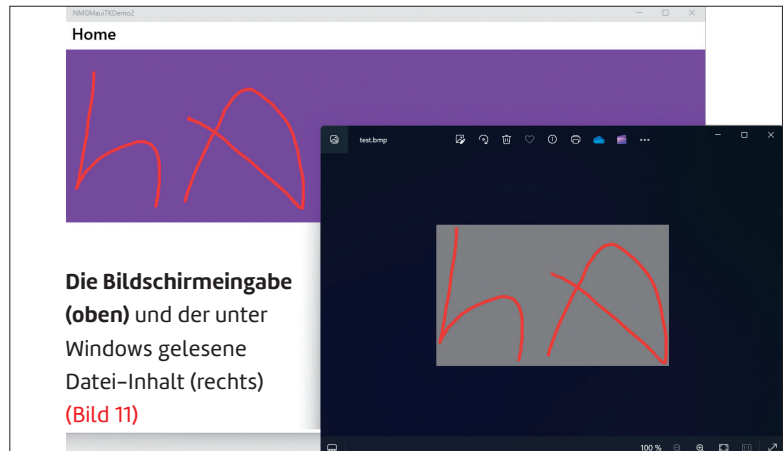
Ordnerauswahl

Während sich die *FileSaver*-Klasse um das Ablegen von Elementen im Dateisystem kümmert, steht mit dem *FolderPicker* eine Klasse zum Öffnen von Verzeichnissen zur Verfügung. Ihre Nutzung setzt auf den Handcomputerbetriebssystemen ebenfalls Modifikationen im Bereich der Permission-Struktur voraus, die hier allerdings nicht besprochen werden. Zur Aktivierung des neuen Helferleins ist es erforderlich, einen weiteren Button in der XAML-Datei anzulegen. Da sein Verhalten nicht von der *DrawingView* abhängig ist, ist es nicht notwendig, das Attribut zur Deaktivierung zu verwenden:

```
<Button x:Name="CmdLoad" Text="Load"
HorizontalOptions="End"
Clicked="CmdLoad_Clicked" />
```

Analog zur weiter oben implementierten Speicherung ist es auch hier notwendig, einen asynchronen Eventhandler mit dem Button zu verbinden:

```
private async void CmdLoad_Clicked(...)
{
    var cts =
        new CancellationSource();
    var result = await
        FolderPicker.Default.PickAsync(
            cts.Token);
    if (result.IsSuccessful) {
        await Toast.Make(
            $"The folder was picked:
            Name - {result.Folder.Name},
            Path - {result.Folder.Path}",
            ToastDuration.Long).Show(
                cts.Token);
    }
}
```



```
else
{
    await Toast.Make(
        $"The folder was not picked with error:
        {result.Exception.Message}").Show(cts.Token);
}
}
```

Unter Windows führt das Anklicken des Buttons zum Erscheinen des Dialogs, in dem der Benutzer einen beliebigen Ordner auswählen darf. Unter Android ist das Systemverhalten insofern interessant, als sich nicht alle Ordner auch ansprechen lassen. Das Betriebssystem weist darauf allerdings hin, siehe **Bild 13** und **Bild 14**.

Und jetzt mit Avatar

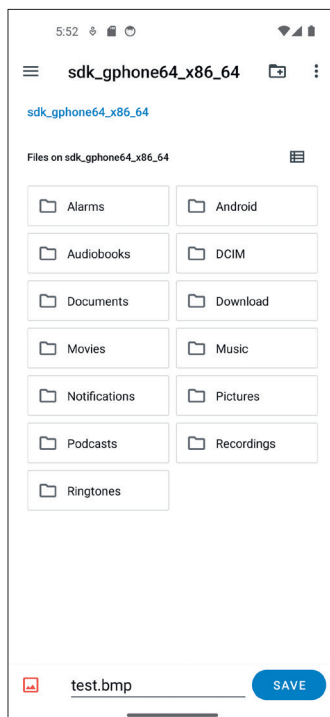
Zu guter Letzt soll in diesem Artikel noch ein hilfreiches Steuerelement folgen: Mit *AvatarView* steht ein Widget zur Verfügung, das auf die Anzeige von Avatar-Informationen spezialisiert ist:

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

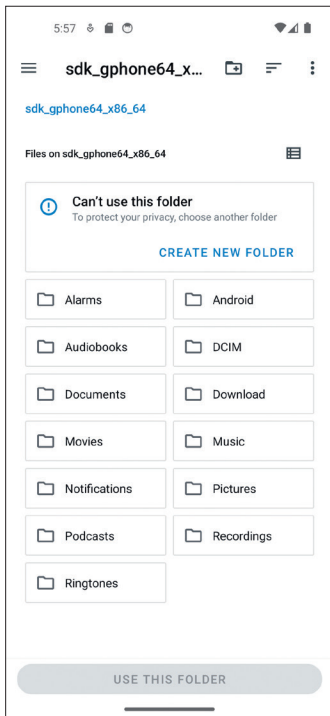
<StackLayout>
    <toolkit:AvatarView Text="TH"
        BackgroundColor="Blue" />
</StackLayout>
```

Das Ergebnis unserer Bestrebungen ist dann das Erscheinen des in **Bild 15** gezeigten Programmverhaltens. Nützlich ist die Möglichkeit, die im Avatar-Steuerelement anzuzeigenden Informationen aus einem URL zu beziehen. Hierzu genügt es, die *ImageSource* nach folgendem Schema festzulegen:

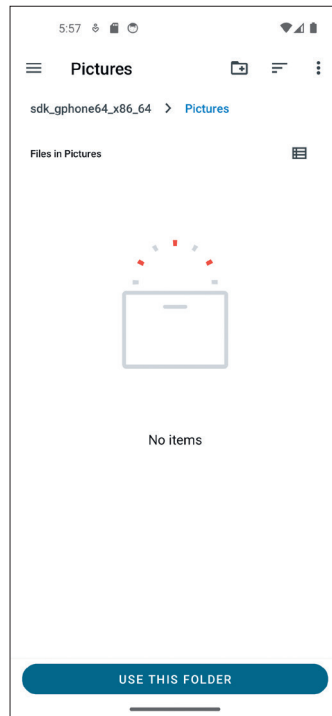
```
<toolkit:AvatarView ImageSource=
    "https://aka.ms/campus.jpg"
    Text="IS"
/>
```



Android 14: Das MAUI Community Toolkit erleichtert das Speichern von Dateien erheblich (**Bild 12**)



Ist ein Ordner nicht ansprechbar, zeigt Android eine Warnung an (Bild 13)



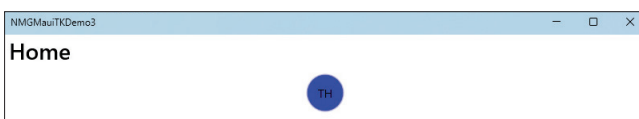
In ansprechbaren Ordnern verhält sich der FolderPicker wie ein CommonDialog (Bild 14)

Die Toolkit-Beispielapplikation

Das MAUI Community Toolkit in einem einzelnen Artikel vorstellen zu wollen ist illusorisch. In der Vergangenheit begegneten die Entwickler des Toolkits dieser Problematik durch Beispiel-Apps, die sie in den App Stores der jeweiligen Plattformen vertrieben. Dies ist insofern unkomfortabel, als Änderungen im Toolkit-Code das erneute Hochladen der App erfordern. Besitzer aktueller Versionen von Visual Studio können dieses Problem allerdings bequem umgehen – nach dem Start der *Welcome Experience* wählen Sie in der Rubrik *Los geht's* die Option *Repository klonen* aus, um den Assistenten zum „Rehydrieren“ von in einer Versionsverwaltung abgelegtem Code zu aktivieren.

Im Fall des Beispielcodes lautet der korrekte URL <https://github.com/CommunityToolkit/Maui.git> – nach der Eingabe parametrisiert Visual Studio den Projektspeicherort automatisch.

Wichtig: Für die Nutzung der GitHub-Rehydrationsfunktion sind ausschließlich URLs nach dem Schema <https://github.com/CommunityToolkit/Maui.git> zulässig, also mit der Endung *.git* ausgestattete URLs. Wer einen Pfad benutzt wie etwa <https://github.com/CommunityToolkit/Maui/tree/main>, wird von Visual Studio mit einer Fehlermeldung abgespeist.



Avatar: Die Initialen des Anwenders erscheinen mit rundem Hintergrund (Bild 15)

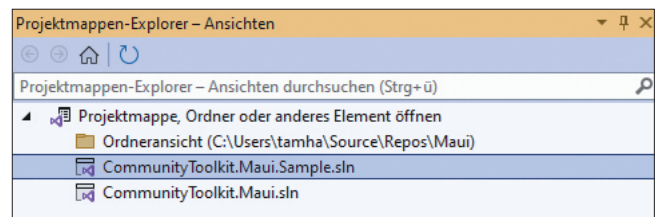
Hat alles geklappt, fordert der Projektmappen-Explorer zur Auswahl der gewünschten Solution auf, wie in **Bild 16** zu sehen ist. Für unsere Anforderungen lautet die korrekte Auswahl *Sample.sln* – danach wird das Projekt geladen und lässt sich ausführen.

Bei der Rehydrierung von in GitHub enthaltenen Projekten kommt es mitunter zu Fehlern, die auf das gescheiterte Auflösen der verwendeten .NET-Version hinweisen. Weitere Informationen zur Behebung dieses Fehlers finden Sie unter [6]. Normalerweise ist das Programm danach zur Ausführung bereit und erlaubt das Experimentieren mit den diversen Komponenten.

Fazit

Mit dem MAUI Community Toolkit bietet die Entwicklergemeinschaft verschiedene Komponenten an, die bei der Arbeit mit MAUI-Projekten hilfreich sind. Obwohl dieser Artikel nur einen vergleichsweise kleinen Teil der Möglichkeiten vorstellen konnte, gilt, dass der Werkzeugkasten nicht auf MVVM beschränkt ist – wer nur diese Funktion nutzen würde, lässt wertvolle Features ungenutzt. ■

- [1] .NET MAUI, www.dotnetpro.de/SL2410MAUICommKit1
- [2] GitHub, .NET MAUI Community Toolkit, www.dotnetpro.de/SL2410MAUICommKit2
- [3] MS Learn, .NET MAUI Community Toolkit, www.dotnetpro.de/SL2410MAUICommKit3



Visual Studio 2022 kommt nicht ins Straucheln, wenn ein Repository mehrere Solutions enthält (Bild 16)

- [4] Snackbar plattformspezifisch einrichten, www.dotnetpro.de/SL2410MAUICommKit4
- [5] Samsung Badge-Provider, www.dotnetpro.de/SL2410MAUICommKit5
- [6] Unable to resolve the .NET SDK Version, www.dotnetpro.de/SL2410MAUICommKit6



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

A2410MAUICommKit

MULTITHREADING

C#-Locking

Damit parallel laufende Threads nicht zu falschen Ergebnissen führen.



Multithreading ist immer ein spannendes Thema. Der Begriff – häufig auch als Nebenläufigkeit bezeichnet – steht in der Informatik für das gleichzeitige (oder quasi-gleichzeitige) Abarbeiten mehrerer Threads (Ausführungsstränge) innerhalb eines einzelnen Prozesses oder eines Tasks [1]. Im Gegensatz zum Multitasking, bei dem mehrere unabhängige Programme voneinander abgeschottet quasi-gleichzeitig ausgeführt werden, sind die Threads eines Anwendungsprogramms nicht voneinander abgeschottet und können somit durch sogenannte Race Conditions Fehler verursachen.

Wenn sich mehrere Threads gleichzeitig um eine Ressource streiten, kommt deshalb das Thema Locking mit C# ins Spiel. Doch was steckt eigentlich hinter einem solchen Lock, und was ändert sich mit der neuesten C#-Version 13?

Warum Locking eine Rolle spielt

Unter einem Lock oder Locking (englisch für Sperre oder Sperren) versteht man in der Informatik das Sperren des Zugriffs auf ein Betriebsmittel [2]. Eine solche Sperre ermöglicht den exklusiven Zugriff eines Prozesses auf eine Ressource, es wird garantiert, dass kein anderer Prozess diese Ressource liest oder verändert, solange die Sperre besteht.

Das Grundproblem beim Thema Locking soll anhand der Beispielsklasse *Konto* (Bild 1) demonstriert werden. Die Klasse besitzt ein öffentliches Feld namens *kontostand* vom Typ *int*. Die Beispielanwendung startet über den *ThreadPool* zwei Aufrufe derselben Methode *Konto.Manipulate*, welche den Kontostand kontinuierlich erhöht. In einer idealen Welt würde die Methode dazu führen, dass der Kontostand auf eine Million anwächst.

Da die Methode zwei Mal aufgerufen wird, müsste der Kontostand, nachdem beide Aufrufe abgearbeitet sind, auf zwei Millionen ansteigen. So weit die in einer idealen Welt erdachte Theorie.

Startet man die Beispielanwendung mehrfach, kommt es bei fast jedem Durchlauf zu einem anderen Ergebnis. Bild 2 zeigt ein Beispiel eines Durchlaufs. Die erwarteten zwei Millionen sind nicht dabei. Im Gegenteil: Die Ergebnisse sehen so aus, als würde die Anwendung bei ihrer Ausführung mit einem Würfel hantieren.

Der Grund für dieses Verhalten liegt darin, dass beide Threads ohne jede Synchronisation dasselbe Datenfeld ändern. Das Inkrementieren eines Integer-Wertes (*kontostand++*) lässt sich (etwas vereinfacht) herunterbrechen auf die folgenden Operationen:

- Der aktuelle Wert wird ausgelesen.
- Der Wert wird erhöht.
- Der neue Wert wird in den Speicherbereich zurückgeschrieben.

Dadurch, dass wir bei Threading nie wissen, welcher Thread am Zug ist, und es auch nicht sicher ist, dass die beiden überhaupt gleichzeitig laufen, kann es zu Unterbrechungen in der Abarbeitung kommen. Einem Thread kann CPU-Zeit zugeordnet werden, er kann aber auch kurzzeitig angehalten werden. Dies führt dazu, dass keine Operation atomar ist und es im Beispiel zu uner-

```

2  Konto meinKonto = new Konto();
3  ThreadPool.QueueUserWorkItem(_ => meinKonto.Manipulate());
4  ThreadPool.QueueUserWorkItem(_ => meinKonto.Manipulate());
5  Console.ReadLine();
6  Console.WriteLine($"Neuer Kontostand: {meinKonto.kontostand}");
7
8  class Konto
9  {
10     public int kontostand = 0;
11     public void Manipulate()
12     {
13         for (int i = 0; i < 1_000_000; i++)
14         {
15             kontostand++;
16         }
17     }
18 }

```

Die Beispielsklasse *Konto* mit dem öffentlichen Feld *Kontostand* vom Typ *Integer* (Bild 1)

```

Debug  LockSample x
[Icons] Threads & Variables Console
Neuer Kontostand: 645652

```

Unerwartete und unterschiedliche Ergebnisse bei jedem Programmdurchlauf (Bild 2)

klärlichen Ergebnissen kommen kann. Das Gemeine daran: Niemand warnt den Entwickler davor. Er muss solche Probleme selbst erkennen und beheben.

Lösungen

Je nach Problemstellung gibt es für solche Szenarien unterschiedliche Lösungsansätze. Und für manche Spezialfälle gibt es sogar sehr einfache Lösungen. Für ein Integer-Feld wie in der Beispielklasse hat Microsoft die Klasse *Interlocked* geschaffen, welche das Inkrementieren des Integers atomar ausführt. Bereits eine kleine Anpassung des Codes der Klasse führt sofort zur optimalen Lösung. **Bild 3** zeigt die zusätzliche Zeile *Interlocked.Increment(ref kontostand)*, die dazu führt, dass das Problem nicht mehr auftritt. In dieser Programmfassung wird der Kontostand korrekt berechnet und erreicht die erwartete Zielsumme von 2 Millionen.

Doch so schön diese Lösung auch sein mag, sie garantiert nur die „korrekte“ Abarbeitung. Es ist hier durchaus fraglich, ob eine Parallelisierung mit mehreren Threads für eine schnellere Abarbeitung sorgt, da sich in diesem Spiel zwei Threads um diesen Integer streiten (Thema: Parallelisierung ist nicht in jedem Fall sinnvoll).

```

12 public void Manipulate()
13 {
14     for (int i = 0; i < 1_000_000; i++)
15     {
16         Interlocked.Increment(ref kontostand);
17     }
18 }

```

Debug LockSample x

Neuer Kontostand: 2000000

Interlocked behebt das Problem für Integer-Werte und liefert das erwartete Ergebnis (**Bild 3**)

Außerdem: Für Integer-Felder ist die Klasse *Interlocked* die perfekte Lösung, allerdings hilft sie nicht weiter, wenn der Entwickler es mit komplexeren Datentypen oder eigenen Klassen zu tun hat.

Der Hammer: lock

Ein universeller „Hammer“, wie ich es gerne umschreibe, ist das Schlüsselwort *lock* in C#. Es erlaubt unabhängig vom Datentyp, dass der Entwickler einen sogenannten Sperrbereich definiert, in den jeweils nur ein Thread „eintreten“ und darin seinen Code ausführen darf.

Die grundsätzliche Verwendung ist hierbei, dass mittels des Schlüsselworts *lock* ein .NET-Objekt als Schlüsselobjekt ausgewählt und dieses abgesperrt wird. Ein zweiter Thread kann den gesperrten Bereich so lange nicht betreten, bis dieser wieder freigegeben wird. In **Bild 4** sehen Sie den entspre-

```

10 private object _syncObject = new object();
11 public int kontostand = 0;
12 public void Manipulate()
13 {
14     for (int i = 0; i < 1_000_000; i++)
15     {
16         lock (_syncObject)
17         {
18             kontostand++;
19         }
20     }
21 }

```

Debug LockSample x

Neuer Kontostand: 2000000

Lock sperrt den Zugriff auf den Kontostand und führt ebenfalls zum richtigen Endergebnis (**Bild 4**)

chend modifizierten Code der Beispielanwendung. Ein Programmlauf beweist auch hier wieder, dass damit der korrekte Kontostand ermittelt wird.

Aufgrund des Locks können wir Entwickler nun innerhalb des Sperrbereichs mit den Daten arbeiten, wie wir wollen, da garantiert wird, dass der aktuelle Thread der einzige ist, welcher den Bereich betreten darf. Sollte ein weiterer Thread den Sperrbereich betreten wollen, so wird dieser in der Zeile des *lock* festhängen, bis die Sperre freigegeben wurde.

Die Common Language Runtime (CLR) von .NET sorgt dafür, dass ein Lock immer speziell für einen Thread gilt, weshalb auch ein doppeltes Versperren möglich ist (**Bild 5**). Das ist übrigens auch der Grund dafür, dass sich *async* und *await* mit *lock* nicht gut vertragen, da nach einem *await* eventuell ein ►

```

17 lock (_syncObject)
18 {
19     lock (_syncObject)
20     {
21         kontostand++;
22     }
23 }

```

Debug LockSample x

Neuer Kontostand: 2000000

Ein doppelter Lock (**Bild 5**)

Threadwechsel stattfindet und somit ein anderer Thread das gesperrte Objekt wieder freigibt, das er gar nicht hat. Damit das nicht passiert, quittiert der Compiler den Versuch, *lock* mit *async* und *await* zu kombinieren, ebenfalls mit einer Fehlermeldung.

Hinter den Kulissen

Spannend ist, was konkret passiert, wenn man den Lock-Befehl benutzt. Der Compiler setzt den Lock (bisher) immer mit der Klasse *System.Threading.Monitor* um. Diese beherbergt die Magie, um den Sperrbereich korrekt zu realisieren. In einem *try/finally*-Block wird versucht, diese Sperre über die *Monitor*-Klasse abzubilden und beim Verlassen wieder freizugeben, siehe **Bild 6**. Unter der Haube wird dabei im Fall von Windows die Funktion *EnterCriticalSection* aus dem Win32-API [3] genutzt, wie der Code-Schnipsel in **Bild 7** zeigt. Das bedeutet: Eigentlich liefert das Betriebssystem die für einen Lock erforderliche Funktion.

```
object syncObject = this._syncObject;
bool lockTaken = false;
try
{
    Monitor.Enter(syncObject, ref lockTaken);
    ++this.kontostand;
}
finally
{
    if (lockTaken)
        Monitor.Exit(syncObject);
}
```

Die Klasse *System.Threading.Monitor* setzt den Lock um ...
(Bild 6)

C# 13 ändert etwas ...

Mit C# 13 ändert sich genau in diesem Zusammenhang nun etwas. Microsoft hat einen neuen Datentyp namens *Lock* eingeführt. Einerseits soll er die Lesbarkeit verbessern, da gerade die Signatur unseres Feldes *object syncobject* eine sehr übliche Form ist, aber es purzelt ein einfaches *System.Object*

```
private void AcquireCore()
{
    Interop.Kernel32.EnterCriticalSection(&pMonitor->_criticalSection);
}

private void ReleaseCore()
{
    Interop.Kernel32.LeaveCriticalSection(&pMonitor->_criticalSection);
}
```

... verwendet wird dabei die Funktion *EnterCriticalSection* aus dem Win32-API (Bild 7)

```
Lock.Scope scope = this._syncObject.EnterScope();
try
{
    ++this.kontostand;
}
finally
{
    scope.Dispose();
}
```

Ein Objekt sperren mit C#13 und der neuen Lock-Klasse (Bild 8)

einfach so herum, was sich 2024 doch etwas „falsch“ anfühlt. Mit dem Datentyp *Lock* kann einerseits ganz klar ausgedrückt werden, dass es sich hier um ein *Lock*-Objekt handelt. Der betreffende Code im Beispiel sieht damit dann so aus:

```
private Lock _syncObject = new Lock();
```

Andererseits ist es nun nicht mehr zwangsweise so, dass der Compiler immer die *Monitor*-Klasse verwendet, sondern er kann mit C# in Version 13 auch auf alternative Wege zurückgreifen.

Sieht man sich den erzeugten IL-Code an, so erkennt man, dass im Fall von .NET 9 und dem neuen Datentyp *Lock* die Methode *EnterScope* verwendet wird und das Objekt am Ende in einem *try-finally*-Block mit *Dispose* wieder freigegeben wird (Bild 8). Die neue *Lock*-Klasse stützt sich dabei also nicht mehr auf die *Monitor*-Klasse (die das Win32-API nutzt), sondern verschiebt die Sperrlogik in den .NET-Bereich – weg vom Betriebssystem.

Fazit

Locking ist ein Thema, welches bei Multithreading immer aufkommt, und das Thema ist nicht nur alt, sondern auch unangenehm komplex. Aus diesem Grund wurde hier noch einmal klargestellt, warum Locking ein wichtiges Thema für Entwickler ist, und vor allem, dass Microsoft hier eine Änderung in C# 13 einführt. ■

[1] Multithreading, www.dotnetpro.de/SL2411NETirol1

[2] Lock, www.dotnetpro.de/SL2411NETirol2

[3] Die Funktion *EnterCriticalSection* aus dem Win32-API, www.dotnetpro.de/SL2411NETirol3



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

A24011NETirol



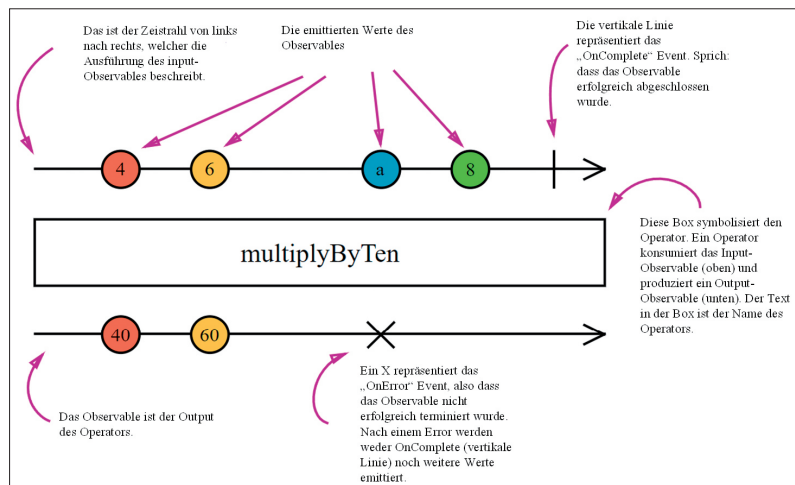
REAKTIV PROGRAMMIEREN, TEIL 2

LINQ für Events

Die wichtigsten Operatoren für die Praxis.

Der erste Teil dieser Artikelserie [1] hat die Open-Source-Bibliothek Rx.NET als Implementierung der Reactive Extensions [2] vorgestellt. Sie ist eine Implementierung des Observer-Design-Patterns, das die Programmierung Push-basierter Benachrichtigungen erleichtert. Rx.NET [3] kann dazu verwendet werden, native .NET-Events zu ersetzen, die mit einigen Schwächen behaftet sind. Dabei wird ein funktionaler Ansatz verfolgt.

Inspiziert vom Objekt *IEnumerable* wird das Objekt *IObservable* verwendet. Dieses informiert über neue Ereignisdaten (*OnNext*), ob es abgeschlossen ist (*OnCompleted*) oder ob ein Fehler aufgetreten ist (*OnError*). Es werden so lange neue Daten per *OnNext* emittiert, bis *OnCompleted* oder *OnError* aufgerufen wird.



Gängige Notation für Observables und Operatoren (Bild 1)

Ein *Observer*-Objekt wird bei einem *IObservable* registriert und verarbeitet die Daten ähnlich wie ein von herkömmlichen Anwendungen bekannter Eventhandler. Ein *Subject* ist beides: sowohl ein *Observable* als auch ein *Observer*. Das bedeutet, dass ein *Subject* sowohl Daten empfangen (*Observer*) als auch Daten an seine Abonnenten weitergeben kann (*Observable*). Deshalb werden *Subjects* in der Praxis genutzt, um Daten manuell zu emittieren.

Zwischen *Observer* und *Observables* befinden sich die modularen Operatoren, die nach dem Vorbild von LINQ beliebig aneinandergereiht werden können. In diesem zweiten Teil der Artikel-Trilogie über Rx.NET sollen die wichtigsten Bausteine zur Programmierung mächtiger Event-Ketten vorgestellt werden.

Operator-Notation

Operatoren lassen sich nur schwer mit Worten beschreiben. Deswegen hat sich eine gängige Diagramm-Notation etabliert, die man häufig in Dokumentationen antrifft. Bild 1 zeigt, wie ein *Observable* durch einen Operator *multiplyByTen* in ein anderes *Observable* umgewandelt wird.

Ein *Observable* wird als Zeitstrahl von links nach rechts dargestellt. Die einzelnen Daten, die per *OnNext* geliefert werden, sind als Elemente auf dem Zeitstrahl aufgetragen (hier 4, 6, a, 8.). Am Ende eines *Observables* kann eine vertikale Linie für das *OnComplete*-Ereignis stehen. Alternativ symbolisiert ein X, dass es zu einem Fehler kam (*OnError*). Nach *OnComplete* beziehungsweise *OnError* werden keine Daten mehr geliefert. Es gibt auch *Observables*, die weder *OnComplete* noch *OnError* erzeugen. Diese würden unendlich lang weitere Daten liefern.

Aktiv werden

RxMarbles [4] bietet eine schöne Möglichkeit, interaktiv mit Operatoren zu experimentieren. Die Website ist zwar auf RxJS fokussiert, aber die gängigen Operatoren gibt es sowohl in Rx.NET als auch in RxJS. Es lohnt sich daher auch für Rx.NET-Anwender, einen Blick darauf zu werfen. Auf der Website RxJS Marbles können die Events, welche – wie bei der gerade besprochenen Notation – als Kugeln dargestellt werden, per Drag-and-drop zeitlich verschoben werden, und man sieht auf der unteren Leiste in Echtzeit, wie sich das Ergebnis verändert (Bild 2).

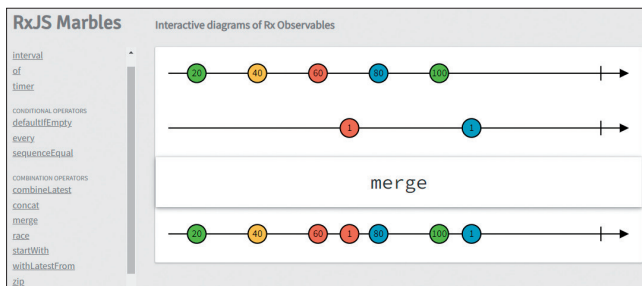
So richtig interaktiv wird es mit RxJS-fruits [5] von Gregor Biswanger. Diese Website richtet sich ebenfalls in erster Linie an die Java-

Script/TypeScript-Version von Rx, ist aber auch für Rx.NET-Entwickler sehr interessant, um erste Gehversuche zu wagen. In 16 kurzweiligen Levels besteht die Aufgabe darin, Obst (sinnbildlich für Events auf dem *Observable*) per Operatoren zu filtern, zu manipulieren, zu vereinen et cetera (Bild 3).

Tipp: Auch nach einiger Erfahrung in Rx.NET und dessen Operatoren kann man unerwartete Überraschungen erleben. Es empfiehlt sich, ein kleines Konsolenprojekt zum Experimentieren zu erstellen.

Die wichtigsten Operatoren

In Rx gibt es Dutzende Operatoren, die vergleichbar mit LINQ beliebig hintereinandergereiht werden können. Die Operatoren ermöglichen es, mit nur wenig Code sehr ►



RxJS Marbles: Website zum interaktiven Experimentieren mit Operatoren (Bild 2)

mächtigen Verarbeitungsketten zu definieren. Auf der Website von ReactiveX [6] werden einige dieser Operatoren zusammengetragen. Um alles aus Rx.NET herauszuholen, ist es von Bedeutung, die wichtigsten Operatoren zu kennen und diese sicher einsetzen zu können. In **Tabelle 1** finden Sie eine Übersicht der Operatoren. Nachfolgend werden die gemäß der Erfahrung des Autors in der Praxis relevantesten Operatoren vorgestellt. Für alle dabei verwendeten Code-Sequenzen wird die Extension-Methode aus **Listing 1** vorausgesetzt.

Where (Find)

Where filtert alle Events mit dem übergebenen Predicate. In Rx-Implementierungen für andere Programmiersprachen wird *Where* [7] oftmals als *Filter* bezeichnet. Da in .NET aber *Where* durch LINQ bereits etabliert war, heißt *Filter* in Rx.NET *Where*.

Achtung: Durch die Bezeichnung *Where* ist Rx.NET zwar konsistent zu LINQ, aber gleichzeitig kann man bei komplexeren Queries, insbesondere dann, wenn man eine Collection aus Observables verarbeitet, nicht mehr klar erkennen, ob es sich um ein *Where* auf einer Collection oder um einen Rx.NET-Operator handelt. Hier empfiehlt es sich, im Code einen Kommentar zu spendieren, wie im folgenden Code-Schnipsel gezeigt wird. Darin wird eine Sequenz der Zahlen von 0 bis 5 nach ungeraden Werten gefiltert:

```
Observable.Range(0, 5)
    .Where(x =>
        x % 2 == 1)
    // IObservable<int>
    .Print();
```

Ausgabe:
OnNext(1)
OnNext(3)
Completed

Das Beispiel verwendet *Observable.Range(start, count)* – ein sogenanntes Cold-Observable (mehr dazu später), das zum Zeitpunkt der Registrie-

rung Werte ab dem Parameter *start* mit der Anzahl *count* ausgibt und abschließend *completed*.

Select (Map)

Select projiziert alle Events mit der übergebenen Transformations-Funktion. In anderen Rx-Implementierungen wird *Select* oftmals als *Map* [8] bezeichnet. Da in .NET aber *Select* durch LINQ bereits etabliert war, heißt *Map* auch in Rx.NET *Select*.

Achtung: Analog zum Hinweis zu *Where* ist es auch bei *Select* ratsam, einen Kommentar anzufügen. Im folgenden Beispiel wird eine Sequenz der Zahlen von 0 bis 5 mit 10 multipliziert.

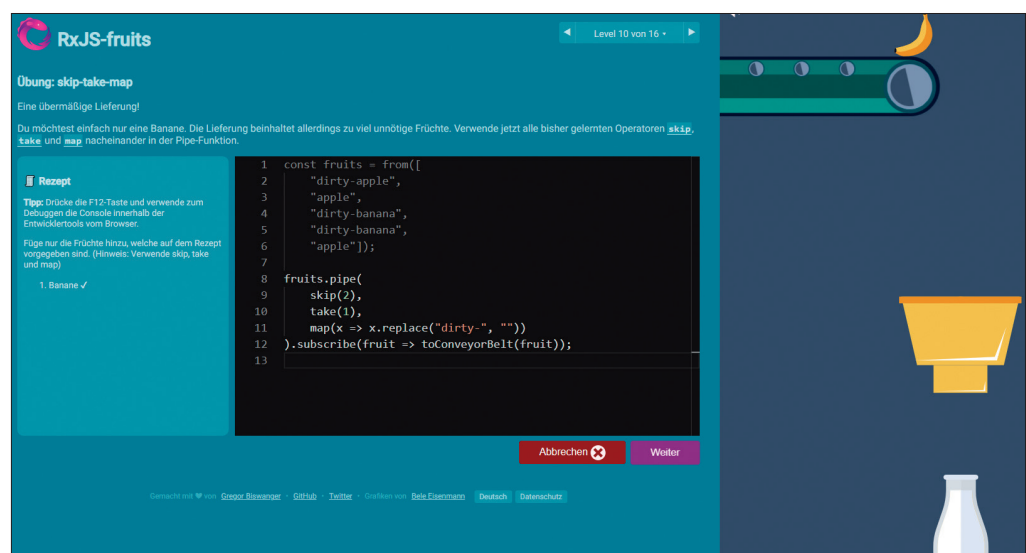
```
Observable.Range(0, 5)
    .Select(x => x * 10) // IObservable<int>
    .Print();
```

Ausgabe:
OnNext(0)
OnNext(10)
OnNext(20)
OnNext(30)
OnNext(40)
Completed

CombineLatest

CombineLatest ist ein sehr nützlicher und in der Praxis häufig anzutreffender Operator. Er dient dazu, mehrere Observables in einem einzigen Observable zusammenzufassen, vergleiche **Bild 4**. Dieser Fall tritt auf, wenn man mehrere Datenquellen verarbeitet, welche unabhängig voneinander Informationen veröffentlichen.

Im nachfolgenden Code-Schnipsel werden zunächst ein Temperatur- und ein Luftfeuchtigkeitssensor simuliert und die Ergebnisse anschließend miteinander in einem String kombiniert. Aus Platzgründen wird hier darauf verzichtet, den Heat-Index zu berechnen.



RxJS-fruits: Die Website bietet Übungsmöglichkeiten im Umgang mit Operatoren (Bild 3)

● Tabelle 1: Übersicht der Operatoren

Erzeugende Operatoren	
Return	Erstellt ein Observable, das einen einzigen Wert liefert und dann abgeschlossen wird.
Empty	Erstellt ein Observable, das sofort abgeschlossen wird, ohne einen Wert zu liefern.
Never	Erstellt ein Observable, das nie Werte liefert und nie abgeschlossen wird.
Throw	Erstellt ein Observable, das sofort einen Fehler ausgibt.
Interval	Gibt periodisch aufeinander folgende ganze Zahlen in einem angegebenen Intervall aus.
Timer	Gibt einen Wert nach einer Verzögerung aus und kann optional periodisch Werte ausgeben.
Range(Start, Count)	Erstellt ein Observable, das die in <i>Count</i> festgelegte Anzahl Werte ab <i>Start</i> ausgibt.
Filternde Operatoren	
Where	Filtert Elemente basierend auf einer Bedingung.
Distinct	Unterdrückt aufeinander folgende doppelte Elemente.
DistinctUntilChanged	Unterdrückt aufeinander folgende doppelte Elemente, erlaubt jedoch unterschiedliche aufeinander folgende Werte.
Take	Gibt die ersten n Elemente aus einer Sequenz zurück.
Skip	Überspringt die ersten n Elemente aus einer Sequenz.
TakeWhile	Gibt Elemente aus, solange eine Bedingung wahr ist.
SkipWhile	Überspringt Elemente, solange eine Bedingung wahr ist.
Transformierende Operatoren	
Select	Projiziert jedes Element einer Sequenz.
SelectMany	Projiziert jedes Element einer Sequenz in eine Observable-Sequenz und gibt die abgeflachten Ergebnisse zurück. Tipp: Eine Überladung erlaubt es, für jedes Element einen Task auszuführen und die Ergebnisse in Form eines Observables zu verarbeiten.
Buffer	Gruppert Elemente in Arrays der angegebenen Größe.
Window	Teilt eine Observable-Sequenz in mehrere kleinere Observable-Sequenzen auf.
Kombinierende Operatoren	
Merge	Kombiniert mehrere Observable-Sequenzen in eine einzige.
Zip	Kombiniert mehrere Observable-Sequenzen, indem es jeweils ein Element aus jeder Sequenz in ein Element zusammenfasst.
CombineLatest	Kombiniert die neuesten Werte aus mehreren Observable-Sequenzen.
Join	Kombiniert Elemente aus zwei Observable-Sequenzen basierend auf Selektoren.
GroupJoin	Kombiniert Elemente aus zwei Observable-Sequenzen basierend auf einem Schlüssel und gruppiert die Ergebnisse.
Switch	Wechselt zu einer neuen Observable-Sequenz, wenn eine neue Observable-Sequenz emittiert wird, und deregistriert die vorherige.
Fehlerbehandlung	
OnErrorResumeNext	Wirft ein Observable eine Exception, wird mit dem nachfolgenden Observable fortgefahren.
Catch	Verhält sich wie <i>OnErrorResumeNext</i> , bietet aber die Möglichkeit, auf den Fehler zu reagieren.
Retry	Wiederholt die Ausführung eines Observables bei Fehlern so lange, bis sie erfolgreich beendet ist.
Zeitverhalten	
Interval	Erzeugt eine Sequenz, die nach jedem Zeitraum einen aufsteigenden Wert produziert.
Delay	Verschiebt die beobachtbare Sequenz um die angegebene Zeitdauer. Die Zeitabstände zwischen den Werten bleiben erhalten.
Throttle	Verwirft Elemente so lange, bis eine gewisse Zeit keine Elemente ausgegeben werden (Beispiel: Autocomplete)
Nebenläufigkeit	
SubscribeOn	Bestimmt, auf welchem Scheduler die Logik des Registrierens und Deregistrierens ausgeführt wird.
ObserveOn	Bestimmt, auf welchem Scheduler Beobachtungen empfangen werden.
SelectMany	Kann dazu genutzt werden, um Tasks für jedes Element in der Sequenz auszuführen.

```
var temperatureSensor = new Subject<double>();
var humiditySensor = new Subject<double>();
```

```
Observable
```

```
.CombineLatest(temperatureSensor,
    humiditySensor)
.Select(x => $"temperature = {x[0]}°C ;
    humidity = {x[1]}%")
.Print();
```

```
temperatureSensor.OnNext(35);
temperatureSensor.OnNext(28);
```

```
humiditySensor.OnNext(45);
humiditySensor.OnNext(55);
```

```
temperatureSensor.OnNext(25);
temperatureSensor.OnCompleted();
humiditySensor.OnCompleted();
```

```
Ausgabe:
```

```
OnNext(temperature = 28°C ; humidity = 45%)
OnNext(temperature = 28°C ; humidity = 55%)
OnNext(temperature = 25°C ; humidity = 55%)
Completed
```

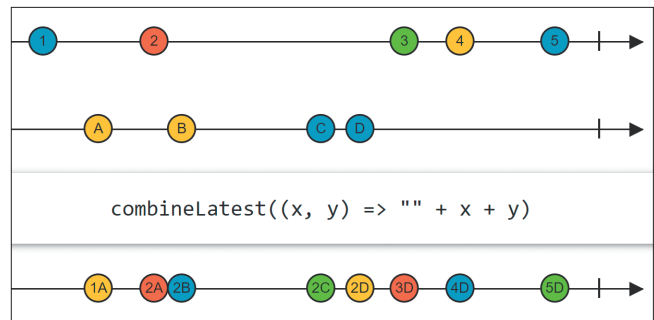
Der erste Temperaturwert (35) wird hier nicht ausgegeben, da der Operator *CombineLatest* so lange wartet, bis alle Observables mindestens einen Wert geliefert haben.

Wie findet man den passenden Operator?

Es ist wichtig, den zur Aufgabe passenden Operator zu finden, um Rx.NET maximal effizient einsetzen zu können. Hierzu gibt es für Rx.JS einen interaktiven Entscheidungsbaum [9], mit dem man durch Beantwortung einfacher Fragen den richtigen Operator finden kann. Ein nicht interaktiver Entscheidungsbaum existiert auch für Rx [10].

● Listing 1: PrintExtension

```
public static class PrintExtension
{
    public static void Print<T>(
        this IObservable<T> source)
    {
        source.Subscribe(
            value => Console.WriteLine(
                $"OnNext({value})"),
            ex => Console.WriteLine(
                $"OnError() lead to exception:
                {ex.Message}"),
            () => Console.WriteLine("Completed"));
    }
}
```



Schematische Darstellung von CombineLatest (Bild 4)

Achtung: Bei beiden Angeboten ist zu beachten, dass man in der Dokumentation den entsprechenden Operator für Rx.NET verwendet. Gerade Rx.NET weicht von der Rx-Namensgebung einige Male ab, wie bereits bei *Map* und *Select* gesehen.

Hot- und Cold-Observables

Ein sehr wichtiges Konzept ist die Unterscheidung zwischen Hot- und Cold-Observables, auch Hot- und Cold-Sources genannt. Diese Eigenschaft beschreibt, wie sich das Observable im Zusammenspiel mit dem Subscriber verhält. Da *OnNext()* die Daten zeitlich verteilt an das Observable liefert, kann es relevant sein, zu welchem Zeitpunkt sich der Subscriber am Observable registriert.

Registriert sich der Observer bei einem Hot-Observable, bekommt der Subscriber neue Daten erst ab dem Zeitpunkt, zu dem er sich registriert hat. Das bedeutet im Umkehrschluss, dass alle Daten vor der Registrierung für den Subscriber verloren sind.

Bei Cold-Observables sieht es anders aus. Sie können Informationen intern speichern und beispielsweise zum Zeitpunkt der Registrierung ausgeben.

Hot-Observables

Typische Beispiele für Hot-Observables sind aktuelle Sensordaten, Click-Streams oder Börsenticker. Hot-Observables taugen also immer dann, wenn man nur an zukünftigen Ereignissen interessiert ist.

Das folgende Code-Beispiel simuliert einen Temperatursensor, der kontinuierlich Daten liefert. Der Subscriber registriert sich hier erst, nachdem die Temperaturen 24°C und 28°C emittiert wurden. Deswegen werden nur die danach folgenden Temperaturen 30°C und 35°C ausgegeben.

```
Subject<int> temperatureSubject = new Subject<int>();
// let's emit some values
temperatureSubject.OnNext(24);
temperatureSubject.OnNext(28);

// now we register
temperatureSubject.Select(x =>
    $"Aktuelle Temperatur: {x}°C").Print();

// continue committing data
temperatureSubject.OnNext(30);
temperatureSubject.OnNext(35);
```

● Listing 2: Await und Observables

```
var rangeObservable = Observable.Range(0, 5);

// Liefert das erste Element der Observable-Sequenz.
int result1 = await rangeObservable.FirstAsync();

// Liefert das erste Element der Observable-
// Sequenz oder einen Default-Wert.
int result2 =
    await rangeObservable.FirstOrDefaultAsync();

// Liefert das letzte Element der Observable-Sequenz.
// Das ist der Standardfall, wenn man await benutzt.
int result3_1 = await rangeObservable.LastAsync();
int result3_2 = await rangeObservable;

// Liefert das letzte Element der Observable-
// Sequenz oder einen Default-Wert.
int result5 =
    await rangeObservable.LastOrDefaultAsync();

// Ruft eine Aktionsmethode für jedes Element in
// der Observable-Sequenz auf und gibt einen Task
// zurück, der signalisiert, dass die Sequenz
// beendet ist.
await rangeObservable.ForEachAsync(
    x => Console.WriteLine(x));

var singleObservable = Observable.Return(1);
// Gibt das einzige Element der Observable-
// Sequenz zurück oder wirft eine Exception.
int result6 =
    await singleObservable.SingleAsync();

// Gibt das einzige Element der Observable-
// Sequenz oder einen Default-Wert zurück.
// Gibt es mehr als ein Element, wird eine
// Exception geworfen.
int result7 =
    await singleObservable.SingleOrDefaultAsync();
```

Bei Hot-Observables teilen sich alle Subscriber denselben Datenstrom. Ein späterer Abonnent verpasst somit all diejenigen Ereignisse, die schon aufgetreten sind, bevor er sich registriert hat.

Cold-Observables

Cold-Observables sind immer dann die richtige Wahl, wenn man unabhängig vom Zeitpunkt der Registrierung sein muss. Möchte man sich irgendwann in der Zukunft beim Temperatur-Sensor registrieren und alle vorherigen Temperaturwerte geliefert bekommen, geht das wie folgt:

```
ReplaySubject<int> temperatureSubject =
    new ReplaySubject<int>();

// let's emit some values
temperatureSubject.OnNext(24);
temperatureSubject.OnNext(28);

// now we register
temperatureSubject.Select(
    x => $"Aktuelle Temperatur: {x}°C").Print();

// continue committing data
temperatureSubject.OnNext(30);
temperatureSubject.OnNext(35);
```

Das Beispiel verwendet anstelle eines *Subjects* ein *ReplaySubject*. Dieses speichert alle bereits emittierten Werte ab. Sobald sich ein Abonnent bei einem *ReplaySubject* registriert, werden alle bereits emittierten Werte ausgegeben und danach alle zukünftigen Daten. Das Ergebnis hier wäre also

24°C, 28°C, 30°C, 35°C, unabhängig davon, dass der Observer sich erst registriert hat nachdem die Werte 24°C und 28°C emittiert wurden.

Cold-Observables erzeugen für jeden Abonnenten einen eigenen, unabhängigen Stream.

Mischform

In der Realität gibt es oftmals kein Schwarz und Weiß. Angenommen, man benötigt nach dem Registrieren beim Temperatur-Sensor sofort den aktuellen (also zuletzt gemessenen Wert) und ab diesem Zeitpunkt alle zukünftigen Messungen. Wie ist das einzuordnen?

Der Fakt, dass man sofort die aktuelle Temperatur bekommt, würde für ein Cold-Observable sprechen. Da es sich aber nur auf den letzten Wert und nicht auf die gesamte Historie bezieht, ist es weder ein reines Hot- noch ein Cold-Observable. Hier hilft ein *BehaviorSubject*:

```
BehaviorSubject<int?> temperatureSubject =
    new BehaviorSubject<int?>(null);

// let's emit some values
temperatureSubject.OnNext(24);
temperatureSubject.OnNext(28);

// now we register
temperatureSubject.Select(x =>
    $"Aktuelle Temperatur: {x}°C").Print();

// continue committing data
temperatureSubject.OnNext(30);
temperatureSubject.OnNext(35);
```

Das *BehaviorSubject* in diesem Beispiel merkt sich den zuletzt emittierten Wert und gibt diesen unmittelbar bei der Registrierung eines Observers aus. BehaviorSubjects müssen mit einem initialen Wert erzeugt werden, daher der Parameter *null*. Die Ergebnisse in diesem Beispiel wären 28°C, 30°C sowie 35°C. BehaviorSubjects sind äußerst nützlich, wenn man den aktuellen (zuletzt gemessenen) Sensorwert unmittelbar benötigt und nicht erst darauf warten kann, bis der nächste Sensorwert aufgenommen wird.

Observables in Tasks konvertieren

In .NET modelliert man nebenläufige Aktionen am besten mit Tasks, auf deren Ergebnis man mit *await* wartet. Rx.NET bietet eine Menge Wege an [11], um aus der Observable-Welt in die Task-Welt zu gelangen. Listing 2 zeigt einige Beispiele.

Tasks in Observables konvertieren

Ebenso gibt es mehrere Wege, um aus der Task-Welt in die Observable-Welt zu kommen [12]. Hierbei ist es entscheidend zu bedenken, welches Ausführungsverhalten man erzielen möchte, wenn sich ein Subscriber beim Observable registriert. Siehe dazu den Abschnitt zu Hot- und Cold-Observables.

Fall 1: Ein Task soll exakt einmal starten, und unabhängig vom Zeitpunkt der Registrierung soll das Ergebnis im Observable emittiert werden. Jeder Subscriber erhält das Ergebnis, sobald es vorliegt.

```
var observable = Task.Run(() =>
{
    Console.WriteLine("I am executing");
    return 0;
}).ToObservable();
observable.Print();
observable.Print();
```

```
Ausgabe:
I am executing
OnNext(0)
Completed
OnNext(0)
Completed
```

Daran, dass *I am executing* nur einmal in der Ausgabe erscheint, sieht man, dass der Task nur einmal ausgeführt wurde. Beide Subscriber erhalten dasselbe Ergebnis.

Fall 2: Jedes Mal, wenn sich ein Subscriber registriert, soll ein neuer, separater Task gestartet werden (Cold-Observable).

```
var observable = Observable.FromAsync(() =>
{
    Console.WriteLine("I am creating a new Task");
    return Task.Run(() =>
    {
        Console.WriteLine("I am executing");
        return 0;
    });
});
```

Listing 3: Task und Observables kombiniert

```
Task<string> download1 =
    Task.Run(async () =>
    { await Task.Delay(1000);
      return "Hallo"; });

Task<string> download2 =
    Task.Run(async () =>
    { await Task.Delay(2000);
      return "Welt"; });

var downloadResult =
    await Task
        .WhenAll(download1, download2)
        // Operation auf Task<string>[]

        .ToObservable()
        // Umwandlung in Observable<string[]>

        .Timeout(TimeSpan.FromSeconds(10))
        // Wirft eine Exception, wenn die Observable-
        // Sequenz nicht innerhalb von 10 Sekunden
        // terminiert.

        .FirstAsync();
        // Gibt ein Array mit zwei Elementen und den
        // jeweiligen Downloads zurück.

Console.WriteLine(
    $"{downloadResult[0]} {downloadResult[1]}");
```

```
observable.Print();
observable.Print();
```

```
Ausgabe:
I am creating a new Task
I am executing
OnNext(0)
Completed
I am creating a new Task
I am executing
OnNext(0)
Completed
```

Observable.FromAsync erwartet eine Task-Factory als Parameter. Diese Task-Factory wird jedes Mal aufgerufen, wenn sich ein neuer Subscriber beim Observable registriert.

Hin und zurück

Kombiniert man Tasks und Observables, kann man mächtige, nebenläufige Event-Ketten verarbeiten. Das in Listing 3 gezeigte Beispiel simuliert zwei Downloads, die beide klassisch als Task modelliert sind. Die Aufgabe besteht darin, beide

Downloads parallel zu starten, maximal zehn Sekunden zu warten und dann entweder eine Exception zu werfen oder das Ergebnis zurückzugeben.

Im Beispiel werden die beiden Tasks in ein Objekt vom Typ `Observable<string[]>` umgewandelt, dann mit dem Timeout-Operator [13] kombiniert und das Ergebnis als `download-Result-Task` zurückgegeben.

Fazit und Ausblick

In dieser zweiten von drei Serienfolgen wurden Operatoren und deren Verwendung mit Rx.NET vorgestellt. Operatoren sind die wesentliche Zutat, die es Entwicklern erlaubt, mit Rx.NET effiziente Datenverarbeitungsketten zu gestalten. Zusätzlich wurden Hot- und Cold-Observables erläutert und gezeigt, wie man diese durch die Verwendung verschiedener `Subject`-Typen erzeugt. Den Abschluss bildet der Übergang von der Observable- in die Task-Welt und zurück.

Die kommende dritte und letzte Folge der Artikelreihe wird sich mit fortgeschrittenen Themen wie Threading, Scheduling und Testen beschäftigen. ■

[1] Tim Borowski, *Reaktiv programmieren, Teil 1, Einführung in Rx.NET*, *dotnetpro* 10/2024, Seite 51 ff., www.dotnetpro.de/A2410RxNET

[2] *ReactiveX*, <https://reactivex.io>

[3] *Reactive Extensions for .NET*, www.dotnetpro.de/SL2411RxNET1

[4] *RxJS Marbles*, <https://rxmarbles.com>

[5] *RxJS-fruits*, <https://www.rxjs-fruits.com>

[6] *Operators in der Rx-Dokumentation*, www.dotnetpro.de/SL2411RxNET2

[7] *Filter-Operator*, www.dotnetpro.de/SL2411RxNET3

[8] *Map-Operator*, www.dotnetpro.de/SL2411RxNET4

[9] *Entscheidungsbaum für RxJS*, www.dotnetpro.de/SL2411RxNET5

[10] *Entscheidungsbaum für Rx*, www.dotnetpro.de/SL2411RxNET6

[11] *Weitere Informationen zu Observables*, www.dotnetpro.de/SL2411RxNET7

[12] *Weitere Informationen zum Konvertieren von Tasks*, www.dotnetpro.de/SL2411RxNET8

[13] *Timeout in der Rx-Dokumentation*, www.dotnetpro.de/SL2411RxNET9



Tim Borowski

ist freiberuflicher Softwareentwickler und -berater. Seine Schwerpunkte sind die .NET/C#-Entwicklung, Angular sowie AWS, K8s und Istio Service Mesh. Zudem ist er Fachartikelautor und Konferenzsprecher. Sie erreichen ihn unter tim@borowski-software.de.

dnpCode

A2411RxNET

DEV ACADEMY

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



2 TAGE

AI-driven Software Development mit GPT und Co.

TRAINING!

Was wird behandelt

- GPT-Modelle und ihre Unterschiede
- AI-Tools für Software-Entwickler
- Grundlagen des Prompt Engineering
- Verwenden der OpenAI API
- AI Search und Vektordatenbanken
- Eigene LLMs bereitstellen mit Azure OpenAI
- Entwicklung eigener Assistenten
- Anbindung Backendsysteme



Jörg Neumann,
MVP OpenAI
Trainer

VERSCHLÜSSELUNGSKONZEPTE ERKLÄRT

Grundlagen der Kryptografie

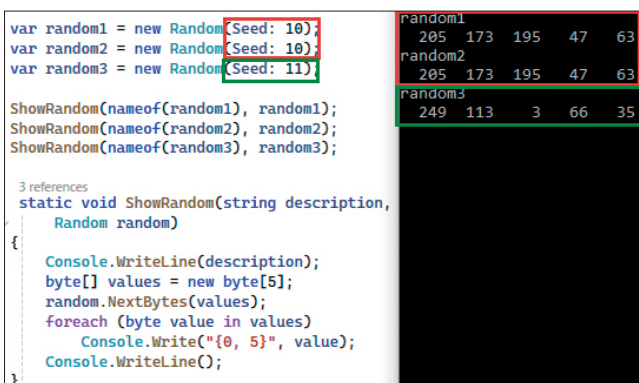
Die Kryptografie ist ein komplexes Thema. Das Verständnis einiger Begriffe macht das Entwicklerleben leichter.

Bei der Kryptografie geht es darum, Daten zu verschlüsseln, um zu verhindern, dass diese für nicht autorisierte Parteien zur Verfügung stehen. Hierzu werden die Ausgangsdaten – bezeichnet als Klartext – von einem Algorithmus unter Verwendung eines Schlüssels so umgewandelt, dass das Ergebnis – der sogenannte Geheimtext – nicht mehr lesbar ist. Nur wer im Besitz des passenden Schlüssels ist, kommt wieder an die Daten. Dieser Artikel stellt einige wichtige Konzepte, Begrifflichkeiten und Zusammenhänge vor.

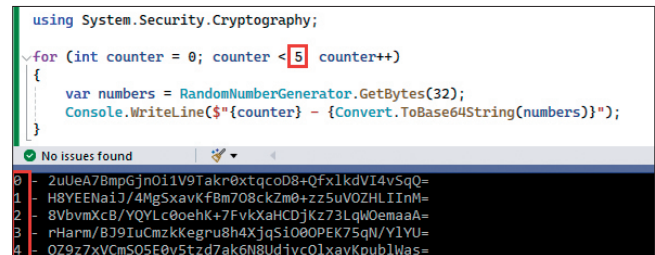
Zufallszahlen

Den Schlüssel, der für die Ver- und Entschlüsselung benötigt wird, sollte es möglichst nur einmalig geben. Um dies sicherzustellen, werden Zufallszahlen generiert. Wer nun jedoch seine Gedanken in Richtung des .NET-Typs *System.Random* [1] lenkt, sollte vorsichtig sein, da dieser bei der Instanzierung einen Seed verwendet, das heißt einen Wert, der für die Initialisierung benötigt wird. In Bild 1 ist ein Beispiel zu sehen, wie zwei Instanzen mit dem identischen Seed generiert werden (rot markiert). Die zugehörigen rot markierten Ausgaben zeigen, dass hierdurch auch zweimal der identische „Zufallswert“ generiert wurde. Die grünen Markierungen dagegen zeigen, wie mit einem anderen Seed ein anderer Zufallswert berechnet wurde.

Der parameterlose Standard-Konstruktor ermittelt einen Seed anhand der Systemuhr, was bedeutet, dass es durchaus passieren kann, dass bei mehreren nacheinander erstellten *System.Random*-Instanzen diese den gleichen „Zufallswert“ produzieren. Da die Zufallszahlen für die Key-Generierung zwingend benötigt werden, ist somit *System.Random* für den Einsatz in der Kryptografie ungeeignet.



Der .NET-Typ *System.Random* (Bild 1)



Verwendung des *RandomNumberGenerator* (Bild 2)

.NET stellt für die Generierung von Zufallszahlen im Kontext der Kryptografie den Typ *RandomNumberGenerator* zur Verfügung, der bei Generierung zwar langsamer ist, aber gegenüber dem Stellenwert der Absicherung von Daten fällt dieser Nachteil nicht ins Gewicht. Bild 2 zeigt die Verwendung dieses Typs. Die Ausgabe enthält die fünf generierten Zufallswerte, es gibt keinerlei Übereinstimmung.

Mit dem Parameterwert 32 beim Aufruf von *RandomNumberGenerator* wird angegeben, dass 32 Bytes von Zufallswerten zu generieren sind.

```
RandomNumberGenerator.GetBytes(32)
```

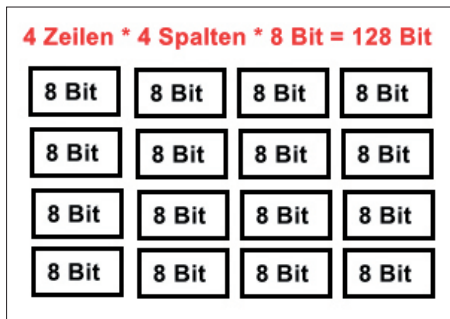
Um die generierten Zufallswerte in einem lesbaren Format zu erhalten, werden diese in Base64 konvertiert. Obwohl auch für diesen Vorgang der Begriff Encodieren verwendet wird, handelt es sich hier nicht um eine Verschlüsselung; der Base64-String kann problemlos wieder in das ursprüngliche Byte-Array-Format zurückverwandelt (decodiert) werden.

Symmetrische Verschlüsselung – AES

Bei der symmetrischen Verschlüsselung kommt lediglich ein Schlüssel zum Einsatz. Dieser eine Schlüssel wird sowohl zum Verschlüsseln als auch zum Entschlüsseln verwendet.

Die symmetrische Verschlüsselung hat folgende Merkmale:

- Sie ist sehr sicher. Der Advanced Encryption Standard (AES [3]) Standard wurde von Joan Daemen und Vincent Rijmen unter der Bezeichnung „Rijndael“ entwickelt und löste den Data Encryption Standard (DES [4]) ab, der nur eine geringe Schlüssellänge von 56 Bit unterstützte.
- Die symmetrische Verschlüsselung ist im Vergleich zur asymmetrischen Verschlüsselung, die später erläutert wird, deutlich schneller.
- Den Schlüssel sicher auszutauschen ist eine Herausforderung. Sollte dieser in die falschen Hände geraten, könnte



AES-Tabelle mit 128 Bit (Bild 3)

```
internal static class EncryptionAes
{
    1 reference
    public static byte[] Encrypt(byte[] dataToEncrypt, byte[] key, byte[] initVector)
    {
        using var aes = Aes.Create();
        aes.Key = key;
        return aes.EncryptCbc(dataToEncrypt, initVector);
    }

    1 reference
    public static byte[] Decrypt(byte[] dataToDecrypt, byte[] key, byte[] initVector)
    {
        using var aes = Aes.Create();
        aes.Key = key;
        return aes.DecryptCbc(dataToDecrypt, initVector);
    }
}
```

Klasse zur AES-Verschlüsselung (Bild 4)

zwischen den rechtmäßigen Sender und Empfänger ein „Man in the middle“ installiert werden, der die versendeten Daten vor dem Empfänger entgegennimmt, entschlüsselt und sie bei Bedarf manipuliert, verschlüsselt und weiterleitet. Der Empfänger könnte somit mit falschen Informationen versorgt werden.

AES verwendet folgende Parameter:

- Eine Schlüssellänge von 128, 192 oder 256 Bit.
- Einen 16-Byte-Initialisierungsvektor. Dieser Wert wird sowohl für die Ver- als auch die Entschlüsselung eingesetzt, um eine gewissen Entropie, das heißt eine Unordnung, zu erreichen.

Beim AES-Verfahren handelt es sich um eine sogenannte „Blockchiffre“ oder auch „block cipher“. Damit wird ein deterministisches Verschlüsselungsverfahren bezeichnet, das einen Klartextabschnitt fester Länge (die zu verschlüsselnden Daten in lesbarer Form) in einen Geheimtext abbildet. Wie bereits beschrieben kann beim verwendeten Schlüssel zwischen einer Größe von 128, 192 oder 256 Bit gewählt werden.

Beim AES wird zunächst jeder Block in eine Tabelle mit vier Zeilen geschrieben. Die Anzahl der Spalten hängt von der Größe ab. Jede Zelle in der Tabelle enthält 8 Bit; bei 128 Bit hat die Tabelle also vier Spalten (Bild 3). Bei einer Länge von 256 Bit hätte die Tabelle acht Spalten.

Der Vorgang der Verschlüsselung durchläuft mehrere Runden. Jede Runde vollzieht hierbei folgende Schritte:

- **Substitution:** Jedes Byte wird unter Verwendung einer Substitutionsbox, auch S-Box genannt, verschlüsselt. Enthielt beispielsweise eine Zelle ursprünglich das Symbol $A(0,0)$, enthält sie danach das Symbol $G(0,0)$. Jedes Byte wird durch einen anderen Wert ersetzt.
- **Shift Row:** Jede Zeile in der Tabelle wird um eine bestimmte Anzahl von Spalten nach links verschoben.
- **Mix Column:** Die Spalten werden mithilfe von Matrixmanipulationen vermischt.
- **Key Addition:** Die Matrix wird mit einem Rundenschlüssel, der aus dem Eingabeschlüssel generiert wird, mit der XOR-Operation verknüpft.

Diese Runden werden mehrfach durchlaufen. Das Verfahren ist vollständig umkehrbar, das heißt, die verschlüsselten Daten können tatsächlich auch wieder entschlüsselt werden.

So viel zur Theorie. In der Praxis kann dies, wie in Bild 4 zu sehen, in C# umgesetzt werden. Die Klasse *EncryptionAes* bietet die beiden Funktionen *Encrypt* und *Decrypt* an. Erste wird mit den zu verschlüsselnden Daten, dem Schlüssel, der sowohl dem Sender als auch dem Empfänger bekannt sein muss, und dem Initialisierungsvektor versorgt. Die zweite Funktion erwartet dagegen die verschlüsselten Daten, den gemeinsamen Schlüssel und den Initialisierungsvektor.

Die Verwendung dokumentiert Bild 5. Zunächst werden der gemeinsame Verschlüsselungs-Key und der Initialisierungsvektor mit Aufrufen an

`RandomNumberGenerator.GetBytes`

erzeugt, welche für die Verschlüsselung (rot markiert) und Entschlüsselung (grün) angewendet werden.

Asymmetrische Verschlüsselung – RSA

Die symmetrische Verschlüsselung ist eine gute Sache. Die Herausforderung ist jedoch, sicherzustellen, dass der Key sicher zwischen den Kommunikationspartnern ausgetauscht wird; dies wurde bereits zuvor mit dem Hinweis auf ►

```
originalMessage: DotNetPro ist klasse!
Encrypted message: ZQXwrjHk5XMIy5RzFVoXOE169+KN4vv8aD0pBY1/4yE=
Decrypted message: DotNetPro ist klasse!

ryptographie - Program.cs
ryptographie
const string originalMessage = "DotNetPro ist klasse!";
var key = RandomNumberGenerator.GetBytes(32);
var initVector = RandomNumberGenerator.GetBytes(16);

var encrypted = EncryptionAes.Encrypt(Encoding.UTF8.GetBytes(originalMessage), key, initVector);
var decrypted = EncryptionAes.Decrypt(encrypted, key, initVector);

var decryptedMessage = Encoding.UTF8.GetString(decrypted);

Console.WriteLine($"originalMessage: {originalMessage}");
Console.WriteLine($"Encrypted message: {Convert.ToBase64String(encrypted)}");
Console.WriteLine($"Decrypted message: {decryptedMessage}");
```

Daten mittels AES ver- und entschlüsseln (Bild 5)

den „Man in the middle“ erklärt. Hier kommt die asymmetrische Verschlüsselung ins Spiel, die zwei unterschiedliche Schlüssel nutzt. Sie arbeitet mit einem „Public Key“, der prinzipiell jedem zur Verfügung steht und zur Verschlüsselung zum Einsatz kommt (Bild 6). Die Entschlüsselung kann nur durch den „Private Key“ erfolgen, der mathematisch mit dem Public Key verknüpft ist. Der Private Key steht nur dem Empfänger zur Verfügung, um die verschlüsselte Nachricht zu entschlüsseln.

Dieses Verfahren wird als asymmetrisch bezeichnet, da im Gegensatz zur symmetrischen Verschlüsselung zwei Schlüssel mit unterschiedlichen Aufgaben zum Einsatz kommen: der Public Key zum Verschlüsseln und der Private Key zum Entschlüsseln. Bei der symmetrischen Verschlüsselung dagegen wird lediglich ein Schlüssel genutzt, der sowohl für die Verschlüsselung als auch für die Entschlüsselung Verwendung findet.

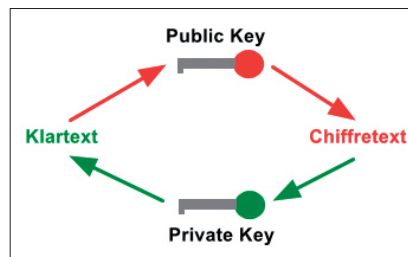
Einige Merkmale der asymmetrischen Verschlüsselung:

- Im Vergleich zur symmetrischen Verschlüsselung ist der Vorgang, Nachrichten zu verschlüsseln und zu entschlüsseln, deutlich langsamer.
- Auch die Menge der auf einmal zu verschlüsselnden Daten ist limitiert.
- Daher wird sie gerne verwendet, um symmetrische Schlüssel zu verschlüsseln.
- Die Schlüssellänge kann zwischen 1024, 2048 und 4096 Bit variieren, wobei 1024-Bit-Schlüssel heute als nicht mehr sicher gelten.

Bild 7 zeigt den ersten Abschnitt der C#-Klasse. Zunächst wird im Konstruktor eine Instanz des .NET-Typs `RSA` erstellt. Die Factory-Funktion wird mit der Information der Schlüssellänge von 2048 aufgerufen. Als Nächstes folgt die Funktion `Encrypt`, sie wandelt vor der Verschlüsselung die zu verschlüsselnde Zeichenkette zunächst in ein Byte-Array um. Das Gegenstück hierzu, die Funktion `Decrypt`, dient dazu, ein Byte-Array mit verschlüsselten Daten wieder zu entschlüsseln.

In Bild 8 sind die Funktionen der `RSA`-Klasse zu sehen, die für den Export und den Import des Public und des Private Keys genutzt werden. Bild 9 veranschaulicht, wie die Klasse zur Ver- und Entschlüsselung verwendet wird.

Wie die Schlüsselinformationen exportiert und wieder importiert werden, zeigt Bild 10. Die erste Instanz der `RSA`-Klasse (grün markiert) exportiert diese und verschlüsselt anschließend die Daten. Die grünen Teile entsprechen somit dem Versender. Die zweite `RSA`-Klasse (rot) steht für den Empfänger. Sie importiert beide Schlüsselinformationen und entschlüsselt die verschlüsselte Information.



Asymmetrische Verschlüsselung (Bild 6)

Hybride Verschlüsselung mit AES und RSA

Der Inhalt der vorangegangenen Abschnitte kann damit wie folgt zusammengefasst werden:

- Die symmetrische Verschlüsselung ist recht flott im Vergleich zur asymmetrischen Verschlüsselung.
- Der sichere Austausch eines symmetrischen Schlüssels ist schwierig und birgt Sicherheitsrisiken.
- Der Schlüsselaustausch der asymmetrischen Verschlüsselung ist deutlich einfacher. Der Zugriff auf den Public Key kann grundsätzlich jedem gewährt werden. Nur der Private Key muss sicher verwahrt werden.

Daher bietet es sich an, das Beste aus beiden Welten miteinander zu kombinieren, was als hybride Verschlüsselung bezeichnet wird. Symmetrische Schlüssel werden mit RSA verschlüsselt und können dadurch sicher ausgetauscht werden.

Als Beispiel soll der sichere Nachrichtenaustausch zwischen zwei Chat-Partnern, Frank und Martina, dienen. Frank

```

internal class EncryptionRSA
{
    private readonly RSA _rsa;
    0 references
    public EncryptionRSA()
    {
        const int keySize = 2048;
        _rsa = RSA.Create(keySize);
    }

    0 references
    public byte[] Encrypt(string dataToEncrypt)
    {
        return _rsa.Encrypt(Encoding.UTF8.GetBytes(dataToEncrypt),
            RSAEncryptionPadding.OaepSHA256);
    }

    0 references
    public byte[] Decrypt(byte[] dataToDecrypt)
    {
        return _rsa.Decrypt(dataToDecrypt, RSAEncryptionPadding.OaepSHA256);
    }
}
  
```

ctor und erste Funktionen der `RSA`-Klasse (Bild 7)

```

public byte[] ExportPrivateKey(int iterationCount, string password)
{
    var keyParams = new PbeParameters(
        PbeEncryptionAlgorithm.Aes256Cbc, HashAlgorithmName.SHA256, iterationCount);

    var encryptedPrivateKey =
        _rsa.ExportEncryptedPkcs8PrivateKey(Encoding.UTF8.GetBytes(password), keyParams);
    return encryptedPrivateKey;
}

1 reference
public void ImportEncryptedPrivateKey(byte[] encryptedPrivateKey, string password)
{
    _rsa.ImportEncryptedPkcs8PrivateKey(Encoding.UTF8.GetBytes(password), encryptedPrivateKey, out _);
}

1 reference
public byte[] ExportPublicKey()
{
    return _rsa.ExportRSAPublicKey();
}

1 reference
public void ImportPublicKey(byte[] publicKey)
{
    _rsa.ImportRSAPublicKey(publicKey, out _);
}
  
```

Private und Public Key der `RSA`-Klasse (Bild 8)

möchte eine Nachricht an Martina senden und vollzieht dazu für den Versand folgende Schritte:

- Generierung eines AES-Session-Keys (symmetrische Verschlüsselung).
- Generierung eines Initialisierungsvektors (Zufallswert, dient der Entropie).
- Verschlüsselung der Nachricht mit dem AES-Schlüssel und dem Initialisierungsvektor.
- Verschlüsselung des AES-Session-Keys mit dem Public Key von Martina (RSA).
- Versand (1.) der verschlüsselten Daten, (2.) des verschlüsselten Session Keys und (3.) des Initialisierungsvektors an Martina.

Auf der Empfangsseite muss Martina folgende Schritte durchlaufen:

- Entschlüsselung des empfangenen, mit dem RSA-Public-Key verschlüsselten AES-Session-Keys mit dem zugehörigen RSA-Private-Key.
- Entschlüsselung der Nachricht mit dem entschlüsselten AES-Session-Key und dem Initialisierungsvektor.

Nachdem Martina die Nachricht gelesen hat, möchte sie Frank antworten und vollzieht hierfür folgende Schritte:

- Generierung eines neuen AES-Session-Keys (symmetrische Verschlüsselung). Der empfangene Session Key wird nicht wieder verwendet.
- Generierung eines neuen Initialisierungsvektors.
- Martina verschlüsselt die Nachricht für Frank mit dem neuen Session Key mittels RSA (asymmetrisch), aber diesmal mit Franks Public Key.
- Martina versendet (1.) die verschlüsselten Daten, (2.) den verschlüsselten Session Key und (3.) den Initialisierungsvektor.

Um die von Martina empfangene Nachricht zu lesen, durchläuft Frank folgende Sequenz:

- Entschlüsselung des mit dem RSA-Public-Key verschlüsselten AES-Session-Keys mit dem eigenen RSA-Private-Key.
- Mit dem entschlüsselten AES-Session-Key kann die Nachricht entschlüsselt werden.

Die Implementierung beginnt mit dem „Datenpaket“, das in Bild 11 dargestellt ist und über die folgenden drei Member verfügt:

```
Original: DotNetPro ist klasse!
Encrypted: uiX8uGKIUIsRglqJp6u9WXT3Ny25WskXe7ZG1NOHzb+ccqYfAbrUBZyMiNiTwZ4vLTC5euDn1NRQRVR754iV1aAIJ...
Decrypted: DotNetPro ist klasse!

DotNetPro.Cryptographie - Program.cs
am.cs x
DotNetPro.Cryptographie
5 const string originalMessage = "DotNetPro ist klasse!";
6 var rsa = new EncryptionRSA();
7
8 var encrypted = rsa.Encrypt(originalMessage);
9 var decrypted = rsa.Decrypt(encrypted);
10
11 Console.WriteLine($"Original: {originalMessage}{Environment.NewLine}");
12 Console.WriteLine($"Encrypted: {Convert.ToBase64String(encrypted)}{Environment.NewLine}");
13 Console.WriteLine($"Decrypted: {Encoding.Default.GetString(decrypted)}{Environment.NewLine}");
```

RSA-Klasse in Verwendung (Bild 9)

```
Original: DotNetPro ist klasse!
Encrypted: g1KCWAqagB7qA1rTEDmnerDuAfPDCrZ0An8S/tmHmxnNRarWgwNoC3rLn0oax3tYivaAhqy4oH+dsETAZC...
Decrypted: DotNetPro ist klasse!

Cryptographie - Program.cs
Cryptographie
var rsa = new EncryptionRSA();
byte[] encryptedPrivKey = rsa.ExportPrivateKey(iterationCount: 1_000, "mypw");
byte[] publicKey = rsa.ExportPublicKey();

const string originalMessage = "DotNetPro ist klasse!";
var encrypted = rsa.Encrypt(originalMessage);

var rsa2 = new EncryptionRSA();
rsa2.ImportPublicKey(publicKey);
rsa2.ImportEncryptedPrivateKey(encryptedPrivKey, "mypw");

var decrypted = rsa2.Decrypt(encrypted);

Console.WriteLine($"Original: {originalMessage}{Environment.NewLine}");
Console.WriteLine($"Encrypted: {Convert.ToBase64String(encrypted)}{Environment.NewLine}");
Console.WriteLine($"Decrypted: {Encoding.Default.GetString(decrypted)}{Environment.NewLine}");
```

RSA-Klasse mit Import/Export der Schlüssel (Bild 10)

- einen AES-Key in verschlüsselter Form,
- die Nachricht, die mit dem AES-Key verschlüsselt wurde,
- den Initialisierungsvektor.

Das hybride Verschlüsselungsverfahren wird in einer Klasse gekapselt. In Bild 12 ist die Funktion zu sehen, mit der die ►

```
internal class EncryptedMessagePacket
{
    /// <summary>
    /// Der verschlüsselte AES Key (symmetrischer Schlüssel).
    /// </summary>
    public byte[] EncryptedAESSessionKey;
    /// <summary>
    /// Die mit dem AES Key verschlüsselten Daten.
    /// </summary>
    public byte[] DataEncryptedWithAESSessionKey;
    /// <summary>
    /// Zufallswert
    /// </summary>
    public byte[] InitializationVector;
}
```

Das Datenpaket mit seinen Membern (Bild 11)

Daten verschlüsselt werden. Die Bestandteile, die die symmetrische Verschlüsselung behandeln, sind rot markiert, während die asymmetrischen Verschlüsselungsbestandteile grün markiert sind.

Zunächst wird der AES-Session-Key erstellt und in das Datenpaket gereicht. Die eigentlichen Daten werden im nächsten Schritt mit diesem AES-Session-Key verschlüsselt und ebenfalls dem Datenpaket übergeben. Final wird der symmetrische Schlüssel asymmetrisch verschlüsselt.

Bild 13 zeigt die Funktion, die der hybriden Entschlüsselung dient. Auch hier wurden die symmetrischen Anweisungen (rot markiert) und die asymmetrischen (grün) Anweisungen farblich gekennzeichnet. Der asymmetrisch verschlüsselte symmetrische Schlüssel wird zunächst entschlüsselt. Anschließend wird mit diesem die symmetrisch verschlüsselte Nachricht entschlüsselt.

Bild 14 dokumentiert die Anwendung. Im Konstruktor der *EncryptionRSA*-Klasse wird das für die asymmetrische Verschlüsselung benötigte Objekt erstellt, siehe Listing 1. Als Nächstes wird dieses Objekt gemeinsam mit der zu verschlüsselnden Nachricht hybrid verschlüsselt und im Anschluss wieder entschlüsselt.

```
public static EncryptedMessagePacket Encrypt(byte[] dataToEncrypt, EncryptionRSA rsaParams)
{
    var aesSessionKey = RandomNumberGenerator.GetBytes(32);
    var messagePacket = new EncryptedMessagePacket
    {
        InitializationVector = RandomNumberGenerator.GetBytes(16)
    };

    messagePacket.DataEncryptedWithAESSessionKey =
        EncryptionAes.Encrypt(dataToEncrypt, aesSessionKey, messagePacket.InitializationVector);

    messagePacket.EncryptedAESSessionKey = rsaParams.Encrypt(aesSessionKey);

    return messagePacket;
}
```

Hybride Verschlüsselung (Bild 12)

```
public static byte[] DecryptData(EncryptedMessagePacket encryptedMessagePacket, EncryptionRSA rsaParams)
{
    var aesSessionKey = rsaParams.Decrypt(encryptedMessagePacket.EncryptedAESSessionKey);
    var decryptedData = EncryptionAes.Decrypt(encryptedMessagePacket.DataEncryptedWithAESSessionKey,
        aesSessionKey, encryptedMessagePacket.InitializationVector);

    return decryptedData;
}
```

Hybride Entschlüsselung (Bild 13)

```
const string originalMessage = "DotNetPro ist klasse!";
var rsaParams = new EncryptionRSA();

var encryptedBlock = EncryptionHybrid.Encrypt(Encoding.UTF8.GetBytes(originalMessage), rsaParams);
var decrypted = EncryptionHybrid.DecryptData(encryptedBlock, rsaParams);

Console.WriteLine($"Original: {originalMessage}");
Console.WriteLine($"Encrypted: {Encoding.UTF8.GetString(decrypted)}");
```

Verwendung der hybriden Verschlüsselung (Bild 14)

Hashing

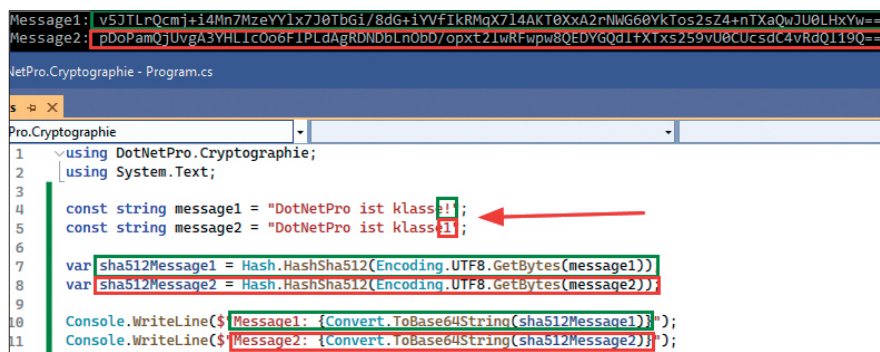
Bei dem Thema Kryptografie kommt man unweigerlich mit dem Begriff „Hashing“ in Berührung. Bei einem Hash handelt es sich um einen digitalen Fingerabdruck von Daten. Wird für eine Textnachricht der Hash berechnet, wird eine scheinbar zufällig zusammengewürfelte Zeichenfolge generiert. Das Hashing hat bestimmte Merkmale:

- Die Berechnung eines Hash ist recht einfach.
- Es ist nicht möglich, eine Nachricht mit einem bestimmten Hash-Wert zu berechnen.
- Eine Änderung der Nachricht, ohne den Hash zu verändern, ist nicht möglich.
- Es ist nicht möglich, unterschiedliche Nachrichten mit dem gleichen Hash zu erhalten.

- Hashing ist somit eine „One way operation“. Hingegen ist die Verschlüsselung eine „Two way operation“.

Wird vor dem Versand einer Nachricht deren Hash berechnet und mit der Nachricht an den Empfänger geliefert, kann der Empfänger selbst den Hash der empfangenen Nachricht berechnen und mit dem übertragenen Hash vergleichen. Sind beide Hash-Werte identisch, kann davon ausgegangen werden, dass die Nachricht unverändert empfangen wurde.

Bild 15 zeigt, wie von zwei nahezu identischen Nachrichten der Hash berechnet wird. Der Unterschied ist das letzte Zeichen, auf welches der rote Pfeil hinweist. Zu sehen ist, dass die berechneten Hash-Werte absolut unterschiedlich sind.



```
Message1: v5JTLnQcmj+14Mn7MzeYY1x7J0TbGi/8dG+iYVfIkRMqX714AKT0XxA2rNMG60YkTos2sZ4+nTXaQwJU0LHxYw==
Message2: pD0PamQJUVgA3YHL1c006F1PLdAgR0ND0Ln00b/opxt21wRfwpw8QEDYGQd1fX1xs259vU0cUcsdC4YRdQ119Q==

letPro.Cryptographie - Program.cs
s + X
Pro.Cryptographie
1 using DotNetPro.Cryptographie;
2 using System.Text;
3
4 const string message1 = "DotNetPro ist klasse!";
5 const string message2 = "DotNetPro ist klasse!";
6
7 var sha512Message1 = Hash.HashSha512(Encoding.UTF8.GetBytes(message1));
8 var sha512Message2 = Hash.HashSha512(Encoding.UTF8.GetBytes(message2));
9
10 Console.WriteLine($"Message1: {Convert.ToBase64String(sha512Message1)}");
11 Console.WriteLine($"Message2: {Convert.ToBase64String(sha512Message2)}");
```

Hash-Berechnung für zwei sehr ähnliche Nachrichten (Bild 15)

HMAC – Hashed Message Authentication Code

Die Kombination aus einem „One way“-Hash und einem kryptografischen Schlüssel wird als HMAC bezeichnet. Der HMAC lässt sich ebenfalls wie ein Hashcode für die Prüfung der Nachrichtenintegrität verwenden. Der HMAC kann zudem zur Authentifizierung der Nachricht verwendet werden – nur wer im Besitz des kryptografischen Schlüssels ist, kann den HMAC berechnen.

Listing 1: Konstruktor der RSA-Kapselung

```
private readonly RSA _rsa;
public EncryptionRSA()
{
    const int keySize = 2048;
    rsa = RSA.Create(keySize);
}
```

```
internal class EncryptedMessagePacket
{
    /// <summary> Der verschlüsselte AES Key (symmetrischer Schlüssel).
    public byte[] EncryptedAESSessionKey;
    /// <summary> Die mit dem AES Key verschlüsselten Daten.
    public byte[] DataEncryptedWithAESSessionKey;
    /// <summary> Zufallswert
    public byte[] InitializationVector;
    /// <summary>
    /// Authentifizierte Hash der verschlüsselten Daten
    /// </summary>
    public byte[] Hmac;
```

Für den HMAC erweitertes Datenpaket (Bild 16)

Listing 2 zeigt, wie die Berechnung eines HMAC in einer Funktion gekapselt wird. Übergeben werden die Daten, für die der Hash berechnet wird, und der kryptografische Schlüssel. Die Verwendung der zugehörigen .NET-Klasse ist trivial.

Auf das Beispiel von Frank und Martina bezogen wäre noch ein zusätzlicher Schritt beim Versand von Frank zu Mar-

tina nötig: Der HMAC der verschlüsselten Daten unter Verwendung des AES-Session-Keys müsste berechnet werden.

Zunächst wird hierzu das Datenpaket, wie es in Bild 16 zu sehen, ist erweitert. Die hybride Verschlüsselung ist in der Klasse *EncryptionHybrid* gekapselt. Sie ähnelt dem bisher gesehenen Code, jedoch ergänzt um die Erweiterung zur Berechnung des HMAC (Bild 17). Der HMAC berechnet den Wert aus den symmetrisch verschlüsselten Daten und dem Initialisierungsvektor. Beide werden in der Funktion *Combine* zusammengeführt.

Bei der Entschlüsselung kann nun der HMAC wie bereits zuvor aus den symmetrisch verschlüsselten Daten und dem Initialisierungsvektor berechnet werden und mit dem HMAC der empfangenen Daten verglichen werden (Bild 18).

Fazit

Die Kryptografie ist ein wichtiges Thema. Dieser Beitrag hat hoffentlich zum Verständnis einiger Begriffe aus dieser Domäne beigetragen.

```
internal class EncryptionHybrid
{
    1 reference
    public static EncryptedMessagePacket Encrypt(byte[] dataToEncrypt, EncryptionRSA rsaParams)
    {
        var aesSessionKey = RandomNumberGenerator.GetBytes(32);
        var messagePacket = new EncryptedMessagePacket
        {
            InitializationVector = RandomNumberGenerator.GetBytes(16)
        };

        messagePacket.DataEncryptedWithAESSessionKey =
            EncryptionAes.Encrypt(dataToEncrypt, aesSessionKey, messagePacket.InitializationVector);

        messagePacket.EncryptedAESSessionKey = rsaParams.Encrypt(aesSessionKey);

        using var hmac = new HMACSHA256(aesSessionKey);
        messagePacket.Hmac = hmac.ComputeHash(
            Combine(messagePacket.DataEncryptedWithAESSessionKey, messagePacket.InitializationVector));

        return messagePacket;
    }
    2 references
    private static by
    1 reference
```

Berechnung des HMAC beim Verschlüsseln (Bild 17)

```
public static byte[] DecryptData(EncryptedMessagePacket encryptedMessagePacket, EncryptionRSA rsaParams)
{
    var aesSessionKey = rsaParams.Decrypt(encryptedMessagePacket.EncryptedAESSessionKey);
    using var hmac = new HMACSHA256(aesSessionKey);
    var hmacToCheck = hmac.ComputeHash(
        Combine(encryptedMessagePacket.DataEncryptedWithAESSessionKey, encryptedMessagePacket.InitializationVector));
    if (!Compare(encryptedMessagePacket.Hmac, hmacToCheck))
    {
        throw new CryptographicException("HMAC check failed");
    }
}
```

Berechnung des HMAC beim Entschlüsseln (Bild 18)

Listing 2: HMAC berechnen

```
public static byte[] HmacSha512(byte[] toGetHash,
byte[] key)
{
    using var hmac = new HMACSHA512(key);
    return hmac.ComputeHash(toGetHash);
}
```



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.

dnpCode

A2411Krypto

- [1] Microsoft Learn, System Random Constructors, www.dotnetpro.de/SL2411Krypto1
- [2] Microsoft Learn, RandomNumberGenerator, www.dotnetpro.de/SL2411Krypto2
- [3] AES bei Wikipedia, www.dotnetpro.de/SL2411Krypto3
- [4] DES bei Wikipedia, www.dotnetpro.de/SL2411Krypto4

Gelungene Integration?

Unit-Tests mit Wolverine waren einfach – gilt dies auch für Integrationstests?

Wie in Teil 4 dieser Serie [1] bereits erläutert, haben Unit-Tests den Nachteil, dass sie nicht das komplette System testen können. Hier kommen Integrationstests ins Spiel. Diese laufen etwas länger und sind aufwendiger aufzusetzen, sind dafür aber im Hinblick auf die Stabilität des Gesamtsystems verlässlicher.

Auch beim Thema Integrationstests kommt uns Wolverine mit Unterstützung entgegen. Nehmen wir als Ausgangsszenario an, wir haben einen Service, der einen Dateiordner auf neu darin abgelegte Dateien überwacht und bei neuen Dateien eine Nachricht publizieren soll, damit andere Funktionen oder Services diese Datei weiterverarbeiten können.

Die Funktionalität basiert auf einem *PhysicalFileProvider*, der auch in der Lage ist, Windows-Shares von einem Linux-Server aus zu überwachen. Dies geschieht über Polling. Das Polling-Intervall beträgt vier Sekunden (dies lässt sich nicht ändern). Damit hat der Integrationstest auch einen temporären Aspekt, da wir zunächst einen Test-Host starten müssen, in dessen Kontext der FileWatcher aktiv ist. Nach dem Start des Hosts kopieren wir eine PDF-Datei in den zu überwachenden temporären Ordner und erwarten, dass unser FileWatcher eine Message vom Typ *FileAdded* publiziert. Außerdem möchten wir auch sicherstellen, dass für diese Nachricht mindestens ein Handler registriert ist.

Da der FileWatcher relativ umfangreich ist, soll hier nur die relevante Methode *OnFilesChanged* gezeigt werden, die eine Nachricht publiziert, wenn neue Dateien hinzugefügt wurden. Der vollständige Quellcode für den FileWatcher ist in den Downloads zum Artikel zu finden.

```
private async void OnFilesChanged(
    object? state
)
{
    var allFiles = FolderReader.GetAllFiles(
        _watchDirectory, _searchPattern);
    var changes = _existingFiles.GetFileChanges(allFiles);
    if (changes.Added.Count != 0)
    {
        foreach (var fileInfo in changes.Added)
        {
            logger.LogDebug(
                "New PDF file added: {FileInfoName}",
                fileInfo.Name);
            await _messageBus.PublishAsync(
                new FileAdded(fileInfo.Name));
        }
    }
}
```

```
    WatchForFileChanges();
}
```

Der zugehörige Test sieht wie folgt aus – zunächst das *SetUp*:

```
[SetUp]
public async Task InitializeAsync()
{
    host = await IntegrationTestHost.InitializeAsync();
    testFiles = _host.Services.GetService<TestFiles>();
    session = await _host.TrackActivity()
        .Timeout(5.Seconds())
        .ExecuteAndWaitAsync(
            (Func<IMessageContext, Task>)(
                async _ =>
                {
                    _testFiles?.CopyTestFileToProjectFolder();
                    await Task.Delay(3.Seconds());
                })
        );
}
```

Der Host, der im Test-Kontext gestartet wird, ist ein Host, der mittels Alba initialisiert wird. Bei Alba handelt es sich um ein NuGet-Paket, das von denselben Entwicklern betreut wird wie Wolverine und Marten.

Alba bietet auf Basis der *WebApplicationFactory* einen Test-Host an. Bei den *TestFiles* handelt es sich um eine Klasse, die das Handling der Test-Dokumente vereinfacht und mit dem Test-Host zusammenspielt.

Wie wir in den Code-Zeilen für den FileWatcher gesehen haben, ist der Aufruf des *MessageBus* in einem Callback enthalten, der vom *ChangeToken* des File Providers aufgerufen wird.

Dank eines weiteren Wolverine-Features können wir sogar solche Szenarien testen. Bei dem Feature handelt es sich um die sogenannte *Tracked Session*. Damit ist es möglich, alle Nachrichten, die im laufenden System ab Aktivierung des Trackings übertragen werden, zu inspizieren.

Wir starten das Tracking in unserem Host und definieren einen Time-out von fünf Sekunden. Die *ExecuteAndWaitAsync*-Methode erlaubt uns, eine Aktion auszuführen. In unserem Fall ist es das Kopieren einer PDF-Datei in den überwachten Ordner. Das erwartete Verhalten ist, dass der FileWatcher die neue Datei erkennt und über den *MessageBus* die Nachricht vom Typ *FileAdded* sendet.

Um festzustellen, ob dies tatsächlich der Fall ist, bietet uns die gestartete Test-Session mehrere Eigenschaften an, die wir

inspizieren können. Im einfachsten Fall prüfen wir nur, ob überhaupt Nachrichten gesendet wurden:

```
[Test]
public void should_contain_messages()
{
    session.AllRecordsInOrder()
        .Length.ShouldBeGreaterThan(0);
}
```

AllRecordsInOrder beinhaltet alle Nachrichten der Session.

Wenn wir sicherstellen möchten, dass tatsächlich diese eine Nachricht gesendet wurde, können wir folgende Assertion verwenden:

```
[Test]
public void should_have_
    one_sent_message()
{
    session
        .Sent
        .AllMessages()
        .Count()
        .ShouldBe(1);
}
```

Die *Sent*-Collection beinhaltet Nachrichten, die erfolgreich versendet wurden.

Möchten wir sicherstellen, dass die Nachricht auch die Eigenschaften zur Datei enthält, können wir auch die Inhalte der Nachricht abrufen:

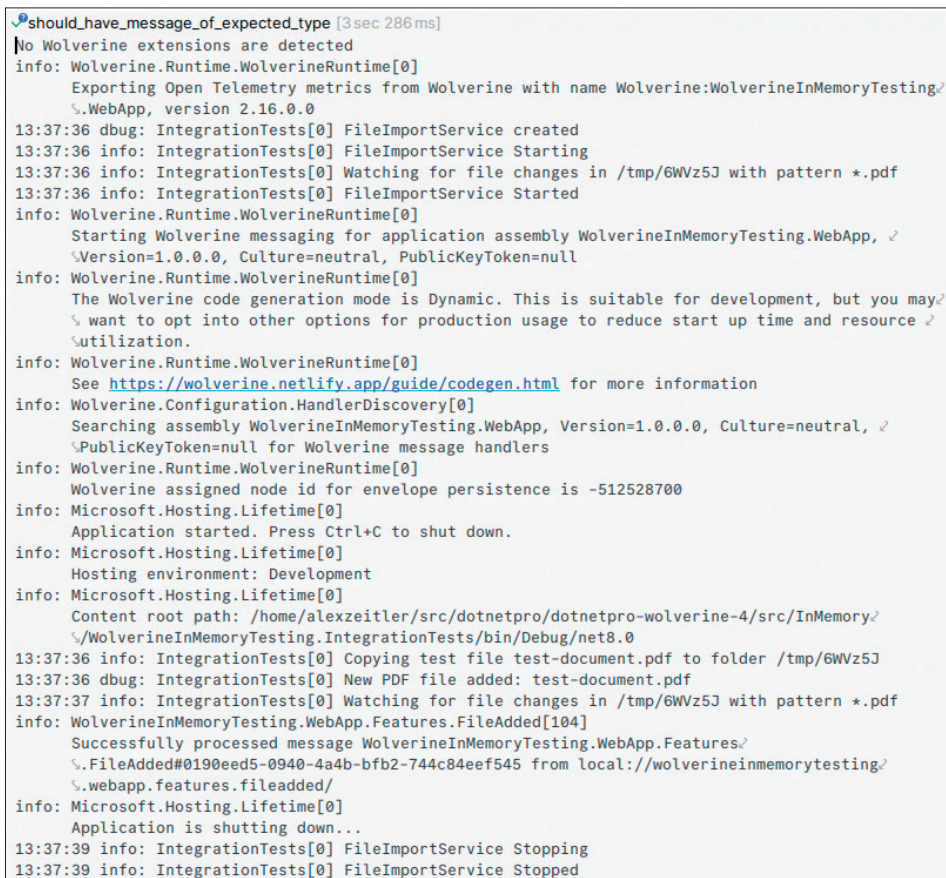
```
[Test]
public void should_have_
    _message_of_expected_type()
{
    var message = _session
        .Sent
        .AllMessages()
        .First();

    message.ShouldBeOfType<FileAdded>();

    message
        .As<FileAdded>()
        .FileName
        .ShouldBe(_testFiles.GetTestPdfFilename());
}
```

Schließlich wollen wir noch der Anforderung Rechnung tragen, dass mindestens ein Handler vorhanden sein soll:

```
[Test]
public async Task should_return_file_info()
{
    session
        .NoRoutes
        .AllMessages()
        .Count()
        .ShouldBe(0);
}
```



```
✓should_have_message_of_expected_type [3 sec 286 ms]
No Wolverine extensions are detected
info: Wolverine.Runtime.WolverineRuntime[0]
    Exporting Open Telemetry metrics from Wolverine with name Wolverine:WolverineInMemoryTesting
    %WebApp, version 2.16.0.0
13:37:36 dbug: IntegrationTests[0] FileImportService created
13:37:36 info: IntegrationTests[0] FileImportService Starting
13:37:36 info: IntegrationTests[0] Watching for file changes in /tmp/6WVz5J with pattern *.pdf
13:37:36 info: IntegrationTests[0] FileImportService Started
info: Wolverine.Runtime.WolverineRuntime[0]
    Starting Wolverine messaging for application assembly WolverineInMemoryTesting.WebApp,
    %Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
info: Wolverine.Runtime.WolverineRuntime[0]
    The Wolverine code generation mode is Dynamic. This is suitable for development, but you may
    % want to opt into other options for production usage to reduce start up time and resource
    %utilization.
info: Wolverine.Runtime.WolverineRuntime[0]
    See https://wolverine.netlify.app/guide/codegen.html for more information
info: Wolverine.Configuration.HandlerDiscovery[0]
    Searching assembly WolverineInMemoryTesting.WebApp, Version=1.0.0.0, Culture=neutral,
    %PublicKeyToken=null for Wolverine message handlers
info: Wolverine.Runtime.WolverineRuntime[0]
    Wolverine assigned node id for envelope persistence is -512528700
info: Microsoft.Hosting.Lifetime[0]
    Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
    Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
    Content root path: /home/alexzeitler/src/dotnetpro/dotnetpro-wolverine-4/src/InMemory
    %/WolverineInMemoryTesting.IntegrationTests/bin/Debug/net8.0
13:37:36 info: IntegrationTests[0] Copying test file test-document.pdf to folder /tmp/6WVz5J
13:37:36 dbug: IntegrationTests[0] New PDF file added: test-document.pdf
13:37:37 info: IntegrationTests[0] Watching for file changes in /tmp/6WVz5J with pattern *.pdf
info: WolverineInMemoryTesting.WebApp.Features.FileAdded[104]
    Successfully processed message WolverineInMemoryTesting.WebApp.Features
    %FileAdded#0190eed5-0940-4a4b-bfb2-744c84eef545 from local://wolverineinmemorytesting
    %webapp.features.fileadded/
info: Microsoft.Hosting.Lifetime[0]
    Application is shutting down...
13:37:39 info: IntegrationTests[0] FileImportService Stopping
13:37:39 info: IntegrationTests[0] FileImportService Stopped
```

Das Protokoll eines Integrationstests (Bild 1)

Statt der Collection *Sent* greifen wir nun auf die Collection *NoRoutes* zu. Diese beinhaltet Nachrichten, für die das Routing keinen Handler finden konnte. Dies ist zum Beispiel für Publish/Subscribe-Szenarien hilfreich, da dort nicht zwingend ein interner Handler vorhanden sein muss.

Wird statt der *PublishAsync*-Methode zum Beispiel die *InvokeAsync*-Methode aufgerufen, erwartet Wolverine einen Handler. Die *Sent*-Collection wäre dann leer, und die *NoRoutes*-Collection würde unsere nicht gesendete Nachricht enthalten. Analog zu *Sent* und *NoRoutes* existieren noch weitere Collections, abhängig vom Status der Nachrichten, zum Beispiel *MovedToErrorQueue* oder *Requeued*.

Bild 1 zeigt das Log der Test-Ausführung. Hier sieht man vom Starten des Hosts über die Wolverine-Statusmeldungen bis hin zu den Ausgaben des FileWatchers und dem Stoppen des Hosts alle Details. ►



Das Session-Tracking bietet noch viele weitere Möglichkeiten: Zum Beispiel ist es möglich, ähnlich wie in unserem Ping-Pong-Beispiel der vorangegangenen Serienfolgen mehrere Hosts zu starten und per Message-Tracking zu asserten, dass eine aus einem Host versendete Nachricht im anderen Host zugestellt wurde – die entsprechende Methode lautet `.WaitForMessageToBeReceivedAt<TMessage>(host)`.

Harte Realität

Die Freude über die zugegebenermaßen einfach zu testende Integration währt möglicherweise nicht lange: Was tun, wenn Wolverine wie im dritten Teil dieser Serie [2] mit einem Broker wie RabbitMQ eingesetzt wird?

Mit dem Wissen aus Teil 3 [2] müsste der Umstieg von In-Memory-Messaging zu Messaging mit RabbitMQ eigentlich relativ einfach zu bewerkstelligen sein. Allerdings liegt noch eine Hürde vor uns, die wir überwinden müssen: RabbitMQ muss vor dem Testlauf verfügbar sein, damit die Tests erfolgreich durchgeführt werden können.

Ein Ansatz wäre, RabbitMQ mittels der TestContainers-Library vor jedem Testlauf als Docker-Container zu starten und RabbitMQ je Testlauf in der Test-Anwendung mit unterschiedlichen Ports zu konfigurieren. Das macht die Angelegenheit sehr einfach, hat aber einen entscheidenden Nachteil: RabbitMQ benötigt selbst auf einer performanten Entwicklermaschine noch immer mindestens drei bis vier Sekunden, um betriebsbereit zu sein. Das heißt, wir hätten in jedem Test einen entsprechenden Overhead, der sich bei mehreren Hundert oder gar bei Tausenden Tests signifikant auswirken

würde und dem Ziel, schnell Antworten zu bekommen, entgegensteht.

Ein anderer Ansatz ist, virtuelle Hosts in RabbitMQ zu verwenden. Worum handelt es sich hierbei? Die Übersetzung der RabbitMQ-Dokumentation lautet wie folgt:

- **RabbitMQ** ist ein mandantenfähiges System: Verbindungen, Austausch, Warteschlangen, Bindungen, Benutzerrechte, Policies und einige andere Dinge gehören zu virtuellen Hosts, logischen Gruppen von Einheiten.
- (...)
 - **Virtuelle Hosts** ermöglichen die logische Gruppierung und Trennung von Ressourcen. Die Trennung physischer Ressourcen ist nicht das Ziel virtueller Hosts, obwohl bestimmte Ressourcen für einzelne virtuelle Hosts begrenzt werden können.

Virtuelle Hosts schaffen also eine Isolation. Damit können wir je Test einen virtuellen Host erzeugen, der sich ähnlich einer isolierten Instanz verhält. Läuft bereits eine RabbitMQ-Instanz, lässt sich ein virtueller Host über das HTTP-API von RabbitMQ praktisch ohne Verzögerung erstellen.

Zunächst sorgen wir also dafür, dass vor dem Testlauf bereits eine RabbitMQ-Instanz vorhanden ist, indem wir sie mit folgendem Compose-File und Docker starten:

services:

```
rmq:
  image: "rabbitmq:3-management"
```



```

container_name: rabbitmq
ports:
  - "5672:5672"
  - "15672:15672"

```

Da der .NET-Client für RabbitMQ keine Möglichkeit bietet, virtuelle Hosts zu verwalten, schaffen wir uns diese mittels *HttpClient* selbst:

```

public static async Task<string> CreateVirtualHost()
{
    var vhostName = "testing";

    using (HttpClient client = new HttpClient())
    {
        client.BaseAddress =
            new Uri("http://localhost:15672");

        var byteArray =
            Encoding.ASCII.GetBytes($"guest:guest");
        client.DefaultRequestHeaders.Authorization =
            new AuthenticationHeaderValue(
                "Basic", Convert.ToBase64String(byteArray));

        var endpoint = $"/api/vhosts/{vhostName}";

        await client.PutAsync(endpoint, null);
    }

    return vhostName;
}

```

Rufen wir den Code bei laufender RabbitMQ-Instanz auf, wird ein neuer virtueller Host mit dem Namen *testing* erstellt. Für den Rest des Artikels belassen wir es des einfacheren Verständnisses halber dabei. Für den Echtbetrieb empfiehlt es sich allerdings, den Namen dynamisch zu gestalten, indem er zum Beispiel eine GUID enthält.

Im nächsten Schritt fügen wir RabbitMQ zu unserer Wolverine-Applikation hinzu. Im Unterschied zu Teil 3 dieser Serie konfigurieren wir allerdings noch die RabbitMQ-ConnectionFactory:

```

hostBuilder.UseWolverine(
    options =>
    {
        options.PublishMessage<FileAdded>()
            .ToRabbitExchange("files");

        options
            .ListenToRabbitQueue(
                "files",
                queue => queue.TimeToLive(15.Seconds())
            );

        options
            .UseRabbitMq(

```

● Wolverine-Funktionsupdate

Während der Arbeit an dem Artikel ist dem Autor aufgefallen, dass es nützlich wäre, neben der *Sent*-Collection des Session-Trackings hierfür weitere Collections bereitzustellen, anstatt die Nachrichten anhand des Status aus der Collection *AllRecordsInOrder* auszufiltern. Nach Rücksprache mit dem Maintainer des Wolverine-Projekts ist ein Pull-Request entstanden, der zur Version 2.16.0 geführt hat. Ab dieser Version sind die genannten Collections wie zum Beispiel *NoRoutes* enthalten.

```

cf =>
{
    cf.HostName = rabbitMQConfiguration.Host;
    cf.VirtualHost =
        rabbitMQConfiguration.VirtualHost;
}
)
.DeclareExchange(
    "files",
    exchange => exchange.BindQueue("files")
)
.AutoProvision();
}
);

```

Starten wir unsere Tests nun erneut, werden diese noch immer erfolgreich ausgeführt. Ein Blick auf die Queue *files* im virtuellen Host *testing* im RabbitMQ-Dashboard zeigt, dass RabbitMQ tatsächlich verwendet wurde (Bild 2).

Fazit: Gelungene Integration!

Nach den Unit-Tests im vierten Teil dieser Serie [1] haben wir in diesem Teil gesehen, wie wir Integrationstests mit Wolverine durch Tracked Sessions einfach und elegant umsetzen können. Ein Artikel in der nächsten dotnetpro-Ausgabe wird sich mit den Themen Diagnose und Troubleshooting von Wolverine-Applikationen auseinandersetzen. ■

[1] Alexander Zeitler, *Red/Green mit Wolverine*, dotnetpro 10/2024, Seite 66 ff., www.dotnetpro.de/A2410Wolverine

[2] Alexander Zeitler, *Hattu Nachricht?*, dotnetpro 9/2024, Seite 82 ff., www.dotnetpro.de/A2409Wolverine



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

A2411Wolverine

SQLKATA/QUERYBUILDER

Die Kunst, Abfragen zu erzeugen

SQL-Abfragen zu schreiben kann mühsam sein. Bibliotheken wie QueryBuilder möchten den Prozess vereinfachen, um dabei Fehler zu vermeiden.

Die Effizienz bei der Verwaltung und Verarbeitung von Daten spielt in vielen Anwendungen eine zentrale Rolle. Auch in .NET-Anwendungen ist die Fähigkeit, präzise und leistungsfähige Datenbankabfragen zu generieren, von entscheidender Bedeutung. Diese Abfragen dienen als Schnittstelle zwischen der Anwendung und der zugrunde liegenden Datenbank, um relevante Daten abzurufen, zu filtern und zu manipulieren.

Um auf eine Datenbank zuzugreifen, existieren verschiedene Technologien und Vorgehensweisen: Datenbankabfragen über SQL, objektrelationale Mapper (ORM) oder Abfragen, die in einer Programmiersprache wie C# geschrieben werden. Letztere werden dann von einer Bibliothek in einen SQL-Dialekt überführt – wie das beispielsweise die Bibliothek SqlKata, auch QueryBuilder genannt, übernimmt, um die es in dieser Episode der Kolumne Frameworks und mehr gehen soll.

Was ist SqlKata beziehungsweise QueryBuilder?

QueryBuilder ist eine .NET-Bibliothek, die es ermöglicht, SQL-Abfragen programmgesteuert und auf möglichst intuitive Weise zu erstellen. Das Projekt nennt sich SqlKata [1]. Das Repository [2] auf GitHub trägt allerdings den Namen QueryBuilder, da es im Kern auch genau darum geht: SQL-Abfragen aus Code erzeugen.

Anstatt SQL-Befehle in Form von Zeichenketten zu schreiben, lässt sich mit SqlKata eine Abfrage mithilfe von Methoden und Eigenschaften zusammenstellen. Dieses Prinzip, eine Schnittstelle aufzubauen, wird oft als Fluent API bezeichnet und erlaubt es, Abfragen schrittweise und lesbar zu definieren, was den Code klarer und besser wartbar macht.

Die Hauptfunktionalität von SqlKata besteht darin, Abfragen zu generieren, die direkt mit einer Datenbank kommunizieren können. Dabei werden alle typischen SQL-Operationen wie *SELECT*, *INSERT*, *UPDATE* und *DELETE* unterstützt.

Darüber hinaus ermöglicht die Bibliothek die einfache Erstellung komplexer Abfragen, die Joins, Gruppierungen, Aggregatfunktionen und Unterabfragen enthalten können. Durch die methodische Struktur von SqlKata wird nicht nur die Erstellung der Abfragen vereinfacht, sondern auch das Risiko von Syntaxfehlern reduziert, da der Code zur Laufzeit generiert und validiert wird.



Foto: EMGenie

Beim Schreiben dieses Artikels liegt die Bibliothek in Version 2.4.0 vor, die am 19. September 2022 veröffentlicht wurde. Auf der Webseite ist zudem ein Playground [3] verfügbar, um die Funktionen der Bibliothek direkt auszuprobieren.

Gründe für den QueryBuilder

Die Verwendung von QueryBuilder bietet im Vergleich zu SQL-Befehlen, die direkt im Code als Zeichenketten eingebettet sind, eine Reihe von Vorteilen. Durch das Fluent API lassen sich Abfragen in einer klaren, strukturierten und gut lesbaren Form schreiben. Anstatt komplexe SQL-Abfragen als Strings im Code zu verstecken, werden sie durch eine methodenbasierte Syntax ersetzt, die einfacher zu verstehen und zu debuggen ist. Dies reduziert die Komplexität und erleichtert die Zusammenarbeit im Team. Außerdem ist das Risiko von Syntaxfehlern erheblich minimiert. Typische Fehler wie vergessene oder falsche Anführungszeichen, Kommas oder Schlüsselwörter lassen sich so vermeiden.

Darüber hinaus erlaubt QueryBuilder die dynamische Erstellung von Abfragen zur Laufzeit. Dies ist besonders nützlich, wenn Abfragen basierend auf Benutzereingaben oder anderen Laufzeitbedingungen erzeugt werden müssen. Das funktioniert mit Zeichenketten zwar auch, aber auf diese Weise ist es nicht notwendig, eine ganze Reihe von Zeichen,

Zeichenketten und Variablen zusammenzuhängen, um einen SQL-Befehl dynamisch zu erzeugen.

Diese Vorteile sprechen für QueryBuilder. Trotzdem bleibt die Frage im Raum stehen, warum nicht direkt Frameworks wie das Entity Framework oder NHibernate zum Einsatz kommen. Diese sind ebenfalls seit Jahren und Jahrzehnten in der Entwicklung, ausgereift, performant und bieten neben ähnlichen Funktionen wie in QueryBuilder noch viele weitere Features und Möglichkeiten.

QueryBuilder möchte ausgewachsene ORMs wie die beiden genannten gar nicht ersetzen. Das ist nicht das Ziel der Bibliothek. Was das Projekt aber bietet, ist zum Beispiel eine feinere Kontrolle über SQL-Abfragen, da die volle Kontrolle über die generierten Abfragen besteht. Zudem ist QueryBuilder als Bibliothek einfacher und flexibler in Projekte einzubinden. Wenn es notwendig ist, auch nur an der ein oder anderen Stelle, wo es gerade benötigt wird. Zudem kommen so weniger Abhängigkeiten mit einem geringeren Overhead ins Projekt.

Ein großer Punkt ist aber die Abstraktion. Während ORM-Tools wie das Entity Framework eine hohe Abstraktion bieten, ist genau das nicht in allen Projekten erwünscht oder notwendig. Nicht in allen Projekten ist die gesamte Datenbanklogik hinter einer Abstraktion zu verbergen, um damit arbeiten zu können. Manchmal ist eine Zwischenlösung zwischen selbst geschriebenen SQL-Abfragen und einem ORM erwünscht. Genau hier setzt QueryBuilder an.

Installation und Einrichtung

QueryBuilder ist für .NET-Projekte konzipiert und setzt eine kompatible .NET-Version voraus. Die meisten modernen .NET-Versionen, wie .NET Core 3.1 und höher oder .NET 5/6/7 unterstützen QueryBuilder. Für Projekte, die auf älteren Versionen des .NET Frameworks basieren, funktioniert QueryBuilder ab der Version 4.6.

Die Installation von QueryBuilder ist beispielsweise als NuGet-Paket unkompliziert. Über das dotnet CLI genügt die Ausführung der folgenden beiden Befehle, um die primären Pakete zu einem Projekt hinzuzufügen:

```
dotnet add package SqlKata
dotnet add package SqlKata.Execution
```

Das zweite Paket ist nur erforderlich, wenn der Support für die Ausführung von SQL-Befehlen gewünscht ist. Was das bedeutet, zeigt der Artikel im weiteren Verlauf.

Sind die Pakete installiert, lässt sich QueryBuilder direkt nutzen, wie der folgende Code zeigt:

```
using SqlKata;

var query = new Query("Users")
    .Where("Name", "Fabian")
    .Where(q =>
        q.Where("Country", "DE")
    );
```

Die erzeugte SQL-Abfrage sieht, mit einer Kompatibilität für den SQL Server, wie folgt aus:

```
SELECT * FROM [Users]
WHERE [Name] = 'Fabian' AND ([Country] = 'DE');
```

Wird die Abfrage mit einer Kompatibilität für PostgreSQL erzeugt, erstellt QueryBuilder die nachfolgende Zeichenkette:

```
SELECT * FROM "Users"
WHERE "Name" = 'Fabian' AND ("Country" = 'DE');
```

Die Bibliothek beherrscht somit verschiedene Dialekte und kann die erzeugten SQL-Abfragen daraufhin anpassen.

Dieses erste Beispiel zeigt zudem die vielleicht einfachste Nutzung des Projekts: Wir nutzen die Bibliothek lediglich dafür, aus strukturiertem Code eine strukturierte SQL-Abfrage zu erzeugen. Ausgeführt wird diese jedoch nicht, und mit der Abfrage oder Manipulation von Daten hat das auch noch nichts zu tun. Dazu zeigt der Artikel ebenfalls im Verlauf weitere Beispiele.

Grundlegende Konzepte

Die Bibliothek fokussiert sich auf die Erstellung von SQL-Anweisungen, während die Ausführung der Abfragen von einer anderen Schicht übernommen wird. Dies ermöglicht eine klare Trennung der Verantwortlichkeiten und macht den Code modularer und flexibler.

Die Struktur von SqlKata ist so aufgebaut, dass sie ein einfaches und intuitives API für die Erstellung von SQL-Abfragen bietet. Der Kern von SqlKata besteht aus dem Query-Objekt, das als Ausgangspunkt für die Erstellung von Abfragen dient. Dieses Objekt ermöglicht es, schrittweise SQL-Abfragen zu konstruieren, indem Methoden aufgerufen werden, die den gewünschten SQL-Befehlen entsprechen. Ein typisches Beispiel für die Nutzung von SqlKata in der Praxis ist:

```
var query =
    new Query("Users")
        .Where("IsActive", true)
        .Select("Id", "Name");
```

Das Beispiel nutzt im Unterschied zum vorherigen Listing die *Select*-Methode, um die Spalten einzuschränken, die zurückgegeben werden. Die Ausgabe dieser SQL-Anweisung sieht wie folgt aus:

```
SELECT "Id", "Name" FROM "Users"
WHERE "IsActive" = true;
```

Listing 1: Ein INNER JOIN über zwei Tabellen

```
var query = new Query("Orders")
    .Join("Users", "Orders.UserId", "Users.Id")
    .Select("Orders.Id", "Users.Name",
        "Orders.TotalAmount");
```

● Listing 2: Komplexere WHERE-Klausel in einer Abfrage

```
var query = new Query("Users")
    .Where("IsActive", true)
    .WhereIn("Role", new[] { "Admin", "User" })
    .WhereNotNull("Email");
```

Die Bibliothek ist modular aufgebaut, sodass sie leicht erweiterbar ist. Sie basiert auf einer Reihe von Kernklassen, die die verschiedenen Aspekte der Abfrageerstellung abdecken, wie beispielsweise den Klassen *Query*, *Compiler* und *BaseQuery*:

- *Query*: Die zentrale Klasse zur Erstellung von SQL-Abfragen. Sie bietet Methoden zur Angabe von Tabellen, Feldern, Bedingungen, Sortierungen und weiteren SQL-Konstrukten.
- *Compiler*: Diese Klasse ist für die Übersetzung der erstellten Abfrage in den SQL-Text verantwortlich, der von der spezifischen Datenbank ausgeführt werden kann. *SqlKata* stellt verschiedene Compiler für unterschiedliche Datenbanken zur Verfügung.
- *BaseQuery*: Dies ist die Basisklasse, von der die *Query*-Klasse erbt. Sie enthält allgemeine Methoden und Eigenschaften, die für die Abfrageerstellung notwendig sind.

SqlKata unterstützt zudem verschiedene Arten von Joins, wie *InnerJoin*, *LeftJoin*, *RightJoin* und *CrossJoin*. Diese Metho-

den, etwa durch Hinzufügen von Aggregatfunktionen oder durch die Kombination mit anderen SQL-Komponenten.

JOIN-Operationen sind beispielsweise unerlässlich, wenn Daten aus mehreren Tabellen abgerufen werden sollen, die über gemeinsame Werte miteinander verknüpft sind. Ein Beispiel für eine einfache *INNER-JOIN*-Abfrage ist in [Listing 1](#) dargestellt.

Diese Abfrage verknüpft die Tabelle *Orders* mit der Tabelle *Users* über die *UserId* in *Orders* und die *Id* in der Tabelle *Users*. Sie wählt die *Id* der Bestellung, den Namen des Benutzers und den Gesamtbetrag der Bestellung aus. Die resultierende SQL-Anweisung sieht folgendermaßen aus:

```
SELECT "Orders"."Id", "Users"."Name",
       "Orders"."TotalAmount"
FROM "Orders"
INNER JOIN "Users" ON "Orders"."UserId" = "Users"."Id";
```

SqlKata erleichtert die Erstellung von *JOINS* erheblich, indem es ein klares und lesbares API zur Verfügung stellt, das die zugrunde liegende SQL-Syntax abstrahiert.

WHERE-Klauseln sind essenziell, um die Abfrageergebnisse auf bestimmte Kriterien zu beschränken. Die Bibliothek bietet eine Vielzahl von Methoden, um Bedingungen festzulegen, darunter einfache Bedingungen, *IN*-Operatoren, *EXISTS*-Klauseln und mehr. Erste *WHERE*-Klauseln haben die vorherigen Listings gezeigt. Ein Beispiel für eine komplexere *WHERE*-Klausel zeigt [Listing 2](#).

Dieses *Beispiel* wählt nur Benutzer aus, die aktiv sind, eine Rolle als Admin oder User haben und eine nicht leere E-Mail-Adresse besitzen. Die entsprechende SQL-Anweisung lautet wie folgt:

```
SELECT * FROM "Users" WHERE "IsActive" = true
AND "Role" IN ('Admin', 'User')
AND "Email" IS NOT NULL;
```

Auf diese Weise lassen sich zahlreiche SQL-Anweisungen über Code erzeugen und in verschiedene SQL-Dialekte über-

● Listing 3: Ein GROUP BY in einer Abfrage

```
var query = new Query("Orders")
    .Select("CustomerId")
    .SelectRaw("SUM(TotalAmount) AS TotalSpent")
    .GroupBy("CustomerId");
```

den ermöglichen es, Abfragen über mehrere Tabellen zu erstellen, indem die Tabellen auf der Grundlage bestimmter Bedingungen miteinander verknüpft werden.

Eine der mächtigsten und am häufigsten verwendeten Methoden ist die *Where*-Methode. Sie ermöglicht die Definition von Bedingungen, die in der SQL-Abfrage berücksichtigt werden sollen. *SqlKata* bietet eine Vielzahl von *Where*-Methoden, wie beispielsweise *Where*, *WhereIn*, *WhereNotIn*, *WhereExists* und viele mehr.

Zudem unterstützt die Bibliothek auch die Verwendung von Aggregatfunktionen wie *Count*, *Sum*, *Avg*, *Min* und *Max*. Diese Funktionen lassen sich direkt auf die Abfrage anwenden, um aggregierte Daten zu erhalten.

Erstellung einfacher Abfragen

Erste Beispiele für das Erzeugen von *SELECT*-Abfragen haben die vorangegangenen Listings bereits gezeigt. Mit *SqlKata* lassen sich zudem komplexere *SELECT*-Abfragen erstel-

● Listing 4: COUNT mit GROUP BY

```
var query = new Query("Orders")
    .Select("CustomerId")
    .SelectRaw("COUNT(*) AS OrderCount")
    .GroupBy("CustomerId");
```

● Listing 5: Beispiel für eine Unterabfrage in einer weiteren Abfrage

```
var subQuery = new Query("Orders")
    .SelectRaw("MAX(TotalAmount)")
    .Where("CustomerId", 1);
```

```
var query = new Query("Orders")
    .Select(subQuery, "TotalAmount");
```

führen – ganz ohne Syntaxfehler und ohne Konkatination von Zeichenketten mit Daten in Variablen.

Erweiterte Abfrageerstellung

Die erweiterte Abfrageerstellung in SQL erfordert das Verständnis komplexerer Konzepte wie Gruppierung, Aggregatfunktionen, Unterabfragen und die dynamische Erstellung von Abfragen. Gruppierungen und Aggregatfunktionen sind essenziell, wenn es darum geht, Daten in zusammengefasster Form darzustellen, wie etwa die Summe, den Durchschnitt oder die Anzahl von Datensätzen. Ein Beispiel für die Gruppierung und Anwendung von Aggregatfunktionen zeigt Listing 3. Diese Abfrage summiert den Gesamtbetrag *TotalAmount* für jede *CustomerId* in der *Orders*-Tabelle. Die Abfrage gruppiert die Daten nach *CustomerId*, sodass der Gesamtbetrag pro Kunde berechnet wird. Die entsprechende SQL-Abfrage sieht folgendermaßen aus:

```
SELECT "CustomerId", SUM(TotalAmount) AS TotalSpent
FROM "Orders" GROUP BY "CustomerId";
```

Wollen wir beispielsweise die Anzahl der Bestellungen pro Kunde ermitteln, lässt sich das wie in Listing 4 dargestellt erreichen. Diese Abfrage gibt die Anzahl der Bestellungen pro

● Listing 6: Beispiel für eine dynamisch erstellte Abfrage

```
var query = new Query("Products");

if (!string.IsNullOrEmpty(userInput.Category))
{
    query.Where("Category", userInput.Category);
}

if (userInput.MinPrice.HasValue)
{
    query.Where("Price", ">=",
        userInput.MinPrice.Value);
}

if (userInput.MaxPrice.HasValue)
{
    query.Where("Price", "<=",
        userInput.MaxPrice.Value);
}
```

● Listing 7: Das Ausführen einer erzeugten Abfrage

```
var connection = new SqlConnection(
    "Data Source=MyDb;User Id=User;Password=*****");
var compiler = new SqlServerCompiler();

var db = new QueryFactory(connection, compiler);

var user = db.Query("Users").Where("Id", 1)
    .Where("Status", "Active").First();
```

Kunde zurück, indem sie die Datensätze nach *CustomerId* gruppiert und die Anzahl der Bestellungen zählt. Der generierte SQL-Befehl sieht wie folgt aus:

```
SELECT "CustomerId", COUNT(*) AS OrderCount
FROM "Orders" GROUP BY "CustomerId";
```

Unterabfragen werden innerhalb anderer Abfragen ausgeführt, um komplexe Datenabfragen zu ermöglichen. *SqlKata* erlaubt es, solche verschachtelten Abfragen zu erstellen, wie das Listing 5 zeigt. Hier wird eine Unterabfrage erstellt, die den maximalen Bestellwert *TotalAmount* für einen bestimmten Kunden *CustomerId = 1* ermittelt. Diese Unterabfrage kommt in der Hauptabfrage zum Einsatz, um alle Bestellungen dieses Kunden mit dem maximalen Bestellwert zu finden. Die resultierende SQL-Abfrage sieht folgendermaßen aus:

```
SELECT (SELECT MAX(TotalAmount) FROM "Orders"
WHERE "CustomerId" = 1) AS "TotalAmount" FROM "Orders";
```

In vielen Anwendungen ist es notwendig, Abfragen dynamisch zu erstellen, abhängig von den Eingaben der Benutzer. *SqlKata* bietet dafür ein flexibles API, das es erlaubt, Bedingungen und Abfrageparameter zur Laufzeit hinzuzufügen. Listing 6 zeigt dazu ein Beispiel.

Der Code erstellt eine Abfrage für Produkte und fügt dynamisch Bedingungen hinzu, basierend auf den Benutzereingaben. Wenn der Benutzer eine Kategorie auswählt, wird diese als Filter hinzugefügt.

Ebenso kommen Mindest- und Höchstpreisbedingungen zum Einsatz, aber nur dann, wenn der Benutzer entsprechende Werte angibt.

Die resultierende SQL-Abfrage variiert je nach den Benutzereingaben und könnte wie folgt aussehen: ▶

```
SELECT * FROM Products WHERE Category = 'Electronics'
AND Price >= 100 AND Price <= 500;
```

Ausführung von SQL-Anweisungen

Wie zu Beginn des Artikels angedeutet, bietet sich die Bibliothek *SqlKata* dafür an, SQL-Anweisungen direkt auszuführen, um mit den Ergebnisdaten im Code weiterarbeiten zu können. Die bisherigen Beispiele haben lediglich die SQL-Anweisungen als Zeichenketten erzeugt, um diese auszuführen.

Listing 7 zeigt das am Beispiel einer SQL-Server-Verbindung, die zunächst erstellt wird. Zusätzlich ist ein spezifischer Compiler notwendig, um den SQL-Dialog korrekt zu erzeugen, in diesem Fall ebenfalls für den SQL Server. Beides wird in die *QueryFactory* hineingegeben, die daraus ein Datenbank-Objekt erzeugt, was nun nicht nur den SQL-Befehl generiert, sondern über Methoden wie *Get* oder *First* auch Datensätze gemäß der Abfrage zurückliefert. Dieses Feature ist in allen Projekten nützlich, in denen keine andere Bibliothek oder kein anderes Framework zum Einsatz kommt, um die Datenbankabfragen abzusetzen und zu verarbeiten. Auf diese Weise kann *SqlKata* beide Features abdecken.

Beispiele zu Pagination, Insert, Update und Delete

Neben der Erstellung komplexer Abfragen unterstützt *SqlKata* auch häufig verwendete Datenbankoperationen wie *Pagination*, also die seitengenaue Datenabfrage, sowie die Operationen *Insert*, *Update* und *Delete*. *Pagination* ist ein wichtiger Aspekt, wenn es darum geht, große Datenmengen in überschaubaren Teilen abzurufen, insbesondere in Webanwendungen.

SqlKata bietet eine einfache Möglichkeit, *Pagination* durchzuführen, indem die Methoden *Limit* und *Offset* verwendet werden. Angenommen, wir möchten eine Liste von Benutzern abrufen, aber nur zehn Benutzer pro Seite anzeigen. Ein Beispiel dazu zeigt **Listing 8**.

Diese Abfrage ruft die ersten zehn aktiven Benutzer *IsActive = true* aus der *Users*-Tabelle ab. Die *Limit*-Methode bestimmt die Anzahl der zurückzugebenden Datensätze, in diesem Fall also zehn, während die *Offset*-Methode bestimmt, ab welchem Datensatz die Abfrage beginnen soll. Die resultierende SQL-Abfrage sieht beispielsweise wie folgt aus:

```
SELECT "Id", "Name", "Email" FROM "Users"
WHERE "IsActive" = true LIMIT 10;
```

Das Einfügen von neuen Datensätzen in eine Datenbank ist ebenfalls eine der grundlegendsten Operationen, die *SqlKata* unterstützt. Die *Insert*-Methode ermöglicht es, Daten ein-

● Listing 8: Beispiel für die Pagination einer Abfrage

```
var page = 1;
var pageSize = 10;

var query = new Query("Users")
    .Select("Id", "Name", "Email")
    .Where("IsActive", true)
    .Limit(pageSize)
    .Offset((page - 1) * pageSize);
```

fach und effizient in eine Tabelle einzufügen. Das Beispiel in **Listing 9** geht davon aus, einen Benutzer in die *Users*-Tabelle einzufügen. Die entsprechende SQL-Anweisung sieht wie folgt aus:

```
INSERT INTO "Users" ("Name", "Email", "IsActive")
VALUES ('John Doe', 'john.doe@example.com', true);
```

SqlKata generiert diese Anweisung automatisch und fügt die entsprechenden Parameter hinzu, wodurch SQL-Injection-Angriffe verhindert werden.

Listing 10 zeigt ein Beispiel für eine *Update*-Anweisung, um gezielt Daten in Datensätzen einer Tabelle zu verändern. Diese Abfrage aktualisiert die E-Mail-Adresse des Benutzers mit der *Id = 1* in der *Users*-Tabelle. Die generierte SQL-Anweisung sieht wie folgt aus:

```
UPDATE "Users" SET "Email" = 'new.email@example.com'
WHERE "Id" = 1;
```

Das Löschen von Datensätzen aus einer Datenbank ist eine weitere grundlegende Operation, die *SqlKata* durch die *Delete*-Methode unterstützt.

Diese Methode ermöglicht das sichere und effiziente Entfernen von Datensätzen basierend auf bestimmten Bedingungen. **Listing 11** zeigt dazu ein Beispiel, um Benutzer aus der *Users*-Tabelle zu löschen, die nicht mehr aktiv sind. Nachfolgend ist die erzeugte SQL-Anweisung zu sehen:

```
DELETE FROM "Users" WHERE "IsActive" = false;
```

Diese Beispiele zeigen, dass auch CRUD-Operationen und häufige Anwendungsfälle wie die seitenweise Datenabfrage mit *SqlKata* kein Problem darstellen.

● Listing 9: Das Hinzufügen von Daten über ein INSERT-Statement

```
var query = new Query("Users").AsInsert(
    new {
        Name = "John Doe",
        Email = "john.doe@example.com",
        IsActive = true
    });
```


Alternative Projekte

Dapper [4] ist keine direkte Alternative zu SqlKata im Sinne eines Query Builders, sondern ein Micro-ORM, der SQL-Abfragen direkt ausführt und die Ergebnisse auf .NET-Objekte mappt. Es ist besonders beliebt wegen seiner Geschwindigkeit und Einfachheit. Während SqlKata hauptsächlich die Erstellung von Abfragen erleichtert, setzt Dapper darauf, dass Entwickler ihre SQL-Abfragen selbst schreiben. Dapper gilt als sehr schnell und leichtgewichtig, mit einer direkten Kontrolle über die SQL-Abfragen. Es ist zudem einfach zu integrieren und verlangt keine komplexe Konfiguration. Eine Abstraktion der SQL-Abfragen besteht aber nicht, und ein Fluent API ist auch nicht vorhanden.

Entity Framework (EF) [5] ist ein umfassender ORM, der SQL-Abfragen durch LINQ-Abfragen abstrahiert und automatisch SQL für eine Vielzahl von Datenbanktypen generiert. Es ist Teil des .NET Frameworks und bietet eine mächtige, aber dennoch einfache Möglichkeit, Datenbankoperationen durchzuführen, ohne dass man sich um die Details der SQL-Syntax kümmern muss. Das Entity Framework bietet eine starke Integration in das .NET-Ökosystem, die Unterstützung für Migrationen und automatisches Erstellen von Datenbankmodellen. Das Framework kommt aber mit einem höheren Overhead im Vergleich zu leichtgewichtigeren Alternativen. Es bietet zudem weniger Kontrolle über die generierten SQL-Abfragen und ist insgesamt für einfache Projekte häufig überdimensioniert.

● Listing 10: Daten aktualisieren über UPDATE

```
var query = new Query("Users")
    .Where("Id", 1)
    .AsUpdate(new {
        Email = "new.email@example.com"
    });
```

NHibernate [6] ist ein weiterer populärer ORM für .NET, der eine mächtige Abstraktionsebene für Datenbankzugriffe bietet. Es ähnelt EF Core, ist jedoch älter und bietet eine Vielzahl von Konfigurationsmöglichkeiten und Anpassungen. Das Framework gilt als sehr mächtig sowie flexibel und bietet eine Unterstützung für eine Vielzahl von Datenbanken inklusive umfangreichen Mapping-Optionen. Es ist somit gut geeignet für komplexe und große Anwendungen. NHibernate ist aber auch komplexer in der Einrichtung und Nutzung. Es hat eine steilere Lernkurve im Vergleich zu anderen ORMs und ist mittlerweile weniger populär als EF Core.

SQLBuilder [7] ist ein weniger bekanntes, aber einfach zu verwendendes Query-Builder-Tool für .NET. Es bietet ein ähnliches Fluent API wie SqlKata. Es ist flexibel in Bezug auf SQL-Datenbanken. Das Projekt bietet allerdings weniger Funktionen und Flexibilität als SqlKata und hat eine kleinere Community.

● Listing 11: Das Löschen von Daten

```
var query = new Query("Users")
    .Where("IsActive", false)
    .AsDelete();
```

Fazit

SqlKata ist ein leistungsstarker und vielseitiger Query Builder für .NET, der die Erstellung von SQL-Abfragen vereinfacht und den Entwicklungsprozess beschleunigt. Mit seinem intuitiven Fluent API ermöglicht SqlKata die Erstellung komplexer Abfragen, ohne dass tiefgehende SQL-Kenntnisse erforderlich sind. Die Bibliothek abstrahiert die Details der SQL-Syntax und bietet eine plattformunabhängige Lösung, die mit verschiedenen Datenbanktypen kompatibel ist.

Die Nutzung von SqlKata ist einfach und simpel, und die Schnittstelle in Form des Fluent API macht genau das, was das Projekt verspricht. Wer keine Lust auf das Schreiben von Abfragen als Zeichenkette hat, findet mit SqlKata eine schnelle und einfache Möglichkeit, durch Code SQL-Abfragen zu erstellen. Ein ORM ist SqlKata aber nicht.

Das Mapping von Daten auf Objekte ist ebenfalls nicht enthalten, wie auch keine Abstraktion von der Datenbank oder Datenbankfunktionen. Wer das in einem Projekt braucht, ist mit SqlKata wahrscheinlich schlecht bedient und sollte weitersehen. Für alle anderen bietet die Bibliothek eine solide Lösung für SQL-Anweisungen. Das Projekt verdient sich ein „Gut +“. Die Dokumentation ist sehr gut. Das Projekt ist zwar in die Jahre gekommen, was aber nichts heißen muss, da es gut funktioniert. Aktuell ist allerdings nicht sicher, ob es weiterentwickelt und damit gepflegt wird. ■

[1] Die Website zu SqlKata, <https://sqlkata.com>

[2] Das Repository auf GitHub zur Bibliothek, www.dotnetpro.de/SL2411Frameworks1

[3] Der Playground auf der Website zu SqlKata, <https://sqlkata.com/playground>

[4] Die Website zu Dapper, www.learnmapper.com

[5] Die Website zu Entity Framework, www.dotnetpro.de/SL2411Frameworks2

[6] Die Website zu NHibernate, <https://nhibernate.info>

[7] Die Website zu SQLBuilder, <https://openhms.sourceforge.io/sqlbuilder>



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.

@FDeitelhoff

dnpCode

A2411Frameworks

KI UND DATA

Vom Hype zum Mehrwert

KI und Daten bestimmen viele Bereiche des Lebens – auch die Softwareentwicklung. Die Artikelserie untersucht, wo die Mehrwerte liegen und wie Wertschöpfung gelingen kann.

Künstliche Intelligenz (KI) hat sich in den letzten Jahren von einer Vision der Zukunft zu einer greifbaren Realität entwickelt, die unsere Art zu arbeiten, zu leben und Geschäfte zu machen tiefgreifend verändert. Die rapide Weiterentwicklung von KI-Technologien hat nicht nur die Effizienz in vielen Branchen gesteigert, sondern auch völlig neue Geschäftsfelder eröffnet. Parallel dazu hat der explosionsartige Anstieg der verfügbaren Datenmengen die Grundlage für diesen Wandel gelegt. Daten und KI bilden gemeinsam das Rückgrat der modernen digitalen Wirtschaft.

Diese Artikelserie beleuchtet die verschiedenen Aspekte und Anwendungen von KI, mit einem besonderen Fokus auf generative Modelle, die als potenzieller Gamechanger in vielen Bereichen gelten. Bei aller Begeisterung bleibt indes die Frage: Inwiefern sind diese Technologien wirklich in der Praxis angekommen, und welchen messbaren Nutzen bieten sie heute schon? Diese Serie beleuchtet verschiedene Möglichkeiten für die Umsetzung und Integration von Modellen und Technologien und untersucht, wie sich daraus Mehrwert schaffen lässt.

Die Bedeutung von KI

Künstliche Intelligenz, oft als die „nächste industrielle Revolution“ bezeichnet [1], ist weit mehr als nur ein Schlagwort. Sie umfasst eine Vielzahl von Technologien, die Maschinen in die Lage versetzen können, Aufgaben zu erledigen, die bisher menschliche Intelligenz erforderten. Dies reicht von einfachen Automatisierungsprozessen bis hin zu komplexen Systemen, die lernen, vorhersagen und Entscheidungen treffen können. Der Einsatz von KI hat bereits signifikante Auswirkungen auf verschiedene Branchen gezeigt, von der Fertigung über das Gesundheitswesen bis hin zur Finanzdienstleistung.

Ein zentraler Faktor für die wachsende Bedeutung von KI ist ihre Fähigkeit, große Datenmengen effizient zu verarbeiten und daraus Muster zu erkennen, die für Menschen kaum erfassbar wären. Diese Fähigkeit hat zu bedeutenden

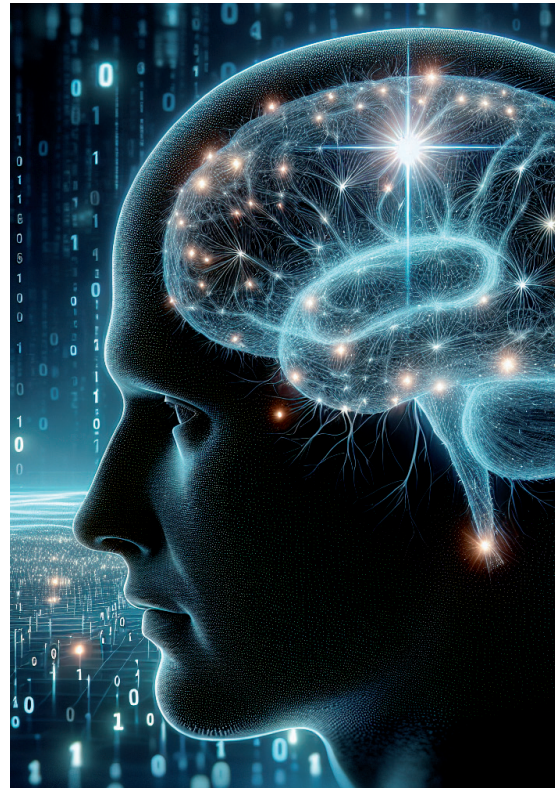


Foto: EMGenie

Durchbrüchen in Bereichen wie der Bild- und Spracherkennung, der prädiktiven Analyse und der Prozessoptimierung geführt. Unternehmen nutzen KI heute, um ihre Geschäftsprozesse zu optimieren, Kosten zu senken, die Kundenerfahrung zu verbessern und letztlich ihre Wettbewerbsfähigkeit zu steigern.

Trotz dieser Fortschritte bleibt die Implementierung von KI eine Herausforderung [2]. Die Integration in bestehende Systeme erfordert nicht nur technisches Know-how, sondern auch ein tiefes Verständnis der jeweiligen Geschäftsprozesse. Zudem sind ethische und datenschutzrechtliche Fragen zu berücksichtigen, die die breite Anwendung von KI in der Praxis weiterhin bremsen. Neben der Modellauswahl sind Fragen zur Integration von KI-unterstützten Algorithmen zu klären. Ein Themengebiet, das im weiteren Verlauf der Artikelserie eingehender betrachtet wird.

Die Bedeutung von Daten

Daten bilden die Grundlage für fundierte Entscheidungen, ermöglichen präzise Vorhersagen und dienen als Treibstoff [3] für künstliche Intelligenz. Ohne qualitativ hochwertige

Daten wären die beeindruckenden Fortschritte in der KI kaum möglich gewesen.

Die Bedeutung von Daten zeigt sich besonders in der Datenwissenschaft, die als Schnittstelle zwischen Informatik, Statistik und Fachwissen dient. Datenwissenschaftler analysieren und interpretieren Daten, um wertvolle Erkenntnisse zu gewinnen, die Unternehmen dabei helfen, ihre Strategien zu optimieren und neue Geschäftsmöglichkeiten zu identifizieren. Die Qualität der Daten ist dabei entscheidend: Nur mit sauberen, vollständigen und aktuellen Daten als Basis lassen sich zuverlässige Modelle entwickeln, die in der Praxis Bestand haben [4].

Mit der weltweit zunehmenden Digitalisierung und Vernetzung entstehen heute unvorstellbare Mengen an Daten. Diese Datenflut erfordert nicht nur neue Technologien zur Speicherung und Verarbeitung, sondern auch innovative Ansätze zur Analyse und Nutzung. Hier kommen KI und insbesondere maschinelles Lernen ins Spiel, die es ermöglichen, aus diesen riesigen Datenmengen wertvolle Informationen zu extrahieren.

Diese Informationen sind aber zunächst aufzufinden und aufzubereiten. Auch die Migration von Daten spielt hierbei eine wichtige Rolle, um nach dieser Migration KI-Systeme nutzen zu können. Diese Themengebiete werden ebenfalls im weiteren Verlauf der Artikelserie im Fokus stehen.

Generative AI als Gamechanger

Unter den zahlreichen KI-Technologien hat sich in den letzten Jahren ein Bereich besonders hervor getan: die generative künstliche Intelligenz. Modelle wie GPT (Generative Pretrained Transformer) haben weltweit für Aufsehen gesorgt, indem sie Texte, Bilder, Musik und sogar Videos generieren können, die oftmals kaum von Inhalten zu unterscheiden sind, die von Menschen erstellt wurden. Diese Fähigkeit, kreativ zu sein, hat das Potenzial, zahlreiche Branchen zu revolutionieren. Die prognostizierte Entwicklung der generativen KI auf dem deutschen Markt zeigt Bild 1.

Generative AI wird oft als Gamechanger [5] bezeichnet, da sie die Art und Weise, wie Inhalte erstellt werden, grundlegend verändert. Sie ermöglicht es Unternehmen, personalisierte Marketingkampagnen zu gestalten, kreative Prozesse zu automatisieren und neue Produkte schneller und kostengünstiger zu entwickeln. Im Gesundheitswesen könnten generative Modelle dazu beitragen, neue Medikamente zu entwerfen oder personalisierte Behandlungspläne zu erstellen. Im Unterhaltungssektor könnten sie die Produktion von Filmen, Musik und Videospielen transformieren.

Doch trotz solcher vielversprechenden Anwendungen steckt die generative KI noch in den Kinderschuhen. Die Qualität der erzeugten Inhalte variiert stark, und oft sind die Modelle noch nicht in der Lage, den Kontext oder die Absicht hinter einer Anfrage vollständig zu verstehen. Darüber hinaus bestehen ethische Bedenken hinsichtlich

der Nutzung von generativen Modellen, insbesondere wenn es um die Erstellung von Inhalten geht, die als täuschend „echt“ wahrgenommen werden können.

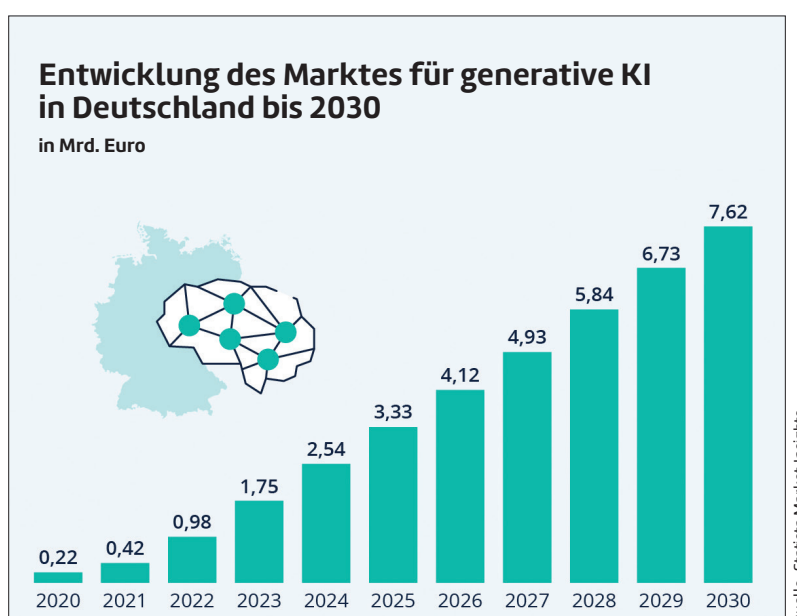
Der tatsächliche Wert von generativer KI in der Praxis wird daher erst in den kommenden Jahren messbar sein. Es wird darauf ankommen, wie gut es gelingt, diese Technologie in bestehende Geschäftsprozesse zu integrieren und gleichzeitig die damit verbundenen Herausforderungen zu meistern. Dabei wird es entscheidend sein, nicht nur auf die technischen Fähigkeiten zu achten, sondern auch die gesellschaftlichen und ethischen Implikationen zu berücksichtigen. Ein Schwerpunkt dieser Artikelserie liegt daher auf dem Verständnis und der Nutzung von generativen Modellen sowie deren Integration in eigene Unternehmensanwendungen und Prozesse.

Der Wandel von Welten

In den letzten Jahren hat sich das Feld der Datenwissenschaften erheblich weiterentwickelt, und ebenso sieht sich die Softwareentwicklung in einem ständigen Wandel. Beide Disziplinen, einst getrennte Welten, nähern sich durch die zunehmende Bedeutung von künstlicher Intelligenz und datengetriebenen Prozessen immer mehr an.

Die Datenwissenschaft hat sich von einer spezialisierten, akademisch geprägten Disziplin zu einem wesentlichen Bestandteil der modernen Geschäftswelt entwickelt. Diese Transformation wird oft als Übergang von einer handwerklichen zu einer industriellen Herangehensweise beschrieben. Während in den frühen Tagen der Datenanalyse einzelne Experten in mühsamer Kleinarbeit Muster in Daten zu erkennen suchten und Modelle entwickelten, stehen heute standardisierte Prozesse und skalierbare Technologien im Vordergrund.

Die handwerkliche Ära der Datenwissenschaft war vor allem geprägt von individuellen Projekten, die stark von ►



Der deutsche Markt für generative KI soll Prognosen zufolge bis 2030 ein Volumen von 7,62 Milliarden Euro erreichen (Bild 1)

den spezifischen Fähigkeiten und Erfahrungen der jeweiligen Datenwissenschaftler abhingen. Diese Experten arbeiteten oft isoliert, und ihre Methoden waren ebenso vielfältig wie ihre Hintergründe. Das Ergebnis war eine Vielzahl von Ansätzen, die in vielen Fällen nur schwer replizierbar waren und stark auf die manuelle Arbeit des Wissenschaftlers angewiesen waren. Dieser Ansatz ermöglichte zwar Innovation und Kreativität, stieß jedoch schnell an seine Grenzen, wenn es darum ging, datenbasierte Lösungen in großem Maßstab zu implementieren.

Mit der zunehmenden Verbreitung von Big Data und der Notwendigkeit, riesige Datenmengen in Echtzeit zu analysieren, begann sich ein Wandel abzuzeichnen. Unternehmen erkannten, dass die bisherige handwerkliche Herangehensweise ineffizient war und nicht die erforderliche Skalierbarkeit bot. Dies führte zur Entwicklung und Einführung industrieller Methoden in der Datenwissenschaft.

Die industrielle Herangehensweise an Datenwissenschaft zeichnet sich durch den Einsatz standardisierter Tools, Prozesse und Frameworks aus. Ziel ist es, datenbasierte Lösungen schnell und in großem Maßstab zu entwickeln und zu imple-

mentieren. Der Fokus liegt auf Effizienz, Wiederholbarkeit und Automatisierung. Hier spielen Plattformen wie Databricks, Hadoop und Apache Spark eine zentrale Rolle, die es ermöglichen, Datenpipelines zu automatisieren und maschinelle Lernmodelle in Produktionsumgebungen zu skalieren.

mentieren. Der Fokus liegt auf Effizienz, Wiederholbarkeit und Automatisierung. Hier spielen Plattformen wie Databricks, Hadoop und Apache Spark eine zentrale Rolle, die es ermöglichen, Datenpipelines zu automatisieren und maschinelle Lernmodelle in Produktionsumgebungen zu skalieren.

Ein weiterer Aspekt dieser Entwicklung ist die zunehmende Integration von DevOps-Prinzipien in die Datenwissenschaft. DataOps, eine Erweiterung von DevOps, zielt darauf ab, den Lebenszyklus von Datenanalysen zu automatisieren und zu optimieren. Dies umfasst die Automatisierung von Datenflüssen, die Integration von Prozessen zur Qualitätssicherung und die kontinuierliche Bereitstellung von Analysen. Durch diese Ansätze wird der Datenwissenschaftler zunehmend zu einem Manager von Prozessen und weniger zu einem handwerklichen Datenkünstler.

Dieser Wandel hat auch das Berufsbild des Datenwissenschaftlers verändert. Wo früher kreative Problemlösung und individuelle Expertise im Vordergrund standen, sind heute technisches Know-how, Prozessmanagement und die Fähigkeit, in interdisziplinären Teams zu arbeiten, entscheidend. Die industrielle Herangehensweise an Datenwissenschaft

Die Softwareentwicklung zwischen Aufbruch und Abbruch

Parallel zur Transformation der Datenwissenschaften durchläuft auch die Softwareentwicklung einen tiefgreifenden Wandel (Bild 2). Die Einführung von KI und Cloud-Technologien und der zunehmende Einsatz von Automatisierung verändern die Art und Weise, wie Software entwickelt, getestet und bereitgestellt wird. Dieser Wandel stellt die Branche vor neue Herausforderungen, bietet aber auch enorme Chancen.

Traditionell war die Softwareentwicklung ein langwieriger, linearer Prozess, der stark auf manuelle Codierung und iterative Verbesserungen angewiesen war. Mit dem Aufkommen von agilen Methoden und DevOps-Prinzipien hat sich dieser Prozess jedoch beschleunigt und flexibilisiert. Diese Entwicklung hat es ermöglicht, Software schneller und effizienter zu entwickeln, indem sie den Übergang von starren, planbasierten Modellen zu flexiblen, inkrementellen Ansätzen vollzog.

Heute steht die Softwareentwicklung an einem Scheideweg: Einerseits gibt es einen Aufbruch zu neuen Methoden und Technologien, die die Art und Weise, wie Software entwickelt wird, revolutionieren (Bild 3). Andererseits besteht das Risiko eines Abbruchs, wenn alte Methoden und Techniken nicht schnell genug durch moderne Ansätze ersetzt werden.

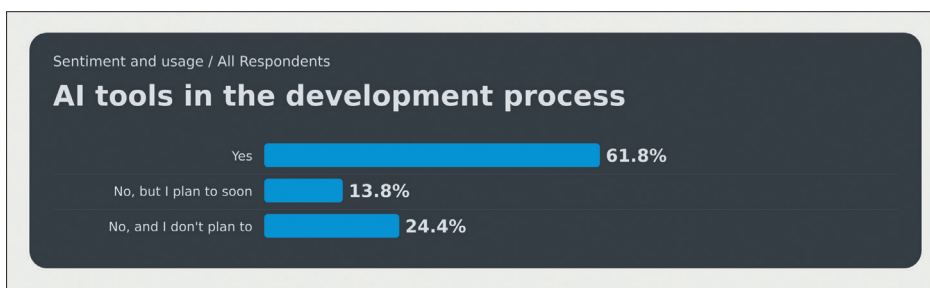
Vor allem der Einsatz von KI und maschinellem Lernen treibt den Aufbruch in der Softwareentwicklung voran [6]. Diese Tech-

nologien ermöglichen es, Code zu generieren, zu testen und zu optimieren, wobei viele der manuellen Aufgaben, die traditionell mit der Softwareentwicklung verbunden sind, automatisiert werden. KI-gestützte Entwicklungstools wie GitHub Copilot oder Tabnine unterstützen dabei, effizienter zu arbeiten und komplexe Probleme schneller zu lösen.

Ein weiterer wichtiger Aspekt dieses Aufbruchs ist die zunehmende Verlagerung von Prozessen der Softwareentwicklung in die Cloud. Cloud-basierte Entwicklungsumgebungen bieten eine nie da gewesene Flexibilität und Skalierbarkeit, die es Unternehmen ermöglicht, Entwicklungsressourcen nach Bedarf zu nutzen und Software schneller auf den Markt zu bringen. Diese Entwicklung hat auch zur Verbreitung von Microservices-Architekturen beigetragen, die es ermöglichen, Anwendungen modularer und anpassungsfähiger zu gestalten.

Doch trotz dieser Fortschritte steht die Softwareentwicklung vor der Herausforderung, sich von veralteten Technologien und Methoden zu lösen. Der Abbruch droht, wenn Unternehmen nicht in der Lage sind, den Übergang zu modernen Entwicklungspraktiken zu vollziehen. Dies zeigt sich ins-

Quelle: survey.stackoverflow.co/2024



76 Prozent der Befragten planen, 2024 KI-Tools in ihren Entwicklungsprozessen einzusetzen, oder tun dies bereits, so ein Ergebnis des jährlichen Stack Overflow Survey (Bild 2)

besondere in der anhaltenden Abhängigkeit von Legacy-Systemen, die oft schwer zu modernisieren sind und die Flexibilität neuer Technologien einschränken.

KI in der modernen Softwareentwicklung

Sprachmodelle, insbesondere solche, die auf maschinellem Lernen basieren, haben sich zu einem unverzichtbaren Werkzeug für Entwickler entwickelt. Modelle wie GPT (Generative Pre-trained Transformer) und BERT (Bidirectional Encoder Representations from Transformers) sind in der Lage, natürliche Sprache zu verstehen und zu generieren, was zu einer neuen Art der Interaktion zwischen Entwicklern und ihren Werkzeugen geführt hat.

Ein bedeutendes Beispiel für den Einsatz von Sprachmodellen in der Softwareentwicklung ist die Code-Vervollständigung. Werkzeuge wie GitHub Copilot nutzen fortschrittliche Sprachmodelle, um Entwicklern beim Schreiben von Code zu helfen. Diese Modelle analysieren den Kontext des geschriebenen Codes und schlagen passende Codezeilen oder sogar ganze Funktionen vor. Dies beschleunigt nicht nur den Entwicklungsprozess, sondern hilft auch, Fehler zu vermeiden, indem es kontextbezogene Vorschläge liefert.

Darüber hinaus lassen sich Sprachmodelle bei der Dokumentation von Code einsetzen. Sie sind in der Lage, automatisch verständliche Kommentare und Dokumentationen basierend auf dem Code zu generieren, was die Verständlichkeit und Wartbarkeit des Codes erheblich verbessert. Dies ist besonders in großen Teams von Vorteil, in denen Konsistenz und Klarheit entscheidend sind.

Ein weiterer interessanter Aspekt ist die Möglichkeit, Sprachmodelle zur Erstellung von Testszenarien zu verwenden. Diese Modelle können basierend auf den Code-Änderungen mögliche Tests generieren, die sicherstellen, dass die neuen Funktionen ordnungsgemäß ablaufen und keine bestehenden Funktionen beeinträchtigen.

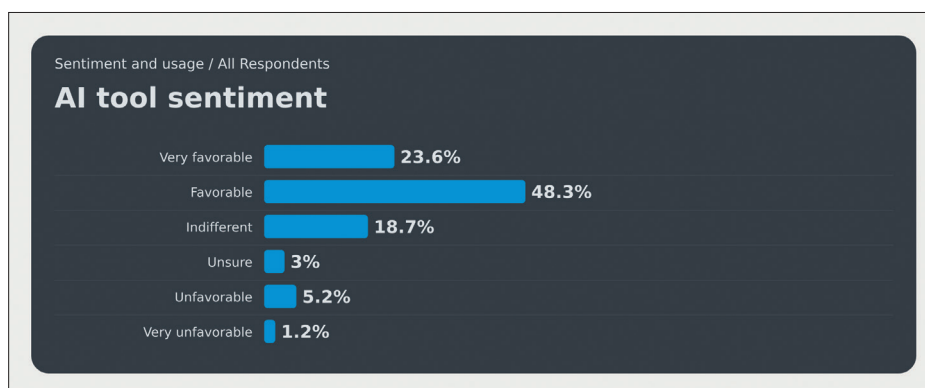
Die Integration von Sprachmodellen in den Entwicklungsprozess birgt jedoch auch Herausforderungen. Eine davon ist die Abhängigkeit von den Daten, auf denen die Modelle trainiert wurden. Wenn diese Daten verzerrt oder unvollständig sind, können die Vorschläge der Modelle suboptimal sein oder sogar Fehler verursachen. Darüber hinaus besteht die Gefahr, dass Entwickler sich zu sehr auf die Vorschläge der Modelle verlassen und ihre eigenen Fähigkeiten zur Problemlösung vernachlässigen.

Sprachmodelle als Schwunggeber

Wenige Technologien haben in den vergangenen zwei Jahren so viel Aufmerksamkeit erregt wie Sprachmodelle. Diese Modelle, die darauf trainiert sind, menschliche Sprache zu

verstehen und zu generieren, stehen im Mittelpunkt der aktuellen Diskussionen über die Zukunft von KI. Von Chatbots über Textgenerierung bis hin zu Übersetzungen: Sprachmodelle scheinen das Potenzial zu haben, unsere Art, zu kommunizieren und Informationen zu verarbeiten, grundlegend zu verändern. Doch ist diese Technologie ein Schwunggeber für die nächste Stufe der Digitalisierung, oder herrscht eher eine vorsichtige Skepsis gegenüber den Risiken, die sie mit sich bringt?

Die Einführung leistungsfähiger Sprachmodelle wie GPT-3 und GPT-4 hat sowohl Begeisterung als auch Besorgnis ausgelöst. Auf der einen Seite stehen die unglaublichen Möglichkeiten, die diese Modelle bieten. Unternehmen und Entwickler sind begeistert von den neuen Wegen, die sich durch die Nutzung von Sprachmodellen eröffnen, sei es in der Automatisierung von Kundeninteraktionen oder der Erstellung von Inhalten in einem noch nie da gewesenen Maßstab. Die Geschwindigkeit, mit der Text generiert, Anfragen beantwortet und sogar komplexe Zusammenhänge erklärt werden können, ist beeindruckend.



72 Prozent der befragten Entwickler stehen laut Stack Overflow Survey dem Einsatz von KI-Tools positiv gegenüber (Bild 3)

Diese Euphorie lässt sich als eine neue Aufbruchstimmung beschreiben. Es scheint, als ob wir an der Schwelle zu einer Ära stehen, in der Mensch-Maschine-Interaktionen so natürlich und effektiv werden wie zwischenmenschliche Gespräche. Sprachmodelle versprechen die Art und Weise, wie wir arbeiten, lernen und kommunizieren, radikal zu verändern. Besonders im Unternehmensumfeld wird intensiv daran gearbeitet, diese Technologien in den Alltag zu integrieren und daraus einen Wettbewerbsvorteil zu ziehen.

Doch wo Licht ist, da ist auch Schatten. Die rasante Entwicklung der Sprachmodelle hat auch Ängste geweckt. Viele fragen sich, ob diese Technologie zu schnell voranschreitet, ohne dass ihre Auswirkungen vollständig verstanden werden. Die Sorge, dass Sprachmodelle in falschen Händen großen Schaden anrichten könnten, ist allgegenwärtig. Auch die Frage, ob diese Modelle irgendwann die Kontrolle über die von ihnen generierten Inhalte verlieren könnten, steht im Raum. Die Vorstellung, dass KI-gestützte Systeme in der Lage sind, Texte zu generieren, die von menschengemachten ►

Inhalten nicht zu unterscheiden sind, ist gleichermaßen faszinierend und beängstigend.

Es bleibt die Frage: Ist die Begeisterung berechtigt, oder sollten wir uns mehr Zeit nehmen, um die möglichen Konsequenzen zu durchdenken? Diese Diskussion ist nicht nur technischer Natur, sondern hat auch tiefgreifende ethische, gesellschaftliche und wirtschaftliche Dimensionen, die im zweiten Teil dieser Artikelserie genauer beleuchtet werden.

Chatbots, Textgenerierung, Übersetzungen ...

Generative KI gilt als transformative Technologie mit breiten Anwendungsmöglichkeiten in verschiedenen Bereichen. In der Medien- und Unterhaltungsindustrie wird sie zur Erstellung von Texten, Bildern, Musik und Videos eingesetzt, beispielsweise um Kurzgeschichten im Stil eines bestimmten Autors zu verfassen oder Musikstücke zu komponieren. In der Industrie und Fertigung unterstützt generative KI die Produktentwicklung und optimiert Produktionsprozesse, während sie in der Kreativwirtschaft Künstlern und Designern als Inspirationsquelle dient. Im Handel und Marketing findet sie Anwendung in der Generierung von Werbung und automatisierten Produktbeschreibungen.

Darüber hinaus wird die generative KI in Unternehmensanwendungen integriert, um Aufgabenmanagement und Kommunikation zu verbessern, sowie in der Dokumentenverarbeitung und Übersetzung, wo sie große Textmengen analysieren und eigene Texte generieren kann. Diese vielseitigen Einsatzmöglichkeiten optimieren Prozesse und eröffnen neue Chancen in zahlreichen Branchen.

Ein Anwendungsfeld für Sprachmodelle ist die Entwicklung von Chatbots. Diese digitalen Assistenten sind inzwischen in vielen Unternehmen zum Standard geworden und haben die Kundenkommunikation revolutioniert. Dank fortschrittlicher Sprachmodelle können moderne Chatbots nicht nur einfache Anfragen beantworten, sondern auch komplexere Gespräche führen, die auf den individuellen Bedürfnissen der Nutzer basieren. Dies ermöglicht es Unternehmen, ihre Kunden rund um die Uhr zu betreuen, ohne dafür auf menschliche Mitarbeiter angewiesen zu sein.

Die Textgenerierung ist ein weiteres Feld, in dem Sprachmodelle große Fortschritte gemacht haben. Ob es darum geht, Artikel, Berichte oder gar ganze Bücher zu schreiben – die Fähigkeiten moderner KI-Modelle, kohärente und kontextuell passende Texte zu erstellen, sind beeindruckend. Diese Technologie wird zunehmend in der Content-Erstellung eingesetzt, um Texte zu generieren, die für Marketingkampagnen, soziale Medien oder andere Plattformen benötigt werden. Die Effizienz und Geschwindigkeit, mit der KI Texte erstellen kann, stellt eine massive Ressourceneinsparung dar und erlaubt es Unternehmen, mehr Inhalte in kürzerer Zeit zu produzieren.

Übersetzungsdienste haben ebenfalls von der Entwicklung fortschrittlicher Sprachmodelle profitiert. Durch den Einsatz von KI können Texte in Echtzeit und mit hoher Genauigkeit in mehrere Sprachen übersetzt werden. Dies hat das Potenzial, Sprachbarrieren abzubauen und die globale Kommunikation zu fördern. Sprachmodelle haben die Fähigkeit, nicht

nur Wort-für-Wort-Übersetzungen zu liefern, sondern auch den Kontext und die kulturellen Nuancen einer Sprache zu berücksichtigen, was zu qualitativ hochwertigeren Übersetzungen führt.

Doch so faszinierend diese Anwendungen auch sein mögen, sie bringen auch Herausforderungen mit sich. Wie können wir sicherstellen, dass die von Sprachmodellen generierten Inhalte immer korrekt und angemessen sind? Welche Verantwortung tragen die Entwickler und Nutzer dieser Technologien?

Diese und ähnliche Fragen sind Diskussionsgrundlage in weiteren Teilen der Artikelserie. Sie schaffen ein besseres Verständnis der Chancen und Risiken, die mit der Nutzung von Sprachmodellen verbunden sind.

Aus- und Einblick

Wie in der Einleitung des vorliegenden Textes bereits erwähnt, befasst sich die Artikelserie mit verschiedenen Bereichen rund um KI und Daten, mit einem Schwerpunkt auf Softwareentwicklung und generative Modelle. Neben der allgemeinen Einführung in die Themenbereiche in diesem Beitrag stehen in den kommenden Ausgaben unterschiedliche Sprachmodelle, Daten, Datenaufbereitung und Datenmigrationen, die Verarbeitung von strukturierten Daten sowie zahlreiche Beispiele im Fokus.

Bei diesen Beispielen handelt es sich um konkrete Aufgabenstellungen, die mit den genannten Technologien in den vergangenen Monaten umgesetzt wurden. Eingeschlossen sind dabei die Entwicklung von Prototypen für Software-as-a-Service-Modelle und weitere konkreten Implementierungen. Diese Beispiele sollen zudem zeigen, was möglich ist, wo es schnell hakt und worauf zu achten ist, um erfolgreiche Umsetzungen zu realisieren. ■

- [1] *Künstliche Intelligenz als industrielle Revolution*, www.dotnetpro.de/SL2411KIundData1
- [2] *Die Implementierung von KI als Herausforderung*, www.dotnetpro.de/SL2411KIundData2
- [3] *Daten als Grundlage und Treibstoff*, www.dotnetpro.de/SL2411KIundData3
- [4] *Modelle für die Praxis*, www.dotnetpro.de/SL2411KIundData4
- [5] *Generative KI als Gamechanger*, www.dotnetpro.de/SL2411KIundData5
- [6] *KI als Treiber der Softwareentwicklung*, www.dotnetpro.de/SL2411KIundData6



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.

@FDeitelhoff

dnpCode

A2411 KlundData

GENAI – MIT OPENAI UND DEM AZURE-API NEUE INHALTE GENERIEREN

Was siehst du auf dem Bild?

Die Integration von Bildeingaben in GPT-Modelle ermöglicht eine Vielzahl praktischer Anwendungen.

GPT-Modelle haben sich in den letzten Jahren als revolutionäre Technologie in der Welt der künstlichen Intelligenz (KI) etabliert (siehe den Kasten **GPT-Modelle**). Sie sind für ihre Fähigkeit bekannt, natürliche Sprache zu verstehen und sinnvolle Antworten zu generieren. Diese Modelle wurden von OpenAI entwickelt und haben verschiedene Iterationen durchlaufen, von GPT-1 bis zu den aktuellsten der GPT-4-Modellfamilie [1][2]. Jedes dieser Modelle baut auf den Fortschritten seiner Vorgänger auf und bringt erhebliche Verbesserungen in der Fähigkeit, kontextualisierte und kohärente Texte zu erstellen.

Die Bedeutung der GPT-Modelle liegt in ihrer Vielseitigkeit und Leistung. Sie sind in der Lage, eine Vielzahl von Aufgaben im Bereich der natürlichen Sprachverarbeitung (Natural Language Processing, NLP [3]) zu bewältigen, darunter Text- und Code-Generierung, maschinelles Übersetzen, das Beantworten von Fragen und sogar das Schreiben von Gedichten und Geschichten. Diese Fähigkeiten machen GPT-Modelle zu einem unverzichtbaren Werkzeug für Entwickler, Forscher und Unternehmen, die fortschrittliche KI-Lösungen in ihre Anwendungen integrieren möchten.

Ein herausragendes Merkmal der GPT-Modelle ist ihre Fähigkeit, große Mengen an Textdaten zu verarbeiten und daraus kontextbezogene Antworten zu generieren. Dies wird durch das Training auf umfangreichen Datensätzen erreicht,

die eine breite Palette von Themen und Stilen abdecken. Dadurch können GPT-Modelle präzise und relevante Informationen liefern, die auf den spezifischen Anforderungen und Fragen der Benutzer basieren.

Die Integration von Bildeingaben in GPT-Modelle wie GPT-4-Turbo und GPT-4o stellt einen weiteren Meilenstein in der Entwicklung der GPT-Technologie dar. Diese Erweiterung befähigt die Modelle, nicht nur Text, sondern auch visuelle Informationen zu verarbeiten und zu interpretieren. Das eröffnet neue Möglichkeiten für Anwendungen, bei denen Text- und Bilddaten kombiniert werden können, wie zum Beispiel in der medizinischen Bildgebung, der visuellen Datenanalyse, der Erstellung multimedialer Inhalte und anderen Anwendungsbereichen. Und genau dieser spannende Anwendungsfall wird im Folgenden weiter vorgestellt und mit Praxisexperimenten überprüft.

Integration von Bildeingaben in GPT-Modelle

Im Kern basiert die Integration von Bildeingaben in GPT-Modelle auf der Erweiterung der Transformer-Architektur, die ursprünglich für die Verarbeitung natürlicher Sprache entwickelt wurde. Mit der Einführung von Modellen wie DALL-E [4] und CLIP [5] hat OpenAI gezeigt, wie leistungsfähig diese Architektur sein kann, wenn sie auf visuelle Daten angewendet wird. DALL-E, ein Modell, das in der Lage ist, aus Textbeschreibungen Bilder zu generieren, und CLIP, das Bilder und Text miteinander in Beziehung setzen kann, sind herausragende Beispiele für die Integration von visuellen und textuellen Daten.

Auf der Plattform von OpenAI [6] können Entwickler jetzt die erweiterten Fähigkeiten dieser Modelle nutzen, um innovative Anwendungen zu erstellen. Diese Integration erfordert eine sorgfältige Vorbereitung und das Verständnis der zugrunde liegenden Technologien. Die Modelle werden mit großen Datensätzen trainiert, die sowohl Text als auch Bilder umfassen, um die Beziehung zwischen den beiden Datenformen zu lernen. Dies ermöglicht ihnen, präzise und kontextbezogene Antworten zu generieren, die sowohl textuelle als auch visuelle Informationen berücksichtigen.

Azure, die Cloud-Computing-Plattform von Microsoft, bietet mit Azure OpenAI Service [7] ebenfalls umfassende Unterstützung für die Integration von Bildeingaben in GPT-Modelle. Durch die Nutzung von Azure können Entwickler von den robusten Infrastruktur- und Skalierungsmöglichkeiten profitieren, die für das Training und die Bereitstellung solcher Modelle erforderlich sind. Azure bietet eine Vielzahl von ►

● GPT-Modelle

Generative vortrainierte Transformatoren oder kurz GPTs [21] (Generative Pre-trained Transformers) sind eine Variante der Großsprachmodelle [22] (Large Language Models, LLM) und ein prominentes Framework für generative künstliche Intelligenz (GenAI). GPTs sind künstliche neuronale Netzwerke, die in Aufgaben der natürlichen Sprachverarbeitung verwendet werden. GPTs basieren auf der Transformer-Architektur [8], sind anhand großer Datensätze unmarkierten Textes vortrainiert und in der Lage, neuartige Inhalte zu generieren, die den von Menschen erzeugten Inhalten ähneln. Ab 2023 verfügen die meisten LLMs über diese Eigenschaften und werden manchmal allgemein als GPTs bezeichnet.

Das erste GPT wurde 2018 von OpenAI eingeführt [23]. OpenAI hat sehr einflussreiche GPT-Grundmodelle veröffentlicht, die fortlaufend nummeriert wurden und die „GPT-n“-Reihe bilden.

Diensten und Tools, die speziell für die Arbeit mit künstlicher Intelligenz und maschinellem Lernen entwickelt wurden, einschließlich der Unterstützung für OpenAI-Modelle.

Die Kombination von OpenAI und Azure ermöglicht es Entwicklern, leistungsfähige KI-Lösungen zu entwickeln, die Text- und Bildverarbeitung nahtlos integrieren. Diese Plattformen bieten die notwendigen Werkzeuge und Ressourcen, um komplexe Modelle zu erstellen, zu trainieren und in großem Maßstab bereitzustellen.

In diesem Artikel werden wir tiefer in die technischen Details und praktischen Beispiele der Bildeingabe-Fähigkeiten in GPT-Modellen eintauchen. Die Einrichtung und Nutzung dieser Technologien wird erläutert und durch praktische Experimente und Beispiele veranschaulicht. Interessierte Leser können so die Vorteile dieser fortschrittlichen Technologie in ihren eigenen Projekten und Anwendungen voll ausschöpfen.

Wie Bildeingabe funktioniert

Die Integration von Bildeingaben in GPT-Modelle beruht auf der Transformer-Architektur [8]. Dieses neuronale Netzwerk, das ursprünglich für die Verarbeitung großer Mengen natürlicher Sprachdaten entwickelt wurde, kann nun auch visuelle Daten effizient verarbeiten. Der Schlüssel dazu liegt in der Art und Weise, wie die Daten verarbeitet werden: Sowohl Text- als auch Bilddaten werden in Vektoren umgewandelt, die als Eingaben für das Netzwerk dienen. Diese Vektorisierung ermöglicht es dem Modell, Beziehungen zwischen verschiedenen Datenpunkten zu erkennen und zu nutzen.

Bilder müssen vor der Verarbeitung durch ein GPT-Modell in eine für das Modell verständliche Form gebracht werden. Dies erfolgt durch den Prozess der Vektorisierung, bei dem Bilder in numerische Darstellungen umgewandelt werden. Diese Vektoren repräsentieren die verschiedenen Merkmale des Bildes, beispielsweise Farben, Formen und Texturen. Ein häufig verwendeter Ansatz ist die Nutzung von Convolutional Neural Networks (CNNs) [9], um die relevanten Merkmale (Features) aus den Bildern zu extrahieren und sie in Vektoren zu überführen. Diese Vektoren können dann in das GPT-Modell eingespeist werden.

Ein wesentlicher Aspekt der Bildeingabe in GPT-Modellen ist die nahtlose Integration von Text- und Bilddaten. Dies wird durch die Verwendung von Embeddings [10] erreicht, die sowohl Text als auch Bilder in einem gemeinsamen Vektorraum darstellen. Dadurch kann das Modell Zusammenhänge zwischen Text- und Bildinformationen erkennen und nutzen. Zum Beispiel kann das Modell auf Basis eines Bildes eine entsprechende Textbeschreibung generieren oder umgekehrt, basierend auf einer Textbeschreibung, ein Bild identifizieren oder erstellen.

Um ein GPT-Modell mit Bildeingaben effektiv nutzen zu können, ist ein umfassendes Training erforderlich. Dies beinhaltet das Training des Modells auf großen Datensätzen, die sowohl Text- als auch Bildinformationen enthalten. Während des Trainings lernt das Modell, die Beziehungen zwischen Text- und Bilddaten zu erkennen und zu nutzen. Nach dem initialen Training kann das Modell weiter verfeinert werden, um spezifische Aufgaben oder Anwendungsfälle besser zu

erfüllen. Dies erfolgt durch Fine-Tuning, bei dem das Modell auf spezialisierte Datensätze trainiert wird, um seine Leistung in bestimmten Bereichen zu optimieren.

OpenAI- und Azure-Implementierungen

Die Implementierung von Bildeingabe-Funktionalitäten in den GPT-Modellen GPT-4 Turbo und GPT-4o wird sowohl von OpenAI als auch der Azure-Cloud-Plattform angeboten. Die beiden Plattformen bieten leistungsfähige Tools und Modelle für die Verarbeitung und Integration von Bilddaten in die GenAI-Prozesse [11], unterscheiden sich jedoch in bestimmten Aspekten und Ansätzen ihrer Technologien.

OpenAI hat Pionierarbeit geleistet und als Vorreiter die Entwicklung von GPT-Modellen etabliert. Angeboten werden spezialisierte Modelle mit Vision-Fertigkeiten wie DALL-E, CLIP, GPT-4 Turbo und GPT-4o:

- **DALL-E:** Dieses Modell ist darauf spezialisiert, aus Textbeschreibungen Bilder zu generieren [4]. Es nutzt eine Transformer-Architektur, um textuelle Eingaben in visuelle Ausgaben zu übersetzen. DALL-E ist nützlich für kreative Anwendungen wie die Erstellung von Kunstwerken, Designprototypen und Illustrationen auf Basis von Textvorgaben.
- **CLIP:** Das Contrastive Language-Image Pretraining (CLIP) kann Bilder und Text miteinander verknüpfen [5]. Es wird verwendet, um die Beziehung zwischen Bildern und ihren Beschreibungen zu lernen, was es ermöglicht, Bildinhalte zu verstehen und textuelle Beschreibungen zu generieren oder umgekehrt. CLIP ist vielseitig einsetzbar in Bereichen wie Bildersuche, Klassifizierung und Annotation.
- **GPT-4 Turbo:** Ein großes, multimodales Modell, das sowohl Text- als auch Bildeingaben akzeptiert und Text ausgibt [1]. Dank seines breiteren Allgemeinwissens und fortschrittlichen logischen Denkvermögens kann es schwierige Probleme mit größerer Genauigkeit lösen als frühere Modelle.
- **GPT-4o:** Das fortschrittlichste Modell der neu vorgestellten multimodalen Modelle von OpenAI [1] (das „o“ steht für „omni“). Wie GPT-4 Turbo ist es multimodal (akzeptiert Text- oder Bildeingaben und gibt Text aus) und hat die gleiche hohe Intelligenz, ist jedoch viel effizienter – es generiert Text doppelt so schnell und ist 50 Prozent günstiger. Darüber hinaus bietet GPT-4o im Vergleich mit den anderen OpenAI-Modellen die beste visuelle Leistung und die beste Performance in nicht englischen Sprachen.

OpenAI bietet API-Zugänge zu diesen Modellen, die es Entwicklern ermöglichen, die Bildverarbeitungsfähigkeiten in ihre Anwendungen zu integrieren. Die Nutzung der APIs ist unkompliziert, erfordert jedoch ein gewisses Maß an Verständnis für die Modellparameter und die Verarbeitung von Eingabedaten.

Azure, die Cloud-Computing-Plattform von Microsoft, bietet ebenfalls umfassende Unterstützung für die Integration von Bildeingaben in den Modellen GPT-4 Turbo und GPT-4o. Dafür wird ein OpenAI-Service auf Azure bereitgestellt, der es Azure-Kunden ermöglicht, die leistungsfähigen Modelle von OpenAI direkt über die Azure-Plattform zu nutzen. Diese Integration bietet den Vorteil einer skalierbaren und sicheren

**Bildanalyse–
Vorlage 1:**
Big Ben in
London (**Bild 1**)



**Bildanalyse–
Vorlage 2:** Pont
Alexandre III
in Paris (**Bild 2**)



ren Infrastruktur, die speziell für die Bedürfnisse von Unternehmen ausgelegt ist. Die Plattform bietet Tools für die Verwaltung und Überwachung von KI-Anwendungen sowie umfangreiche Dokumentation und Support für Entwickler.

Im Vergleich zeigen die Plattformen OpenAI und Azure im Hinblick auf Lösungen mit multimodalen GPT-Modellen leicht unterschiedliche Schwerpunkte:

- **Benutzerfreundlichkeit:** Beide Plattformen stellen benutzerfreundliche Schnittstellen zur Verfügung, aber Azure punktet mit seiner integrierten Entwicklungsumgebung und den zahlreichen vorgefertigten Diensten. OpenAI erfordert etwas mehr technisches Wissen, insbesondere bei der Feinabstimmung und Implementierung spezialisierter Modelle.
- **Flexibilität:** OpenAI liefert spezialisierte Modelle, die besonders leistungsfähig in ihren jeweiligen Anwendungsbereichen sind. Azure verfügt dagegen über eine umfassendere Infrastruktur und eine größere Auswahl an KI-Diensten, was es flexibler für unterschiedliche Anwendungsfälle macht.
- **Kosten und Skalierbarkeit:** Azure stellt eine skalierbare Infrastruktur bereit, die besonders für Unternehmen attraktiv ist, die große Datenmengen verarbeiten müssen. OpenAI stellt leistungsstarke Modelle zur Verfügung, deren Nutzung jedoch mit höheren Kosten verbunden sein kann, vor allem wenn umfangreiche Rechenressourcen benötigt werden.

In der Summe bieten sowohl OpenAI als auch Azure leistungsfähige Lösungen für die Integration von Bildeingaben in GPT-Modelle. Die Wahl der Plattform hängt von den spezifischen Anforderungen des Projekts ab, einschließlich der gewünschten Benutzerfreundlichkeit, Flexibilität sowie ge-

planten Kosten und erforderlichen Skalierbarkeit. Im nächsten Abschnitt werden wir die Schritte zur Einrichtung und Nutzung dieser Technologien detailliert beschreiben.

Schnellstart

Die grundlegenden Vorbereitungsschritte bis zum ersten API-Aufruf und die Einrichtung der verwendeten Entwicklungsumgebungen für OpenAI- und/oder Azure-Plattformen sind bereits ausführlich im vorangegangenen Artikel des Autors über Textgenerierung mit GPT-Modellen [12] beschrieben worden. Deshalb soll an dieser Stelle eine stichpunktartige Zusammenfassung genügen:

- **API-Zugang erhalten:** Um das OpenAI-API zu nutzen, benötigt man einen API-Schlüssel. Registrieren Sie sich deshalb (soweit nicht bereits geschehen) auf der OpenAI-Plattform (siehe [13] für Details) oder auf Azure (siehe [14] für Details). Erstellen Sie auf der OpenAI-Plattform einen *API-Key* (siehe *Account Setup* in [13]) und/oder erstellen Sie auf Azure einen OpenAI-Service beziehungsweise gehen Sie zu einem vorhandenen und kopieren Sie *API-Key*, *END-POINT* und *DEPLOYMENT-NAME* (siehe *Retrieve key and endpoint* in [14]).
- **Umgebung einrichten:** Richten Sie die Umgebungsvariablen ein (siehe *Setup your API Key* in [13], *Environment variables* in [14] und in der *README*-Datei [15])
- **Installation der benötigten Python-Pakete:**
`pip install --upgrade openai`
- **Installation der benötigten C#-Pakete:** Fügen Sie die OpenAI-Client-Bibliothek zu Ihrem .NET-Core-Projekt hinzu:
`dotnet add package Azure.AI.OpenAI --prerelease`

Wenn alles richtig eingerichtet ist, steht Ihnen nichts mehr im Weg, um erste Experimente mit multimodalen GPT-Modellen durchzuführen. Legen Sie los – der Autor wird Sie bei der spannenden Entdeckungsreise begleiten!

Was siehst du auf dem Bild?

Um die Leistungsfähigkeit der GPT-Modelle mit Bildeingabe zu demonstrieren, führen wir mehrere Experimente durch, bei denen die Modelle Bilder analysieren und interpretieren. Diese Experimente veranschaulichen die Fähigkeiten der GPT-Modelle, visuelle Informationen zu verarbeiten und darauf basierend sinnvolle Textausgaben zu generieren. Alle vorgestellten Experiment-Programmeingaben und die jeweiligen Modell-Antworten können dem Artikeltext und dem zugehörigen Demo-Projekt unter [16] entnommen und ausprobiert werden.

Im ersten Experiment wird untersucht, wie die multimodalen GPT-Modelle auf die Frage „Was siehst du auf dem Bild?“ reagieren. Auf **Bild 1** ist der bekannte Big Ben von London dargestellt, und die **Listings 1.A** bis **1.D** enthalten dazu mögliche Varianten derselben API-Anfrage via jeweils reinen REST-Aufruf in Python und in C#, mit OpenAI und AzureOpenAI-Python-Paketen. ►

Führt man nun eine oder alle vier Varianten derselben Modellanfrage aus, erhält man verblüffende Antworten, die einen glauben lassen, dass das Modell das Bild tatsächlich „sieht“. Hier eine der Modell-Antworten, die der Autor beim Experimentieren bestaunen durfte:

```
This image shows the iconic clock tower known as
Big Ben, part of the Palace of Westminster in London,
United Kingdom.
The Palace of Westminster, also known as the Houses of
Parliament, is visible in the background.
The picture is taken from across the River Thames with
Westminster Bridge in the foreground,
where people and vehicles can be seen crossing. The sky
is mostly cloudy with patches of blue.
```

Vergleicht man die Listings 1.A bis 1.D, wird deutlich, dass diese fast identisch sind. Man sieht dennoch, dass die Varianten mit Python-Paketen durch die Verwendung von jeweils OpenAI- oder AzureOpenAI-Python-Paketen transparenter und besser lesbar sind.

Alle sich hieran anschließenden Experimente des Artikels wurden mit Python OpenAI durchgeführt. Wie sich erwiesen hat, ist es unter dem Aspekt der gelieferten Ergebnisse tatsächlich weitgehend Geschmackssache, welche Programmiersprache (Python, C# oder weitere), welche API-Plattform (OpenAI oder Azure) beziehungsweise welches API-Programmiermodell (REST-Aufruf oder ein OpenAI- beziehungsweise AzureOpenAI-Python-Paket) man dazu verwen-

det. Und sollten im konkreten Fall spezifische technische Anforderungen vorliegen, lassen sich die entsprechenden Modalitäten schnell und problemlos umgestalten – entweder erledigt man dies selbst, oder man bittet sogar einen digitalen Code-Assistenten wie ChatGP oder GitHub CoPilot darum.

Das zweite Experiment diente dem Versuch, mittels Prompt-Engineering-Methoden (siehe auch [12] und [17]) aus dem GPT-Modell einen „Dichter“ zu machen. Dafür reichte es aus, eine System-Meldung davorzuschalten:

```
{"role": "system",
  "content": "You are a poetic assistant, skilled
in explaining different facts and concepts with
creative flair."}
```

Und tatsächlich lieferte das GPT-Modell als Ergebnis ein Gedicht über das weltberühmte Bauwerk. Details können Sie im Demo-Projekt zu diesem Artikel unter [16] in Listing 2 (als ►

● Listing 1.A. OpenAI-API-Aufruf via REST in Python

```
from openai import OpenAI

response = OpenAI().chat.completions.create(
    model="gpt-4o",
    messages=[
        {
            "role": "user",
            "content": [
                {"type": "text", "text":
                "What do you see in this image?"},
                {
                    "type": "image_url",
                    "image_url": {
                        "url": "https://bit.ly/img_city_1"
                    }
                }
            ]
        }
    ], max_tokens=1000
)

print(response.choices[0].message.content)
```

● Listing 1.B. OpenAI-API-Aufruf via REST in C#

```
import os
import requests
import json
from openai import OpenAI

api_key = os.getenv("OPENAI_API_KEY")

headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {api_key}"
}

payload = {
    "model": "gpt-4o",
    "messages": [
        {
            "role": "user",
            "content": [
                {"type": "text", "text":
                "What do you see in this image?"},
                {"type": "image_url", "image_url": {
                    "url": "https://bit.ly/img_city_1" }}
            ]
        }
    ], "max_tokens": 1000
}

response = requests.post("https://api.openai.com/v1/
chat/completions", headers=headers, json=payload)
response_data = json.loads(response.text)

print(response_data.get("choices")[0].get(
    "message").get("content"))
```


● Listing 1.C. API-Aufruf mit Python-Paket

```

using System.Net.Http.Headers;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

class Program
{
    static async Task Main(string[] args)
    {
        string apiKey =
            Environment.GetEnvironmentVariable(
                "OPENAI_API_KEY");

        var client = new HttpClient();
        client.DefaultRequestHeaders.Accept.Add(
            new MediaTypeWithQualityHeaderValue(
                "application/json"));
        client.DefaultRequestHeaders.Authorization =
            new AuthenticationHeaderValue(
                "Bearer", apiKey);

        var payload = new
        {
            model = "gpt-4o",
            messages = new[]
            {
                new
                {
                    role = "user",
                    content = new object[]
                    {
                        new { type = "text",
                            text = "What do you see in
                                this image?" },
                        new { type = "image_url",
                            image_url = new {
                                url = "https://bit.ly/
                                    img_city_1" } } } } } }

        var jsonPayload =
            JsonConvert.SerializeObject(payload);
        var content = new StringContent(
            jsonPayload, System.Text.Encoding.UTF8,
            "application/json");

        var response = await client.PostAsync(
            "https://api.openai.com/v1/chat/
                completions", content);
        var responseString = await
            response.Content.ReadAsStringAsync();
        var responseData =
            JsonConvert.DeserializeObject<JObject>(
                responseString);

        string result = responseData["choices"]?[0]?
            ["message"]?["content"]?.ToString();
        Console.WriteLine(result);
    }
}

```

● Listing 1.D. Azure-OpenAI-Service-API-Aufruf mit Python-Paket

```

import os
from openai import AzureOpenAI

# Azure OpenAI endpoint like:
# https://<MY_RESOURCE>.openai.azure.com/openai/
# deployments/<MY_DEPLOYMENT>/extensions"

response = AzureOpenAI(
    api_key=os.getenv("AZURE_OPENAI_API_KEY"),
    api_version='2023-12-01-preview',
    base_url=os.getenv("AZURE_OPENAI_ENDPOINT"))
    .chat.completions.create(
        model="<MY_DEPLOYMENT>",
        messages=[
            {
                "role": "user",
                "content": [
                    {"type": "text", "text":
                        "What do you see in this image?"},
                    {"type": "image_url", "image_url": {
                        "url":
                            "https://bit.ly/img_city_1" }}
                ]
            }
        ], max_tokens=1000
    )

print(response.choices[0].message.content)

```

Datei *Listing_2.py*) und in der dazugehörigen Text-Antwortdatei *Listing_2_Response.txt* nachschlagen.

Im dritten Experiment erschweren wir die Aufgabenstellung für das GPT-Modell wiederum ein wenig: Kombiniert wurden dort eine Bildeingabe der europäischen Sehenswürdigkeit Pont Alexandre III in Paris (**Bild 2**) und zwei Bitten um Antworten in Textform – einmal darüber zu schreiben, was man sieht, und dann ein Gedicht auf Deutsch zu kreieren (siehe **Listing 3** für Details).

Bei seiner Antwort verblüffte das GPT-4o-Modell den Autor mit folgenden doch sehr beachtlichen Strophen:

Auf der Brücke aus vergangener Zeit,
Die Lichter strahlen, golden und breit.
Wasser fließt, ein sanfter Strom,
Die Nacht erwacht, ein stiller Ton.

Statuen aus Bronze, stolz und stark,
Halten Wacht bis hin zum Morgenmark.
Gelbgolden schimmert ihr feines Licht,
Ein zauberhaftes, zartes Gedicht.

Am Himmel ruht der Abendstern,
Ein Versprechen, wundersam und fern.
Geflüster der Stadt, so sanft und leis,
Verträumt, doch majestätisch preis.

Inmitten dieser Pracht und Ruh,
Die Brücke, sie erzählt im Nu,
Von Zeiten voller Glanz und Ehr,
Von Herzen leicht und Herzen schwer.

Vergleich von Sehenswürdigkeiten

Big Ben und das Parlament, London

Vorteile

- **Historische Bedeutung:** Ein Meisterwerk der neogotischen Architektur und Symbol der britischen Geschichte.
- **Prominente Lage:** Zentrale Lage in London, zugänglich durch verschiedene öffentliche Verkehrsmittel.
- **Touristischer Anziehungspunkt:** Ein Muss für Erstbesucher in London, bietet zahlreiche Fotomöglichkeiten.
- **Geführte Touren:** Möglichkeit, an Führungen teilzunehmen und mehr über die Geschichte des Gebäudes zu erfahren.

Nachteile

- **Menschenmassen:** Oft überfüllt, besonders in der Hochsaison, was das Erlebnis beeinträchtigen kann.
- **Eingeschränkter Zugang:** Der Zugang zum Inneren des Parlaments ist teilweise eingeschränkt und vorab zu reservieren.
- **Lärm:** Hohes Verkehrsaufkommen in der Umgebung kann störend sein.

Pont Alexandre III, Paris

Vorteile

- **Ästhetik:** Bekannt für seine prachtvolle und kunstvolle Gestaltung mit Lampen, Statuen und Dekorationen.
- **Romantische Atmosphäre:** Besonders bei Nacht beleuchtet, bietet es eine romantische Kulisse für Spaziergänge.
- **Zentrale Lage:** Verbindung zwischen den bedeutenden Sehenswürdigkeiten wie Les Invalides und dem Champs-Élysées.
- **Kultur und Geschichte:** Eine der schönsten Brücken in Paris, reich an Geschichte und kultureller Bedeutung.

Nachteile

- **Verkehr:** Starker Verkehr kann das Erlebnis beeinträchtigen und zu Lärmbelästigung führen.
- **Wetterabhängigkeit:** Schlechtes Wetter kann den Besuch unangenehm machen, da es sich um eine Freiluftattraktion handelt.
- **Sicherheit:** In den belebten Gegenden gibt es ein erhöhtes Risiko für Taschendiebstahl.

„Reiseanalyse“ für London und Paris (**Bild 3**)

Paris, Stadt der Liebe, im Abendschein,
Lädt ein zum Träumen, Groß und Klein.
Und so bleib ich hier und schau,
Ins Wunder dieser stillen Au.

Das vierte Experiment ist eine Kombination aus sowohl mehreren Text- als auch mehreren Bildeingaben und bittet das Modell, die beiden vorher genannten Sehenswürdigkeiten von London und Paris zu vergleichen und ein Ergebnis als *README*-Markdown zu liefern, etwa so:

● Listing 3: Multimodale Anfrage an das OpenAI-GPT-4o-Modell

```
import json
from openai import OpenAI

with open("./demo/vision/images_list.json", 'r') as file:
    images_urls = json.load(file)

response = OpenAI().chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "You are a poetic assistant, skilled in explaining different facts and concepts with creative flair."},
        {
            "role": "user",
            "content": [
                {"type": "image_url", "image_url": {
                    "url": images_urls.get("Paris") }},
                {"type": "text",
                 "text": "Write about what do you see."},
                {"type": "text",
                 "text": "Write a poem in German."}
            ]
        }
    ], max_tokens=1000
)

# print & save result
result = response.choices[0].message.content
print(result)
with open(f"./demo/vision/Listing_3_Response.txt",
        "w", encoding="utf-8") as outfile:
    outfile.write(result)
```

```
cal-packages\Python38\site-packages\openai\_base_client.py", line 988, in _request
    raise self._make_status_error_from_response(err.response) from None
openai.BadRequestError: Error code: 400 - {'error': {'message': "You uploaded an unsupported image. Please make sure your image is below 20 MB in size and is of one the following formats: ['png', 'jpeg', 'gif', 'webp']"}, 'type': 'invalid_request_error', 'param': None, 'code': 'invalid_image_format'}}
```

Ausnahme wegen unbekannten Dateiformats (Bild 4)

```
"content": [
  {"type": "text", "text": "Make detailed description and comparison of the pros and cons of the given sightseens."},
  {"type": "text", "text": "Make output in German as README-markdown."},
  {"type": "image_url", "image_url":
    { "url": images_urls.get("London") }},
  {"type": "image_url", "image_url":
    { "url": images_urls.get("Paris") }},
]
```

Als Ergebnis erzeugte das Modell die in Bild 3 dargestellte „Reiseanalyse“-README. Die Details dazu finden Sie im Demo-Projekt zu diesem Artikel unter [16] in Listing 4 (als Datei Listing_4.py) und in der dazugehörigen Antwortdatei Listing_4_Response.md.

Das fünfte Experiment fordert das Modell mit der Analyse der IT-Architektur einer auf dem Bild dargestellten Finanz-

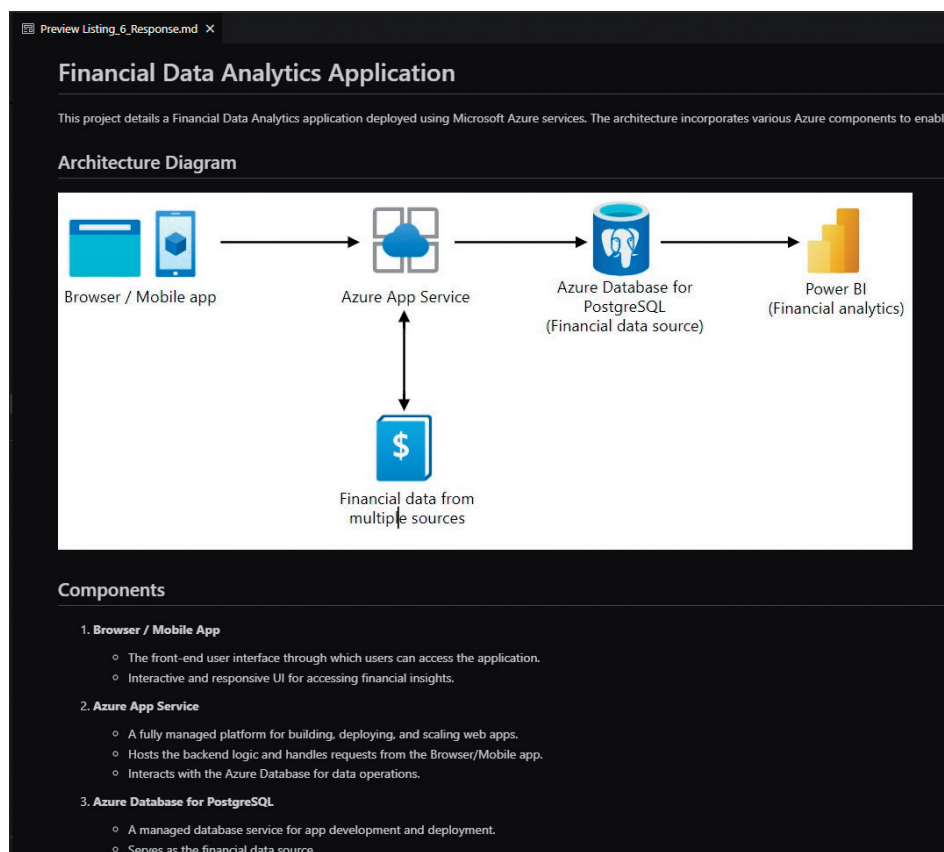
management-App heraus. Dies führte allerdings zu einem herben Rückschlag in Form einer 400er-Fehlermeldung (siehe Bild 4) [18]. Wie der Error-Code zeigt, hängt das offenbar damit zusammen, dass dem GPT-4o-Modell ein Bild in einem nicht unterstützten Format übergeben wurde. Details dazu finden Sie im Demo-Projekt zu diesem Artikel unter [16] in Listing 5 (als Datei Listing_5.py) und in der dazugehörigen Antwortdatei Listing_5_Response.md.

Eine Abhilfe für diesen Fehler schafft das sechste Experiment. In diesem wurde die Architekturbild-Datei zuerst heruntergeladen und lokal mit Base64-Encoding [19] vom binären ins textuelle Format überführt. Dann wurde die im UTF-8-Format codierte Architekturbindeingabe mit weiteren Texteingaben an das Modell mit übergeben (siehe Listing 6). Als Ergebnis produzierte das GPT-4o-Modell eine „durchdachte“ IT-Architektur-Analyse im Markdown-Format (siehe Bild 5).

Im sechsten Experiment ist also nicht nur der Fehler des fünften Experiments behoben, sondern es zeigt darüber hinaus auch einen Weg auf, wie man eine lokale Datei via

OpenAI-API an das Modell mit übergeben kann. Die zugehörige Antwortdatei finden Sie im Demo-Projekt unter [16] in Listing_6_Response.md.

Und zum Abschluss lautete beim siebten Experiment die Fragestellung, ob das multimodale GPT-Modell aus einem Bild den Programmiercode generieren kann. Dafür wurde ein Bild verwendet, das von ChatGPT für einen früheren Artikel des Autors [20] erstellt wurde – es handelt sich um ein Klassendiagramm für ein Autovermietungs-System (siehe Bild 6). Führt man damit ein ähnliches Programm wie bei den vorangegangenen Experimenten aus, kann man ein Analyse-Dokument mit generierten Code-Ausschnitten wie in Bild 7 dankend als Antwort empfangen. Details dazu finden Sie im Demo-Projekt zum Artikel unter [16] in Listing 7 (als Datei Listing_7.py) und in der dazugehörigen Antwortdatei Listing_7_Response.md. ►



IT-Architektur-Analyse auf der Grundlage eines Bildes (Bild 5)

Listing 6: Lokale Datei als Bildeingabe

```
import json
import base64
from openai import OpenAI

# encode the image
image_filename = "ArchitecturePic1.jpg"
with open(f"./demo/vision/{image_filename}", "rb")
    as image_file:
    base64_image =
        base64.b64encode(image_file.read()).decode('utf-8')

# send encoded image via OpenAI API
response = OpenAI().chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "You are a
        professional assistant for Cloud Architecture
        solutions like Azure, AWS etc.."},
        {
            "role": "user",
            "content": [
                {"type": "text", "text":
                "Analyze architecture."},
                {"type": "text", "text":
                "Make output as README-markdown."},
                {"type": "text", "text": f"Include hyperlink
                for Architecture diagramm stored in
                {image_filename}.",
                {"type": "image_url", "image_url": { "url":
                f"data:image/jpeg;base64,{base64_image}" }},
            ]
        }, max_tokens=2000
    )

# print & save result
result = response.choices[0].message.content
print(result)
with open(f"./demo/vision/Listing_6_Response.md", "w",
    encoding="utf-8") as outfile:
    outfile.write(result)
```

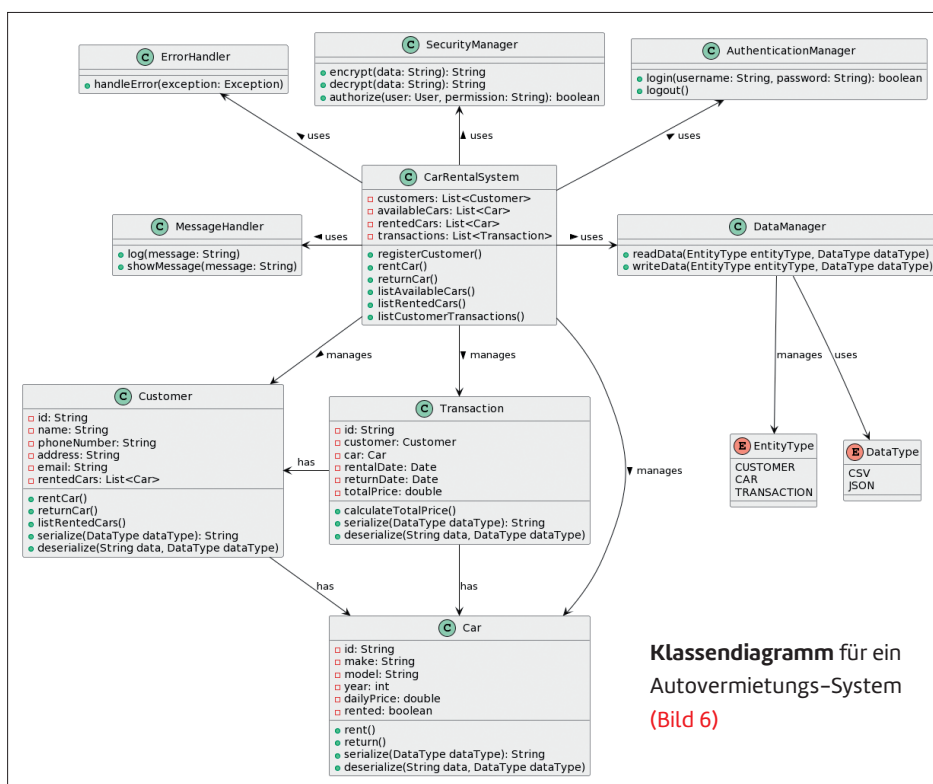
Fazit

Die Integration von Bildeingaben in GPT-Modelle stellt einen bedeutenden Fortschritt in der generativen künstlichen Intelligenz dar, der eine Vielzahl praktischer Anwendungen er-

möglicht. Die GPT-4-Turbo- und GPT-4o-Modelle sind nicht die einzigen verfügbaren multimodalen Modelle für diesen Einsatzzweck, sie stehen aber als prominente Beispiele auf den Plattformen OpenAI und Azure bereit.

Die Anwendungsfälle der multimodalen GPT-Modelle sind nahezu unbegrenzt und können schon jetzt oder in der nächsten Zukunft fast alle kreativen Prozesse unterstützen. Allerdings müssen auch Herausforderungen wie die Sicherstellung der Datenqualität, die Vermeidung von Verzerrungen im Training, ethische Überlegungen sowie regulatorische Rahmenbedingungen für Daten und KI berücksichtigt werden.

Sowohl OpenAI als auch Azure stellen leistungsstarke Lösungen für die Integration von Bildeingaben in GPT-Modelle bereit. Die Wahl der geeigneten Plattform sollte auf den spezifischen Anforderungen des Projekts basieren, einschließlich Benutzerfreundlichkeit, Flexibilität, Kosten und Skalierbarkeit. Mit den richtigen Werkzeugen und Strategien haben Softwarearchitekten und



Klassendiagramm für ein
Autovermietungs-System
(Bild 6)

Entwickler die Möglichkeit, beeindruckende und innovative GenAI-Anwendungen zu schaffen, die nicht nur für ihre Nutzer Erfolg bringen werden, sondern auch ihren Beitrag für Allgemeinwohl und Fortschritt leisten können. ■

- [1] OpenAI, GPT-Models, <https://platform.openai.com/docs/models>
- [2] Microsoft Learn, OpenAI Service-Modelle, www.dotnetpro.de/SL2411AzureOpenAI1
- [3] NLP – Natural language processing bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI2
- [4] OpenAI, DALL·E, Creating images from text, <https://openai.com/index/dall-e/>
- [5] OpenAI, CLIP, Connecting text and images, <https://openai.com/index/clip/>
- [6] OpenAI Platform, www.dotnetpro.de/SL2411AzureOpenAI3
- [7] Azure OpenAI Service, www.dotnetpro.de/SL2411AzureOpenAI4
- [8] Transformer (deep learning architecture) bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI5
- [9] Convolutional neural network bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI6
- [10] Word embedding bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI7
- [11] Generative artificial intelligence bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI8
- [12] Mykola Dobrochynskyy, Erzähl mir doch von Paris, dotnetpro 10/2024, Seite 84 ff., www.dotnetpro.de/A2410OpenAIPython
- [13] OpenAI, Developer Quickstart, www.dotnetpro.de/SL2411AzureOpenAI9
- [14] Microsoft Learn, Quickstart: Use images in your AI chats, www.dotnetpro.de/SL2411AzureOpenAI10
- [15] Mykola Dobrochynskyy, LeanGPTConnect – Beispiel-Projekt für das OpenAI-API bei GitHub, www.dotnetpro.de/SL2411AzureOpenAI11
- [16] Mykola Dobrochynskyy, LeanGPTConnect – Vision-Demos für das OpenAI-API bei GitHub, www.dotnetpro.de/SL2411AzureOpenAI12
- [17] Prompt-Engineering bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI13
- [18] Microsoft Learn, Finance management apps using Azure Database for PostgreSQL, www.dotnetpro.de/SL2411AzureOpenAI14
- [19] Base64 encoding bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI15

```

Preview Listing_7_Response.md X

CarRentalSystem

Manages the core functionality of the car rental system.

using System;
using System.Collections.Generic;

public class CarRentalSystem
{
    private List<Customer> customers = new List<Customer>();
    private List<Car> availableCars = new List<Car>();
    private List<Car> rentedCars = new List<Car>();
    private List<Transaction> transactions = new List<Transaction>();

    public void RegisterCustomer(Customer customer)
    {
        // Implementation for registering a customer
        customers.Add(customer);
    }

    public void RentCar(Customer customer, Car car)
    {
        // Implementation for renting a car
        if (availableCars.Contains(car))
        {
            availableCars.Remove(car);
            rentedCars.Add(car);
            customer.RentCar(car);
        }
    }

    public void ReturnCar(Customer customer, Car car)
    {
        // Implementation for returning a car
        if (rentedCars.Contains(car))
        {
            rentedCars.Remove(car);
            availableCars.Add(car);
        }
    }
}

```

Der generierte Code für das Autovermietungs-System (Bild 7)

- [20] Mykola Dobrochynskyy, Die Code-Revolution, dotnetpro 7/2023, Seite 16 ff., www.dotnetpro.de/A2307Codegenerierung
- [21] Generative pre-trained transformer bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI16
- [22] Large language models bei Wikipedia, www.dotnetpro.de/SL2411AzureOpenAI17
- [23] OpenAI, Improving language understanding with unsupervised learning, www.dotnetpro.de/SL2411AzureOpenAI18



Mykola Dobrochynskyy

ist Senior Architekt bei der Porsche AG. Seine Schwerpunkte sind Softwarearchitektur, modellgetriebene Softwareentwicklung, generative Programmierung, KI sowie Cloud- und dienstorientierte Softwarearchitekturen.

www.xing.com/profile/Mykola_Dobrochynskyy

dnpCode

A2411AzureOpenAI



DATA ACCESS

Die Datenzugriffsberatung mit DR. HOLGER SCHWICHTENBERG

NEUERUNGEN IN ENTITY FRAMEWORK CORE 9.0

AOT noch in Arbeit

Die Preview-Versionen 1 bis 6 von EF Core 9.0 enthalten zahlreiche Neuerungen, aber das größte Projekt, die Kompatibilität mit dem Native-AOT-Compiler, ist „Work in Progress“.

Entity Framework Core 9.0 wird im Rahmen von .NET 9.0 im November 2024 erscheinen. EF Core 9.0 läuft aber nicht nur auf .NET 9.0, sondern auch auf .NET 8.0.

Entity Framework Core besitzt den gleichen Support-Zyklus wie .NET selbst, das bedeutet: Anders als der Vorgänger EF Core 8.0 besitzt die 9.0-Version nur einen „Standard-Term-Support“ (18 Monate) anstelle eines „Long-Term-Support“ für 36 Monate.

Die Datenzugriffskolumne in der dotnetpro-Ausgabe 10/2024 hat bereits die Verbesserungen in Entity Framework Core 9.0 im Bereich der komplexen Typen vorgestellt [1]. Der Artikel in dieser Ausgabe thematisiert die weiteren Neuerungen.

Data Type Test		Enter a SQL expression to filter results (use Ctrl+Space)
Grid	Row #1	
010 011 110	ByteArray	
ABC	ShortArray	[1,2,3]
ABC	IntArray	[1,2,3]
ABC	LongArray	[1,2,3]
ABC	StringArray	["www.IT-Visionen.de","www.dotnet-doktor.de","www.dotnet8.de"]
ABC	DateTimeArray	["2024-06-06T06:54:31.2592251Z","2024-06-07T06:54:31.2592251Z","2024-06-08T06:54:31.2592315Z"]
ABC	BoolArray	[true,false,true]
ABC	GuidArray	["c23a0505-61c2-4005-94a7-64a51e09a181","05e7bc50-726f-41f8-9b2c-0a6ded48dccb","13f26d44-8f20-484a-8b28-5696b884d1b1"]
ABC	EnumArray	[NULL]
ABC	IntList	[4711,42,2024]
ABC	GuidList	["c23a0505-61c2-4005-94a7-64a51e09a181","05e7bc50-726f-41f8-9b2c-0a6ded48dccb","13f26d44-8f20-484a-8b28-5696b884d1b1"]
ABC	ReadOnlyCollection	[4711,42,2024]
ABC	IReadOnlyCollection	[4711,42,2024]
ABC	IReadOnlyList	[4711,42,2024]
ABC	IPAddress	192.168.1.1
ABC	PhysicalAddress	011E919AEF32
010 011 110	Geometry	[NULL]
010 011 110	GeometryCollection	[NULL]
010 011 110	Point	[NULL]
010 011 110	LineString	[NULL]
010 011 110	MultiLineString	[NULL]
010 011 110	MultiPoint	[NULL]
010 011 110	Polygon	[NULL]
010 011 110	MultiPolygon	[NULL]
123	KomplexerTyp_Feld1	42
ABC	KomplexerTyp_Feld2	Hallo

In Datenbanken, außer PostgreSQL, werden die schreibgeschützten Objektmengen als JSON-Array gespeichert (Bild 1)

Schreibgeschützte Objektmengen als Property-Typ

Arrays und typisierte Listen wie `List<int>` für einzelne Properties mit primitiven Typen funktionierten vor Entity Framework Core in der Version 8.0 nur in Verbindung mit dem Datenbankmanagementsystem PostgreSQL, weil es dort einen eigenen Mengentyp für elementare Datentypen gibt.

In Entity Framework Core 8.0 hatte Microsoft die Möglichkeit eingeführt, Arrays und typisierte Listen wie `List<int>`, `List<string>` und `List<Guid>` in anderen Datenbankmanagementsystemen auf JSON-Arrays in einer Textspalte (zum Beispiel `nvarchar(MAX)` in Microsoft SQL Server und `TEXT` in SQLite) abzubilden (Bild 1).

In Version 9.0 von Entity Framework Core erweitert Microsoft diese Mengenabbildungen für einzelne Properties auf die schreibgeschützten .NET-Typen `IReadOnlyList<T>`, `IReadOnlyCollection<T>` und `ReadOnlyCollection<T>`, siehe Listing 1. Bei SQLite und SQL Server entstehen dabei auch wieder JSON-Spalten. Mit PostgreSQL ließ sich dieses Feature



zum Redaktionsschluss für diesen Beitrag noch nicht testen, weil es den Treiber für PostgreSQL erst in Version 9.0 Preview 3 gab, die schreibgeschützten .NET-Typen aber erst in Preview 4 von Entity Framework Core kamen.

Die Verwendung von Properties mit den schreibgeschützten Mengentypen *IReadOnlyList<T>*, *IReadOnlyCollection<T>* und *ReadOnlyCollection<T>* ist auch in LINQ-Abfragen möglich. Entity Framework Core 9.0 übersetzt beispielsweise die folgende LINQ-Befehlsfolge

```
var resultSet = ctx.DataTypeTest
    .Where(x => x.ReadOnlyCollection.Any(
        x => x >= 42))
    .Select(x => new DataTypeTest()
        { ID = x.ID, ReadOnlyCollection =
            x.ReadOnlyCollection })
    .Skip(20)
    .Take(10)
    .ToList();
```



● Listing 1: *IReadOnlyList<T>*, *IReadOnlyCollection<T>* und *ReadOnlyCollection<T>* als OR-Mapping-Spaltentypen

```
public class DataTypeTest
{
    public Int32 ID { get; set; }
    ...
    #region Array-Typ-Mapping -->
        Neu in EFCore 8.0 (zuvor nur für PostgreSQL!)
        [Comment("C#: byte[]")]
        public byte[] ByteArray { get; set; }

        [Comment("C#: short[]")]
        public short[] ShortArray { get; set; }

        [Comment("C#: int[]")]
        public int[] IntArray { get; set; }

        [Comment("C#: long[]")]
        public long[] LongArray { get; set; }

        //[Comment("C#: byte[]")] --> Immer noch nicht
        //möglich in EFC 9.0 :- (
        //public Int128[] Int128Array { get; set; }

        [Comment("C#: string[]")]
        public string[] StringArray { get; set; }

        [Comment("C#: DateTime[]")]
        public DateTime[] DateTimeArray { get; set; }

        [Comment("C#: bool[]")]
        public bool[] BoolArray { get; set; }

        [Comment("C#: Guid[]")]
```

```
public Guid[] GuidArray { get; set; }

    [Comment("C#: ABC[]")]
    public ABC[] EnumArray { get; set; }
    #endregion

    #region Listen --> Neu in EFCore 8.0 (zuvor nur
        für PostgreSQL!)
    [Comment("C#: List<int>")]
    public List<int> IntList { get; set; }

    [Comment("C#: List<Guid>")]
    public List<Guid> GuidList { get; set; }
    #endregion

    #region ReadOnly-Listen (Neu in EFCore 9.0)
    [Comment("C#: ReadOnlyCollection<int>")]
    public ReadOnlyCollection<int> ReadOnlyCollection {
        get; set; }

    [Comment("C#: IReadOnlyCollection<int>")]
    public IReadOnlyCollection<int> IReadOnlyCollection
        { get; set; } = new List<int>();

    [Comment("C#: IReadOnlyList<int>")]
    public IReadOnlyList<int> IReadOnlyList { get; set; }
        = new List<int>();

    #endregion
    ...
}
```

● Listing 2: Einsatz von `HasFillFactor()` mit einem einfachen Schlüssel und einem zusammengesetzten Index

```
modelBuilder.Entity<Person>()
    .HasKey(x => x.ID)
    .HasFillFactor(80);
```

```
modelBuilder.Entity<Person>()
    .HasIndex(x => new { x.Vorname, x.Name })
    .HasFillFactor(80);
```

in nachstehende SQL-Abfrage:

```
SELECT [d].[ID], [d].[ReadOnlyCollection]
FROM [DataTypeTest] AS [d]
WHERE EXISTS (
    SELECT 1
    FROM OPENJSON([d].[ReadOnlyCollection]) WITH (
        [value] int '$') AS [r]
    WHERE [r].[value] >= 42)
ORDER BY (SELECT 1)
OFFSET @__p_0 ROWS FETCH NEXT @__p_1 ROWS ONLY
```

Füllfaktor festlegen

Bei Schlüssel- und Indexspalten können Entwicklerinnen und Entwickler für den Microsoft SQL Server in Entity Framework Core 9.0 nun Angaben zum Füllfaktor machen.

Der Füllfaktor bestimmt den Prozentsatz des Platzes auf einer Seite in der Datenbankdatei, der mit Daten gefüllt werden soll, wobei der Rest auf jeder Seite als freier Platz für zukünftiges Wachstum reserviert wird.

Der Füllfaktor kann nun via `HasFillFactor()` für einfache und zusammengesetzte Schlüsselspalten und Indexe gesetzt werden (siehe [Listing 2](#)).

● Listing 3: Beispiel zum Einsatz von `UseCache()`

```
protected override void OnModelCreating(
    modelBuilder modelBuilder)
{
    // Sequenz erstellen
    modelBuilder.HasSequence<int>("TeamIDSequenz")
        .HasMin(10).HasMax(30) // nur Nummern zwischen 10
        // und 20
        .IsCyclic() // zyklisch wiederholende Nummern
        .StartsAt(11).IncrementsBy(2) // ab 11 jeweils +1
        .UseCache(10); // immer die 10 nächsten Nummern
        // im Hauptspeicher
    // oder: .UseNoCache(); // kein Cache

    // Sequenz auf eine Spalte anwenden
    modelBuilder.Entity<Person>()
        .Property(o => o.TeamID)
        .HasDefaultValueSql(
            "NEXT VALUE FOR TeamIDSequenz");
}
```

Cache-Einstellungen bei der Erstellung von Sequenzen

In Entity Framework Core 9.0 können Entwicklerinnen und Entwickler die Cache-Einstellung von Sequenzen per Fluent-API festlegen. Die Cache-Einstellung einer Sequenz legt fest, wie viele Zahlen auf einmal erzeugt und im RAM gehalten werden, siehe die Dokumentation zu Sequenzen in Microsoft SQL Server unter [2].

Im Standard legt der SQL Server die Größe des Cache selbst fest. Datenbankentwicklerinnen und -entwickler können nun darauf schon bei der Modelldefinition via `UseCache()` und `UseNoCache()` Einfluss nehmen (siehe [Listing 3](#)).

Automatisch kompilierte Modelle

Entity Framework Core erzeugt immer schon erst zur Laufzeit einen Programmcode für das Mapping eines Objektmodells auf ein Datenbankschema. Diese Laufzeitcodegenerierung kann bei großen Objektmodellen zeitaufwendig sein, und Laufzeitcodegenerierung ist zudem nicht kompatibel zu dem Ahead-of-Time-Compiler „Native AOT“, den es seit .NET 7.0 gibt und den Microsoft in .NET 9.0 auch für Datenbankzugriffe mit Entity Framework Core (zumindest in einigen Szenarien) möglich machen will. In .NET 8.0 funktioniert Entity Framework Core leider gar nicht bei aktivierter Native-AOT-Kompilierung.

Seit Version 6.0 von Entity Framework Core gibt es als Alternative zur kompletten Laufzeitkompilierung bereits kompilierte Modelle, bei denen ein Teil der Laufzeitcodegenerierung zur Entwicklungszeit stattfindet.

Der bisherige Weg zum kompilierten Modell sieht so aus: Entwicklerinnen und Entwickler müssen zuerst einen Kommandozeilenbefehl ausführen

```
dotnet ef dbcontext optimize
```

beziehungsweise in der PowerShell-basierten NuGet-Package-Manager-Konsole in Visual Studio eingeben:

```
Optimize-DbContext
```

Danach gilt es im Programmcode noch einen Methodenauf-ruf in `OnConfiguring()` zu ergänzen:

```
.UseModel(Kontextname.Instance)
```

Wichtig dabei: Man muss die kompilierten Modelle an der Kommandozeile immer wieder neu erzeugen, sobald es Änderungen am Objektmodell oder der Kontextklasse gibt.

Alle diese Schritte können in Entity Framework Core 9.0 entfallen: Es gibt Auto-Compiled Models! Das Modell kompilieren zur Entwicklungszeit lässt sich mit einem neuen MS-Build-Task `<EFOptimizeContext>` automatisieren. Dazu muss man zunächst das neue NuGet-Paket `Microsoft.EntityFrameworkCore`.

Tasks [3] (verfügbar seit Entity Framework Core Preview 3) einbinden und dann in der Projektdatei die Einstellung `<EFOptimizeContext>` auf den Wert `true` setzen:

```
<PropertyGroup>
  <EFOptimizeContext>true</EFOptimizeContext>
</PropertyGroup>
```

Bei jedem Übersetzungsvorgang sieht man dann an der Konsole beziehungsweise im Output-Fenster:

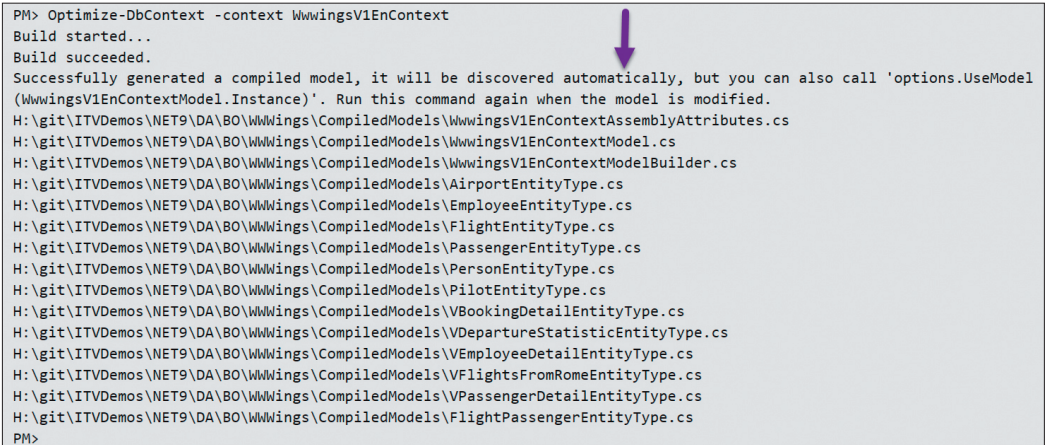
```
Optimizing DbContext...
```

Anpassungen der Modellkompilierung können Entwicklerinnen und Entwickler über weitere Projekteinstellungen wie `<EFStartupProject>`, `<DbContextName>` und `<EFTargetNamespace>` vornehmen.

Der Methodenaufruf `UseModel()` ist beim neuen Vorgehen nicht mehr notwendig, sofern das kompilierte Modell in der gleichen Assembly liegt wie die Kontextklasse. Dies gilt unabhängig davon, ob man den MSBuild-Task `<EFOptimizeContext>` einsetzt oder die Modellkompilierung weiterhin von Hand anstößt (Bild 2).

Typischerere Erstellung von Hierarchie-IDs

In Entity Framework Core 8.0 hatte Microsoft die Unterstützung für hierarchische Datentabellen im Microsoft SQL Ser-



```
PM> Optimize-DbContext -context WwwingsV1EnContext
Build started...
Build succeeded.
Successfully generated a compiled model, it will be discovered automatically, but you can also call 'options.UseModel
(WwwingsV1EnContextModel.Instance)'. Run this command again when the model is modified.
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\WwwingsV1EnContextAssemblyAttributes.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\WwwingsV1EnContextModel.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\WwwingsV1EnContextModelBuilder.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\AirportEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\EmployeeEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\FlightEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\PassengerEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\PersonEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\PilotEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\VBookingDetailEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\VDepartureStatisticEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\VEmployeeDetailEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\VFlightsFromRomeEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\VPassengerDetailEntityType.cs
H:\git\ITVDemos\NET9\DA\BO\WwWings\CompiledModels\FlightPassengerEntityType.cs
PM>
```

Ausgabe bei manuellem Aufruf von `Optimize-DbContext` (Bild 2)

ver mit dem Spaltentyp `hierarchyid` eingeführt. Seitdem gibt es eine korrespondierende .NET-Klasse `Microsoft.EntityFrameworkCore.HierarchyId`.

In .NET 9.0 gibt es in der Klasse `HierarchyId` an der Stelle eine kleine Verbesserung in Form einer neuen Überladung der Methode `Parse()`: Entwicklerinnen und Entwickler können `HierarchyId`-Instanzen nicht nur wie bisher aus einer Zeichenkette (zum Beispiel `"/4/1/3/1/2/"`), sondern auch typischerer auf Basis einer anderen `HierarchyId`-Instanz und Ganzzahlen erstellen, siehe das Beispiel in Listing 4.

ToHashSetAsync()

Analog zur bereits in Entity Framework Core 2.0 eingeführten Materialisierungs-Methode `ToHashSet()` gibt es ab Version 9.0 ein asynchrones Pendant: `ToHashSetAsync()`. Listing 5 zeigt die Verwendung dieser Methode.

Einfluss auf die Parametrisierung

Entwicklerinnen und Entwickler können nun mit den .NET-Funktionen `EF.Parameter()` und `EF.Constant()` erzwingen, dass bei der Übersetzung von LINQ-Abfragen nach SQL ein Wert in der SQL-Abfrage als Parameter beziehungsweise als Konstante übergeben wird. Bisher lag die Entscheidung für die Parametrisierung allein beim objektrelationalen Mapper. Nun ist dies beeinflussbar, um die Nutzung des Query-Cache des Datenbankmanagementsystems zu optimieren. ►

Listing 4: Einsatz der neuen Überladungen der Methode `Parse()` in der Klasse `HierarchyId`

```
for (int j = 1; j < 5; j++)
{
    // alt
    var level2 = $"{startLevel}{i}/{j}/";
    var hid2Alt = HierarchyId.Parse(level2);
    // neu
    var hid2Neu = HierarchyId.Parse(
        abteilungsleiter.Level, j);

    var n = new Employee(hid2Neu, f.Name.FullName(),
        f.Random.Number(1970, 2023));
    n.Company = company;
    ctx.Add(n);
    Console.WriteLine("  " + n.ToString());
    var c3 = ctx.SaveChanges();
}
```

EF.Constant() ist auch als ein Nachtrag zu .NET 8.0 verfügbar, der in Version 8.0.2 erschienen ist. Dies ist insofern ungewöhnlich, da Microsoft behauptet, bei dem modernen .NET dem Semantic Versioning zu folgen, das bei neuen Funktionen eine Änderung der Versionsnummer an der zweiten Stelle vorsieht.

Aus [Listing 6](#) entsteht folgende SQL-Abfrage mit Parametrisierung mit Konstante bei der Spalte *Departure* und Parameter bei der Spalte *Destination*:

```
SELECT[f].[FlightNo], [f].[Airline], [f].[Departure],
[f].[Destination], [f].[FlightDate], [f].[FreeSeats],
[f].[Memo], [f].[NonSmokingFlight],
[f].[Pilot_PersonID], [f].[Seats], [f].[Timestamp]
FROM[Operation].[Flight]
AS[f]
WHERE[f].[Departure] = N'Frankfurt' AND[f].[Destination]
= @__destination_0
```

Aus [Listing 7](#) entsteht folgende SQL-Abfrage mit Konstanten bei beiden Spalten:

```
SELECT[f].[FlightNo], [f].[Airline], [f].[Departure],
[f].[Destination], [f].[FlightDate], [f].[FreeSeats],
[f].[Memo], [f].[NonSmokingFlight],
[f].[Pilot_PersonID], [f].[Seats], [f].[Timestamp]
FROM[Operation].[Flight]
AS[f]
WHERE[f].[Departure] = N'Frankfurt' AND[f].[Destination]
= N'Rome'
```

Aus [Listing 8](#) entsteht folgende SQL-Abfrage mit Parametern bei beiden Spalten:

```
SELECT[f].[FlightNo], [f].[Airline], [f].[Departure],
[f].[Destination], [f].[FlightDate], [f].[FreeSeats],
[f].[Memo], [f].[NonSmokingFlight],
[f].[Pilot_PersonID], [f].[Seats], [f].[Timestamp]
FROM[Operation].[Flight]
AS[f]
WHERE[f].[Departure] = @__p_0 AND[f].[Destination] =
@__destination_1
```

Weitere Verbesserungen bei der Übersetzung von LINQ in SQL

Verbesserungen in der Übersetzung von LINQ in SQL in Entity Framework Core 8.0 sind:

- Falls das Ziel-Datenbanksystem ein Microsoft SQL Server der neuesten Version 2022 ist, verwendet Entity Framework Core 9.0 nun die neu eingeführten T-SQL-Funktionen *LEAST()* und *GREATEST()* bei der Übersetzung von *Math.Min()* und *Math.Max()*.
- Entity Framework Core kann nun die .NET-Methoden *TimeOnly.FromDateTime()* und *TimeOnly.FromTimeSpan()* in LINQ-Abfragen in SQL übersetzen.
- Bei der Übersetzung von Abfragen, die sich auf JSON-Spalten beziehen, werden nun bei *OPENJSON...WITH* nur

● Listing 5: Einsatz von ToHashSetAsync()

```
public async Task ToHashSetAsync()
{
    var ctx =
        new DA.WWWings.WwwingsV1EnContext();
    var q = ctx.Flights.Take(10);

    CUI.Cyan("SQL: " + q.ToQueryString());
    var hashSet = await q.ToHashSetAsync();
    CUI.Cyan("Datensätze: " + hashSet.Count);

    //var firstFlight = hashSet[0]; --> kein Index

    foreach (var f in hashSet)
    {
        CUI.LI($"{f.FlightNo}: {f.Departure}->
        {f.Destination} {f.FlightDate}");
    }
}
```

● Listing 7: Einsatz von EF.Constant()

```
var list = ctx.Flights
    .Where(e => e.Departure == "Frankfurt" &&
        e.Destination ==
            EF.Constant(destination))
    // seit 8.0.2
    .ToList();
```

● Listing 8: Einsatz von EF.Parameter()

```
var list = ctx.Flights
    .Where(e => e.Departure ==
        EF.Parameter("Frankfurt") &&
        e.Destination == destination)
    // seit 9.0
    .ToList();
```


noch diejenigen Teile der JSON-Zeichenkette ausgewertet, die für die Bedingung oder das Resultset notwendig sind.

- Vergleiche mit *Count != 0* und *Count > 0* übersetzt Entity Framework Core nun in SQL mithilfe von *EXISTS* statt *COUNT*. Aus einer LINQ-Abfrage wie zum Beispiel

```
var blogsWithPost = await context.Blogs.Where(
    b => b.Posts.Count > 0).ToListAsync();
```

wird folgende SQL-Abfrage:

```
SELECT "b"."Id", "b"."Name", "b"."SiteUri"
FROM "Blogs" AS "b"
WHERE EXISTS (
    SELECT 1
    FROM "Posts" AS "p"
    WHERE "b"."Id" = "p"."BlogId")
```

● Listing 9: Verschachtelte LINQ-Abfragen

```
var dotnetPosts = context
    .Posts
    .Where(p => p.Title.Contains(".NET"));

var results = dotnetPosts
    .Where(p => p.Id > 2)
    .Select(p => new { Post = p, TotalCount =
        dotnetPosts.Count() })
    .Skip(2).Take(10)
    .ToArray();
```

Bei der Übersetzung von LINQ zu SQL fasst Entity Framework Core 9.0 nun die SQL-Befehle von verschachtelten LINQ-Abfragen zusammen (siehe [Listing 9](#) bis [11](#)).

Verbesserungen für den Azure-Cosmos-DB-Datenbanktreiber

Weitere Verbesserungen finden sich im Datenbanktreiber von Entity Framework Core für Azure Cosmos DB. Der Cosmos-DB-Provider hat bereits seit Entity Framework Core 6.0 primitive Sammlungen in begrenzter Form unterstützt. In Version 9.0 wird diese Unterstützung verbessert, indem die Metadaten und API-Oberflächen für primitive Sammlungen in Dokumentdatenbanken mit denen in relationalen Datenbanken konsolidiert werden. Dadurch können primitive Sammlungen explizit mit dem Modellierungs-API abgebildet werden, das die Konfiguration von Facetten des Elementtyps ermöglicht. Zum Beispiel kann eine Liste von erforderlichen (Nicht-null-)Zeichenfolgen nun wie folgt abgebildet werden:

```
modelBuilder.Entity<Book>()
    .PrimitiveCollection(e => e.Quotes)
    .ElementType(b => b.IsRequired());
```

Zudem wirft der Entity-Framework-Core-Datenbanktreiber für Azure Cosmos DB ab Entity Framework Core 9.0 nun einen Laufzeitfehler aus, wenn Entwicklerinnen und Entwickler synchrone Zugriffe (zum Beispiel mit *ToList()*) versuchen. Der Treiber unterstützt intern nämlich gar keine synchronen Zugriffe. Bisher waren synchrone Zugriffe dennoch möglich, wurden intern asynchron ausgeführt und dann bis zum Ausführungsende blockiert. Das konnte zu Deadlocks führen.

In Version 9.0 gibt es in diesem Fall nun im Standard einen Laufzeitfehler. ▶

● Listing 10: Übersetzung der obigen LINQ-Abfragen in zwei SQL-Abfragen in Entity Framework Core 8.0

```
SELECT COUNT(*)
FROM [Posts] AS [p]
WHERE [p].[Title] LIKE N'%.NET%'

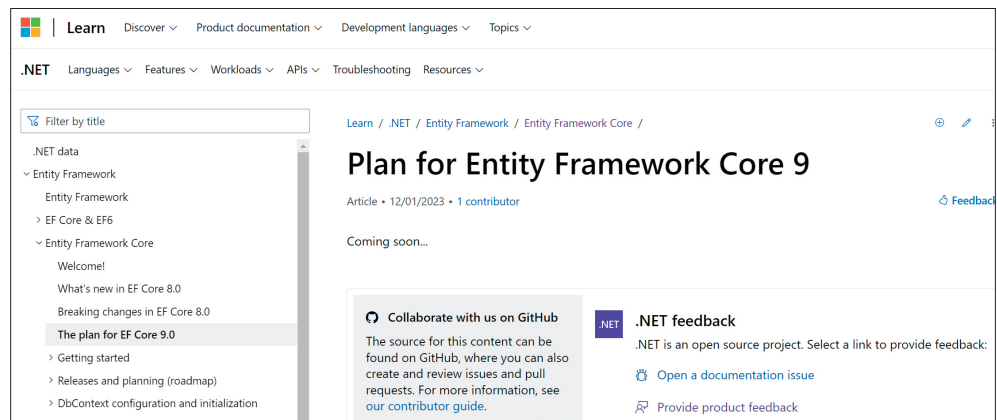
SELECT [p].[Id], [p].[Archived], [p].[AuthorId],
       [p].[BlogId], [p].[Content], [p].[Discriminator],
       [p].[PublishedOn], [p].[Title], [p].[PromoText],
       [p].[Metadata], (
    SELECT COUNT(*)
    FROM [Posts] AS [p0]
    WHERE [p0].[Title] LIKE N'%.NET%')
FROM [Posts] AS [p]
WHERE [p].[Title] LIKE N'%.NET%' AND [p].[Id] > 2
ORDER BY (SELECT 1)
OFFSET @_p_1 ROWS FETCH NEXT @_p_2 ROWS ONLY
```

● Listing 11: Übersetzung der obigen LINQ-Abfragen in eine einzige SQL-Abfrage in Entity Framework Core 9.0

```
SELECT [p].[Id], [p].[Archived], [p].[AuthorId],
       [p].[BlogId], [p].[Content], [p].[Discriminator],
       [p].[PublishedOn], [p].[Title], [p].[PromoText],
       [p].[Metadata], (
    SELECT COUNT(*)
    FROM [Posts] AS [p0]
    WHERE [p0].[Title] LIKE N'%.NET%')
FROM [Posts] AS [p]
WHERE [p].[Title] LIKE N'%.NET%' AND [p].[Id] > 2
ORDER BY (SELECT 1)
OFFSET @_p_0 ROWS FETCH NEXT @_p_1 ROWS ONLY
```

Derzeit kann man das alte Verhalten noch wiederherstellen mit:

```
protected override void
    OnConfiguring(DbContext
        OptionsBuilder
            optionsBuilder)
=> optionsBuilder
    .ConfigureWarnings(b =>
        b.Ignore(CosmosEventId
            .SyncNotSupported));
```



Microsoft hat in der Dokumentation unter [4] angekündigt, dass man binnen zwei Jahren (in Version 11.0 von Entity Framework Core) synchrone Aufrufe im Treiber für Azure Cosmos DB komplett verbieten will. Spätestens dann müssen Entwicklerinnen und Entwickler alle synchronen Materialisierungsoperationen wie *ToList()* in die asynchronen Pendanten *ToListAsync()* umgewandelt haben.

Außerdem unterstützt der Cosmos-DB-Provider für Entity Framework Core ab Version 9.0 auch die rollenbasierte Zugriffskontrolle (RBAC) [5].

Entity Framework Core und Native AOT

Im Vortrag zu Entity Framework Core 9.0 auf der Microsoft-Build-Konferenz [6] zeigte Shay Rojansky Fortschritte bei der Ahead-of-Time-Kompilierung für Entity Framework Core auf Basis von statischer Codeanalyse und C#-Interceptoren, mit denen LINQ-Befehle schon beim Veröffentlichen der Anwendung in SQL übersetzt werden. Allerdings wurde auch klar gestellt, dass es in Version 9.0 immer noch keine stabile Implementierung geben wird und noch nicht alle LINQ-Abfragen damit möglich sein werden. Insbesondere werden dynamisch zusammengebaute LINQ-Abfragen nicht möglich sein, weil die statische Codeanalyse solche Abfragen nicht erfassen kann.

In der Preview 6 von Entity Framework Core gibt es leider noch nichts davon zu sehen. Microsoft schreibt im „What's New“-Dokument zum Thema „AOT and pre-compiled queries“: „At this point in the EF9 release, there is not much available that can be used by you, the app developer.“ [7].

Weitere Neuerungen in Entity Framework Core 9.0

Weitere Verbesserungen in Entity Framework Core 9.0 sind:

- Die Umwandlung einer normalen Tabelle in eine temporale Tabelle ist nun mit einer stark verkürzten Befehlsfolge in den Schemamigrationen möglich, siehe dazu [7].
- Die Entity-Framework-Core-Kommandozeilenwerkzeuge sollen nun weniger häufig eine Neukompilierung des Projekts erfordern.
- Die Erstellung von Konventionen für die Modellerstellung hat Microsoft verschlankt, hierfür gibt es ein umfangreiches Beispiel in der Dokumentation unter [8].

Keine Roadmap für Entity Framework Core 9.0 (Bild 3)

Darüber hinausgehende Ideen des Entity-Framework-Entwicklungsteams muss man sich mühsam auf GitHub [9] zusammensuchen, denn die offizielle Roadmap unter [10] enthält auch zum Redaktionsschluss für diesen Beitrag Stand Ende Juni 2024 immer noch nur „Coming soon“ (Bild 3). ■

[1] Holger Schwichtenberg, *Komplexität im Griff*, dotnetpro 10/2024, Seite 94 ff., www.dotnetpro.de/A2410DataAccess

[2] Microsoft Learn, *CREATE SEQUENCE (Transact-SQL)*, www.dotnetpro.de/SL2411DataAccess1

[3] NuGet-Paket *Microsoft.EntityFrameworkCore.Tasks*, www.dotnetpro.de/SL2411DataAccess2

[4] Microsoft Learn, *Azure Cosmos DB for NoSQL*, www.dotnetpro.de/SL2411DataAccess3

[5] Microsoft Learn, *Azure role-based access control in Azure Cosmos DB*, www.dotnetpro.de/SL2411DataAccess4

[6] Microsoft-Build-Vortrag, *EF Core 9: Evolving Data Access in .NET*, www.dotnetpro.de/SL2411DataAccess5

[7] Microsoft Learn, *What's New in EF Core 9*, www.dotnetpro.de/SL2411DataAccess6

[8] Microsoft Learn, *Make existing model building conventions more extensible*, www.dotnetpro.de/SL2411DataAccess7

[9] Offene Issues zu EF Core 9 auf GitHub, www.dotnetpro.de/SL2411DataAccess8

[10] Microsoft Learn, *Plan for Entity Framework Core 9*, www.dotnetpro.de/SL2411DataAccess9

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

A2411DataAccess

DIE VERBINDUNG ZWISCHEN CLOUD- UND ON-PREMISES-ANWENDUNGEN

Hybride Integration mit Azure Service Bus

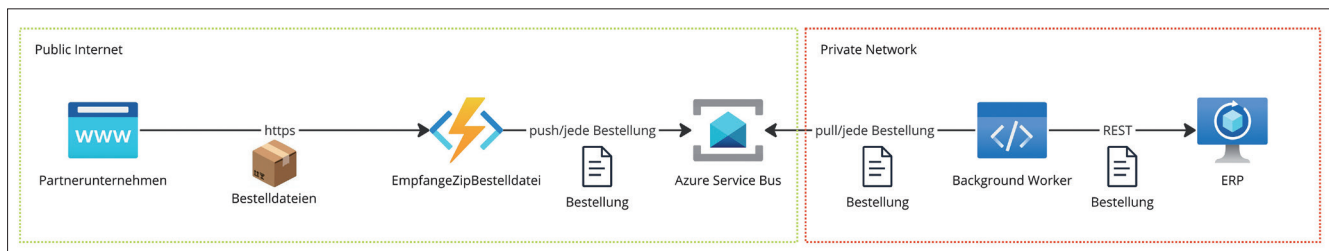
Mit Messaging Netzwerkgrenzen überbrücken, um Anwendungen in der Cloud mit On-Premises-Anwendungen zu verbinden.

Unternehmen stehen in der heutigen IT-Landschaft vor der Herausforderung, flexibel und effizient auf wechselnde Geschäftsanforderungen reagieren zu müssen. Dabei spielt die Bereitstellung von Softwarelösungen eine zentrale Rolle. Die Entscheidung, ob Anwendungen und Daten in einer On-Premises-Umgebung oder in der Cloud betrieben werden, ist längst nicht mehr eine Frage von Entweder-oder. Stattdessen setzen immer mehr Unternehmen auf hybride Integrations-szenarien, um das Beste aus beiden Welten zu kombinieren.

On-Premises-Lösungen bieten nach wie vor unübertroffene Kontrolle und Sicherheit. Besonders in Branchen mit strengen Compliance-Vorgaben bleibt die lokale Datenhaltung

gleichzeitig die Vorteile der Cloud erschließen. So können Unternehmen schnell auf Marktveränderungen reagieren, Innovationszyklen verkürzen und die Gesamtbetriebskosten senken. In einer Welt, in der Agilität und Effizienz entscheidend sind, stellen hybride Integrationsszenarien die Zukunft der IT-Anwendungslandschaft dar.

Wie gelingt es, Cloud und On-Premises miteinander zu verbinden, ohne dafür ein ganzes IT-Security-Team inklusive IT-Netzwerkspezialisten einzustellen beziehungsweise hohen Konfigurations- und Wartungsaufwand zu erzeugen? Die Antworten liefert Ihnen dieser Artikel: Sie erfahren, wie Sie mithilfe von Azure Service Bus dieses Ziel erreichen.



Architekturskizze zur Beispielanwendung (Bild 1)

ein entscheidender Faktor. Doch dieser Sicherheitsgewinn geht oft einher mit hohen Kosten und begrenzter Skalierbarkeit. Hier setzt die Cloud an, die mit ihrer Flexibilität, Skalierbarkeit und den geringeren Betriebskosten punktet. Cloud-Computing ermöglicht es, Ressourcen je nach Bedarf dynamisch zu nutzen und neue Technologien wie künstliche Intelligenz oder Big-Data-Analysen effizient in Geschäftsprozesse zu integrieren.

Doch weder On-Premises noch Cloud-Computing sind allein in der Lage, sämtliche Anforderungen moderner Anwendungen zu erfüllen. Hybride Szenarien bieten hier die optimale Lösung. Sie ermöglichen eine nahtlose Integration von lokal gehosteten Anwendungen mit Cloud-Diensten und bieten so eine flexible, skalierbare und sichere Umgebung, die den individuellen Bedürfnissen eines Unternehmens gerecht wird. Durch hybride Ansätze lassen sich bestehende Investitionen in On-Premises-Infrastrukturen optimal nutzen und

Azure Service Bus: Netzwerkgrenzen überwinden ohne Kompromisse bei der IT-Security

In einer Zeit, in der verteilte Anwendungen über verschiedene Netzwerke und Umgebungen hinweg kommunizieren müssen, stehen Unternehmen vor der Herausforderung, diese Verbindungen sicher und zuverlässig zu gestalten. Dabei gilt es nicht nur, technische Hürden wie Netzwerkgrenzen zu überwinden, sondern auch strenge IT-Sicherheitsrichtlinien einzuhalten. Der Azure Service Bus bietet hier eine effektive Lösung, die es ermöglicht, Kommunikationsbarrieren zu überwinden, ohne die Sicherheit zu gefährden.

Der Azure Service Bus fungiert als vermittelnde Instanz, die Nachrichten zwischen Anwendungen transportiert, selbst wenn diese sich in unterschiedlichen Netzwerkzonen oder hinter Firewalls befinden. Durch die Verwendung von Warteschlangen können Nachrichten sicher zwischengespeichert und erst zu einem späteren Zeitpunkt abgerufen werden, ►

sobald die Ziellanwendung verfügbar ist. Diese asynchrone Kommunikation ermöglicht es, Netzwerk-grenzen zu durchbrechen, ohne direkte Verbindungen herstellen zu müssen, die potenziell Sicherheitsrisiken mit sich bringen.

Ein wesentliches Sicherheitsmerkmal des Azure Service Bus ist die strenge Zugriffskontrolle. Über Role-Based Access Control (RBAC) und Shared Access Signatures (SAS) lässt sich präzise steuern, welche Anwendungen und Benutzer auf welche Ressourcen zugreifen dürfen. Dadurch wird sichergestellt, dass nur autorisierte Anwendungen oder Services Nachrichten senden oder empfangen können. Diese granular einstellbaren Berechtigungen minimieren das Risiko von unbefugtem Zugriff und schützen sensible Daten effektiv.

Zudem unterstützt der Azure Service Bus eine Ende-zu-Ende-Verschlüsselung. Nachrichten werden sowohl während der Übertragung als auch im Ruhezustand verschlüsselt, was den Schutz vor Abhörversuchen und Datenlecks weiter verstärkt. In Kombination mit der Möglichkeit, Nachrichten über dedizierte Kommunikationspfade zu senden, wird ein hohes Maß an Sicherheit gewährleistet, selbst in Szenarien, die komplexe Netzwerkarchitekturen umfassen.

Für hybride Szenarien bietet der Azure Service Bus eine sichere Brücke zwischen On-Premises-Systemen und Cloud-Diensten. Unternehmen können ihre bestehenden Sicherheitsrichtlinien beibehalten und gleichzeitig die Vorteile der Cloud nutzen. Durch die Nutzung von Service Bus können Anwendungen in der Cloud nahtlos mit On-Premises-Lösungen kommunizieren.

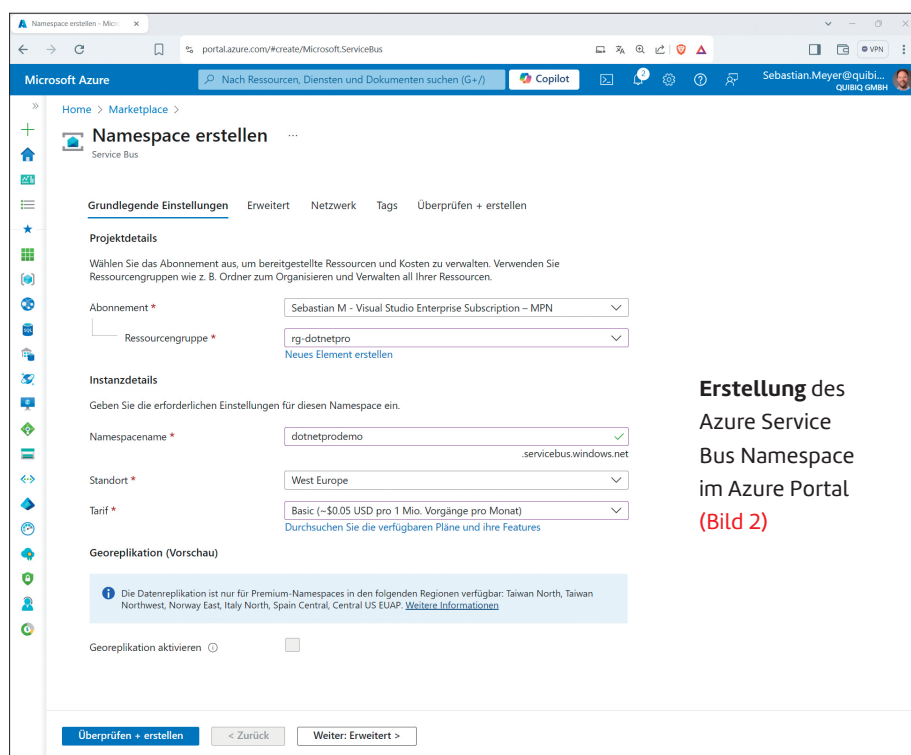
Der Azure Service Bus ermöglicht es, Netzwerk-grenzen sicher zu überwinden und gleichzeitig die IT-Security auf höchstem Niveau zu halten. Unternehmen erhalten damit eine zuverlässige, skalierbare und sichere Plattform, um komplexe Kommunikationsanforderungen zu erfüllen und ihre IT-Infrastruktur zukunftssicher zu gestalten.

Service Bus in der Übersicht

Der Azure Service Bus (komplette Dokumentation unter [1]) zeichnet sich durch eine Reihe von Eigenschaften aus, die ihn zu einem leistungsfähigen und vielseitigen Message-Broker für moderne Anwendungen machen. Hier sind die wichtigsten Merkmale:

1. Warteschlangen (Queues):

- **First In, First Out (FIFO):** Nachrichten werden in der Reihenfolge zugestellt, in der sie eingegangen sind.
- **Lastausgleich:** Mehrere Empfänger können Nachrichten parallel verarbeiten, was die Skalierbarkeit erhöht.



Erstellung des Azure Service Bus Namespace im Azure Portal (Bild 2)

2. Themen und Abonnements (Topics and Subscriptions):

- **Publish/Subscribe-Modell:** Ermöglicht das Verteilen einer Nachricht an mehrere Abonnenten.
- **Filtern von Nachrichten:** Nachrichten können basierend auf Attributen gefiltert und ausschließlich an relevante Abonnenten gesendet werden.

3. Sicherheitsfeatures:

- **Rollenbasierte Zugriffskontrolle (RBAC):** Präzise Steuerung, wer Nachrichten senden und empfangen kann.
- **Ende-zu-Ende-Verschlüsselung:** Schutz sensibler Daten während der Übertragung.

4. Nachrichtenverarbeitung:

- **Dauerhafte Speicherung:** Nachrichten bleiben gespeichert, bis sie erfolgreich verarbeitet wurden.
- **Dead-Letter Queues:** Fehlgeschlagene Nachrichten können zur späteren Analyse in speziellen Warteschlangen abgelegt werden.

5. Integration:

- **Hybrid-Umgebungen:** Verbindung von On-Premises und Cloud-Diensten.
- **Nahtlose Integration mit anderen Azure-Diensten:** Einfache Anbindung an Azure Functions, Logic Apps und mehr.

6. Zuverlässigkeit und Skalierbarkeit:

- **Hohe Verfügbarkeit:** Automatische Replikation der Nachrichten in der Azure-Region.
- **Automatische Skalierung:** Dynamische Anpassung an das Nachrichtenvolumen ohne manuelle Eingriffe.

7. Erweiterte Features:

- **Scheduled Delivery:** Nachrichten können für die zukünftige Zustellung geplant werden.
- **Session Management:** Nachrichten mit logischen Abhängigkeiten können als Gruppe verarbeitet werden.

Azure Service Bus bietet eine robuste, sichere und flexible Plattform für die Kommunikation zwischen verteilten Systemen. Ob für einfache Warteschlangen, komplexe Publish/Subscribe-Szenarien oder hybride Architekturen – Service Bus ist die perfekte Lösung für zuverlässige und skalierbare Anwendungen.

Warum eine Messaging-Lösung sinnvoll ist

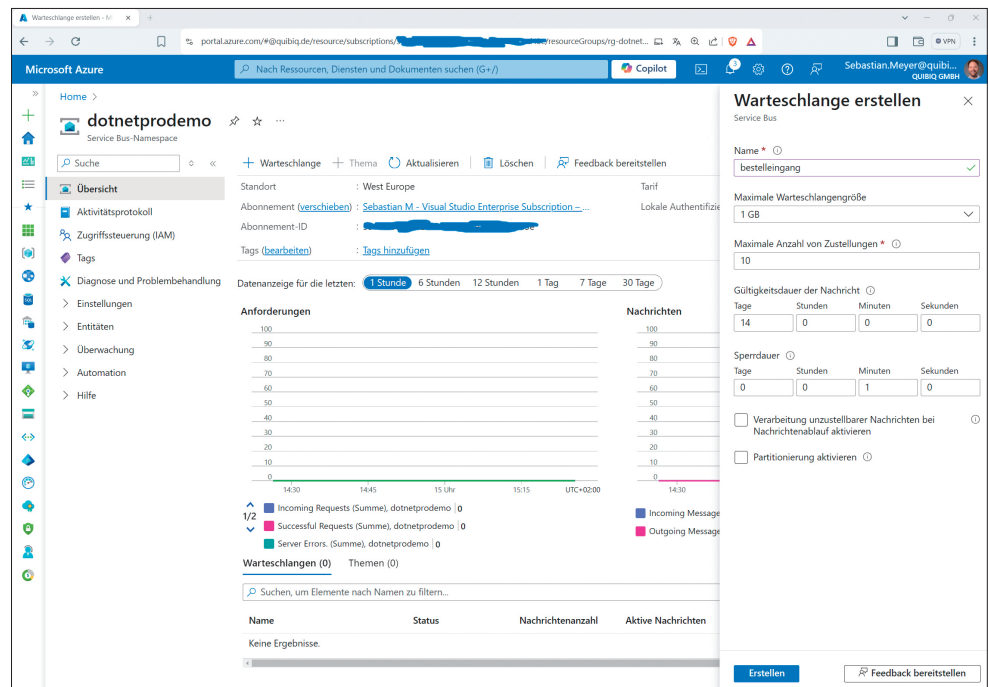
Für Szenarien, in denen Cloud-Komponenten mit On-Premises-Komponenten kommunizieren, bietet sich eine Messaging-Lösung an. Dies gilt auch für ein umgekehrtes Szenario, Cloud-zu-Cloud oder Multicloud-Szenarien. Der Grund ist die asynchrone Kommunikation der Anwendungen mit dem Message Broker. Der Message Broker kümmert sich um die Komplexität von Verfügbarkeit, Speicherung der Nachrichten, Security und die garantierte Zustellung der Nachrichten an den Empfänger oder die Verschiebung in eine DLQ (Dead-Letter Queue), wenn der Empfänger die Nachricht nicht innerhalb der definierten Zeit entgegennimmt. In einer direkten beziehungsweise synchronen Kommunikation müssen diese Dinge immer wieder selbst implementiert, getestet und gewartet werden.

Wartung ist ein gutes Stichwort: Sollten die Anwendungen in einer geplanten Wartung nicht verfügbar sein, bleibt die Nachricht wie beschrieben auf dem Message Broker gespeichert. Sobald die Anwendung den Wartungszustand verlässt und wieder verfügbar ist, werden die Nachrichten zugestellt. Der Gesamtprozess ist durch Wartung einzelner Anwendungen also nicht unterbrochen – ein wichtiges Merkmal von Messaging-Lösungen und ideal für unseren Anwendungsfall.

Der Demo-Anwendungsfall

In diesem Artikel möchte ich Ihnen das Konzept der Messaging-Lösung vorstellen. Daher habe ich ein einfaches, nachvollziehbares Beispiel gewählt, um das zu realisieren. Wir behandeln nur die Kernkomponenten und werden keine ausformulierte Geschäftslogik implementieren. Am Ende erhalten Sie aber ein funktionsfähiges Beispiel für die Ende-zu-Ende-Kommunikation von Cloud zu On-Premises.

Für unser Beispiel werden wir eine Azure Functions App (.NET 8) in Azure bereitstellen, die über einen HTTP-Trigger eine ZIP-Datei entgegennimmt. Diese ZIP-Datei enthält einzelne Bestelldateien, die wir von einem Partnerunternehmen über diese Schnittstelle erhalten. Die Functions App entpackt die Dateien und schickt jede einzelne Bestellung als Nachricht auf den Service Bus in eine Warteschlange. On-Premi-



Einrichtung einer Warteschlange im Service Bus Namespace (Bild 3)

ses ist ein ERP-System vorhanden, das eine standardisierte REST-Schnittstelle zur Verfügung stellt, damit Bestellungen auf diesem Wege übertragen werden können. Um die Nachrichten aus der Warteschlange entgegenzunehmen, wird ein .NET-8-Background-Worker implementiert, der dann auch die Kommunikation in Richtung ERP übernimmt (siehe Bild 1). Steigen Sie ein, der Service Bus auf der Linie 18 fährt heute nicht nach Istanbul, sondern nach On-Premises.

Kurze Übersicht zu Azure Functions

Bevor wir starten, zunächst noch eine kurze Übersicht zu Azure Functions (siehe [2]). Azure Functions ist ein serverloser Compute-Dienst von Microsoft, der Entwicklern die Möglichkeit bietet, Anwendungen schnell und effizient zu erstellen und auszuführen.

Ohne sich um Infrastruktur kümmern zu müssen, können Entwickler sich vollständig auf den Code konzentrieren. Hier sind die wichtigsten Eigenschaften von Azure Functions:

1. Serverless Computing:

- **Automatische Skalierung:** Azure Functions skaliert automatisch basierend auf der Last. Es werden nur Ressourcen genutzt, wenn der Code ausgeführt wird.
- **Kostenoptimierung:** Bezahlen Sie nur für die Rechenleistung, die Ihre Anwendung tatsächlich nutzt. Keine laufenden Kosten für ungenutzte Ressourcen.

2. Unterstützung für mehrere Sprachen:

- **Vielseitigkeit:** Entwickeln Sie Funktionen in einer Vielzahl von Sprachen, darunter C#, JavaScript, Python, Java und PowerShell. Dies ermöglicht Flexibilität bei der Wahl der Programmiersprache.

3. Integration mit Azure Services:

- **Einfache Anbindung:** Azure Functions integriert sich nahtlos mit anderen Azure-Diensten wie Azure Service ►

Bus, Blob Storage, Event Grid und Cosmos DB. Diese Integration erleichtert die Erstellung von Workflows und datengetriebenen Anwendungen.

- **Trigger und Bindungen:** Azure Functions verwendet Trigger, um Funktionen bei bestimmten Ereignissen automatisch auszuführen, und Bindungen, um nahtlos auf Datenquellen zuzugreifen.
4. Entwicklungs- und Bereitstellungsoptionen:
- **Entwicklungsumgebung:** Unterstützt sowohl lokale Entwicklung als auch die Entwicklung direkt im Azure-Portal. Funktionen lassen sich einfach über GitHub, Azure DevOps oder andere CI/CD-Pipelines bereitstellen.
 - **App-Service-Plattform:** Azure Functions nutzt dieselbe bewährte Infrastruktur wie Azure App Service und bietet hohe Verfügbarkeit und Sicherheit.
5. Sicherheit und Compliance:
- **Rollenbasierte Zugriffskontrolle (RBAC):** Fein abgestimmte Sicherheitsmodelle sorgen dafür, dass nur autorisierte Benutzer auf Funktionen zugreifen können.
 - **VNet-Integration:** Ermöglicht die sichere Verbindung von Funktionen mit Ressourcen in einem virtuellen Netzwerk.
6. Zuverlässigkeit und Verfügbarkeit:
- **Hohe Verfügbarkeit:** Azure Functions bietet integrierte Redundanz und Ausfallsicherheit, um eine hohe Verfügbarkeit zu gewährleisten.
 - **Globale Verteilung:** Funktionen können in verschiedenen Azure-Regionen bereitgestellt werden, um eine globale Reichweite und geringe Latenzzeiten zu gewährleisten.
7. Flexible Hosting-Optionen:
- **Consumption Plan:** Automatische Skalierung basierend auf Ereignissen, ideal für variable Lasten.
 - **Premium Plan:** Skalierung ohne Kaltstarts und zusätzliche Funktionen wie VNet-Integration.
 - **Dedicated (App Service) Plan:** Läuft auf dedizierten VMs und bietet volle Kontrolle über die Skalierung.

8. Monitoring und Debugging:

- **Application Insights:** Integrierte Überwachung und Protokollierung ermöglichen es, die Performance von Funktionen zu analysieren und Fehler schnell zu identifizieren.
- **End-to-End-Debugging:** Unterstützung für Remote-Debugging und lokale Tests.

Azure Functions ist die perfekte Lösung, wenn es darum geht, skalierbare, ereignisgesteuerte Anwendungen zu entwickeln, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen. Mit seiner Flexibilität, Leistungsfähigkeit und tiefen Integration in das Azure-Ökosystem bietet es Entwicklern alles, was sie brauchen, um schnell und effizient auf Geschäftsanforderungen zu reagieren.

Nachdem nun alle notwendigen Dienste erklärt sind, beginnen wir mit der Bereitstellung in Azure unter [3].

Einen Azure Service Bus Namespace erstellen

Lassen Sie uns starten, indem wir einen Service Bus Namespace anlegen. Der Namespace ist der Ort, an dem der Message Broker lebt.

Die Erstellung des Service Bus Namespace kann über den Azure Marketplace (siehe Bild 2) gestartet werden. Suchen Sie im Marketplace nach *Service Bus* und bestätigen Sie über *Erstellen*. Unter *Grundlegende Einstellungen* wählen Sie Ihr Abonnement aus und wählen eine vorhandene Ressourcen-Gruppe oder erstellen eine neue. Damit sind die Projektdetails gesetzt. Im Anschluss geht es an die Instanzdetails, wo Sie einen Namen für Ihren Namespace festlegen. Hier müssen Sie einen FQDN (Full Qualified Domain Name) vergeben, der weltweit eindeutig ist. Danach wählen Sie Ihren Standort aus, in dem der Namespace bereitgestellt werden soll, und zum Abschluss wählen Sie Ihren Tarif. Für die Demo ist die der Basic-Tarif völlig ausreichend. Alle weiteren Einstellungen auf den folgenden Reitern können Sie im Standard belassen. Die Provisionierung lässt sich einfach per Klick auf *Überprüfen + erstellen* direkt anstoßen.

Warteschlange erstellen

Wie bereits erwähnt, ist der Service Bus Namespace der Speicherort für die Warteschlangen. Lassen Sie uns daher im nächsten Schritt eine Warteschlange erstellen. Gehen Sie wie in Bild 3 gezeigt vor.

Auf der Übersichtsseite Ihres erstellten Namespaces finden Sie die Schaltfläche *+ Warteschlange*. Klicken Sie hier, um eine neue Warteschlange hinzuzufügen. Im darauffolgenden Dialog vergeben Sie einen Namen für die Warteschlange und passen die Parameter auf Ihre Bedürfnisse an. Sie können die Einstellungen auch im Standard belassen und auf *Erstellen* klicken.

An dieser Stelle ist die Einrichtung des Service Bus (Namespace und Warteschlange) abgeschlossen. Wir werden später

Funktionen-App erstellen (Verbrauch)

Grundlagen | Storage | Netzwerk | Überwachung | Bereitstellung | Tags | Überprüfen + erstellen

Erstellen Sie eine Funktionen-App, mit der Sie Funktionen als logische Einheit gruppieren können, um die Verwaltung, Bereitstellung und Freigabe von Ressourcen zu vereinfachen. Mit Funktionen können Sie Ihren Code in einer serverlosen Umgebung ausführen, ohne zuerst einen virtuellen Computer erstellen oder eine Webanwendung veröffentlichen zu müssen.

Projektdetails

Wählen Sie ein Abonnement aus, um bereitgestellte Ressourcen und Kosten zu verwalten. Verwenden Sie Ressourcen-Gruppen wie z. B. Ordner zum Organisieren und Verwalten all Ihrer Ressourcen.

Abonnement *

Ressourcen-Gruppe * [Neues Element erstellen](#)

Instanzendetails

Name der Funktionen-App * azurewebsites.net

Runtime-Stack *

Version *

Region *

Betriebssystem * ☐ Linux ☒ Windows

[Überprüfen + erstellen](#) [< Zurück](#) [Weiter: Storage >](#)

Erstellen der Azure Functions App (Bild 4)

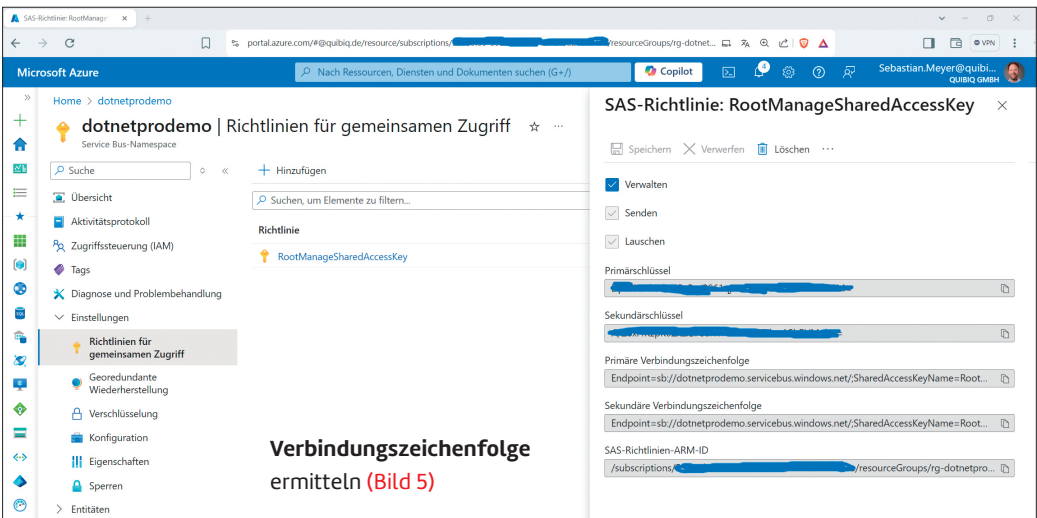
noch einmal zurückkehren, um die Verbindungszeichenfolge abzuholen, die wir für die Verbindung mit Azure Functions und dem Background-Worker benötigen.

Azure Functions App erstellen

Wir schließen die Bereitstellung in Azure mit dem Anlegen der Azure Functions App ab (siehe Bild 4). Diese benötigt eine Laufzeitumgebung für die Bereitstellung und Ausführung.

Der Einstiegspunkt dafür ist wieder der Marketplace, und dort suchen Sie nach *Funktions-App*. Wählen Sie *Verbrauch* als Bereitstellungsmethode aus. Der Funktionsumfang, der mit dieser Bereitstellungsmethode zur Verfügung steht, reicht für unsere Demo aus.

Unter den Grundlagen bei den Projektdetails definieren Sie *Abonnement* und *Ressourcengruppe* für die Bereitstellung. Bei den Instanzendetails wählen Sie wieder einen



Verbindungszeichenfolge ermitteln (Bild 5)

FQDN, der eindeutig sein muss. Als Runtime wählen Sie *.NET* in der Version *8 (LTS)*, *isolated worker model* (beachten Sie hierzu auch den Kasten *Azure Functions Worker Model*).

Zum Abschluss wählen Sie noch Ihre Region und als Betriebssystem den Eintrag *Windows*. Alle anderen Reiter und die damit verbundenen Einstellungen können Sie im Standard belassen. Schließen Sie die Bereitstellung mit *Überprüfen + erstellen* ab.

Die Functions App ist im Endeffekt ein Behälter, in dem die einzelnen Functions bereitgestellt und ausgeführt werden. Die Function, die die Geschäftslogik beinhaltet, wird in Visual Studio erstellt, implementiert und in der Functions App über die Option *Publish* veröffentlicht.

Sie haben damit die Infrastruktur, die in Azure benötigt wird, erfolgreich angelegt. Lassen Sie uns jetzt mit dem starten, was Ihnen am meisten Freude bereitet: die Implementierung!

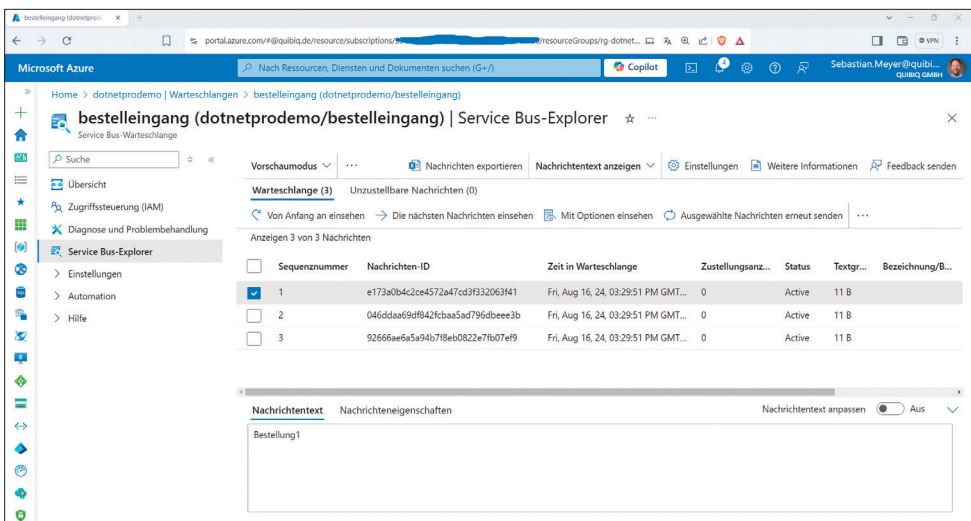
Erstellung der Visual Studio Solution

Alle Artefakte werden in einer Visual Studio Solution angelegt. Starten Sie mit der Erstellung des Azure-Functions-Projekts. Wählen Sie das entsprechende Projekttemplate aus und konfigurieren Sie als Worker-Runtime *.NET 8* im *Isolated-Worker Model*. Wenn Sie hier eine andere Runtime wählen, können Sie die Azure Functions in Azure später nicht als Deployment-Target auswählen. Im nächsten Schritt erstellen Sie eine neue Function und wählen *HttpTrigger* aus. Sie haben nun eine Function erstellt, die einen HTTP-Endpoint bereitstellt. Listing 1 zeigt die vollständige Implementierung für die Demo.

Microsoft stellt ein NuGet-Package zur Anbindung ►

● Azure Functions Worker Model

Azure Functions können als Isolated-Worker (wie in unserem Beispiel) oder im In-Process-Model ausgeführt werden. Das In-Process-Model wird ab dem 10. November 2026 nicht mehr unterstützt. Sie sollten sich daher bei Azure Functions für das Isolated-Model entscheiden.



Die Nachrichten im Service Bus Explorer (Bild 6)

von Service Bus bereit (Azure.Messaging.ServiceBus). Referenzieren Sie dieses in Ihr Projekt. Alle anderen Infrastrukturkomponenten und Abhängigkeiten werden von der Azure Functions Runtime isoliert.

Die Verbindung zum Service Bus bauen wir über die Verbindungszeichenfolge auf. Diese können Sie im Azure Portal im Service Bus Namespace auslesen (siehe [Bild 5](#)). Kopieren

Sie die primäre oder sekundäre Verbindungszeichenfolge und übergeben Sie diese dem *ServiceBusClient*-Konstruktor. Der *ServiceBusClient* stellt die Verbindung zum Service Bus Namespace her.

Im nächsten Schritt muss die Warteschlange initialisiert werden. Dazu wird dem *ServiceBusSender* der Name der Warteschlange als Parameter übergeben.

● Listing 1: Azure Function zum Empfangen der ZIP-Datei

```
using Microsoft.Azure.Functions.Worker;
using Microsoft.Extensions.Logging;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.IO.Compression;
using Azure.Messaging.ServiceBus;

namespace FunctionApp
{
    public class EmpfangZipBestelldatei
    {
        private readonly ILogger<EmpfangZipBestelldatei>
            _logger;
        private readonly ServiceBusClient
            _serviceBusClient;
        private readonly ServiceBusSender
            _serviceBusSender;

        public EmpfangZipBestelldatei(
            ILogger<EmpfangZipBestelldatei> logger)
        {
            logger = logger;
            serviceBusClient = new ServiceBusClient(
                "<CONNECTIONSTRING>");
            serviceBusSender = _serviceBusClient.CreateSender(
                "<QUEUENAME>");
        }

        [Function("EmpfangZipBestelldatei")]
        public async Task<ActionResult> Run([HttpTrigger(
            AuthorizationLevel.Function, "post")]
            HttpRequest req)
        {
            logger.LogInformation("Received a request to
                process and send notifications.");

            if (!req.HasFormContentType ||
                !req.Form.Files.Any())
            {
                return new BadRequestObjectResult(
                    "Please upload a ZIP file.");
            }

            var file = req.Form.Files.First();

            if (file.Length == 0 || !file.FileName.EndsWith(
                ".zip", StringComparison.OrdinalIgnoreCase))
            {
                return new BadRequestObjectResult(
                    "The uploaded file is not a valid ZIP file.");
            }

            try
            {
                using (var zipStream = file.OpenReadStream())
                {
                    using (var archive = new ZipArchive(
                        zipStream, ZipArchiveMode.Read))
                    {
                        foreach (var entry in archive.Entries)
                        {
                            using (var entryStream = entry.Open())
                            {
                                string fileContent = new StreamReader(
                                    entryStream).ReadToEnd();
                                logger.LogInformation($"{{fileContent}}");

                                await
                                    _serviceBusSender.SendMessageAsync(
                                        new ServiceBusMessage(fileContent));
                            }
                        }
                    }
                }

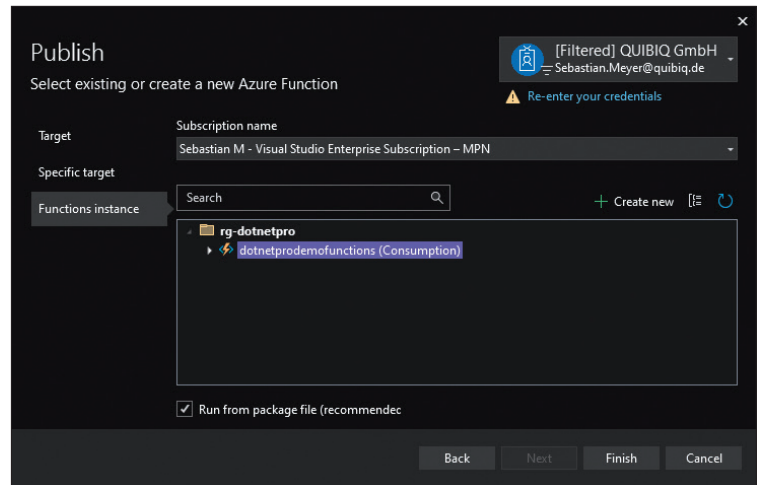
                return new OkObjectResult("ZIP file processed
                    and notifications sent successfully.");
            }
            catch (Exception ex)
            {
                logger.LogError(
                    $"An error occurred: {ex.Message}");
                return new StatusCodeResult(
                    StatusCodes.Status500InternalServerError);
            }
        }
    }
}
```

```
_serviceBusClient =
    new ServiceBusClient("<CONNECTIONSTRING>");
_serviceBusSender =
    _serviceBusClient.CreateSender("<QUEUENAME>");
```

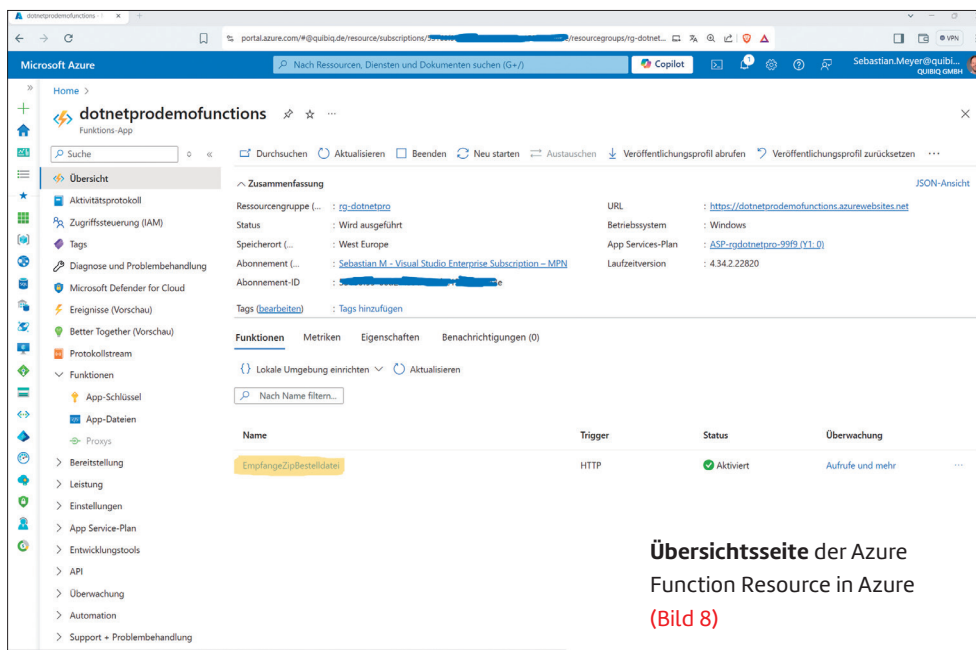
Die Function hat den Namen *EmpfangeZipBestell-datei* und wird über HTTP ausgeführt. Die Function nimmt ausschließlich *POST*-Anfragen an.

```
[Function("EmpfangeZipBestelldatei")]
public async Task<ActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Function,
        "post")] HttpRequest req){...}
```

Neben der Entpack-Logik erzeugt die Function für den Inhalt der jeweiligen Bestellung eine *Service-*



Veröffentlichen der Azure Function (Bild 7)



Übersichtsseite der Azure Function Resource in Azure (Bild 8)

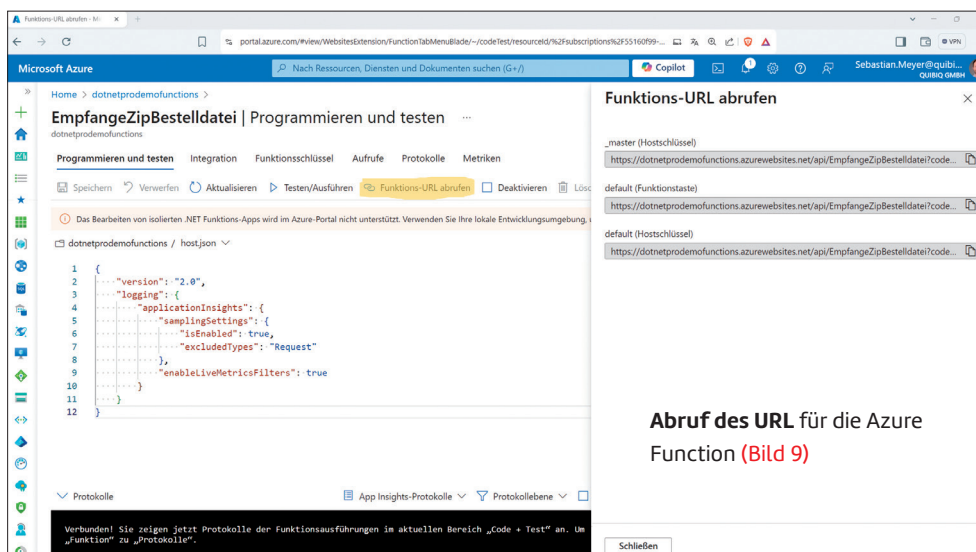
BusMessage und legt diese in die Warteschlange.

```
await _serviceBusSender
    .SendMessageAsync(
        new ServiceBusMessage(
            fileContent));
```

Bei Fehlern wird eine entsprechende Fehlermeldung erzeugt.

Sie können die Function lokal testen und debuggen. Ich empfehle Ihnen den Postman-Client, damit Sie komfortabel eine ZIP-Datei an den HTTP-Endpoint der Function senden können. Sie können aber auch gerne jedes andere Tool verwenden, um den Request abzuschicken. Wenn der Code fehlerfrei ausgeführt wird, können Sie sich die Nachrichten im Service Bus Explorer ansehen (siehe Bild 6). Wichtig zu verstehen ist hier, dass Sie die Nachrichten nur einsehen können, wenn Sie keinen Abnehmer der Daten haben. Zu diesem Zeitpunkt haben wir noch keinen. Den Background-Worker erstellen wir im nächsten Schritt.

Wenn Ihr Test erfolgreich war, können Sie mit dem Veröffentlichen Ihrer Version nach Azure beginnen. Dazu wählen Sie *Veröffentlichen* ►



Abwurf des URL für die Azure Function (Bild 9)

● Listing 2: Empfang der Nachrichten aus dem Service Bus

```
using Azure.Messaging.ServiceBus;

namespace ServiceBusWorker;

public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;
    private readonly ServiceBusClient _serviceBusClient;
    private readonly ServiceBusProcessor
        _serviceBusProcessor;

    public Worker(ILogger<Worker> logger)
    {
        logger = logger;
        serviceBusClient = new ServiceBusClient(
            "<CONNECTIONSTRING>");
        serviceBusProcessor = _serviceBusClient
            .CreateProcessor("<QUEUENAME>");

        serviceBusProcessor.ProcessMessageAsync +=
            _serviceBusProcessor.ProcessMessageAsync;
        serviceBusProcessor.ProcessErrorAsync +=
            _serviceBusProcessor.ProcessErrorAsync;
    }

    private async Task
        _serviceBusProcessor.ProcessMessageAsync(
            ProcessMessageEventArgs arg)
    {
        var bestellung = arg.Message.Body.ToString();

        // Hier findet nun die Verarbeitung der Bestellung
        // statt. Validierung etc., Transfer ins ERP
        // Wenn keine Fehler aufgetreten sind, kann die
        // Nachricht im Service Bus gelöscht werden.
        logger.LogInformation(
            $"On-Premises-Worker: Bestellung empfangen.
            Inhalt der Datei: {bestellung}");

        await arg.CompleteMessageAsync(arg.Message);
    }

    private Task _serviceBusProcessor.ProcessErrorAsync(
        ProcessErrorEventArgs arg)
    {
        logger.LogCritical("Fehler bei der Verarbeitung
            der Bestelldateien.");
        return Task.CompletedTask;
    }

    protected override async Task ExecuteAsync(
        CancellationToken stoppingToken)
    {
        await _serviceBusProcessor.StartProcessingAsync(
            stoppingToken);

        while (!stoppingToken.IsCancellationRequested)
        {
            if (logger.IsEnabled(LogLevel.Information))
            {
                logger.LogInformation("Worker running at:
                    {time}", DateTimeOffset.Now);
            }
            await Task.Delay(1000, stoppingToken);
        }
    }
}
```

im Projektkontextmenü aus und folgen dem Dialog wie in **Bild 7**. Wählen Sie Ihr Abonnement und die entsprechende Ressourcengruppe aus und klicken Sie auf die Azure Function, die Sie am Anfang erstellt haben. Schließen Sie den Dialog ab und warten Sie, bis die Bereitstellung erfolgreich ist. Ihre Function ist jetzt in Azure verfügbar.

Damit Sie den Aufruf gegen die Azure Function in der Cloud ausführen können, brauchen Sie den Endpunkt mit entsprechendem Schlüssel. Dieser wird benötigt, damit kein unautorisierter Aufruf der Function aus dem Internet erfolgen kann. Das ist wichtig, denn die Function steht jetzt im Public Internet zur Verfügung.

Für den Abruf des Endpunkts gehen Sie zur Übersichtsseite der Function App Resource in Azure (siehe **Bild 8**). Klicken Sie auf die entsprechende Funktion und wählen Sie *Funktions-URL abrufen* (siehe **Bild 9**). Kopieren Sie eine der Adressen und führen Sie dann die Anfrage mit dem Tool Ihrer Wahl

aus (siehe **Bild 10**). Wenn die Anfrage erfolgreich war, sehen Sie die Nachrichten im Service Bus Explorer.

Erstellung des .NET 8 Background Worker

Sie haben erfolgreich Nachrichten an die Service-Bus-Warteschlange gesendet. Schon jetzt haben Sie einen großen Vorteil der Messaging-Lösung kennengelernt, der am Anfang erwähnt wurde: Ihre Nachrichten werden gespeichert, ohne dass ein Empfänger für die Nachrichten da ist (quasi nicht zu Hause ist). In Fällen von wartungsbedingten oder unvorhergesehenen Systemausfällen verlieren Sie mit diesem Ansatz keine Daten. Machen wir uns also daran, einen Empfänger für die Nachrichten aus der Warteschlange zu implementieren. Als Template bietet sich der Background-Worker an, da er „unendlich“ läuft.

Legen Sie dazu in der Visual Studio Solution ein neues .NET-8-Worker-Projekt an. In **Listing 2** finden Sie die kom-

plette Implementierung des Workers. Wie bei der Function nutzen wir für die Verbindung zum Service Bus wieder das NuGet-Package `Azure.Messaging.ServiceBus`. Dem `ServiceBusClient` wird die Verbindungszeichenfolge als Parameter übergeben. Damit ist die Verbindung zum Service Bus Namespace aufgebaut.

Jetzt erstellen wir zum Empfang der Nachrichten aus der Warteschlange einen `ServiceBusProcessor`. Dieser bekommt den Namen der Warteschlange als Parameter übergeben.

```
serviceBusClient =
    new ServiceBusClient("<CONNECTIONSTRING>");
serviceBusProcessor = _serviceBusClient.CreateProcessor(
    "<QUEUENAME>");
```

Der `ServiceBusProcessor` ist ein Observer, in dem zwei Events registriert und implementiert werden müssen:

```
info: ServiceBusWorker.Worker[0]
  On-Premises-Worker: Bestellung empfangen. Inhalt der Datei: Bestellung1
info: ServiceBusWorker.Worker[0]
  On-Premises-Worker: Bestellung empfangen. Inhalt der Datei: Bestellung2
info: ServiceBusWorker.Worker[0]
  Worker running at: 08/16/2024 15:40:35 +02:00
info: ServiceBusWorker.Worker[0]
  On-Premises-Worker: Bestellung empfangen. Inhalt der Datei: Bestellung3
info: ServiceBusWorker.Worker[0]
```

Ausgabe der Nachrichten aus dem Service Bus (Bild 11)

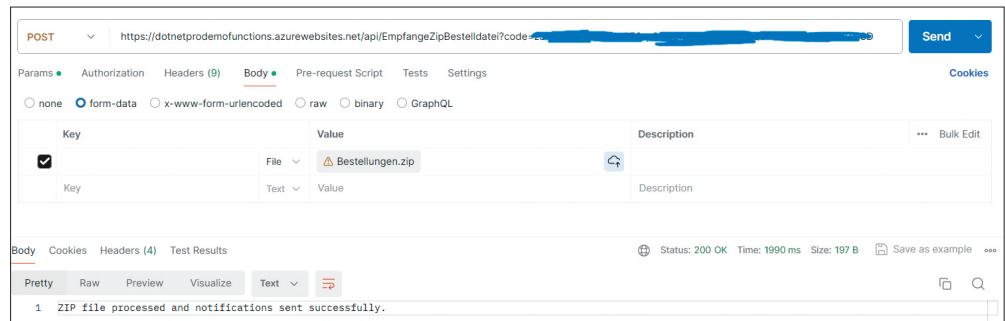
```
serviceBusProcessor.ProcessMessageAsync +=
    _serviceBusProcessor_ProcessMessageAsync;
serviceBusProcessor.ProcessErrorAsync +=
    _serviceBusProcessor_ProcessErrorAsync;
```

`ProcessMessageAsync` wird aufgerufen, sobald eine neue Nachricht in der Warteschlange verfügbar ist, und `ProcessErrorAsync` wird für Fehlerfälle benötigt. In der `ProcessMessageAsync` wird die Logik zum Empfang und zur weiteren Verarbeitung der Nachricht implementiert. In diesem Fall packen wir die Payload der Nachricht – die Bestellung – aus und speichern diese als `variable`, damit wir diese auf der Konsole ausgeben (siehe Bild 11). Die Logik zum ERP entfällt hier.

Zum Abschluss muss die Nachricht im Service Bus als verarbeitet gemeldet werden.

```
await arg.CompleteMessageAsync(arg.Message);
```

Wird das nicht gemacht, wird die Nachricht im definierten Zeitraum immer wieder an den Empfänger übergeben (Guaranteed Delivery), bevor sie in der Dead-Letter Queue landet.



Aufruf der Function mit Postman (Bild 10)

Damit der `ServiceBusProcessor` seine Arbeit aufnehmen kann, müssen Sie ihn starten. Das passiert in der `Execute`-Methode des Workers mit folgendem Aufruf:

```
await _serviceBusProcessor.StartProcessingAsync(
    stoppingToken);
```

Sie haben ein Ende-zu-Ende-Szenario aus der Cloud (Azure) auf Ihre On-Premises-Umgebung (Background-Worker) mit Hilfe von Service Bus implementiert. Vielen Dank für Ihre Fahrt mit unserem Service Bus!

Zusammenfassung

Sichere Kommunikation ist auch dann möglich, wenn Cloud und On-Premises-Systeme nicht über ein VPN-Gateway oder Ähnliches miteinander verbunden sind. Der leichtgewichtige, aber durchaus sichere Ansatz kommt ohne hohen administrativen Aufwand aus und bremst die Innovationsfähigkeit nicht aus.

Auf der anderen Seite sind verteilte Systeme sehr komplex, und Sie oder Ihr Team benötigen das Hintergrundwissen über Azure Functions und Service Bus. Auch wenn es sich hierbei um Serverless-Angebote handelt, ist ein fundamentales Wissen notwendig. ■

[1] Microsoft Learn, Was ist Azure Service Bus?

www.dotnetpro.de/SL2411CloudOnPrem1

[2] Microsoft Learn, Übersicht zu Azure Functions,

www.dotnetpro.de/SL2411CloudOnPrem2

[3] Azure Portal, <https://portal.azure.com>



Sebastian Meyer

ist Lead Architect bei der QUIBIQ Hamburg GmbH und Microsoft MVP. Er berät seit vielen Jahren Unternehmen bei der Architektur und Umsetzung von Integrationsplattformen, ist Sprecher, Trainer und Autor.

www.linkedin.com/in/sebastianmeyerit

dnpCode

A2411CloudOnPrem



LERNSTOFF

dnpCode A2411Lernstoff

ERSTE SCHRITTE MIT SQL-ABFRAGEN

T-SQL

Bewusst für Debütanten konzipiert ist dieser Titel und zugleich ganz auf T-SQL, die Abfragevariante für den Microsoft SQL-Server, beschränkt. Um den Zugang einfach zu halten, wird eine übersichtliche kleine Beispieldatenbank verwendet, sodass sich die Ergebnisse von Abfragen leichter überprüfen lassen und die Struktur der Datenbank immer im Blick bleibt. Als Software werden die frei zugänglichen Versionen von Microsoft SQL Server 2022 zugrunde gelegt.

Im ersten Kapitel wird mit exakt bebilderten Schrittfolgen zunächst die Arbeit mit dem SQL Server Management Studio vorgestellt, um eine praktische Basis für die Arbeit mit Skriptdateien und Projekten zu gewinnen. Systematisch erläutert der Autor zunächst die Grundlagen von T-SQL, um in der Folge die Verwendung der Sprachelemente in der Praxis auszuprobieren: *SELECT*-Abfragen, Filtern, Gruppieren, Aggregieren und Sortieren. Ausführlich werden die verschiedenen Datentypen, die integrierten Funktionen und der Einsatz von Operatoren beschrieben. Eigene Kapitel behandeln das Pivo-



Dirk Angermann

T-SQL-Abfragen für Microsoft SQL-Server 2022. Grundlagen und Konzepte

mitp, 2024, 280 Seiten, Deutsch, ISBN 978-3-7475-0631-8, 34,99 Euro

Zielgruppe:
Datenbankabfragen

Voraussetzungen: keine

Leseprobe:
www.dotnetpro.de/SL2411Lernstoff1

tieren, die Arbeit mit Groupingsets für die Zusammenfassung von Gruppen, die Abfrage von Metadaten und die Arbeit mit gespeicherten Prozeduren.

Programmieren mit T-SQL behandelt die Nutzung von T-SQL-Batch-Dateien. Dazu gehören auch die Fehlerbehandlung und der Umgang mit Transaktionen für Datenänderungsanweisungen und ihrer Steuerung mit *BEGIN*-, *COMMIT*- oder *ROLLBACK*-Anweisungen. Zum Abschluss geht es darum, wie sich Abfragen mithilfe von Indizes optimieren lassen und die Abfrageleistung kontrolliert werden kann. (vh) ■

SOFTWARE MESSEN

Metriken

Ist die Software gut? Ist sie umfangreich und komplex, war sie aufwendig zu programmieren? Wird sie sich leicht pflegen lassen? Wer solche Fragen möglichst objektiv beantworten möchte, braucht Metriken für Software.

Das Buch mit dem Untertitel „Die Vermessung von Applikationen“ präsentiert die Verfahren für die Messung von Software und erläutert deren Einsatzgebiete. Es liefert Methoden und Metriken, um Softwareartefakte von den Anforderungen über das Softwaredesign und den Code bis hin zu den Testmitteln in den drei Dimensionen Quantität, Komplexität und Qualität zu messen und zu bewerten. Die aktuelle zweite Auflage berücksichtigt dabei auch die aktuellen Entwicklungen im Bereich der Metriken und deren Bedeutung im Zusammenhang mit DevOps und Agilität.

Auf rund 290 Seiten, untergliedert in zehn Kapitel, präsentieren die Autoren den aktuellen Stand der Technik zum Thema Softwaremetriken. Die Ausführungen reichen von der



R. Seidl, M. Baumgartner, H. M. Sneed

Software-Metriken. Die Vermessung von Applikationen

Hanser, 2. Auflage 2024, 356 Seiten, Deutsch, ISBN 978-3-446-47687-5, 59,99 Euro

Zielgruppe: Software-Ingenieure

Voraussetzungen: Keine

Leseprobe:
www.dotnetpro.de/SL2411Lernstoff2

Softwarekomplexität nach dem IEEE-Standard über die Ansätze, um Softwarequalität zu messen, bis zur Vorstellung unterschiedlicher Codemetriken. Auch der Messung der Wartungsproduktivität ist ein eigenes Kapitel gewidmet.

Das abschließende Kapitel beleuchtet die Softwaremessung in der Praxis und erläutert, wie man dauerhafte Messverfahren auf vorhandene Metriken aufsetzen kann. Kurz vorgestellt werden Praxisbeispiele von Hewlett Packard und der Siemens AG. (bl) ■

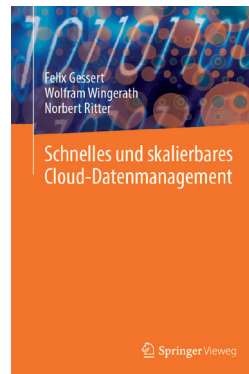


DATENMANAGEMENT

Latenzminimierung

Langsam reagierende Websites verärgern die Kundschaft. Auf Basis der Erfahrungen mit der von ihnen gegründeten Backend-as-a-Service-Plattform Bagend entwickeln die Autoren in diesem Titel ein übersichtliches Klassifikationschema, das ein fundiertes Urteil darüber erlaubt, welches der zahlreichen Datenmanagementsysteme für die eigenen Anforderungen am besten geeignet sein könnte. Es geht dabei um eine an der Praxis orientierte Entscheidungshilfe für die unterschiedlichen Anwendungsszenarien. Dabei stehen insbesondere NoSQL-Speichersysteme im Vordergrund.

Zunächst werden die Auswirkungen unterschiedlicher Architekturen in Cloud-basierten Anwendungen auf die Latenz betrachtet. Ausführlich wird auf die Nutzung von Zwischenspeichern im Web eingegangen. Eine kontrollierte Abschwächung der Konsistenzanforderungen wird dabei als Option betrachtet. Das Verlagern von Komplexität in die Cloud beim Einsatz von NoSQL-Datenbanken rückt Database-as-a-Service- und Backend-as-a-Service-Lösungen in den Vorder-



F. Gessert, W. Wingerath, N. Ritter

Schnelles und skalierbares Cloud-Datenmanagement

Springer, 2024, 210 Seiten, Deutsch, ISBN 978-3-031-54387-6, 129,99 Euro, E-Book 99,99 Euro

Zielgruppe: Cloud-Datenmanagement

Voraussetzungen: keine

Leseprobe (Anmeldung erforderlich): www.dotnetpro.de/SL2411Lernstoff3

grund. Als wesentliche Mittel zur Latenzminimierung werden die verschiedenen Caching-Methoden sowohl auf der Server- als auch auf der Client-Seite betrachtet. Ein weiterer Schwerpunkt ist, wie sich Datenänderungen möglichst effektiv durch verteilte Transaktionen vornehmen lassen.

Den Abschluss bilden ein Überblick über die aktuellen NoSQL-Tools zu den Themen Partitionierung, Replikation, Speicherverwaltung und Anfrageverarbeitung sowie ein Ausblick auf zukünftige Trends. (vh) ■

● Kurz vorgestellt

Bernd Baumgarten

Diskrete Mathematik kompakt

Deckt etliche Grundvorlesungen für Studierende der Informatik ab. Gegenstand sind Mengen, Relationen, Funktionen, Logik und Algebra sowie diskrete Anwendungen wie Zahlen, Graphen und diskrete Wahrscheinlichkeit. Mit Übungsaufgaben und Lösungen.

De Gruyter Oldenbourg, 2. Auflage 2024, 448 Seiten, Deutsch, ISBN 978-3-11-133572-8, 74,95 Euro

Matthias Daigl, Rolf Glunz

ISO 29119

Praxisorientierte Einführung sowie Leitfaden für alle, die ein modernes Software-Testkonzept erstellen wollen und dabei Wert auf Normenkonformität legen. Mit vielen Hintergrundinformationen sowie ausführlichen Fallstudien aus unterschiedlichen Anwendungsbereichen.

O'Reilly, 2. Auflage 2024, 314 Seiten, Deutsch, ISBN 978-3-98889-011-5, 44,90 Euro

Scot W. Stevenson

Einführung in die moderne Assembler-Programmierung

Von den Grundlagen bis zu verfeinerten Anwendungsmöglichkeiten. Umsetzung höherer Assembler-Strukturen (Schleifen, Stapel, Sprungtabellen, Rekursion et cetera) in effektivem RISC-V-Code. Zudem warnt der Autor vor Fallstricken und gibt Tipps.

dpunkt.verlag, 2024, 278 Seiten, Deutsch, ISBN 978-3-98889-007-8, 36,90 Euro

Ulrike Häßler

JavaScript Schnelleinstieg

JavaScript-Grundlagen für die Webprogrammierung mit zahlreichen Übungsaufgaben und Lösungen. Behandelt werden auch das Document Object Model (DOM), Event Handling, asynchrone Verarbeitung, das REST-API sowie das Speichern von Cookies.

Vorausgesetzt werden solide Grundkenntnisse in HTML und CSS. mitp, 2024, 320 Seiten, Deutsch, ISBN 978-3-7475-0778-0, 19,99 Euro

WISEJ.NET HYBRID

Das Web nativ für die Hosentasche

Mit Wisej.NET Hybrid moderne Single-Page-Webapplikationen auf mobilen Apps nativ ausführen.

Wisej.NET ist ein Framework für die Entwicklung moderner Webanwendungen, das auf dem .NET-Ökosystem basiert. Es ist besonders nützlich für Entwickler, die bestehende Windows-Forms- oder WPF-Anwendungen zu Webanwendungen migrieren oder neue plattformübergreifende Anwendungen von Grund auf erstellen möchten.

Wisej.NET Hybrid erweitert diesen Ansatz um die Möglichkeit, Wisej.NET-basierte Webanwendungen nativ auf den Endgeräten auszuführen. Dies gelingt durch eine Kombination mit dem Framework .NET MAUI [1]. Diese Integration ermöglicht es, moderne Anwendungen zu erstellen, die sowohl die Vorteile von Webanwendungen bieten als auch die nativen Funktionen von Desktop- und mobilen Plattformen nutzen können.

Wisej.NET im Schnelldurchlauf

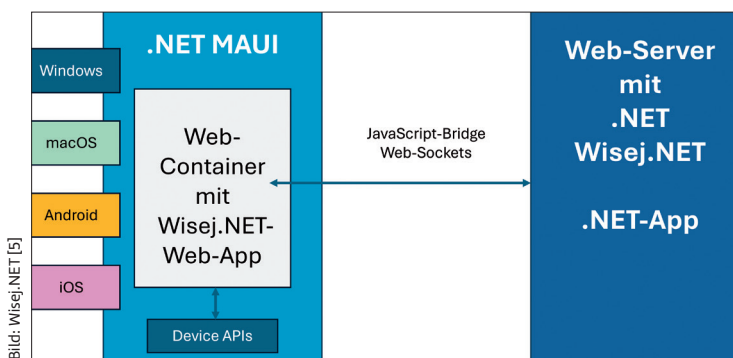
Wisej.NET wurde in der dotnetpro bereits ausführlicher unter [2] und [3] vorgestellt, daher beschränken wir uns hier auf einen kompakten Überblick über die Ziele und den techni-

auf die Webplattform übertragen, da Wisej.NET den Großteil der bestehenden Codebasis und Architektur beibehalten kann. Es erlaubt die Entwicklung von Webanwendungen mit einem Ansatz, der dem von Windows-Forms-Anwendungen ähnelt. Auch hier erfolgt die Programmierung mithilfe von C# oder optional mit Visual Basic .NET. Das Framework ist ebenfalls geeignet, Anwendungen, die mit ähnlichen Technologien wie etwa Visual Basic 6 oder Gupta erstellt wurden, zu einer modernen Webapplikation unter Beibehaltung großer Teile der Softwarestrukturen zu migrieren.

Ein zentrales Merkmal dieses Frameworks ist seine umfangreiche Komponentenbibliothek, die eine breite Palette von Steuerelementen zur Verfügung stellt. Die Gestaltung der Benutzeroberfläche erfolgt während der Entwicklungsphase mithilfe eines grafischen Designers, der in Visual Studio integriert ist. Dies beschleunigt und vereinfacht die Erstellung von Webanwendungen erheblich. Für alle relevanten grafischen Steuerelemente einer bestehenden Windows-Forms-Applikation gibt es eine modernisierte und neu gestaltete Entsprechung mit einem fast identischen Funktionsumfang sowie gleichen Eigenschaften, Methoden und Ereignissen. Diese Analogie vereinfacht den Migrationsprozess und führt dazu, dass einfache Dialogfelder ohne weitere Anpassungen direkt übernommen werden können. Anpassungen der Funktion und des Designs können über neue, leistungsfähige Steuerelemente erreicht werden.

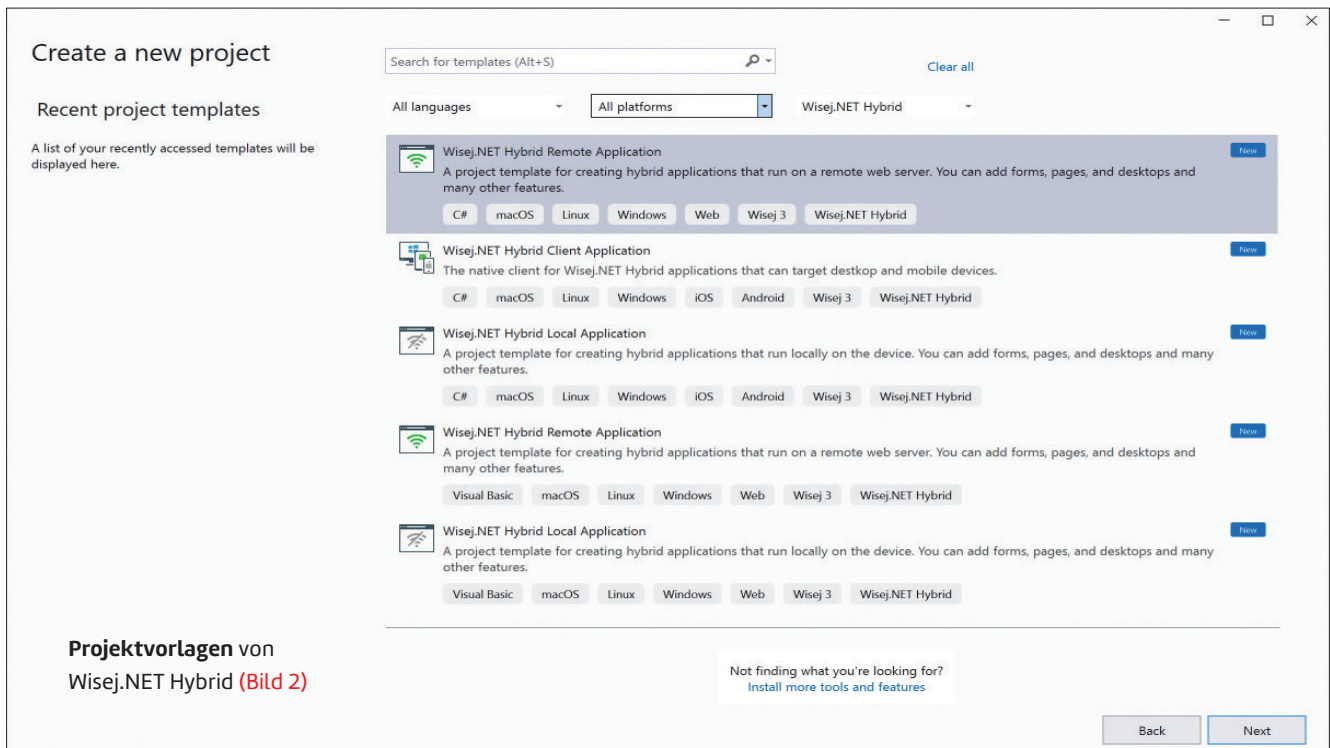
Das Konzept des Echtzeit-Updates (Single Page Application) ermöglicht eine schnelle Aktualisierung der Benutzeroberfläche, ohne dass die gesamte Seite neu geladen werden muss. Verkürzt und stark vereinfacht könnte man es so ausdrücken, dass man die Anwendungslogik bei der Softwremigration vom Client auf den Server verlagert. Das Framework sorgt dann dafür, dass die ursprüngliche .NET-Anwendung auf dem Server ausgeführt wird und die Clients diese über den Browser aufrufen können. Die Unterscheidung zwischen der Server- und der Clientseite gestaltet sich folgendermaßen:

- **Serverseite:** Im Wisej.NET-Framework übernimmt der Server die zentrale Rolle, indem er die Kommunikation zwischen dem Client (dem Browser) und der Anwendung steuert. Der Server empfängt Anfragen vom Client, führt den entsprechenden Code aus und schickt die aktualisierte Be-



Architektur einer Wisej.NET-Hybrid-Anwendung (Bild 1)

schen Ansatz dieses Frameworks. Wisej.NET bietet die Möglichkeit, bestehende Anwendungen aus Windows Forms und WPF in moderne Webanwendungen zu migrieren. Dies geschieht durch die Bereitstellung von Tools und Methoden, die die Portierung der bestehenden Benutzeroberflächen und der Geschäftslogik erheblich vereinfachen. Entwickler, die bereits Erfahrung mit Windows Forms und WPF haben, können ihre bestehenden Anwendungen mit minimalem Aufwand



nutzeroberfläche zurück zum Client. Außerdem verwaltet der Server den Anwendungszustand und speichert Informationen über die aktuelle Benutzersitzung. Die Ausführung der .NET-Anwendung erfolgt auf dem Server.

- **Clientseite:** Der Client bezieht sich hier auf den Webbrowser. Er repräsentiert die Benutzeroberfläche, verarbeitet Nutzerinteraktionen und sendet Anfragen an den Server. In dieser Sphäre werden hauptsächlich HTML, CSS und JavaScript verarbeitet. Die Verbindung zwischen Client und Server wird über Websocket- oder HTTP-Verbindungen hergestellt, um Echtzeitaktualisierungen zu ermöglichen.

Wisej.NET ist sowohl mit dem älteren .NET Framework bis zur Version 4.8 als auch mit .NET 8 kompatibel. Diese Vielseitigkeit ermöglicht die Ausführung sowohl auf Windows als auch auf Linux-Betriebssystemen auf dem Server. Dabei hat sich in der Praxis gezeigt, dass die neu erstellten Webapplikationen auf der Basis von Wisej.NET in der Lage sind, eine Vielzahl gleichzeitiger Verbindungen adäquat zu handhaben, ohne dass dabei Probleme bei höherer Last auftreten.

Wisej.NET Hybrid – Verbindung von Wisej.NET und .NET MAUI

Das Hauptziel von Wisej.NET Hybrid ist, die Entwicklung plattformübergreifender Anwendungen zu vereinfachen. Zielbetriebssysteme sind Windows, macOS, iOS und Android. Durch die Kombination von Wisej.NET und .NET MAUI können Entwickler eine einheitliche Codebasis verwenden, um Anwendungen für Web, Desktop und Mobile zu erstellen [4]. Dies reduziert den Aufwand und die Komplexität, die normalerweise mit der Wartung mehrerer Codebasen für verschiedene Plattformen verbunden sind.

Das Ziel ist, eine Wisej.NET-basierte Webapplikation in einem nativen App-Container auf der jeweiligen Zielpattform auszuführen und auf diese Weise auf die System- und Gerätefunktionen zugreifen zu können. Mithilfe eines speziellen hybriden Offline-Modus ist es auch möglich, dass die Webapplikation ohne permanenten Netzwerkzugriff ausgeführt werden kann. Zu den nativen Funktionen gehören beispielsweise ein Zugriff auf GPS, Kamera, Push-Benachrichtigungen, E-Mail-Schnittstelle, Sensoren und das Dateisystem.

Durch die Kombination der beiden Frameworks .NET MAUI und Wisej.NET Hybrid stellt sich die Architektur gemäß Bild 1 dar [5]. Diese kann in groben Zügen wie folgt beschrieben werden (siehe auch die Ausführungen zu Wisej.NET weiter oben):

- **Client-Server-Architektur:** Wisej.NET Hybrid nutzt eine Client-Server-Architektur. Diese besteht aus mehreren Schichten: dem Webserver, dem Hybrid-Container, der Client-Schicht und der Integration mit nativen APIs.
- **Webserver-Schicht:** Sie bildet das Backend der Wisej.NET-Hybrid-Anwendung und ist für die Verwaltung der Geschäftslogik, die Datenverarbeitung und die Kommunikation mit den Client-Geräten verantwortlich. Wisej.NET Hybrid basiert auf ASP.NET Core. Der Server verarbeitet HTTP-Anfragen, führt die Geschäftslogik aus und sendet die Ergebnisse zurück an den Client.
- **Wisej.NET Framework:** Dieses Framework bietet die Funktionen, um die Webanwendung mit Echtzeit-Updates, serverseitiger UI-Logik und Datenbankbindung zu erstellen. Wisej.NET ermöglicht es, UI-Komponenten serverseitig zu verwalten und Änderungen in Echtzeit an den Client zu übermitteln, was eine Interaktivität und Dynamik der Anwendung ermöglicht. ▶

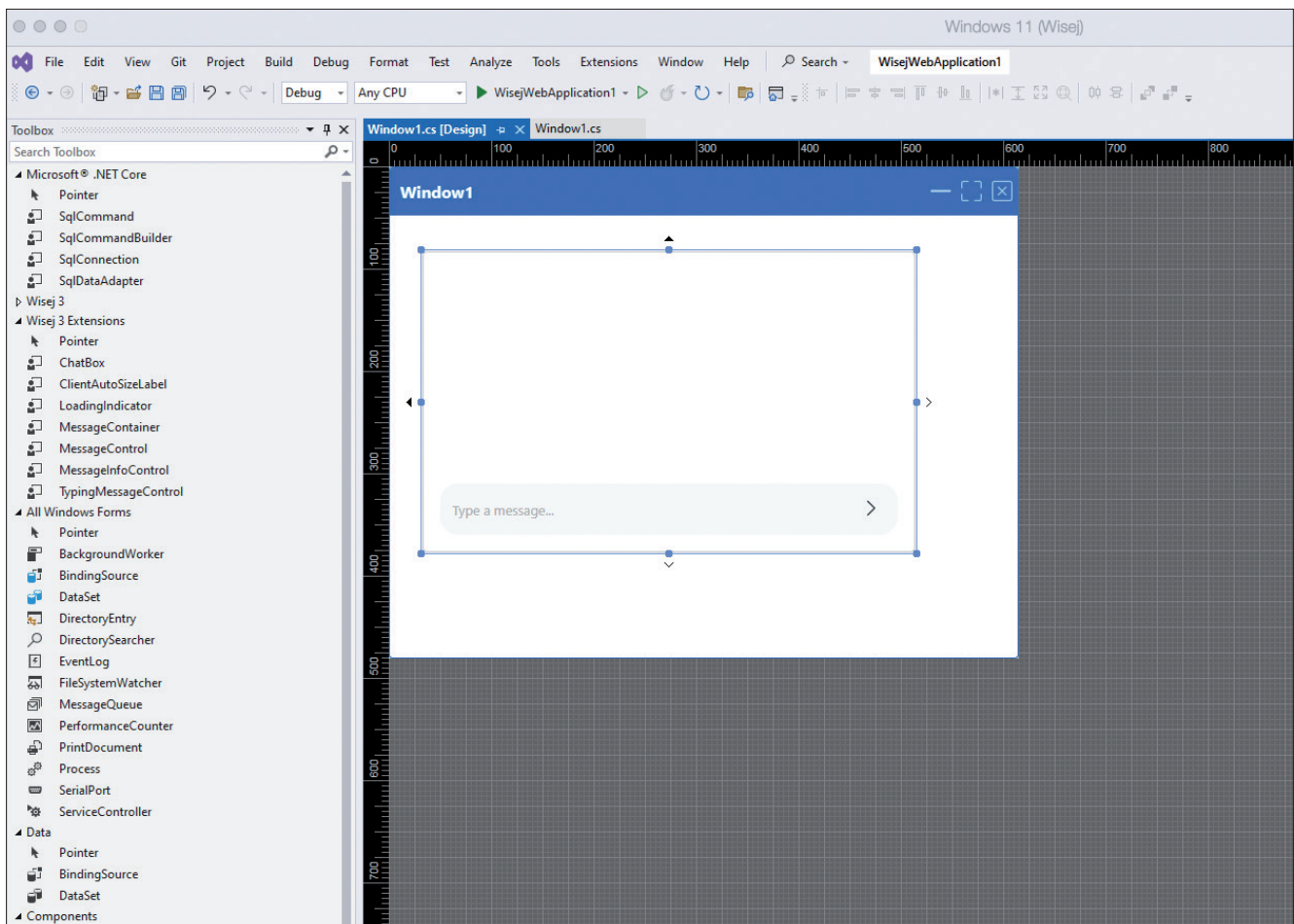
- **Hybrid-Container:** Dieser ist eine entscheidende Komponente in der Wisej.NET-Hybrid-Architektur, da der Hybrid-Container die Webanwendung in eine native Umgebung einbettet. Zum Einsatz kommt hierfür eine *WebView*-Komponente. Sie ermöglicht es, die Webanwendung innerhalb eines nativen Rahmens darzustellen, was die Integration von Webinhalten in native Apps ermöglicht. Diese Komponente wird sowohl auf Desktop- als auch auf mobilen Plattformen genutzt, um eine konsistente Benutzeroberfläche zu bieten.
- **Verbindung zu nativen APIs:** Der Hybrid-Container ermöglicht den Zugriff auf native APIs und Betriebssystemfunktionen, die für den Zugriff auf Hardware-Komponenten und spezifische Plattfordienste erforderlich sind. Dadurch können Funktionen wie Push-Benachrichtigungen, GPS, Kamera und Sensoren genutzt werden.
- **Client-Schicht:** Die Client-Schicht ist für die Benutzeroberfläche und die Interaktion mit dem Benutzer zuständig. Sie umfasst die Darstellung der UI-Komponenten und die Verwaltung der Benutzerinteraktionen. Die Client-Schicht nutzt HTML5 und JavaScript, um die Benutzeroberfläche zu rendern und interaktive Elemente bereitzustellen. Diese Technologien ermöglichen es, moderne und responsive Benutzeroberflächen zu erstellen, die auf verschiedenen Geräten und Bildschirmgrößen gut funktionieren.

- **UI-Komponenten von Wisej.NET:** Die umfangreiche Bibliothek von UI-Komponenten in Wisej.NET bietet Entwicklern die Möglichkeit, komplexe und interaktive Benutzeroberflächen zu erstellen. Diese Komponenten sind anpassbar, sodass sich mit ihnen auch benutzerdefinierte Steuerelemente erstellen lassen. Darüber hinaus steht eine große Auswahl an Komponenten von Drittanbietern zur Verfügung. Weitere JavaScript-basierte Komponenten können ebenfalls in Wisej.NET integriert werden.
- **Echtzeitkommunikation:** Wisej.NET Hybrid basiert auf einer bidirektionalen Kommunikation zwischen Server und Client. Dieser Ansatz unterstützt Echtzeit-Updates und Synchronisierung, die für Anwendungen erforderlich sind, die kontinuierlich aktuelle Daten anzeigen sollen oder sofortige Rückmeldungen von Benutzern benötigen.

Als Entwicklungsumgebung kommt Visual Studio zum Einsatz. Die Effizienz der Entwicklung wird durch die Integration eines grafischen Designers für die Erstellung der Benutzeroberfläche der Webapplikation maßgeblich beschleunigt.

Setup und eine erste App

Installieren Sie Visual Studio ab der Version 17.8 und das .NET Framework Version 8. In Visual Studio sind folgende Workloads über den Visual-Studio-Installer zu installieren:



Grafischer Designer für Wisej.NET (Hybrid) in Visual Studio (Bild 3)

- *.NET desktop development*: Für die Entwicklung von WinForms-Anwendungen.
- *ASP.NET and web development*: Entwicklung von Webapplikationen inklusive Einrichtung eines lokalen Webserver, um diese direkt auf dem Entwicklungsrechner testen zu können.
- *.NET Multi-platform App UI development*: Voraussetzung für die Entwicklung von Apps auf der Basis von .NET MAUI.

Ebenso installieren Sie die Extension von Wisej.NET 3.5 über den Extension-Manager von Visual Studio. Dabei werden folgende Projekttypen eingerichtet (Bild 2):

- *Client-Anwendung*: Diese Projektvorlage wird zum Erstellen eines Wisej.NET-Hybrid-Clients verwendet. Das Client-Projekt erstellt eine ausführbare Datei (.exe, .apk, .ipa und andere), die auf jedem Desktop oder jeder mobilen Plattform bereitgestellt werden kann. Diese Vorlage basiert auf .NET MAUI.
- *Remote-Anwendung*: Erstellt eine Wisej.NET-Anwendung, die auf einem Remote-Webserver wie IIS, Kestrel oder Nginx unter Windows oder Linux bereitgestellt wird und es erlaubt, mit den nativen Funktionen eines entfernten Geräts zu interagieren. Um solche nativen Gerätefunktionen nutzen zu können, müssen Sie über die Wisej.NET-Hybrid-Client-Anwendung auf diese Funktionen zugreifen.
- *Lokale Anwendung*: Erstellt Anwendungen, die vollständig auf dem jeweiligen Gerät ausgeführt werden. Die Projektvorlage enthält ähnliche Funktionen wie die Remote-Anwendungsvorlage. Dieses Projekt muss zur Verwendung mit der Wisej.NET-Hybrid-Client-Anwendung verknüpft werden.

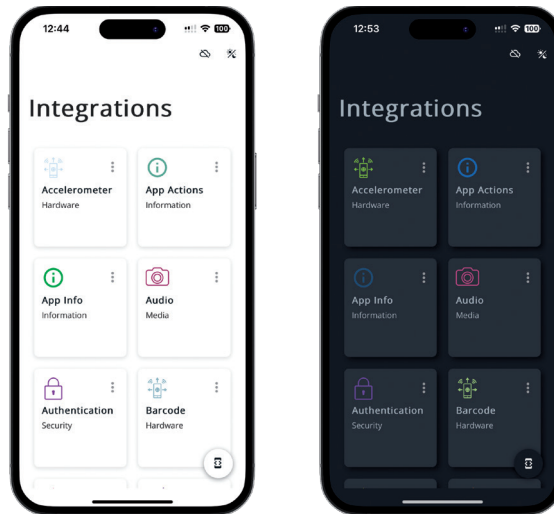
Alle Anwendungstypen sind hybride Apps, das heißt, sie basieren auf einer Integration der Web-App (Wisej.NET) und einer .NET-MAUI-App.

Der folgende Test basiert auf der Projektvorlage *Hybrid Client Application*. Das generierte .NET-MAUI-Projekt initialisiert in der *App*-Klasse einen internen Web-Container für die Ausführung der Wisej.NET-App:

```
MainPage = new HybridShell();
```

Eine Wisej.NET-Hybrid-Projektmappe enthält zwei Projekte:

- *Wisej.NET Web-App*: Dabei handelt es sich um eine Webapplikation, die auf dem Framework Wisej.NET basiert. Durch den Einsatz des Embedded-Servers kann diese auch offline ausgeführt werden, sofern das in Bezug auf die Datenhaltung sinnvoll ist.
- *Wisej.NET Hybrid-App*: Es handelt sich um die hybride Wisej.NET-App auf der Basis von .NET MAUI, die in ihrem internen Web-Container die Web-App ausführt. Die .NET-



Mobile App (.NET MAUI) mit eingebetteter Wisej.NET-Web-App (Bild 4)

MAUI-App kann nativ auf den Zielsystemen Windows, macOS, iOS und Android gestartet werden.

Fassen wir die entstandene Architektur zusammen: Die hybride Web-App auf der Basis von Wisej.NET ist in die .NET-MAUI-App eingebettet. Der Vorteil liegt in der Kombination dieser beiden App-Typen. Man nutzt Wisej.NET, um mit dessen Hilfe eine Webapplikation zu erstellen. Dazu stehen unter anderem der grafische Designer in Visual Studio (Bild 3) und die große Auswahl an visuellen Komponenten zur Verfügung. Gleichwohl können auch bestehende Win-

Forms- oder WPF-Applikationen durch Migration auf effiziente Art und Weise in eine Webapplikation migriert werden. Diese Web-App wird in die .NET-MAUI-App eingebunden (Web-Container). Die Web-App erhält über den internen Web-Container und das Framework .NET MAUI native App-Eigenschaften, beispielsweise einen erweiterten System- und Gerätezugriff, wie es der nachfolgende Abschnitt erläutert.

Im Ergebnis erhält man eine native .NET-MAUI-App (Bild 4), die alle Vorzüge einer nativen App wie die Installation auf dem Device, Deployment über einen Store, System- und Gerätezugriff sowie Offline-Modus aufweist.

Features

Wisej.NET Hybrid bietet beim Erstellen hybrider Remote- und hybrider lokaler Anwendungen zahlreiche Funktionen an, die in diesen Anwendungen zum Einsatz kommen können, siehe Bild 5 und [6]. Beispiele solcher Funktionen sind:

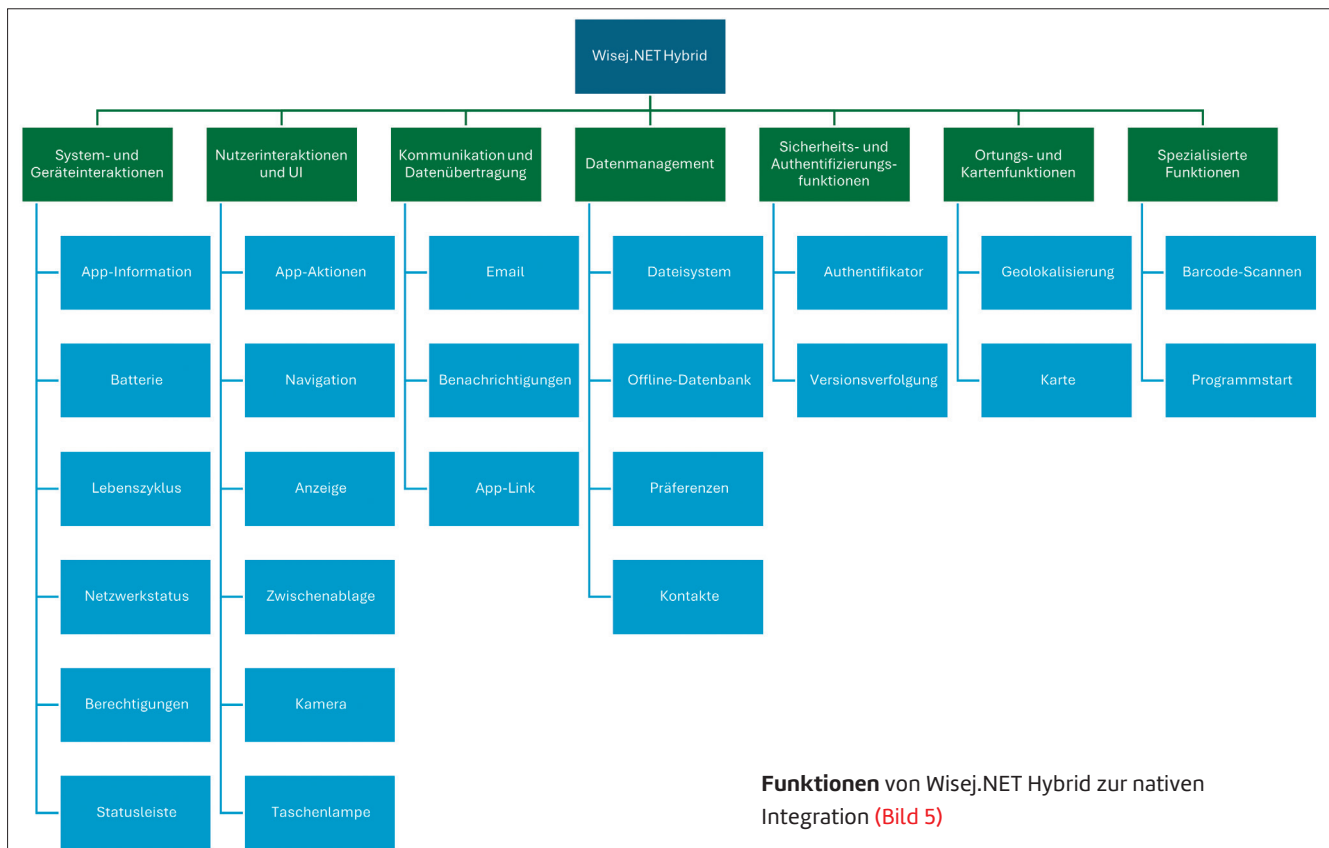
- **System- und Geräteinteraktionen**: Diese Gruppe umfasst Funktionen, die mit der Interaktion zwischen der Anwendung und dem System oder Gerät zusammenhängen. Dazu gehört die Funktion *App Info*, die Methoden zur Interaktion mit den Anwendungsinformationen in den Geräteeinstellungen bietet. Die Funktion *Battery* ermöglicht die Ermittlung des Ladezustands des Geräts und gibt Auskunft, ob der Energiesparmodus aktiviert ist. *Lifecycle Events* helfen dabei, zu bestimmen, wann die Anwendung in den Hintergrund oder Vordergrund tritt oder beendet wird. Die Funktion *Network Events* ermöglicht das Lesen von Informationen über den aktuellen Netzwerkstatus und die verfügbaren Verbindungen. Mit der Funktion *Permissions* können zur Laufzeit Benutzerberechtigungen überprüft und angefordert werden, um den Zugriff auf verschiedene Gerätefunktionen zu ermöglichen. Schließlich lassen sich mit der Funktion *Status Bar* der Text und die Hintergrundfarben der Statusleiste des Geräts manipulieren.
- **Nutzerinteraktionen und UI**: Diese Funktionen betreffen die direkte Interaktion des Nutzers mit der Anwendung und ►

deren Benutzeroberfläche. Die Funktion *App Actions* verwaltet Anwendungsaktionen, wie zum Beispiel Verknüpfungen zu bestimmten Teilen der Anwendung. Die Funktion *Navigation* ermöglicht die Anzeige von nativen .NET-MAUI-XAML-Ansichten als modales Dialogfeld. Mit der Funktion *Display* kann die Sperrung des Bildschirms verhindert und die Bildschirmausrichtung sowie -größe ermittelt werden. Die Funktion *Clipboard* erlaubt das Lesen und Schreiben von Text in die Zwischenablage. Die *Camera*-Funktion ermöglicht das Aufnehmen von Bildern und Videos, während die Funktion *Flashlight* zum Ein- und Ausschalten der Taschenlampe verwendet werden kann.

- **Kommunikation und Datenübertragung:** In dieser Gruppe befinden sich Funktionen, die für die Kommunikation und den Datenaustausch zuständig sind. Mit der Funktion *Email* können E-Mails direkt aus dem Programm verfasst und versendet werden. *Local Notifications* ermöglichen das Versenden von Push-Benachrichtigungen, die zu einem bestimmten Zeitpunkt oder innerhalb eines bestimmten Zeitraums ausgelöst werden. Die Funktion *App-Links* stellt Methoden zur Registrierung und Abmeldung von Anwendungs-Deep-Links bereit. Die *App Info*-Funktion wird genutzt, um Daten über die App abzurufen.
- **Datenmanagement:** In dieser Gruppe sind die Funktionen zusammengefasst, die das Management von Daten betreffen. Die Funktion *File System* bietet Zugriff auf die Dateien des Systems, sodass diese gelesen und geschrieben werden können. Die Funktion *Offline Database* ermöglicht die Nutzung einer SQLite-Datenbank, um Daten zu speichern,

selbst wenn keine Internetverbindung besteht. Mit der Funktion *Preferences* können Schlüssel-Wert-Paare auf dem Clientgerät gespeichert werden. Schließlich ermöglicht die Funktion *Contacts* den Zugriff auf die Gerätekontakte als JSON-Datenstrom.

- **Sicherheits- und Authentifizierungsfunktionen:** Diese Gruppe fokussiert sich auf Sicherheitsfunktionen und Authentifizierung. Der *Authenticator* ermöglicht die Integration von Single Sign-on (SSO) sowie die Nutzung biometrischer Authentifizierung, etwa in Form von Gesichtserkennung oder Fingerabdruck. Die Funktion *Version Tracking* überprüft Versions- und Build-Nummern der Anwendung und zeigt zusätzliche Informationen an, wie etwa, ob die Anwendung zum ersten Mal gestartet wurde.
- **Ortungs- und Kartenfunktionen:** Diese Funktionen betreffen die Nutzung von Standortdaten und Karten. Die Funktion *Geolocation* ermöglicht die Ermittlung des Geräte-Standorts und bietet eine Geocodierungsfunktion, um relevante Orientierungspunkte für einen Standort zu bestimmen. Mit der Funktion *Map* kann die auf dem Gerät installierte Kartenanwendung mit einem vordefinierten Standort oder einer Ortsmarkierung gestartet werden.
- **Spezialisierte Funktionen:** In dieser Gruppe befinden sich Funktionen, die sehr spezifische Aufgaben erfüllen. So bietet *Barcode Scanning* eine Barcode-Scan-Integration für Android und iOS, was das Scannen von Barcodes direkt in der Anwendung ermöglicht. Mit der Funktion *App Actions* können andere installierte Anwendungen auf dem Gerät direkt aus der App heraus geöffnet werden.



Mithilfe dieser Funktionen wird die Webanwendung zu einer nahezu vollständig nativ integrierten Applikation auf dem Zielsystem, was bedeutet, dass diese über Möglichkeiten verfügt, die ihr sonst üblicherweise verwehrt sind.

Fazit und Ausblick

Wisej.NET Hybrid bietet einen interessanten Ansatz, um Webapplikationen einerseits auf effiziente Art und Weise zu erstellen und diese sogleich in einem nativ eingebetteten Web-Container auf den Zielgeräten, inklusive einem Geräte- und Systemzugriff, auszuführen.

Die Webapplikation kann dabei ihrerseits auf der Basis eines Migrationsprojekts (WinForms, WPF) oder als Neuentwicklung erstellt werden.

Die Ziele beim Einsatz dieser Technologie sind, mit einer einzigen Codebasis mehrere Plattformen zu erreichen, die Entwicklungskosten zu verringern und damit die Markteinführungszeit erheblich zu verkürzen. Inbegriffen ist dabei auch die Möglichkeit, für bestehende Legacy-Applikationen vollständig neue Benutzeroberflächen auf der Basis moderner Webtechnologien zur Verfügung zu stellen, bestehenden Quellcode weiterhin zu nutzen (Businesslogik) und gleichzeitig native Features auf Desktop- und Mobilgeräten zu verwenden. ■

[1] .NET Multi-platform App UI,
www.dotnetpro.de/SL2411WisejMAUI1

[2] Veikko Krypczyk, *Der Stand der Dinge*, dotnetpro 1/2018, Seite 120 ff., www.dotnetpro.de/A1801Wisej

[3] Veikko Krypczyk, *Alter Wein in neuen Schläuchen?*, dotnetpro 4/2017, Seite 64 ff., www.dotnetpro.de/A1704Wisej

[4] Jon Hilton, *Going Native with Wisej.NET Hybrid*, www.dotnetpro.de/SL2411WisejMAUI2

[5] *Einführung in Wisej.NET Hybrid*,
<https://docs.wisej.com/hybrid>

[6] *Features in Wisej.NET Hybrid*,
www.dotnetpro.de/SL2411WisejMAUI3



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

A2411WisejMAUI

DEV ACADEMY

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

3 TAGE
02.-04.12.2024



Architektur für .NET-Anwendungen

TRAINING!

Was wird behandelt

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



David Tielke
Trainer

MICROSOFT TEAMS IN EIGENE ANWENDUNGEN INTEGRIEREN

Teams als Kommunikationsservice

Microsoft Teams lässt sich umfassend erweitern. Über Schnittstellen ist auch die Integration in eigene Anwendungen möglich. Wie das über das Graph API funktioniert, lesen Sie hier.

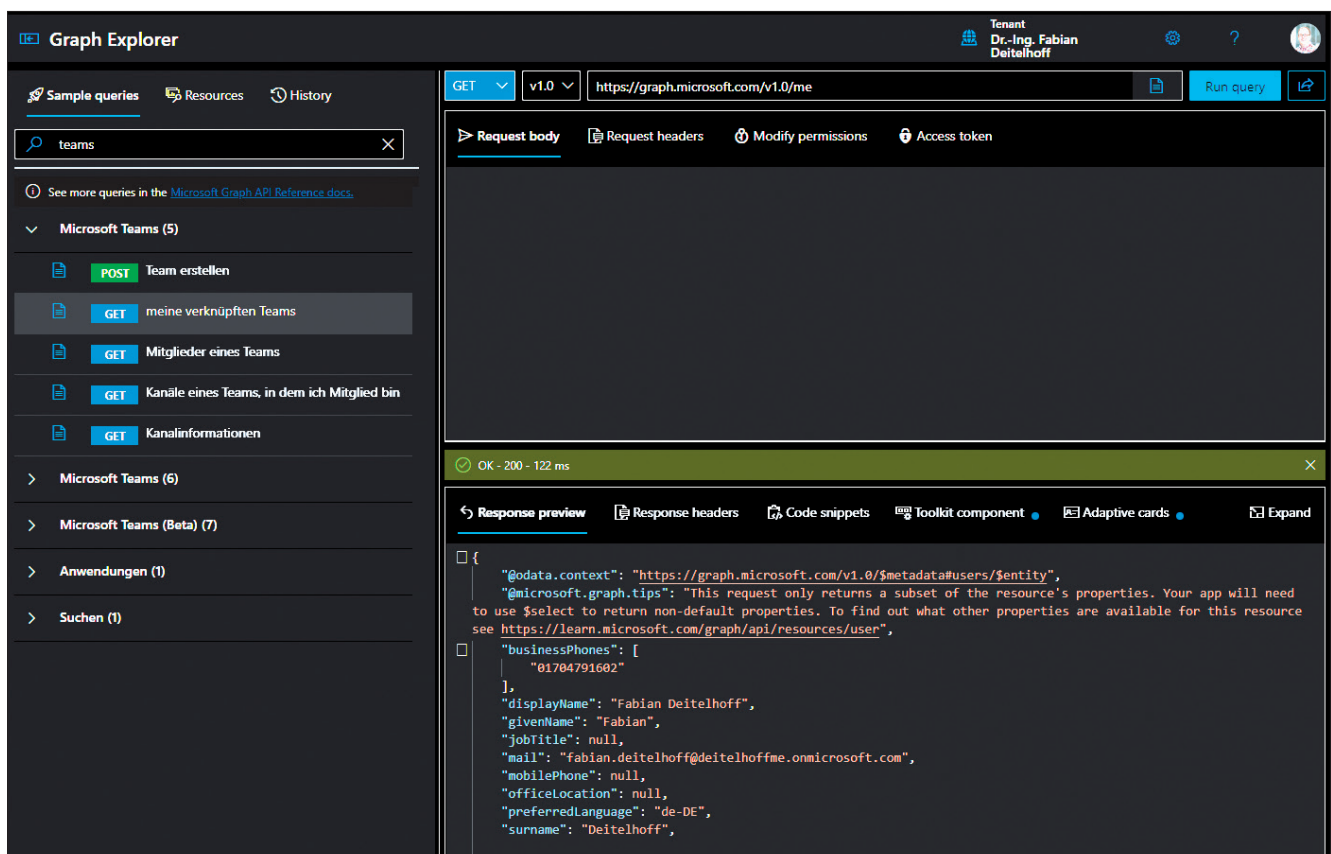
Die Kommunikation in Unternehmen erfolgt zunehmend digital. Dadurch kommt effizienten Kommunikations- und Kollaborationslösungen entscheidende Bedeutung zu. Mit der verstärkten Nutzung von Remote-Arbeit ist der Bedarf an leistungsfähigen Tools weiter gestiegen, um Teamarbeit und Kommunikation nahtlos zu unterstützen.

Eines der bekanntesten Werkzeuge ist Microsoft Teams, das als Teil von Microsoft 365 zur Verfügung steht und Chat, Videoanrufe, Dateiablage sowie die Integration von Anwendungen unter einer Oberfläche kombiniert.

Microsoft Teams bietet vielfältige Möglichkeiten, die Plattform in bestehende Arbeitsprozesse und Anwendungen zu integrieren. Eine der wichtigsten ist die Integration mit der

Microsoft-365-Suite, die den nahtlosen Zugriff auf Tools wie Outlook, SharePoint und OneDrive ermöglicht. Unterstützt wird auch eine breite Palette von Drittanbieter-Apps und -Diensten, von Projektmanagement-Tools über CRM-Systeme bis hin zu IT-Support-Lösungen. Benutzer können so ihre bevorzugten Anwendungen direkt in Teams verwenden. Darüber hinaus lassen sich mithilfe der Microsoft-Teams-Developer-Plattform benutzerdefinierte Apps und Bots erstellen, die spezifische Geschäftsanforderungen erfüllen (siehe [1]).

Häufig besteht jedoch der Wunsch, auf Daten von Microsoft Teams direkt zuzugreifen, ohne erst eine eigene App zu erstellen. Dieser Artikel beschreibt die Integration von Teams-Daten in eine eigene Anwendung – und nicht andersherum.



Der Graph Explorer zum Testen von Graph-API-Anfragen (Bild 1)

Microsoft Entra Admin Center | Nach Ressourcen, Diensten und Dokumenten suchen (G+)

Home > App-Registrierungen > **Anwendung registrieren**

*** Name**
Der dem Benutzer gezeigte Anzeigename für diese Anwendung. (Dieser kann später geändert werden.)
dotnetpro-tests ✓

Unterstützte Kontotypen
Wer kann diese Anwendung verwenden oder auf diese API zugreifen?
☒ Nur Konten in diesem Organisationsverzeichnis (nur "Dr.-Ing. Fabian Deitelhoff" – einzelner Mandant)
☐ Konten in einem beliebigen Organisationsverzeichnis (beliebiger Microsoft Entra ID-Mandant – mandantenfähig)
☐ Konten in einem beliebigen Organisationsverzeichnis (beliebiger Microsoft Entra ID-Mandant – mandantenfähig) und persönliche Microsoft-Konten (z. B. Skype, Xbox)
☐ Nur persönliche Microsoft-Konten
[Entscheidungshilfe...](#)

Umleitungs-URI (optional)
Die Authentifizierungsantwort wird nach erfolgreicher Authentifizierung des Benutzers an diesen URI zurückgegeben. Die Angabe ist zum jetzigen Zeitpunkt optional und kann später geändert werden. Für die meisten Authentifizierungsszenarien ist jedoch ein Wert erforderlich.
 Plattform auswählen:

Registrieren Sie eine App, an der Sie gerade arbeiten. Integrieren Sie Katalog-Apps und andere Apps von außerhalb Ihrer Organisation, indem Sie sie aus [Unternehmensanwendungen](#) hinzufügen.

Indem Sie den Vorgang fortsetzen, stimmen Sie den Microsoft-Plattformrichtlinien zu. [G*](#)

Registrieren

Die Registrierung einer neuen App im Admin-Center (Bild 2)

Kommunikationslösungen in eigenen Anwendungen

Die Integration von Kommunikationslösungen wie Microsoft Teams in eigene Anwendungen bietet zahlreiche Vorteile.

Statt zwischen verschiedenen Anwendungen wechseln zu müssen, können Benutzer innerhalb ihrer gewohnten Arbeitsumgebung auf Kommunikationsfunktionen zugreifen. Durch Einbindung von Teams-Funktionen wie Chat und Anrufe und gemeinsame Dokumentenbearbeitung können Teams zudem effektiver kommunizieren und zusammenarbeiten.

Darüber hinaus erhöht die Integration die Transparenz und Nachvollziehbarkeit von Kommunikationsprozessen, da alle Interaktionen und Dokumentationen in einem einheitlichen System stattfinden. Und schließlich ermöglicht sie eine bessere Anpassung an spezifische Geschäftsanforderungen. Unternehmen können maßgeschneiderte Lösungen entwickeln, die genau auf ihre Bedürfnisse zugeschnitten sind.

Microsoft Teams und das Graph API

Für eine nahtlose und effiziente Integration von Teams in eigene Anwendungen existieren verschiedene APIs und SDKs, die Microsoft Teams bietet, sowie das Microsoft Graph API, das als zentraler Aspekt dieser Integration dient.

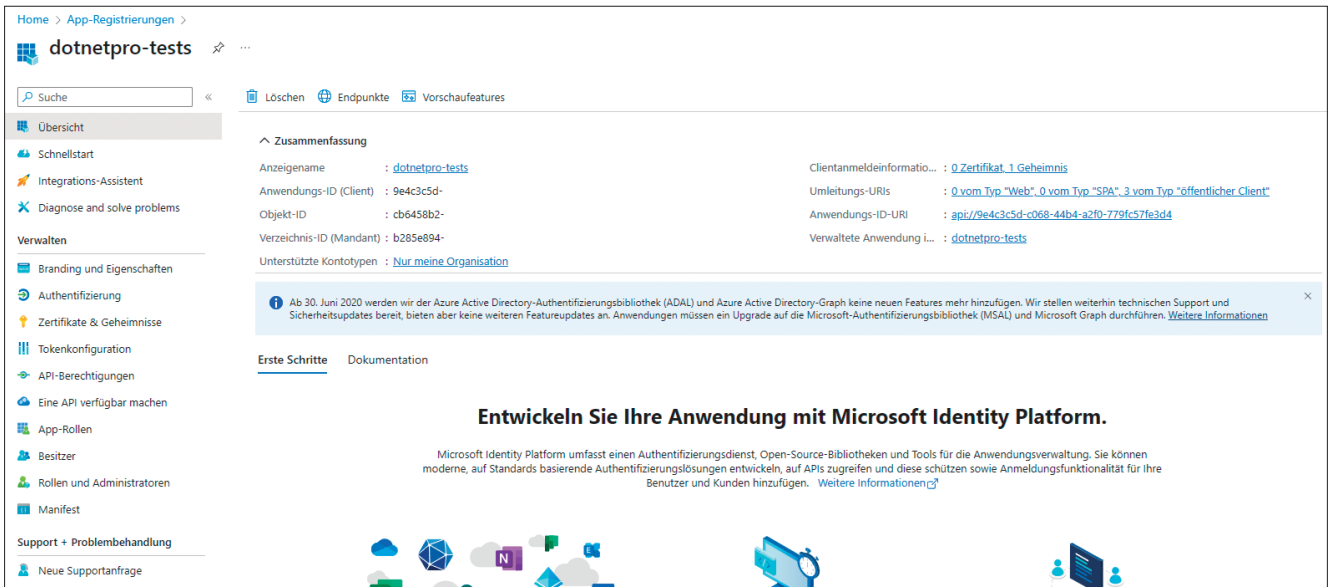
Das Graph API für Microsoft Teams umfasst zahlreiche Funktionen zur Integration und Verwaltung von Teams-Features. Zu den wichtigsten gehören das Erstellen und Verwalten von Teams, Gruppen und Kanälen, das Hinzufügen von Tabs und das Verwalten von Apps im Teams-App-Katalog. Auch das Erstellen von Kanälen und Chats zum Senden und

Empfangen von Nachrichten sowie die Nutzung von Tags zur Klassifizierung von Benutzern oder Gruppen sind möglich. Darüber hinaus lassen sich Anrufe und Online-Meetings erstellen und empfangen. Zu guter Letzt ist die Integration von Arbeitszeitplänen und Zeiterfassung möglich.

Einschränkungen gibt es allerdings bei den Polling-Anforderungen und den Leistungsgrenzen für Teams. Wenn eine Anwendung abfragt, ob sich eine Ressource geändert hat, ist das nur einmal pro Tag möglich. Muss eine Anwendung häufiger über Änderungen informiert werden, ist ein Abonnement für diese Ressource sinnvoller, um Änderungsbenachrichtigungen über Webhooks zu erhalten. Eine dedizierte Website [2] von Microsoft listet alle Möglichkeiten und Einschränkungen auf.

Das Microsoft Teams API, das über das Microsoft Graph API zugänglich ist, ist ein zentrales Werkzeug für Integrationen. Es bietet eine einheitliche Programmierschnittstelle, die den Zugriff auf eine Vielzahl von Microsoft-365-Diensten und -Daten ermöglicht, einschließlich Microsoft Teams. Das Graph API ist REST-basiert und erlaubt es, über HTTP-Anfragen auf Daten zuzugreifen und Aktionen auszuführen.

Bevor auf das API zugegriffen werden kann, müssen Anwendungen authentifiziert und autorisiert werden. Dies erfolgt in der Regel über OAuth 2.0, wobei die Microsoft Identity Platform verwendet wird, um Tokens zu generieren, die den Zugriff auf das API gewähren. Diese Tokens müssen bei jeder API-Anfrage im Header übermittelt werden, um sicherzustellen, dass nur autorisierte Anwendungen Zugriff auf die Daten haben. ►



Die Daten der erstellten App (Bild 3)

Das Graph SDK und der Graph Explorer

Das Microsoft Graph SDK [3] ist ein wichtiges Toolkit, um auf die vielfältigen Microsoft-365-Dienste zuzugreifen. Damit stehen Ressourcen wie Benutzerinformationen, Dateien, E-Mails, Kalender, Aufgaben, Teams-Informationen und viele weitere zur Verfügung. Das SDK vereinfacht den Zugriff auf dieses API, indem es vorgefertigte Methoden und Klassen bietet, welche die Interaktion mit den verschiedenen Endpunkten erleichtern. Im Beispielprojekt ist das Paket Microsoft.Graph installiert, um das Graph SDK zu nutzen.

In diesem Kontext stellt der Microsoft Graph Explorer ein nützliches, webbasiertes Tool dar, das es ermöglicht, das Graph API direkt im Browser zu testen. Der Graph Explorer bietet eine intuitive Benutzeroberfläche, mit der Benutzer API-Abfragen erstellen und ausführen können, ohne Code schreiben zu müssen (Bild 1). Er zeigt die Ergebnisse der API-Aufrufe in Echtzeit an und unterstützt die Authentifizierung, sodass Benutzer mit ihren eigenen Daten arbeiten können. Dadurch ist der Explorer sinnvoll, um die verfügbaren Endpunkte und deren Funktionalitäten kennenzulernen.

Weitere Pakete

Die Microsoft Authentication Library (MSAL) [4] ist eine Bibliothek für die Authentifizierung und den Erhalt von Zugriffstokens, um sicher auf Microsoft Graph und andere geschützte Ressourcen zuzugreifen. MSAL unterstützt mehrere Authentifizierungsflüsse, einschließlich interaktiver, stiller und integrierter Authentifizierung, sowie Authentifizierungsmethoden wie OAuth 2.0 und OpenID Connect.

MSAL ist für eine Reihe unterschiedlicher Programmiersprachen und Plattformen verfügbar, einschließlich .NET, Java, JavaScript und Python. Es vereinfacht die Implementierung von Authentifizierung und Autorisierung in Anwendungen und bietet Funktionen wie Token-Caching, Verwaltung von Benutzerkonten und Unterstützung für Multi-Factor Authentication (MFA).

Die Beispiele des Artikels nutzen MSAL, um ein Token für den Zugriff auf das Graph API zu erhalten. Im Beispielprojekt ist dazu das Paket Microsoft.Identity.Client installiert. Ein Access-Token lässt sich gut über die Website jwt.io [5] anschauen, um zu prüfen, ob alles in Ordnung ist.

● Listing 1: Authentifizierung, um ein Access-Token zu erhalten

```
var clientId = "...";
var tenantId = "...";
var clientSecret = "...";
var scopes = new[] { "User.Read",
    "TeamSettings.ReadWrite.All", "Chat.ReadWrite" };

var app = ConfidentialClientApplicationBuilder.
    Create(clientId)
    .WithClientSecret(clientSecret)
    .WithAuthority(new Uri(
        $"https://login.microsoftonline.com/{tenantId}"))
    .Build();

var result = await app.AcquireTokenForClient(scopes).
    ExecuteAsync();

Console.WriteLine(result.AccessToken);

var accessToken = result.AccessToken;
```

Das Paket Azure.Identity ist eine Bibliothek, die in Azure-SDKs verwendet wird, um die Authentifizierung in Azure-Diensten zu erleichtern. Es bietet eine einheitliche Schnittstelle für die Authentifizierung und unterstützt verschiedene Authentifizierungsmechanismen wie Managed Identity, Shared Token Cache und interaktive Browser-Authentifizierung. Azure.Identity ist besonders nützlich für Anwendungen, die in Azure ausgeführt werden und auf verschiedene Azure-Ressourcen zugreifen müssen. Das Beispielpaket nutzt die Bibliothek für einige Klassen und Typen, die dort definiert sind.

Die Kombination aus Microsoft Graph SDK, Graph Explorer, MSAL und den Paketen Azure.Identity, Microsoft.Graph und Microsoft.Identity.Client erlaubt es, flexibel auf die vielfältigen Dienste und Daten von Microsoft 365 zuzugreifen.

Eine App registrieren

Damit eine App Zugriff auf die angeforderten Daten hat, müssen wir für diese App einen Eintrag im Microsoft Entra Admin Center [6] erstellen und sie damit registrieren. Der Menüpunkt dafür ist *Anwendungen | App-Registrierungen*. Sofern es schon registrierte Anwendungen gibt, werden sie hier alle aufgelistet. Zusätzlich besteht die Möglichkeit, neue Apps zu registrieren, was **Bild 2** verdeutlicht. Neben einem beschreibenden Namen ist lediglich die Option wichtig, wer Zugriff auf diese App haben soll.

Erweiterte Einstellungen

Öffentliche Clientflows zulassen

Folgende Mobilgerät- und Desktopflows aktivieren:

- App erfasst Klartextkennwort (Flow für Kennwortanmeldeinformationen des Ressourcenbesitzers) [Weitere Informationen](#)
- Keine Tastatur (Gerätecodeflow) [Weitere Informationen](#)
- SSO für in die Domäne eingebundene Windows-Geräte (in Windows integrierter Authentifizierungsflow) [Weitere Informationen](#)

Die erweiterten Einstellungen einer App (Bild 4)

Bild 3 zeigt die Beispiel-App, die für diesen Artikel erstellt und konfiguriert ist. Der Screenshot zeigt zudem, woher die IDs kommen, die zuvor für die Authentifizierung notwendig waren. Im Menüpunkt *Zertifikate & Geheimnisse* kann ein App-Secret erstellt werden, das ebenfalls zuvor bei der Authentifizierung erforderlich war. Wichtig ist, die Option der öffentlichen Clientflows zuzulassen, wie **Bild 4** zeigt. Steht diese Option auf *Nein*, erzeugt der Code für den Zugriff auf das Graph API einen Fehler mit dem Hinweis, dass die Konfiguration nicht in Ordnung ist und einen Fehler aufweist.

Sehr wichtig ist zudem noch, der App Zugriff auf die relevanten Bereiche von Microsoft Graph zu gewähren. Das ist im Menüpunkt *API-Berechtigungen* möglich. **Bild 5** zeigt dazu ein Beispiel. Zugriff besteht zum Beispiel auf die Chats, auf Teams und die allgemeinen Benutzerinformationen.

Insgesamt sind das etliche Einstellungen und Optionen, die an verschiedenen Stellen zu berücksichtigen sind. Ande- ►

dotnetpro-tests | API-Berechtigungen

Suche

Aktualisieren

Haben Sie Feedback für uns?

Übersicht

Schnellstart

Integrations-Assistent

Diagnose and solve problems

Verwalten

Branding und Eigenschaften

Authentifizierung

Zertifikate & Geheimnisse

Tokenkonfiguration

API-Berechtigungen

Eine API verfügbar machen

App-Rollen

Besitzer

Rollen und Administratoren

Manifest

Support + Problembehandlung

Neue Supportanfrage

Konfigurierte Berechtigungen

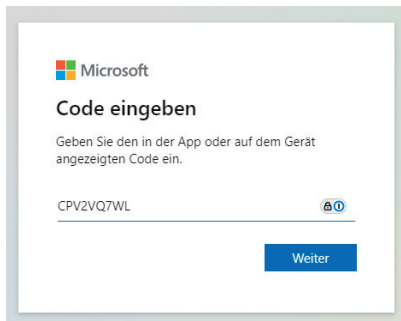
Anwendungen sind zum Aufruf von APIs autorisiert, wenn ihnen im Rahmen des Zustimmungsprozesses Berechtigungen von Benutzern/Administratoren erteilt werden. Die Liste der konfigurierten Berechtigungen muss alle Berechtigungen enthalten, die die Anwendung benötigt. [Weitere Informationen zu Berechtigungen und Zustimmung](#)

+ Berechtigung hinzufügen

✓ Administratorzustimmung für "Dr.-Ing. Fabian Deitelhoff" erteilen

API/Berechtigungsname	Typ	Beschreibung	Administratoreinwill...	Status
Microsoft Graph (33)				
Chat.Create	Anwendung	Create chats	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.ManageDeletion.All	Anwendung	Delete and recover deleted chats	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.Read.All	Anwendung	Read all chat messages	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.Read.WhereInstalled	Anwendung	Read all chat messages for chats where the associated Tea...	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.ReadBasic.All	Anwendung	Read names and members of all chat threads	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.ReadBasic.WhereInstalled	Anwendung	Read names and members of all chat threads where the as...	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.ReadWrite.All	Anwendung	Read and write all chat messages	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.ReadWrite.WhereInstalled	Anwendung	Read and write all chat messages for chats where the asso...	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
Chat.UpdatePolicyViolation.A	Anwendung	Flag chat messages for violating policy	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
ChatMember.Read.All	Anwendung	Read the members of all chats	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
ChatMember.Read.WhereInsta	Anwendung	Read the members of all chats where the associated Team...	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
ChatMember.ReadWrite.All	Anwendung	Add and remove members from all chats	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
ChatMember.ReadWrite.Whe	Anwendung	Add and remove members from all chats where the associ...	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
ChatMessage.Read.All	Anwendung	Read all chat messages	Ja	✓ Gewährt für "Dr.-Ing. Fa..."
offline_access	Delegiert	Zugriff auf Daten beibehalten, für die Sie Zugriff erteilt ha...	Nein	✓ Gewährt für "Dr.-Ing. Fa..."
openid	Delegiert	Benutzer anmelden	Nein	✓ Gewährt für "Dr.-Ing. Fa..."
profile	Delegiert	Grundlegendes Profil von Benutzern anzeigen	Nein	✓ Gewährt für "Dr.-Ing. Fa..."

Die wichtigen API-Berechtigungen, damit Zugriff über die Schnittstelle besteht (Bild 5)



Eingabe eines Codes zur Authentifizierung (Bild 6)

rerseits ist diese Konfiguration nur einmal initial zu tätigen und aus Sicherheitsgründen nachvollziehbar, da über das Graph API von Microsoft Zugriff auf praktisch alle Daten besteht, die bei Microsoft gespeichert sind.

Authentifizierung und Autorisierung

Es existieren verschiedene Wege, um eine Anwendung bei Microsoft zu authentifizieren und die notwendigen Rechte anzufragen, um auf die Daten des Graph API zugreifen zu können. **Listing 1** zeigt den Weg über den *ConfidentialClientApplicationBuilder* des *Microsoft.Identity.Client*-Pakets. Um ein Access-Token zu erlangen, sind eine Cli-



Der App Zugriffsrechte auf ein Microsoft-Konto erteilen (Bild 7)

ent-ID, eine Tenant-ID, ein Client-Secret und die angefragten Scopes erforderlich. Woher diese Daten kommen, zeigt der Artikel in Kürze. Steht das Access-Token zur Verfügung, lassen sich damit simple HTTP-Anfragen absetzen, um Daten abzufragen oder zu setzen.

Alternativ besteht der Zugriff über das Paket *Microsoft.Identity*. Dazu zeigt **Listing 2** ein Beispiel. Zuerst werden die verschiedenen Optionen, die für die Authentifizierung notwendig sind, in einem *DeviceCodeCredentialOptions*-Objekt zusammengefasst. Neben der Client- und der Tenant-ID ist auch ein Callback anzugeben, der die Aufforderung enthält, sich mit einem Code bei Microsoft zu authentifizieren. Erst dann kann es im Code weitergehen. Mit den Optionen lässt sich ein *DeviceCodeCredential*-Objekt erzeugen, das dann zusammen mit den angefragten Scopes für die Instanzierung eines *GraphServiceClient* genutzt wird. Dadurch steht direkt der Graph-Client aus dem Graph SDK zur Verfügung, der in den folgenden Beispielen zum Einsatz kommt.

Beim Ausführen erscheint nach kurzer Zeit auf der Konsole folgende Ausgabe, erzeugt von dem Callback, den wir im vorherigen Beispiel implementiert haben:

```
To sign in, use a web browser to open the page
https://microsoft.com/devicelogin and enter the
code LZUWG6GCU to authenticate.
```

Einen Besuch der Website zeigt **Bild 6**. Nach Eingabe des Codes ist das Konto auszuwählen, das für die Authentifizierung und Verknüpfung der App zum Einsatz kommen soll.

Anschließend müssen die Zugriffsrechte (Scopes) der App bestätigt werden, wodurch die App dann Daten abfragen und schreiben kann. **Bild 7** zeigt dazu ein Beispiel mit einigen Rechten zum allgemeinen Benutzer und zu Teams-Einstellungen und Chats. Nach der Bestätigung auf der Website läuft der Code automatisch weiter. Passen die IDs und der Scope zu den angefragten Daten, führt der Code die Operationen aus. Welche das sind, zeigen die nachfolgenden Beispiele.

Verschiedene Zugriffsarten

Microsoft Identity bietet verschiedene Zugriffsszenarien, die sicherstellen, dass Anwendungen auf die benötigten Ressourcen zugreifen können, ohne die Sicherheit und Datenschutzanforderungen zu gefährden. Die beiden Hauptszenarien sind Delegated Access und App-only Access.

Delegated Access kommt zum Einsatz, wenn eine Anwendung im Namen eines Benutzers auf Ressourcen zugreift. In diesem Szenario authentifiziert sich der Benutzer bei der An-

● Listing 2: Authentifizierung über Azure Identity

```
var options = new DeviceCodeCredentialOptions
{
    AuthorityHost =
        AzureAuthorityHosts.AzurePublicCloud,
    ClientId = clientId,
    TenantId = tenantId,

    // Callback function that receives the user prompt
    DeviceCodeCallback = (code, cancellation) =>
    {
        Console.WriteLine(code.Message);
        return Task.FromResult(0);
    },
};

var deviceCodeCredential =
    new DeviceCodeCredential(options);

var graphClient = new GraphServiceClient(
    deviceCodeCredential, scopes);
```

wendung, und die Anwendung erhält daraufhin ein Zugriffstoken, das die Berechtigungen des Benutzers widerspiegelt. Dieses Zugriffstoken wird dann verwendet, um API-Aufrufe an Microsoft Graph oder andere geschützte Ressourcen zu senden. Ein typisches Beispiel für Delegated Access ist eine Webanwendung, die Zugriff auf die E-Mails und den Kalender eines Benutzers benötigt. Nachdem sich der Benutzer authentifiziert hat, kann die Anwendung API-Anfragen im Namen des Benutzers senden und erhält Zugriff auf die Ressourcen, die der Benutzer selbst sehen und bearbeiten kann.

App-only Access wird verwendet, wenn eine Anwendung ohne Benutzerinteraktion auf Ressourcen zugreifen muss. In den Beispielen dieses Artikels kommt diese Variante zum Einsatz. In einem solchen Szenario authentifiziert sich die Anwendung selbst und erhält ein Zugriffstoken, das auf die Anwendungsidentität beschränkt ist. Dieses Zugriffstoken enthält nur diejenigen Berechtigungen, die der Anwendung explizit zugewiesen wurden. Ein typisches Beispiel für App-only Access ist ein Hintergrunddienst, der periodisch Daten aus Microsoft Graph abrufen oder Änderungen an Ressourcen vornimmt, ohne dass ein Benutzer eingreifen muss. Diese Zugriffsart ist besonders nützlich für Anwendungen, die automatisierte Prozesse durchführen, wie etwa das Synchronisieren von Daten oder das Durchführen von Wartungsaufgaben.

Zugriff auf Daten von Benutzer und Teams

Wenn die umfangreichen Aufgabenbereiche Registrierung, Konfiguration und Authentifizierung abgeschlossen sind, lassen sich über den Graph-Client recht einfach Informationen abfragen oder neue Daten erzeugen. Der folgende Code ruft beispielsweise die Daten des Benutzers ab:

```
var user = await graphClient.Me.GetAsync();
```

Der Screenshot des Graph Explorers, der in **Bild 1** dargestellt ist, hat dieses Beispiel bereits gezeigt und preisgegeben, dass die Daten unter folgenden Endpunkt erreichbar sind:

```
https://graph.microsoft.com/v1.0/me
```

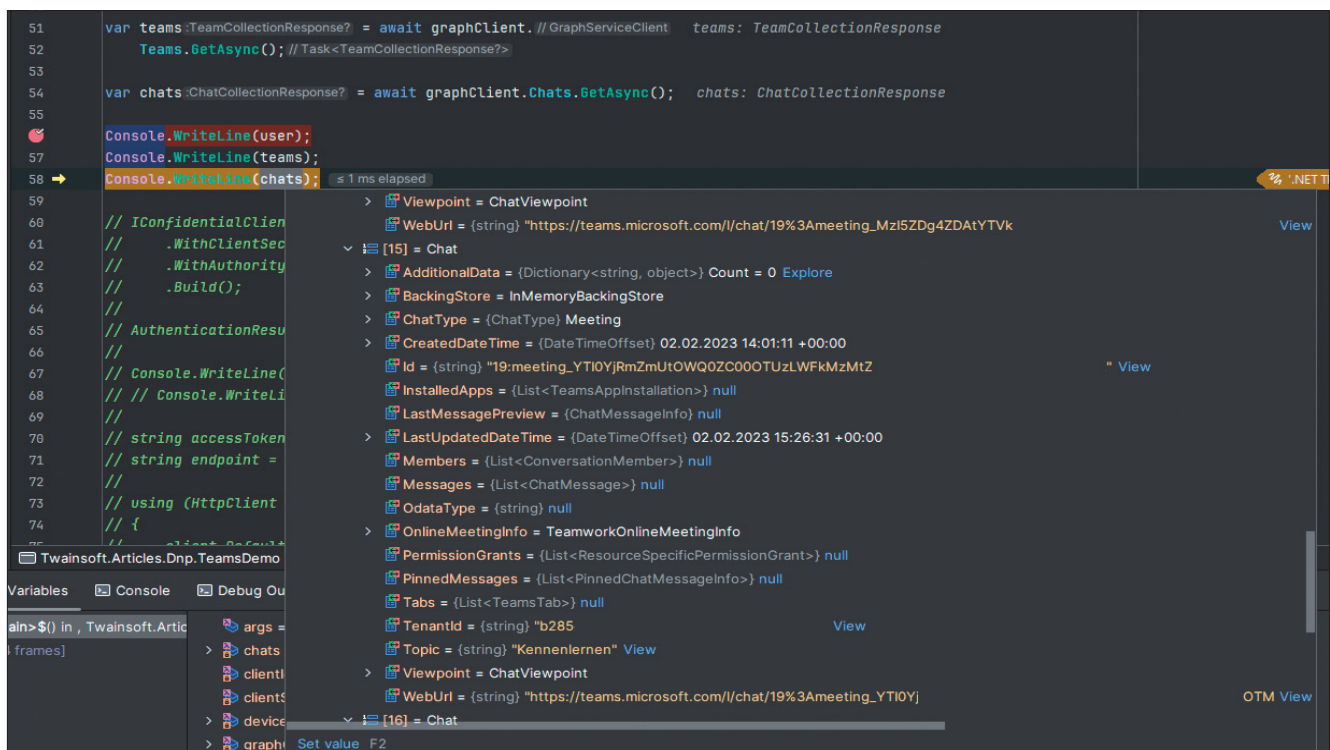
Zurückgeliefert werden Daten zur Kommunikation, Vorname, Nachname, Anzeigenname, präferierte Sprache und dergleichen. Auf diese Weise lassen sich die Daten von Benutzern sehr leicht abfragen, um diese zum Beispiel in der eigenen Anwendung weiterzuverarbeiten.

Sind die Daten aller Benutzer erforderlich, hilft folgende Abfrage weiter:

```
var users = await graphClient.Users.GetAsync();
```

Die Schnittstelle erlaubt es darüber hinaus auch, nur bestimmte Daten abzufragen. Das ist in vielen Fällen sinnvoll, da das Graph API eine ganze Palette an Informationen bereithält und die zurückgelieferte Datenmenge umfangreich sein kann. Dazu ein Beispiel für die eigenen Benutzerdaten:

```
var user = await graphClient
    .Me
    .GetAsync(requestConfiguration =>
    {
        requestConfiguration.QueryParameters.Select =
            ["displayName", "jobTitle"];
    });
```



Ausschnitt aus den Daten eines Chats (Bild 8)

● Listing 3: Erstellen eines neuen Teams über das API

```
using Microsoft.Graph.Models;

var requestBody = new Team
{
    DisplayName = "Neues Team",
    Description = "Beschreibung des neuen Teams",
    AdditionalData = new Dictionary<string, object>
    {
        {
            "template@odata.bind", "https://graph.microsoft.
            com/v1.0/teamsTemplates('standard')"
        },
    },
};

var result = await graphClient.Teams.
PostAsync(requestBody);
```

Die Anweisung fragt lediglich die Daten zum Anzeigenamen und der Berufsbezeichnung ab, was die übertragene Datenmenge deutlich reduziert.

Darüber hinaus lassen sich Daten von Microsoft Teams abfragen, um eine Integration der Kommunikationslösung in eigene Anwendungen umzusetzen. Ein Aufruf von

```
var teams = await graphClient.Teams.GetAsync();
```

fragt allgemeine Daten zu Microsoft Teams ab. Die Informationen zu Microsoft Teams und den Chats stehen dabei unter unterschiedlichen Endpunkten zur Verfügung.

Ein Aufruf von

```
var chats = await graphClient.Chats.GetAsync();
```

ruft alle Chats ab, in denen sich der angemeldete Benutzer befindet. Wie eine solche Antwort aussieht, zeigt Ihnen der Screenshot in **Bild 8**. Dort sind die Daten eines Chats zu sehen.

Über die Schnittstelle lassen sich auch neue Teams erstellen. Ein Beispiel dazu sehen Sie in **Listing 3**. Einige Daten für das neue Team sind ebenso erforderlich wie die Angabe eines Templates, aus dem das neue Team erzeugt wird.

Arbeiten mit Kanälen und Chats

Neben den allgemeinen Benutzerinformationen und Teams-Einstellungen besteht über das Graph API Zugriff auf Chats und Chatnachrichten in Microsoft Teams. Zum Teil befinden sich die Daten unterhalb der *Chat*-Eigenschaft, andere lassen sich über die bereits genutzte *Teams*-Eigenschaft nutzen.

Listing 4 startet damit, zu zeigen, wie sich ein neuer Kanal erstellen lässt. Im Body der Anfrage sind der neue Name des Kanals und seine Beschreibung zu sehen. Die Anfrage wird über die *Channels*-Eigenschaft abgesetzt, die sich pro Team

● Listing 4: Erstellen eines neuen Kanals über das API

```
using Microsoft.Graph.Models;

var requestBody = new Channel
{
    DisplayName = "Neuer Kanal",
    Description = "Neue Beschreibung für den neuen
    Kanal",
};

var result = await graphClient.Teams["{team-id}"].
Channels.PostAsync(requestBody);
```

in Microsoft Teams aufteilt. Daher ist auch die Team-ID anzugeben, unterhalb der sich der neue Kanal einsortieren soll.

Das funktioniert auch bei Chats. **Listing 5** zeigt, wie sich ein neuer 1:1-Chat erstellen lässt. Dazu sind, neben dem Chat-Typ, die Mitglieder des 1:1-Chats einzugeben. Das müssen für diesen Typ zwei sein. Die User-ID lässt sich beispielsweise über die Abfrage der Benutzer ermitteln, wie weiter oben gezeigt. Diese Abfrage ist bei der *Chat*-Eigenschaft des Graph API abzusetzen, da es sich um einen Chat handelt.

Zudem lassen sich Chatnachrichten posten. Dazu zeigt **Listing 6** ein Beispiel. Da eine Nachricht erstellt werden soll, ist bei der Abfrage eine Chat-ID anzugeben. Dort gibt es eine *Messages*-Eigenschaft, um dort neue Nachrichten posten zu können. Nach der Abfrage wird die neue Nachricht erstellt und allen angezeigt, die im Chat verfügbar sind.

Fehlerbehebungen

Bei der Arbeit mit dem Graph API können viele Fehler auftreten. „Klassiker“ wie die Autorisierungsfehler *401 Unauthorized* oder *403 Forbidden* lassen sich meist schnell beheben. Häufige Ursachen dafür sind ungültige Zugriffstoken. Dabei ist sicherzustellen, dass das Zugriffstoken gültig, nicht abgelaufen und im Header korrekt angegeben ist.

Auch falsche Berechtigungen führen häufig zu Fehlern. Dabei ist zu überprüfen, ob die angeforderten Berechtigungen den benötigten API-Aufrufen entsprechen und ordnungsgemäß erteilt wurden. Einige Aktionen erfordern zudem im Admin-Center von Microsoft die Zustimmung eines Administrators – unabhängig davon, welche Berechtigungen eine App hat und welcher Scope angefragt wurde. In diesem Kontext ist auch sicherzustellen, dass das Zugriffstoken zum richtigen API passt, da es ansonsten zu API-Ressourcenfehlern kommt.

Bei delegierten Codeflüssen prüft Microsoft Graph zusätzlich, ob die Anforderung auf der Grundlage der in der Anwendung gewährten Berechtigungen und der Berechtigungen des angemeldeten Benutzers zulässig ist. Im Allgemeinen zeigt dieser Fehler an, dass der Benutzer nicht über die erforderlichen Berechtigungen verfügt, um die Anforderung auszuführen, oder dass der Benutzer nicht für diejenigen Daten lizenziert ist, auf die er zugreift.

Listing 5: Erstellen eines neuen 1:1-Chats über die Schnittstelle

```
var requestBody = new Chat
{
    ChatType = ChatType.OneOnOne,
    Members = new List<ConversationMember>
    {
        new AadUserConversationMember
        {
            OdataType =
                "#microsoft.graph.aadUserConversationMember",
            Roles = new List<string>
            {
                "owner",
            },
            AdditionalData = new Dictionary<string, object>
            {
                {
                    "user@odata.bind" , "https://graph.
                        microsoft.com/v1.0/users('{user-id}')"
                },
            },
        },
    },
};

new AadUserConversationMember
{
    OdataType = "#microsoft.graph.
        aadUserConversationMember",
    Roles = new List<string>
    {
        "owner",
    },
    AdditionalData = new Dictionary<string, object>
    {
        {
            "user@odata.bind" , "https://graph.
                microsoft.com/v1.0/users('{user-id}')"
        },
    },
};

var result = await graphClient.Chats.PostAsync(
    requestBody);
```

Listing 6: Posten einer neuen Nachricht in einem Chat

```
var requestBody = new ChatMessage
{
    Body = new ItemBody
    {
        Content = "Hallo Welt im Chat!",
    },
};

var result = await graphClient.Chats["{chat-id}"].
    Messages.PostAsync(requestBody);
```

Fazit und Einschätzung zur Teams-Integration

Die Integration von Microsoft Teams in eigene Anwendungen mithilfe des Microsoft Graph API bietet umfassende Möglichkeiten, die Zusammenarbeit und Kommunikation innerhalb von Unternehmen zu verbessern. Eine direkte Integration, beispielsweise als Host in einer eigenen Anwendung, ist aktuell allerdings nicht möglich.

Die Integration über das API ist auf der einen Seite etwas aufwendiger, aber auch feingranularer, sodass genau diejenigen Daten und Operationen von Microsoft Teams zum Einsatz kommen können, die im konkreten Anwendungsfall notwendig sind. Zudem bieten es die Schnittstellen an, Teams aus administrativer Sicht zu steuern, auch ohne den Zugriff auf Chats und Chatnachrichten.

Insgesamt ist der Zugriff über das Graph API simpel, wenn die vielen Schritte zur App-Registrierung, Konfiguration und Authentifizierung erst einmal durchgeführt sind. Über die Schnittstelle stehen anschließend viele Interaktionsmöglichkeiten mit Teams und den hinterlegten Daten zur Verfügung, um diese in eigene Anwendungen zu integrieren. ■

[1] Fabian Deitelhoff, *Helferlein in Microsoft Teams*, dotnetpro 10/2024, Seite 100 ff., www.dotnetpro.de/A2410Teams

[2] *Möglichkeiten und Einschränkungen von Microsoft Teams und dem Graph API*, www.dotnetpro.de/SL2411Teams1

[3] *Das Repository zum Graph SDK von Microsoft*, <https://github.com/microsoftgraph/msgraph-sdk-dotnet>

[4] *Wissenswertes zur Microsoft Authentication Library (MSAL)*, www.dotnetpro.de/SL2411Teams2

[5] *Website zur Token-Analyse*, <https://jwt.io>

[6] *Microsoft Entra Admin Center*, <https://entra.microsoft.com>



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.

@FDeitelhoff

dnpCode

A2411Teams

ARBEITSMARKT

TRENDS UND JOBS FÜR ENTWICKLER

Arbeitsplatz 2.0

Kompetenzen im Fokus

Der E-Learning-Anbieter Udemy hat untersucht, wie Unternehmen den Übergang von der traditionellen Personalsuche zu einer „kompetenzbasierten Organisation“ bewältigen. Dabei werden alle Mitarbeiter als Individuen mit einer Reihe von Fähigkeiten betrachtet und nicht auf die Anforderungen eines Stellenprofils reduziert. Dieses Umdenken soll es Unternehmen erlauben, Kompetenzen schneller und strategischer projekt- und abteilungsübergreifend einzusetzen und besser auf die Bedürfnisse ihrer Mitarbeiter einzugehen.

Der Bericht „Arbeitsplatz 2.0: Das Potenzial des Skill-orientierten Unternehmens“ zeigt, dass viele Unternehmen bereits mit dem Übergang zu einer kompetenzbasierten Organisation begonnen haben. 84 Prozent der Befragten gaben an, dass ihr Unternehmen zumin-

dest in Erwägung gezogen hat, im nächsten Jahr kompetenzbasierte Prozesse einzuführen. Bei 75 Prozent wurde bereits mindestens ein kompetenzbasierter Personalprozess eingeführt (Bild 1).

Für die Umsetzung von Skill-orientierten Implementierungsstrategien sind viele verschiedene Konzepte in Umlauf. Die am weitesten verbreitete Strategie konzentriert sich darauf, allen Mitarbeitern Zugang zu externen Ressourcen und Kursen für den Aufbau ihrer Skills bereitzustellen und ihnen damit Tools an die Hand zu geben, mit denen sie sich weiterbilden und neue Skills lernen können. Dicht darauf folgt die Strategie, Entscheidungen in Bezug auf Einstellungen, Beförderungen oder Projektaufträge anhand von Skills-Beurteilungen zu treffen. Während die Strategie, in Stellenanzeigen keinen Hochschulabschluss mehr zu fordern, häufig als wichtiger Schritt für den Übergang zu einem Skill-orientier-

ten Konzept genannt wird, hat die Umfrage ergeben, dass diese Strategie in der Tat am seltensten zum Einsatz kommt, siehe Bild 2. Ein Grund hierfür könnte darin liegen, dass traditionelle Kriterien wie ein Hochschulabschluss erst dann als überflüssig angesehen werden, wenn bessere Praktiken etabliert sind.

GitHub Developer Survey

Bessere Codequalität

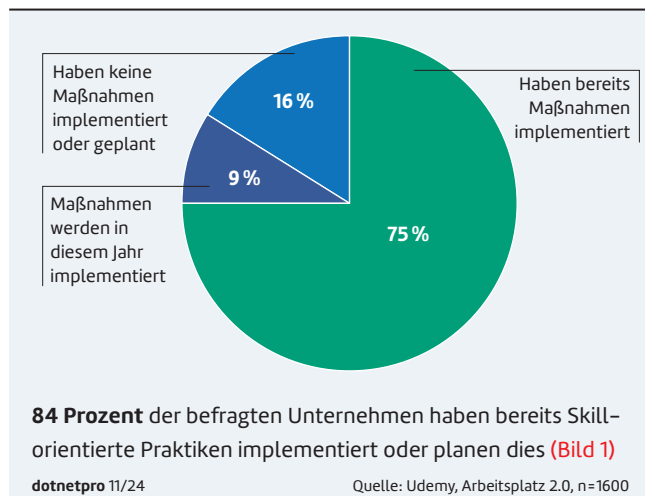
Für seinen neuen Developer Survey hat GitHub rund 2000 Fachleute aus den USA, Brasilien, Deutschland und Indien (je 500 pro Land) in verschiedenen Rollen im Bereich Entwicklung, Ingenieurwesen und Programmierung befragt. Zu den Ergebnissen des Developer Survey gehört, dass generative KI die Softwareentwicklung rasant verändert: Mehr als 97 Prozent der Befragten gaben an, schon einmal AI Coding Tools bei der Arbeit verwendet zu haben – ein Ergebnis, das in allen vier Ländern gleich ist. Allerdings

gab ein geringerer Prozentsatz an, dass ihre Unternehmen die Einführung von KI-Tools zulassen beziehungsweise aktiv fördern. Hier weist Deutschland mit 59 Prozent den niedrigsten Wert auf. Dies zeigt, dass Unternehmen das Interesse ihrer Entwickler an KI-Tools besser unterstützen können.

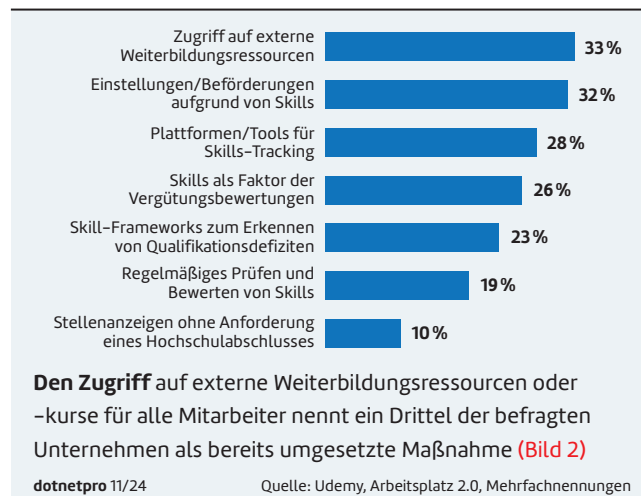
60 Prozent der Befragten in Deutschland berichteten von einer wahrgenommenen Verbesserung der Codequalität durch den Einsatz von KI-Code-Tools. Dies stimmt laut GitHub mit der Forschung überein, die das Unternehmen im vergangenen Jahr gemeinsam mit Accenture durchgeführt hat, um die Auswirkungen von GitHub Copilot auf unternehmensweite Engineering-Teams und deren Einfluss auf die wahrgenommene Codequalität zu analysieren.

Insbesondere von Arbeitssuchenden wird Kompetenz im Umgang mit KI-Code-Tools als großer Vorteil angesehen: 43 Prozent der Befragten in Deutschland sind der Meinung,

● Skill-orientierte Praktiken in Unternehmen



● Skill-orientierte Konzepte (Auszug)



Zahl des Monats

Sich erneut selbstständig machen würden sich **90 Prozent** der rund 3000 von Freelancermap befragten Freiberufler. Während sich **9 Prozent** bei dieser Frage nicht ganz sicher sind, sagt nur **1 Prozent**, dass sie sich nicht noch einmal selbstständig machen würden.

Quelle: Freelancermap, Freelancer-Kompass 2024, n=3042

dass dieses Fachwissen ihre Chancen auf dem Arbeitsmarkt erheblich verbessert.

Übrigens gaben die Befragten am häufigsten an, die durch den Einsatz von KI-Code-Tools eingesparte Zeit für die Systemgestaltung, Zusammenarbeit und Weiterbildung zu nutzen. Konkret nutzten 47 Prozent der Befragten in Deutschland diese zusätzliche Zeit für Zusammenarbeit und Systemdesign.

Freiberufler Selbstständigkeit

Für den Freelancer-Kompass 2024 hat Freelancermap etwas über 3000 Freiberufler befragt, unter anderem danach, aus welchem Grund sich die Befragten selbstständig gemacht haben. An oberster Stelle der Antworten (Bild 3) – mehrere Antworten waren möglich – stehen nicht die besseren Verdienstmöglichkeiten (sie kamen auf 32 Prozent), sondern die Unabhängigkeit mit 68 Prozent. Die meisten Freelancer wollten

ihr eigener Chef sein und flexible Arbeitszeiten haben. Bei den Antworten oben mit dabei waren auch die fachliche Herausforderung und die abwechslungsreiche Tätigkeit als Freiberufler (37 Prozent). Ein knappes Drittel nennt auch die Unzufriedenheit mit dem letzten Unternehmen beziehungsweise dem letzten Vorgesetzten als Grund für den Schritt in die Selbstständigkeit.

Ihre gegenwärtige wirtschaftliche Lage beurteilt mehr als die Hälfte der Befragten als „gut“ oder „sehr gut“ (Bild 4).

Ein knappes Drittel (31 Prozent) bewertet die Lage als „neutral“, und 16 Prozent schätzen die Lage als „schlecht“ oder gar „sehr schlecht“ ein.

Gefragt nach den Zukunftsaussichten erwarten 42 Prozent eine unveränderte Auftragslage, 35 Prozent sagen, sie werde besser werden, und fast ein Viertel der Freiberufler befürchtet eine Verschlechterung.

Der Bruttogewinn der Freiberufler im Jahr 2023 variierte zwischen „unter 25 000 Euro“ (19

Prozent) bis zu „über 125 000“ Euro (17 Prozent). Die große Mehrheit liegt zwischen diesen Extremen. Als durchschnittlichen Nettogewinn aller Befragten nennt der Freelancer-Kompass 62 948 Euro.

Meetings Alle gehen rein, nichts kommt raus

Viele Meetings sind ergebnislos und schlicht Zeitverschwendung, sagt eine neue Studie von Miro. Die Mehrheit der Befragten gibt an, aufgrund von zu vielen Meetings nicht mehr zur eigentlichen Arbeit zu kommen.

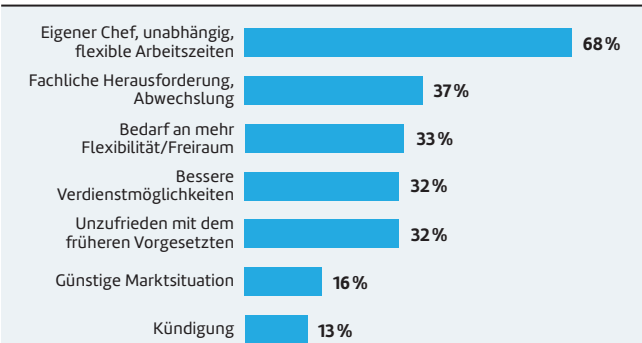
Immerhin sind fast zwei Drittel der Befragten überzeugt, dass Meetings wichtig für eine erfolgreiche Zusammenarbeit sind und dabei helfen, Kontakte zu anderen Teams oder Kollegen zu pflegen (54 Prozent).

Außerdem glaubt fast die Hälfte, dass sich Meetings positiv auf die eigene berufliche Entwicklung auswirken und eine Möglichkeit sind, Beziehungen

zu Führungskräften aufzubauen. Alarmierend ist aber, dass mehr als die Hälfte der Befragten angibt, aufgrund zu vieler Meetings kaum dazu zu kommen, ihre eigentliche Arbeit zu erledigen. Fast drei Viertel kennen Tage mit endlosen Meetings ohne Pausen. Das wirkt sich auf die mentale Gesundheit aus. Fast genauso viele haben sich aufgrund der Meeting-Flut schon einmal ausgebrannt gefühlt. Ein Umstand, der damit zusammenhängen könnte, dass eine deutliche Mehrheit (79 Prozent) auch außerhalb der regulären Arbeitszeiten und fast zwei Drittel gelegentlich auch während des Urlaubs an Meetings teilnehmen.

Bei deutschen Wissensarbeitern scheint außerdem FOMO („Fear of missing out“) ein Problem zu sein: 74 Prozent der Befragten klinken sich in Meetings ein, auch wenn sie nur als optionale Teilnehmer eingeladen sind – so viele wie in keinem anderen der befragten Länder.

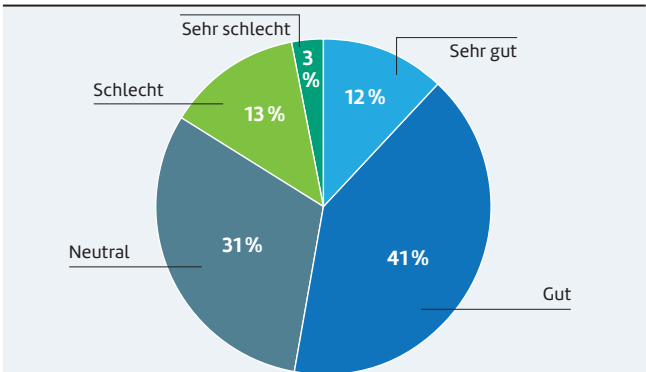
Warum selbstständig?



Die Verdienstmöglichkeiten sind ein wichtiger Punkt, aber weit wichtiger ist die Aussicht, sein eigener Chef zu sein und flexible Arbeitszeiten zu haben (Bild 3)

dotnetpro 11/24 Quelle: Freelancermap, Freelancer-Kompass 24, Mehrfachnennungen

Freelancer: Wirtschaftliche Lage



Mehr als die Hälfte der Befragten bezeichnet ihre wirtschaftliche Lage als „gut“ oder „sehr gut“ (Bild 4)

dotnetpro 11/24 Quelle: Freelancermap, Freelancer-Kompass 24



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 einbuchen unter jobs.developer-media.de



Diese und viele weitere aktuelle Job-Angebote finden Sie auf jobs.developer-media.de

Bergische Universität Wuppertal	Wissenschaftliche*r Mitarbeiter*in IT-Sicherheit	Wuppertal
Ebner Media Group GmbH & Co. KG	Produktmanager Digital (m/w/d)	München
Justus-Liebig Universität Giessen	IT-Mitarbeiter/in (m/w/d) im IT-Support	Gießen
Landesbank Hessen-Thüringen	IT Portfoliomanager (w/m/d)	Offenbach am Main
Johann Wolfgang Goethe-Universität Frankfurt am Main	IT-Projektmitarbeiter*in Campusmanagement (m/w/d)	Frankfurt am Main
SWAGLab GmbH	Mitgestalter (m/w/d)	Hamburg
Bergische Universität Wuppertal	Wissenschaftliche*r Mitarbeiter*in IT-Sicherheit – Projekt Multi-Faktor-Authentifizierung	Wuppertal
Landesbank Hessen-Thüringen	IT-Risikomanager (w/m/d)	Düsseldorf
Bergische Universität Wuppertal	Wissenschaftliche*r Mitarbeiter*in (Doktorand*in) im Bereich Large Language Models	Wuppertal

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**

Bayerstraße 16a, 80335 München

Sitz von Redaktion, Anzeigen, Vertrieb

Bayerstraße 16a, 80335 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, Joachim Pich

Autoren

Elena Bochkor, Tim Borowski, Dr. Fabian Deitelhoff,
Mykola Dobrochynskyy, Christian Giesswein,
Tam Hanna, Christian Havel, André Krämer,
Dr. Veikko Krypczyk, Bernhard Lauer, Sebastian
Meyer, Bernhard Pichler, Dr. Holger Schwichtenberg,
Oliver Sturm, David Tielke, Helmut Vonhoegen,
Christian Wenz, Ralf Westphal, Alexander Zeitler

Art Directorin

Maria-Luise Steinkühler

Gestaltung**EMG Design Unit**

Dagmar Breitenbach, Simone Köhnke,
Petra Reichensperner

Kundenservice / Einzelheftbestellung

EBNER MEDIA GROUP,

Bayerstraße 16a, 80335 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 monatlich.

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 1.1.2019

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 zwölfmal im Jahr.

Das Einzelheft kostet im Inland 19,90 €.

Der Preis für das Premium-Plus-Abonnement

(12 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

Text Control GmbH

www.textcontrol.com

2

Ebner Media Group

www.ebnermedia.de

31, 39-43,

57, 77, 131, 142-143, 147, 148

Autor:in werden!



Schreiben Sie
Artikel für die

dotnetpro



info@dotnetpro.de

Anbietersverzeichnis

Persönliche Beratung bei Fernando Schneider | +49 731 88005-8831 | fernando.schneider@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 12/2024 erscheint am 21. November 2024

Foto: EMI/Genie



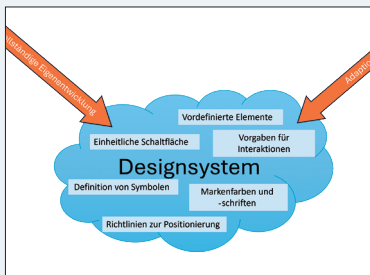
Wie wir Software entwickeln ...

Lange ignoriert, ist das Problem der enormen mentalen Belastung in der Softwareentwicklung inzwischen erkannt. Doch wie geht man als Betroffener damit um? Die **dotnetpro 12/2024** richtet dazu den Blick auf zwei wichtige Aspekte.

Während **Stefan Mintert** Ursachen für die fehlende Zufriedenheit im Job analysiert und Möglichkeiten zur Beseitigung beschreibt, geht **Tom Wendel** der Frage nach, warum Transformationsprojekte selten an der Technik scheitern und weshalb Akzeptanz und Motivation im Team – neben umsichtigem Projektmanagement und einem durchdachten Transformationsplan – Schlüsselfaktoren für den Projekterfolg sind.

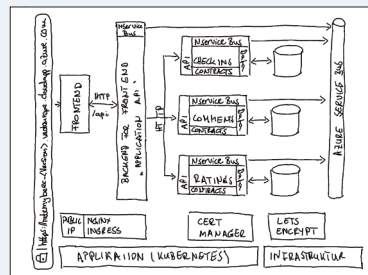
Design mit System

Wie Sie Designsysteme nutzen, um die Qualität der Benutzerschnittstelle signifikant zu steigern.



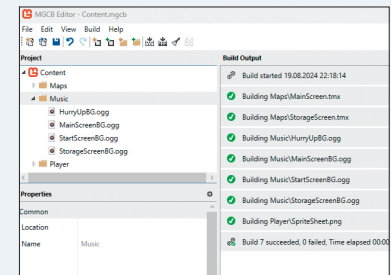
.NET Aspire und Legacy

Aspire verspricht, Entwicklern viel Arbeit abzunehmen. Klappt das aber auch bei betagter Software?



Pimp my (Mono)Game

Erfahren Sie, wie Sie Ihr Spiel mit NPCs bereichern und durch Soundeffekte sowie Musik aufwerten können.



End of file



Computers do not solve problems, they execute solutions.

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>



X: die News im Stream
https://twitter.com/dotnetpro_mag

SAVE THE DATE

DEVELOPER WEEK '25

WIR SEHEN UNS
IM SOMMER 2025!

DWX

developer-week.de | [#DWX25](https://twitter.com/DWX25) | Find us on    

KONFERENZ-
PROGRAMM
IM HEFT!

ab Seite 39



.NET DEVELOPER CONFERENCE '24

DIE KONFERENZ FÜR .NET ENTWICKLER
25. - 28.11.2024 | KÖLN

dotnet-developer-conference.de | #DDC24 | Find us on [f](#) [i](#) [X](#) [in](#)

Präsentiert von:

