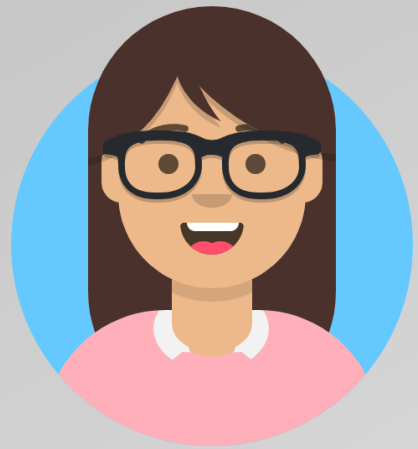

LARinet
ADC
C++
C#
Plattformübergreifende
App-Entwicklung mit .NET MAUI
ppedv

Captain Code

- Maskottchen der ADC 2025
- .NET Experte bei ppedv AG
- Lego MVP, MCT

...about us?

- LARInet (Learn Read Implement)
- IT-Dienstleister
  - Development (App/Web)
  - .NET, WinUI 3, .NET MAUI, Blazor
  - Design
  - Konzepte
  - **Workshops, Seminare**



Wer?

Elena Bochkor
Veikko Krypczyk

Wo?

Erfurt

Kontakt:

https://larinet.com

Und Sie?

Mögen Sie sich kurz vorstellen?

Welchen Hintergrund bringen Sie mit?

- .NET, NET-Framework?
- Xamarin
- Mobile Entwicklung (iOS, Android)
- Xcode
- Erfahrungen auf macOS

Was wünschen Sie sich für heute?

Übersicht

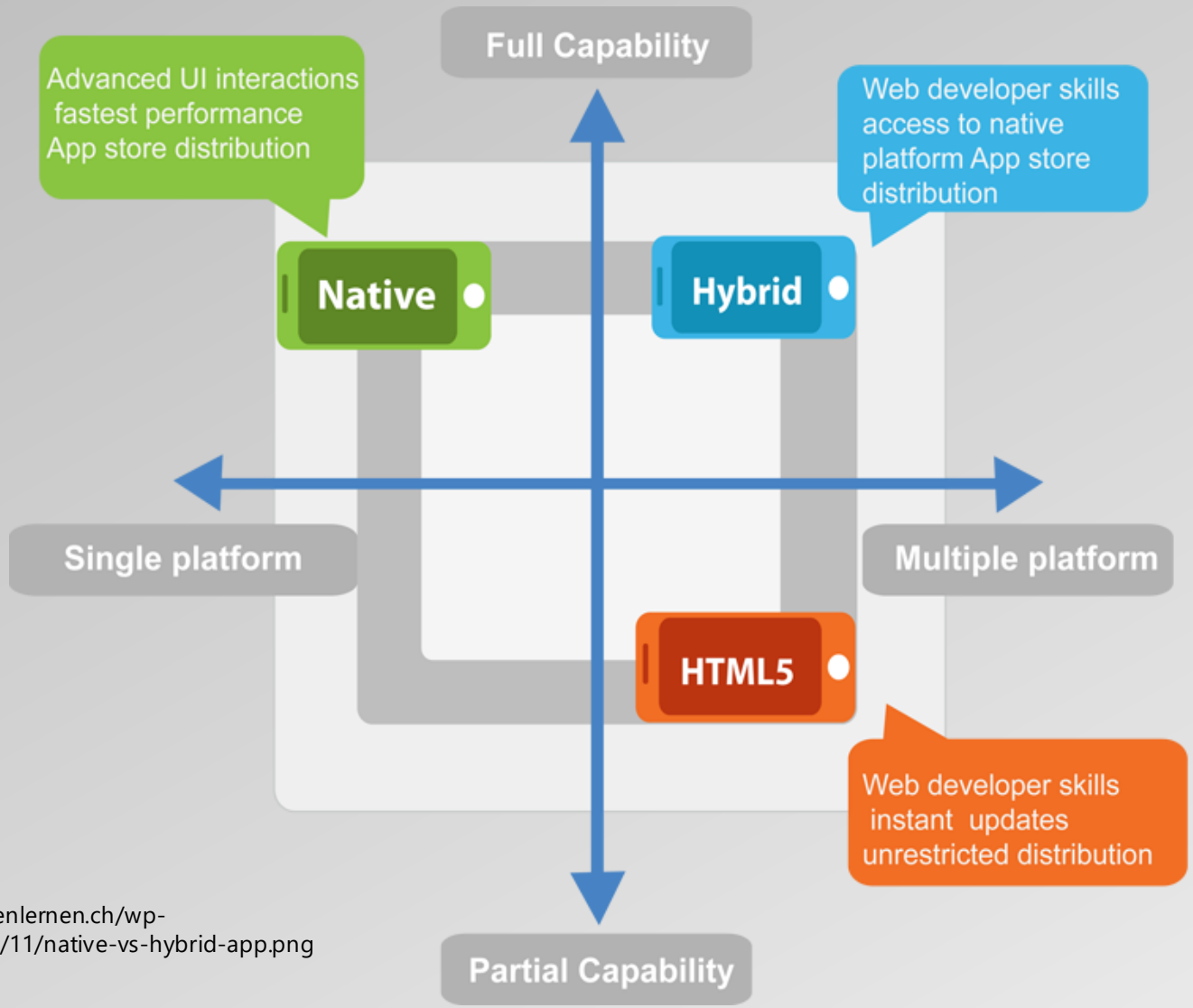
- **Teil 1: .NET MAUI verstehen und einsetzen – Grundlagen**
- Teil 2: Tolle User Interfaces mit XAML gestalten
- Teil 3: Das MVVM-Entwurfsmuster verstehen und anwenden (App-Architektur)
- Teil 4: Zugriff auf Geräte und Sensoren ermöglichen –- mit .NET MAUI und Plattformcode

Übersicht Teil 1

- .NET MAUI im Vergleich zu anderen Technologien
- Grundlagen von .NET MAUI, Aufbau des Frameworks
- System- und Entwicklungsumgebung
- Devices, Emulator, Simulator
- Eine erste App
- Projektstruktur verstehen

Vergleich zu anderen Technologien

App-Typen



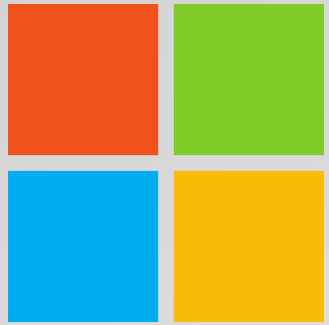
Quelle: https://gestaltenlernen.ch/wp-content/uploads/2018/11/native-vs-hybrid-app.png

Technologien and Frameworks

- Native
  - Android: Java/Kotlin mit Android Studio
  - iOS: Swift/SwiftUI mit Xcode
- Hybrid
  - Apache Cordova mit Web Technologies
- Web
  - HTML/CSS/JavaScript
  - SPA Frameworks

Quelle: https://pixabay.com/de/images/search/app%20mobile/

Cross Platform Entwicklung



- Flutter
- RAD Studio
- NativeScript
- ~~Xamarin/ Xamarin.Forms~~
- …
- . NET Multi-App Platform UI (.NET MAUI)

Quelle: https://pixabay.com/

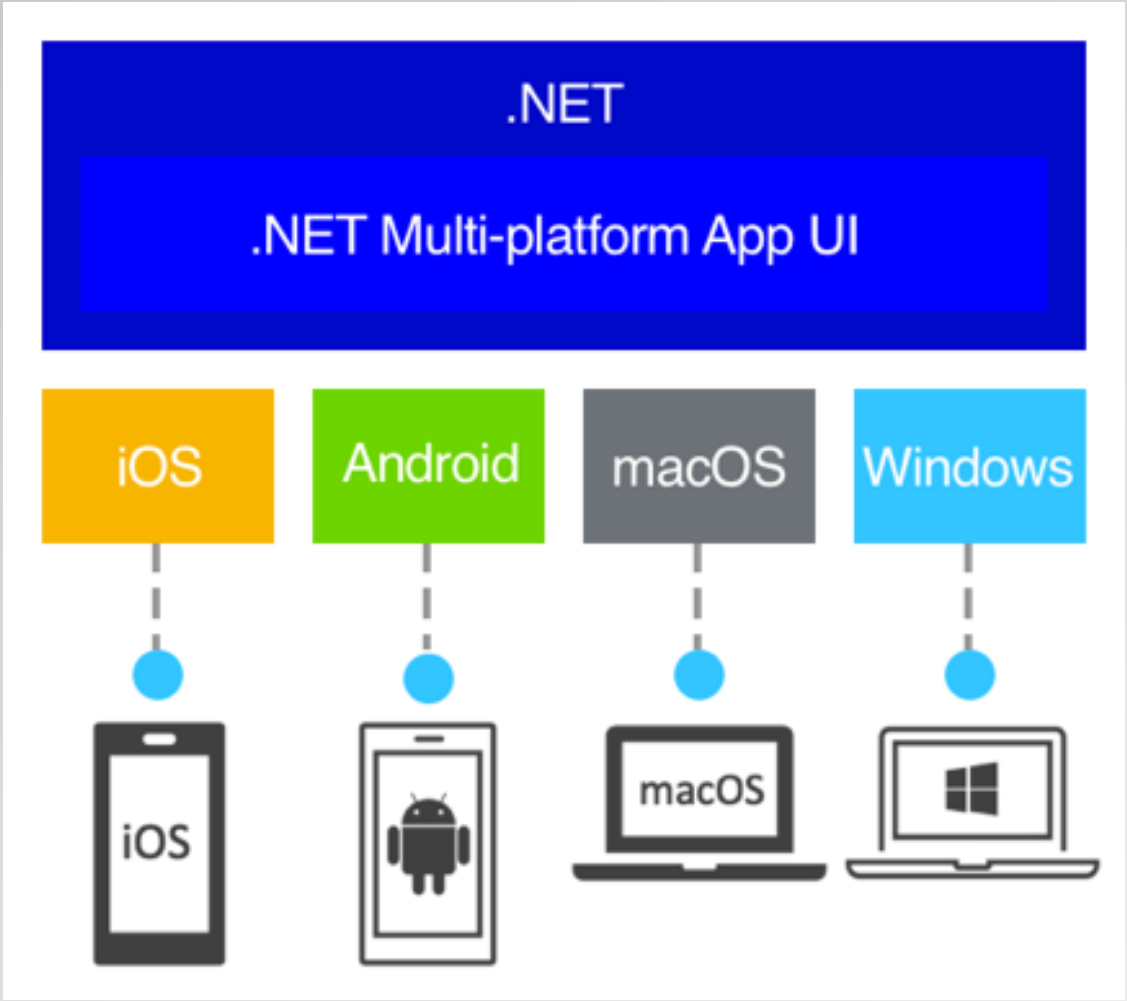
Technologie-Vergleich (MAUI vs. Flutter)

Kriterium	.NET MAUI	Flutter
Sprache	C#	Dart
Zielsysteme	Android, iOS, Windows, macOS, Tizen	Android, iOS, Web, Desktop
User Interface-Framework	deklarativ (XAML, Markup)	deklarativ (Widgets)
Businesslogik	gemeinsame Codebasis für Businesslogik und User Interface	gemeinsame Codebasis für Businesslogik und User Interface
Ökosystem	.NET-Ökosystem	Flutter-Ökosystem
Reifegrad	in Nutzung	in Nutzung
Community	groß und etabliert	groß und aktiv
Performance	gut	gut
bevorzugte Entwicklungsumgebung	Visual Studio	Visual Studio Code
Lernkurve	Abhängig von .NET-Kenntnissen	Abhängig von Dart-Kenntnissen
Unterstützung	Microsoft	Google
Open Source	ja	ja

Grundlagen von .NET MAUI und Aufbau des Frameworks

Was ist .NET MAUI

- plattformübergreifendes Framework zum Erstellen nativer mobiler und Desktop-Apps mit C# und XAML.
- es können Sie Apps entwickeln, die auf Android, iOS, macOS und Windows von einer einzigen gemeinsam genutzten Codebasis ausgeführt werden können



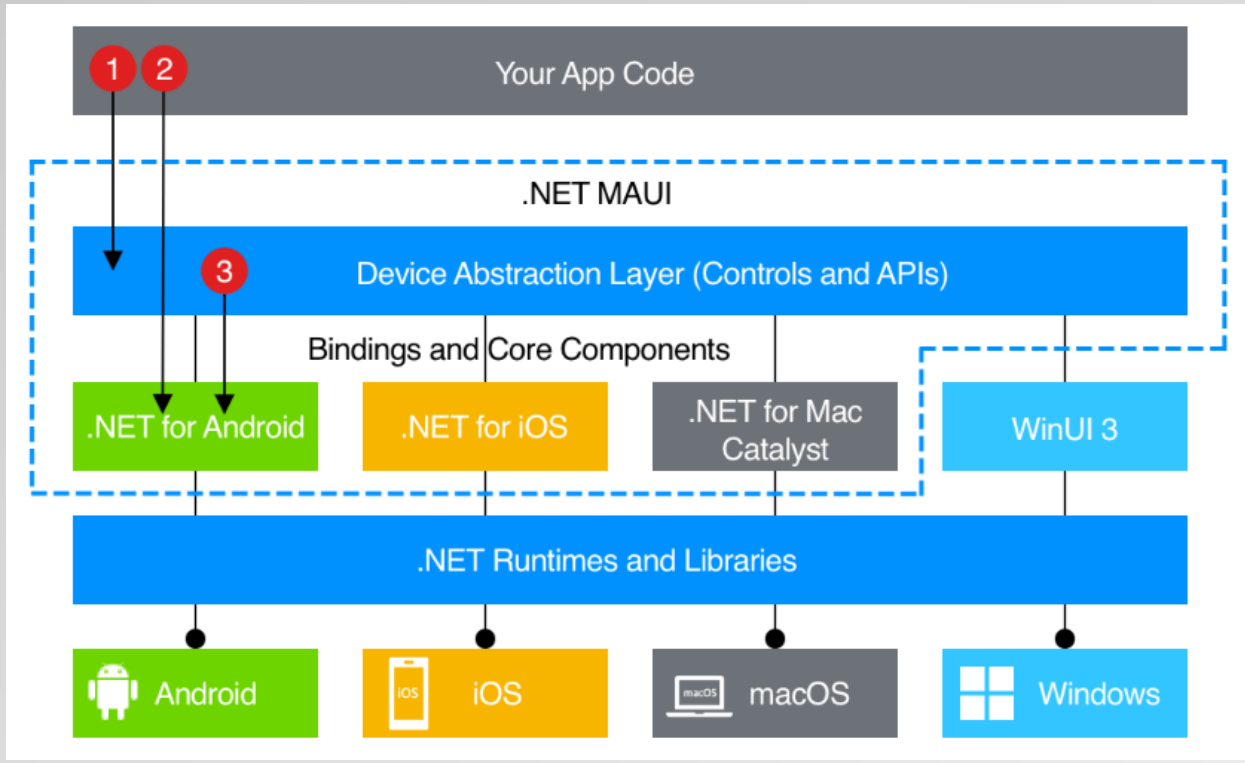
Quelle: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui

Ziele von .NET MAUI

- .NET MAUI ist Open Source und die Weiterentwicklung von Xamarin.Forms, erweitert von mobilen auf Desktop-Szenarien.
- Mit .NET MAUI können Sie plattformübergreifende Apps mit einem einzigen Projekt erstellen, aber Sie können bei Bedarf plattformspezifischen Quellcode und Ressourcen hinzufügen.
- Eines der Hauptziele von .NET MAUI besteht darin, Ihnen zu ermöglichen, so viel Ihrer App-Logik und Ihres UI-Layouts wie möglich in einer einzigen Codebasis zu implementieren.

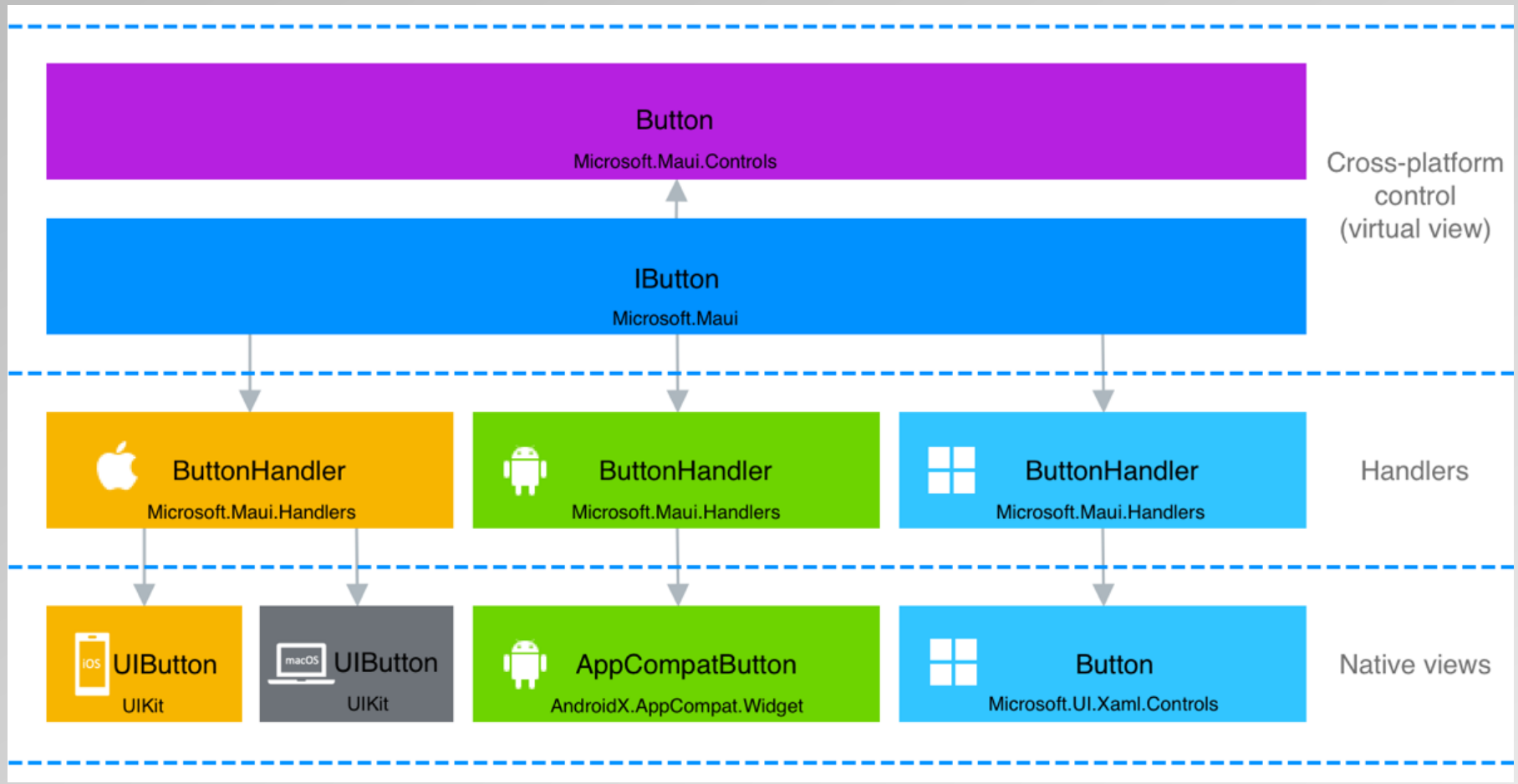
Funktionsweise von .NET MAUI

- >.NET 6 bietet eine Reihe plattformspezifischer Frameworks zum Erstellen von Apps:
  - .NET für Android
  - .NET für iOS
  - .NET für macOS
  - Windows UI 3 (WinUI 3) Bibliothek.
- diese Frameworks haben Zugriff auf die gleiche .NET 6 Base Class Library (BCL). für Android, iOS und macOS wird die Umgebung von Mono implementiert
- unter Windows stellt .NET CoreCLR die Ausführungsumgebung bereit



Quelle: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui

Funktionsweise von .NET MAUI



Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/handlers/

Apps für Android und Windows

- Android-Apps, die mit .NET MAUI kompiliert wurden, werden von C# in Zwischensprache (IL) erstellt, die dann just-in-time (JIT) kompiliert wird, wenn die App gestartet wird.
- Windows-Apps, die mithilfe von .NET MAUI erstellt wurden, verwenden Windows UI 3 (WinUI 3)-Bibliothek, um native Apps zu erstellen, die auf den Windows-Desktop abzielen.

Apps für iOS und macOS

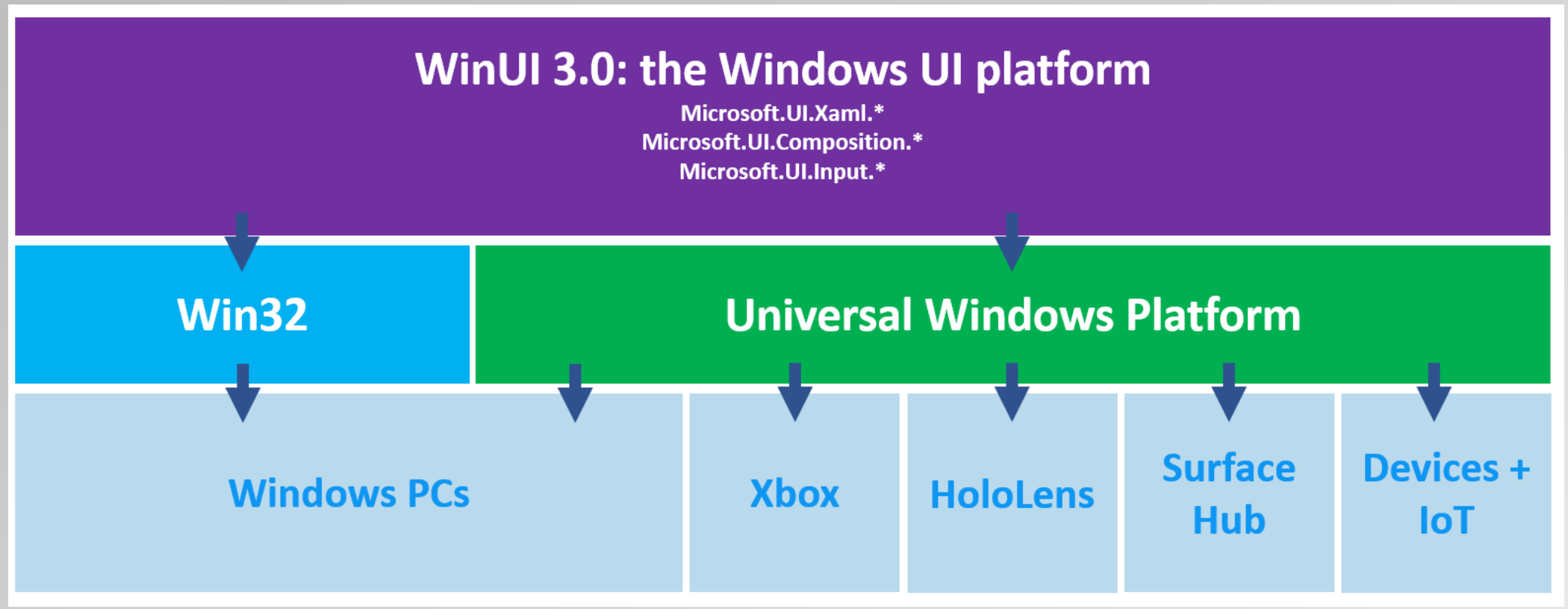
- iOS-Apps, welche die mithilfe von .NET MAUI erstellt wurden nutzen UIKit zur Darstellung des User Interfaces.
- macOS-Apps, die mithilfe von .NET MAUI erstellt wurden, verwenden Mac Catalyst, eine Lösung von Apple, die Ihre iOS-App mit UIKit auf den Desktop erstellt, und erweitert sie mit zusätzlichen AppKit- und Plattform-APIs wie erforderlich.

Exkurs: WinUI 3

- neue technologieunabhängige Grafikschnittstelle
- Bestandteil des Windows App SDK
- Desktop-App mit vollständigem Systemzugriff
- „vereint“ das Design der UWP mit den nativen Features der WPF
- moderne UI-Controls: Fluent Design, Modern Input, Standard Architektur-Pattern: MVVM
- Einschränkungen: nur ab Windows 10

Quelle: https://learn.microsoft.com/en-us/windows/apps/winui/

Exkurs: Architektur WinUI 3



Quelle: https://docs.microsoft.com/de-de/dotnet/architecture/modernize-desktop/windows-migration

Was bietet .NET MAUI

- Sammlung von UI-Steuerelementen
- ein Layoutmodul zum Entwerfen von Seiten
- mehrere Seitentypen zum Erstellen von umfassenden Navigationstypen
- Unterstützung für die Datenbindung und Entwicklungsmuster
- die Möglichkeit, Handler anzupassen, um die Art und Weise zu verbessern, in der UI-Elemente dargestellt werden.
- plattformübergreifende APIs für den Zugriff auf systemeigene Gerätefeatures, wie GPS, Akku- und Netzwerkzustände

Was bietet .NET MAUI (Fortsetzung)

- plattformübergreifende Grafikfunktionen, die einen Zeichenbereich bereitstellen, der Grafiken und Bilder unterstützt
- ein einzelnes Projektsystem, das **multi-targeting** zum Ziel von Android, iOS, macOS und Windows verwendet
- **.NET hot reload**, sodass Sie sowohl Ihren XAML- als auch Ihren verwalteten Quellcode ändern können, während die App ausgeführt wird, und beobachten Sie dann das Ergebnis Ihrer Änderungen, ohne die App neu zu erstellen

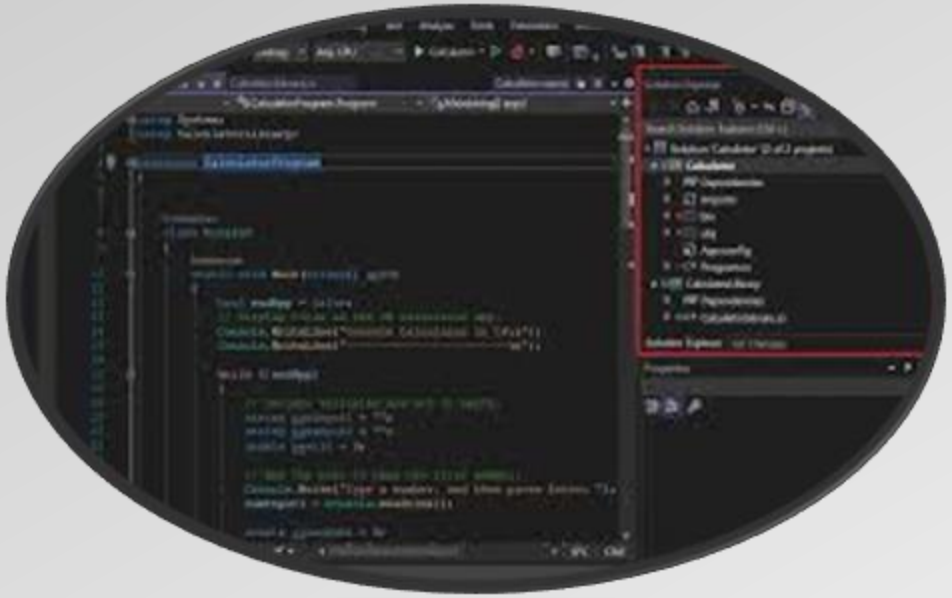
Unterstützte Plattformen

- Android 5.0 (API 21) oder höher
- iOS 11 oder höher, mit der neuesten Version von Xcode
- macOS 10.15 oder höher über Mac Catalyst
- Windows 11 und Windows 10 Version 1809 oder höher
- Tizen über Samsung
- .NET MAUI Blazor-Apps, hybride Apps (Web-Apps auf dem Desktop)
- (Linux)

System- und Entwicklungsumgebung

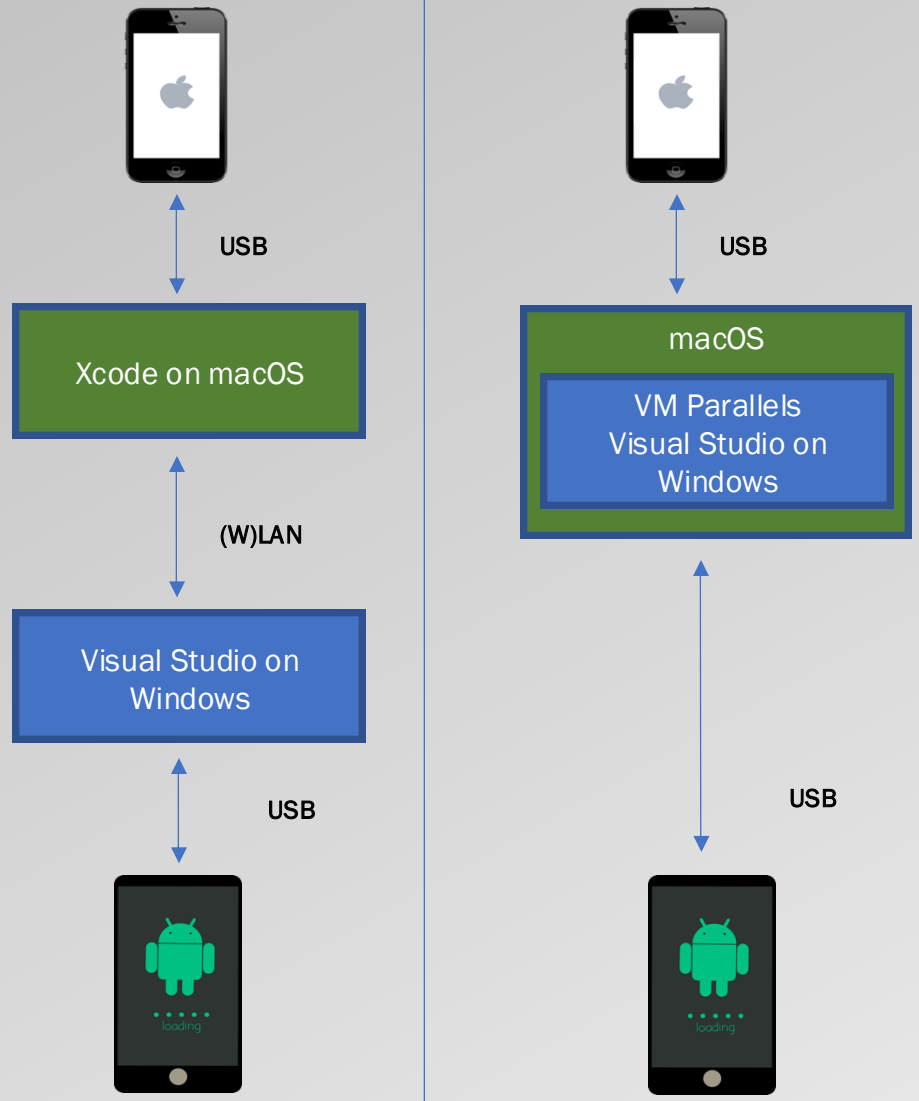
System- und Entwicklungsumgebung

- die gewünschten Zielsysteme bestimmten die System- und Entwicklungsumgebung
- für eine Entwicklung von App für macOS und ist ein Mac-PC notwendig
- Optionen zur IDE:
  - **Entwicklung unter Visual Studio unter Windows**
  - Entwicklung mit Visual Studio Code (plattformübergreifend) mit C# Dev-Kit

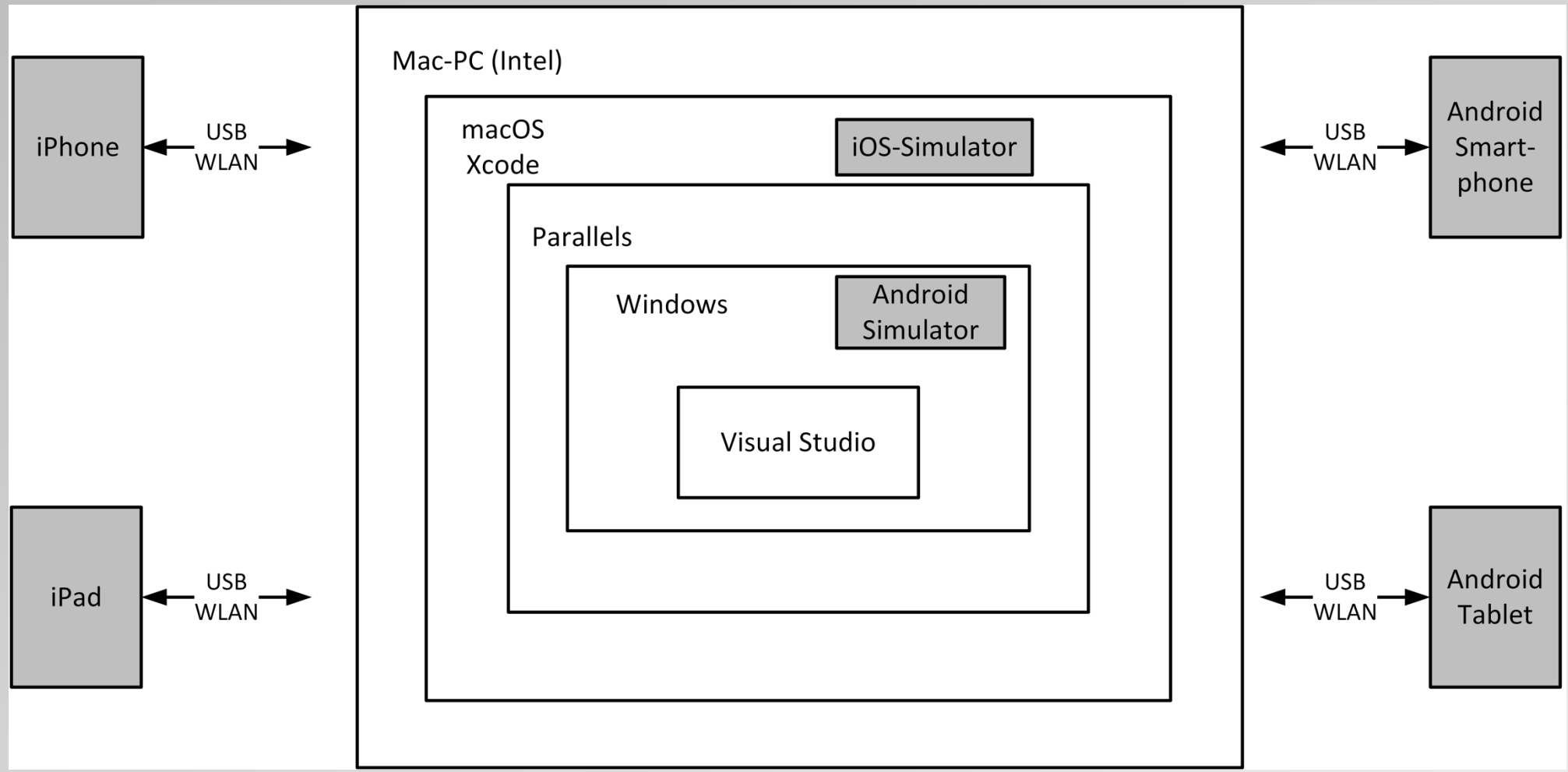


Achtung: Es ist auf die „Kompatibilität“ der Versionen zu achten

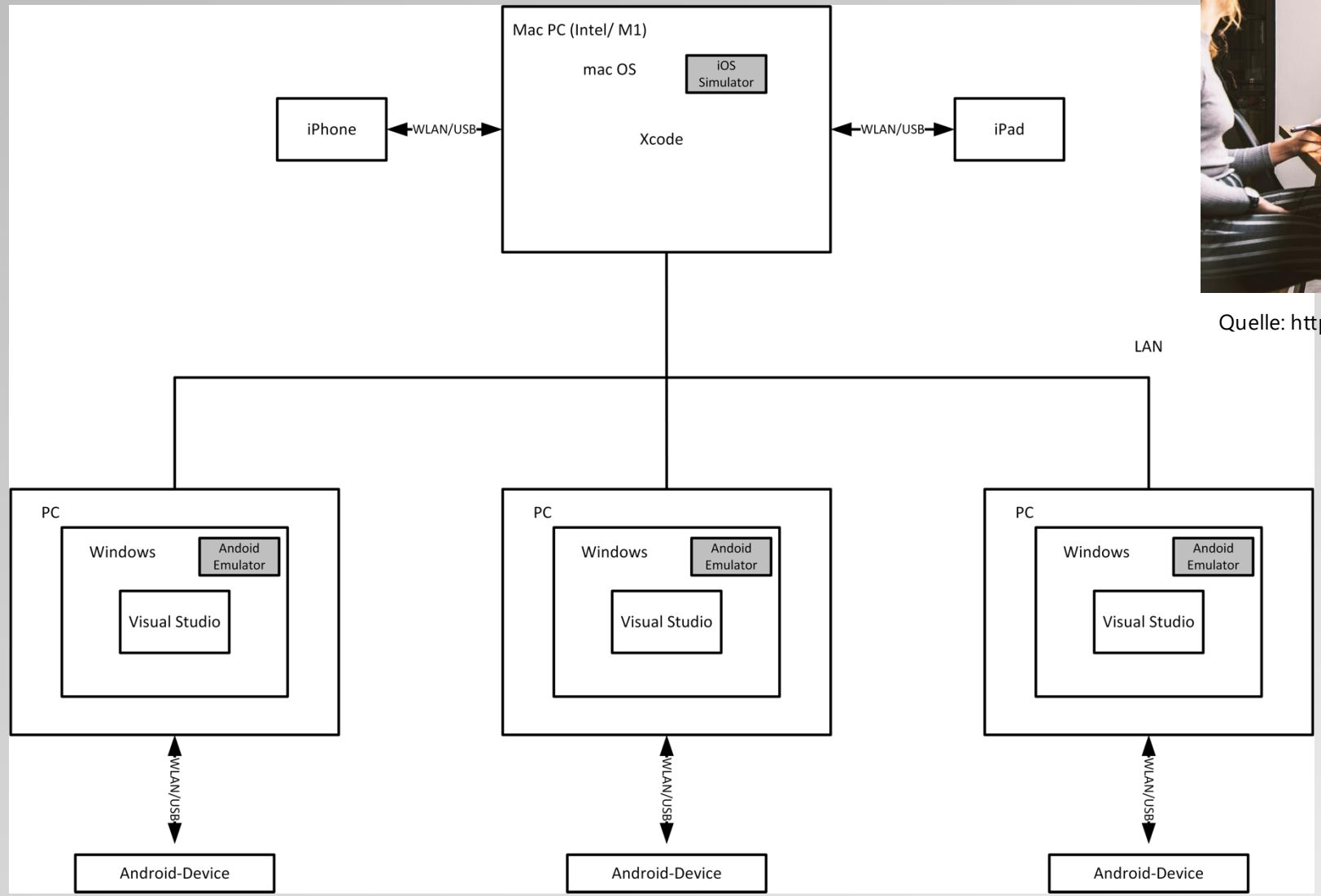
Architektur der Entwicklungsumgebung



Visual Studio in einer virtuellen Maschine

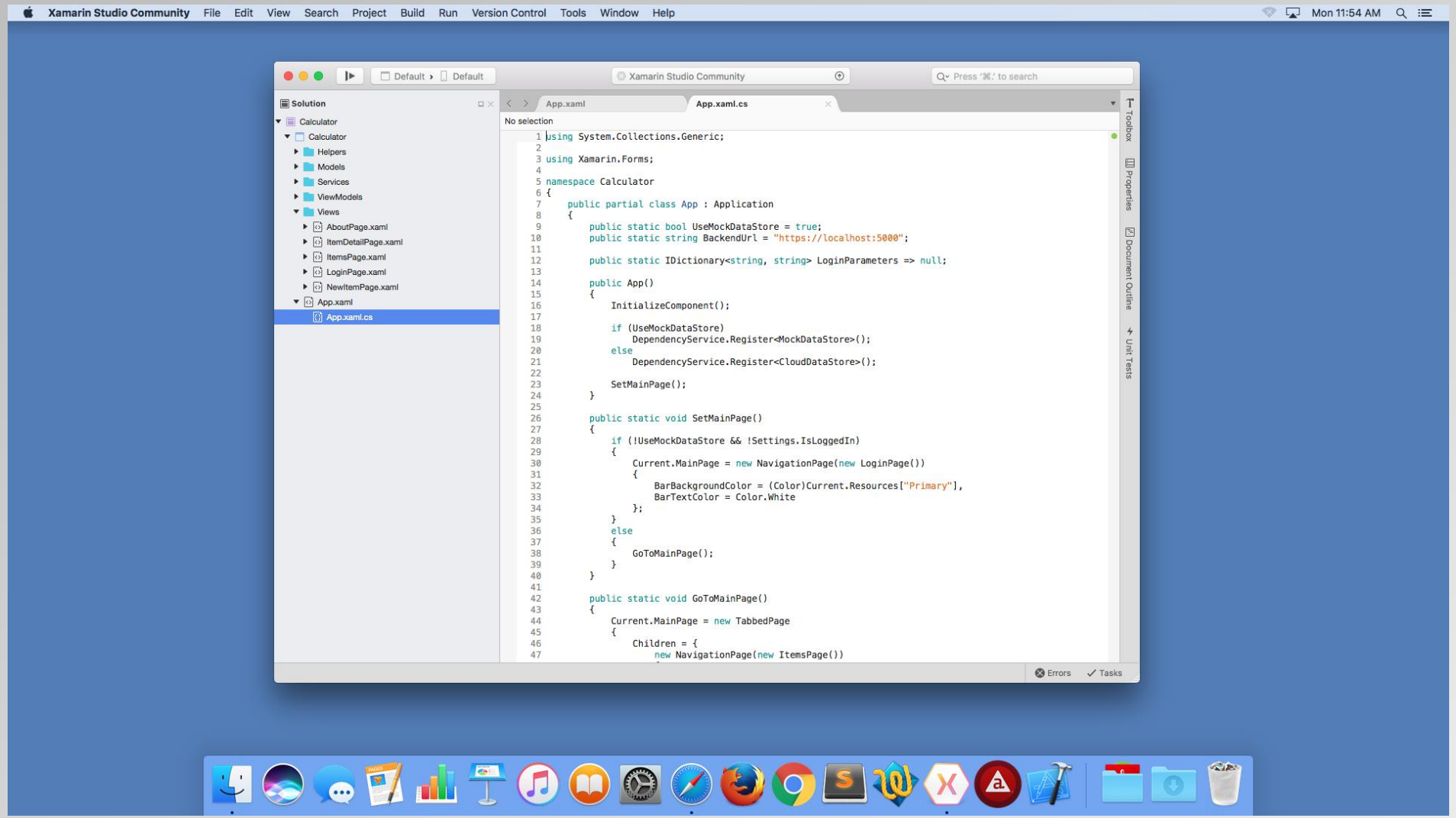


Team-Entwicklung im Netzwerk



Quelle: https://pixabay.com/de/images/search/team/

Alternative für iOS/ macOS: MacInCloud

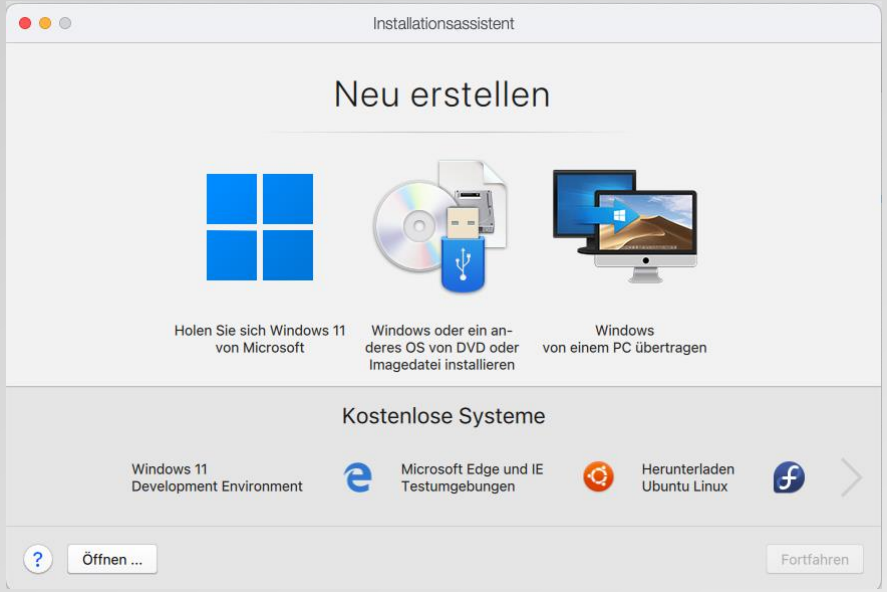



```
using System.Collections.Generic;

using Xamarin.Forms;

namespace Calculator
{
    public partial class App : Application
    {
        public static bool UseMockDataStore = true;
        public static string BackendUrl = "https://localhost:5000";

        public static IDictionary<string, string> LoginParameters => null;

        public App()
        {
            InitializeComponent();

            if (UseMockDataStore)
                DependencyService.Register<MockDataStore>();
            else
                DependencyService.Register<CloudDataStore>();

            SetMainPage();
        }

        public static void SetMainPage()
        {
            if (!UseMockDataStore && !Settings.IsLoggedIn)
            {
                Current.MainPage = new NavigationPage(new LoginPage())
                {
                    BarBackgroundColor = (Color)Current.Resources["Primary"],
                    BarTextColor = Color.White
                };
            }
            else
            {
                GoToMainPage();
            }
        }

        public static void GoToMainPage()
        {
            Current.MainPage = new TabbedPage
            {
                Children = {
                    new NavigationPage(new ItemsPage())
```

Quelle: https://www.macincloud.com/pages/visual-studio-mac.html

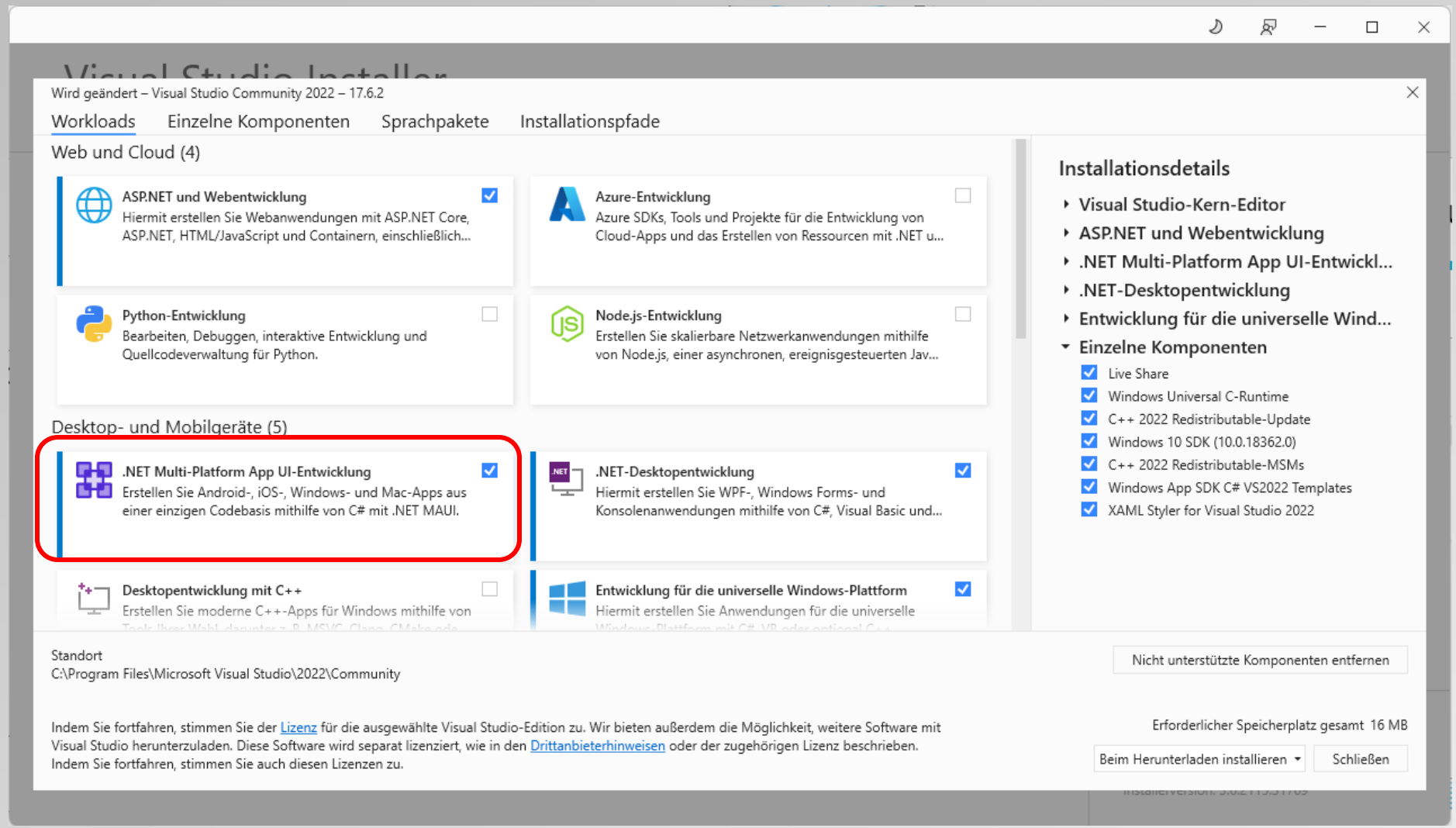
Entwicklung unter Windows

- auf einem Wimdows-Rechner oder in einer VM (unter macOS)

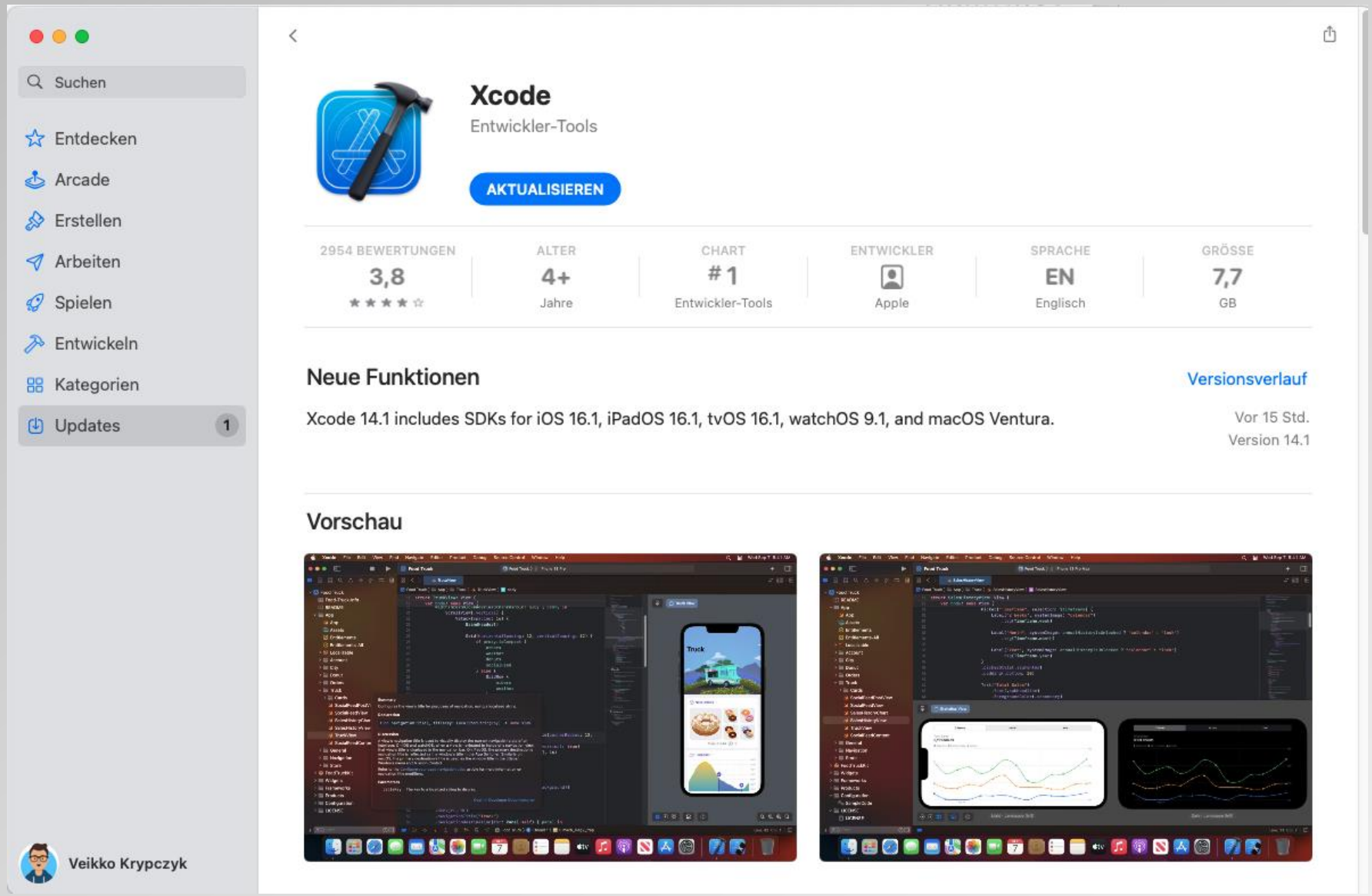


- Achtung: beim einer VM unter macOS ist zwischen Intel und M1-Architektur zu unterscheiden

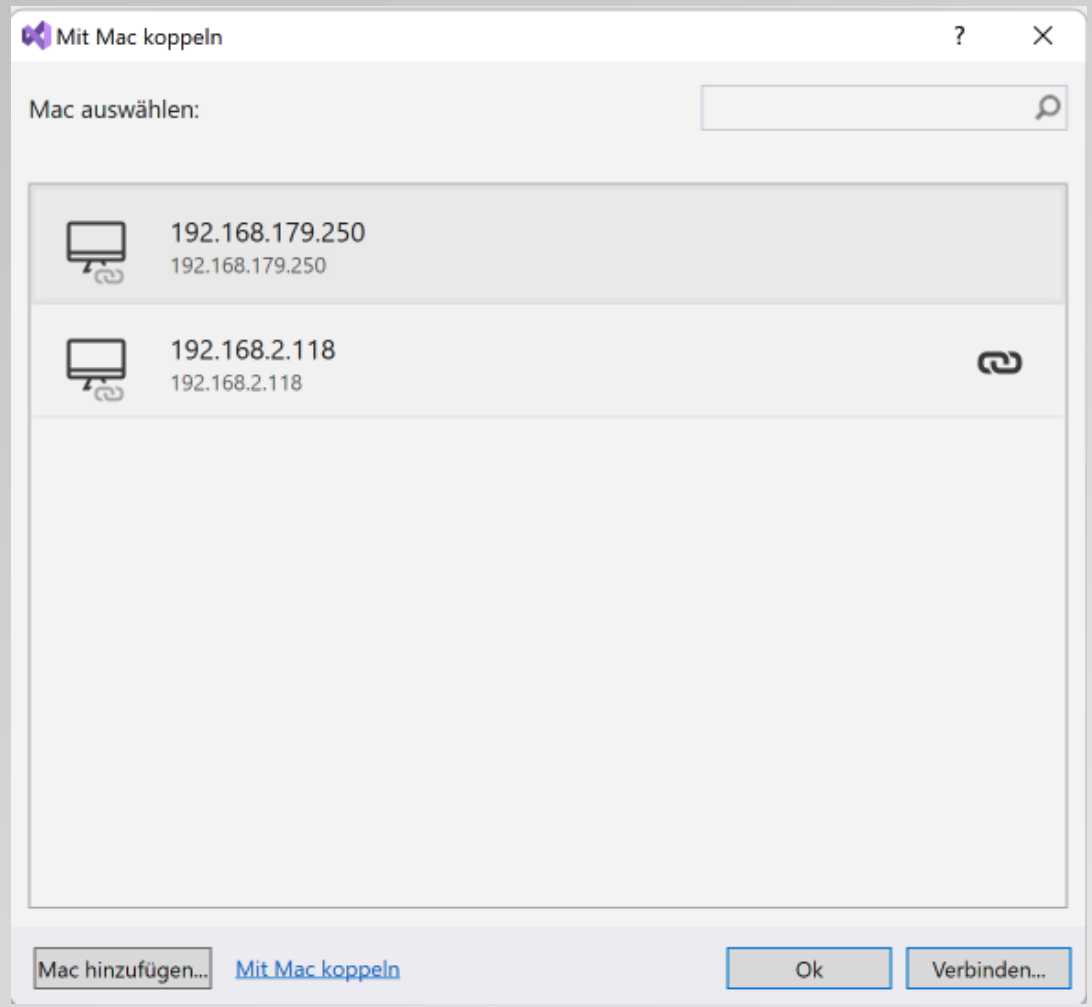
Visual Studio (Windows)



Xcode (macOS)


Suchen
Entdecken
Arcade
Erstellen
Arbeiten
Spielen
Entwickeln
Kategorien
Updates
1
Xcode
Entwickler-Tools
AKTUALISIEREN
2954 BEWERTUNGEN
3,8
ALTER
4+
Jahre
CHART
#1
Entwickler-Tools
ENTWICKLER
Apple
SPRACHE
EN
Englisch
GRÖSSE
7,7
GB
Neue Funktionen
Versionsverlauf
Xcode 14.1 includes SDKs for iOS 16.1, iPadOS 16.1, tvOS 16.1, watchOS 9.1, and macOS Ventura.
Vor 15 Std.
Version 14.1
Vorschau
Veikko Krypczyk

Verbindung VS (Windows) – macOS

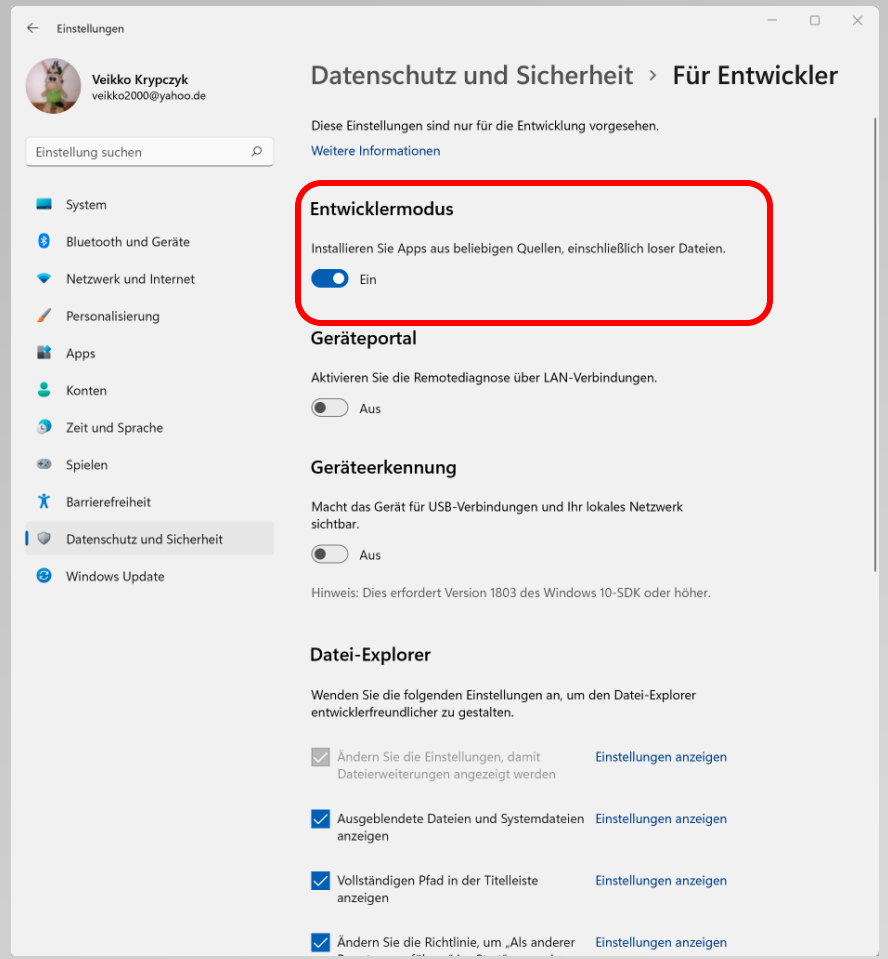


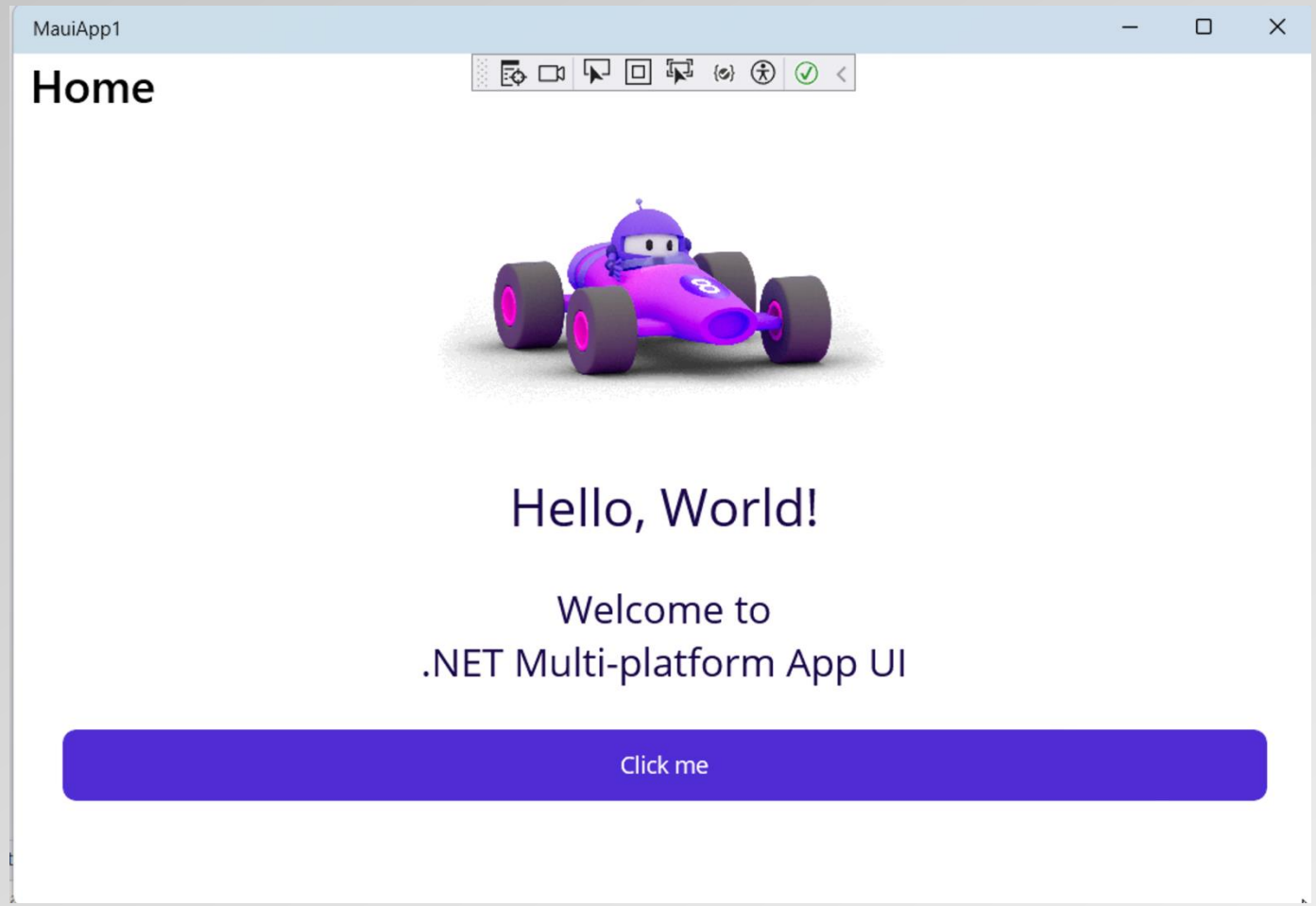
Persönliche Empfehlung

- iMac/ MacBook-Pro
- Parallels/ Windows 11
- Visual Studio in Windows

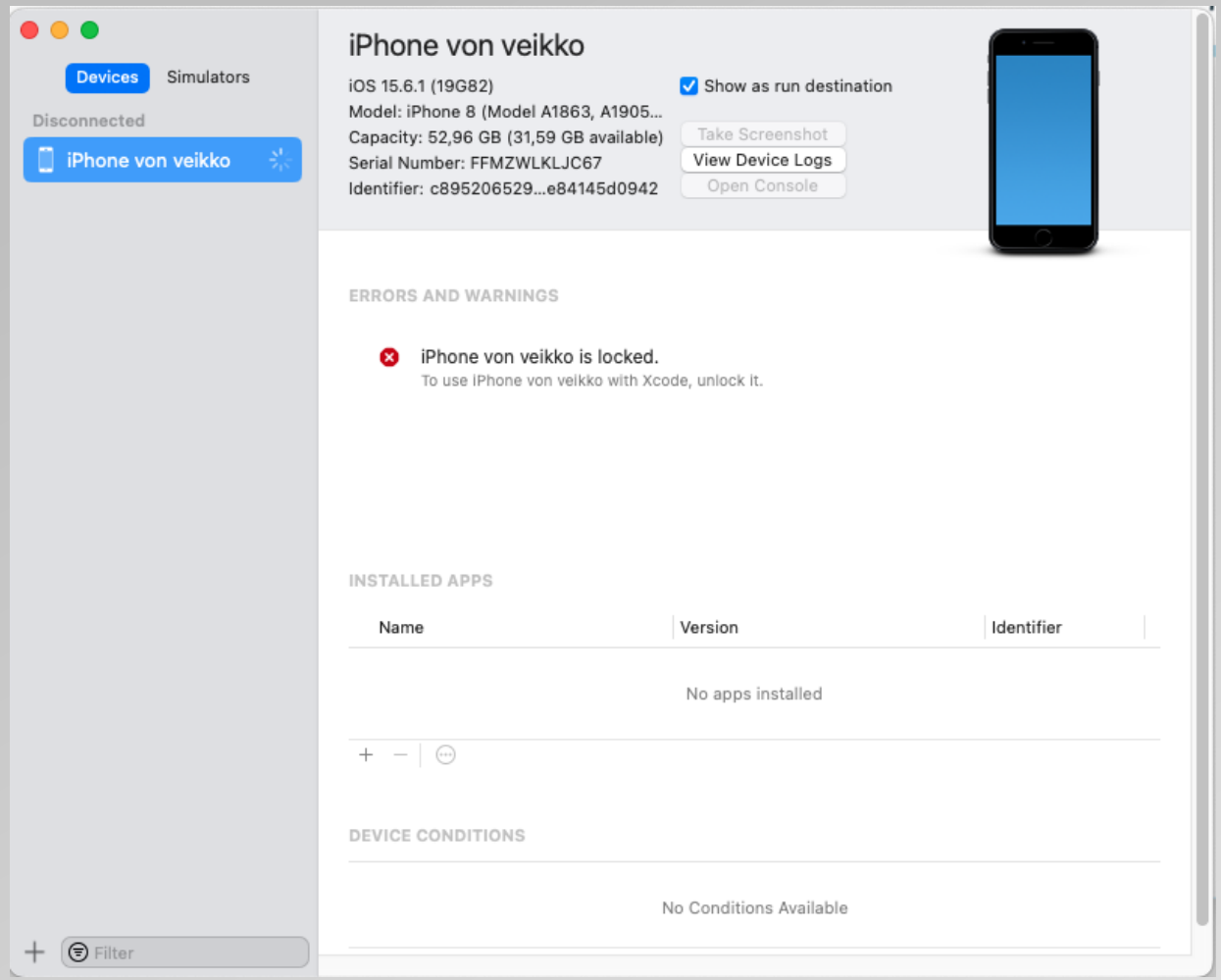
Devices, Emulator, Simulator

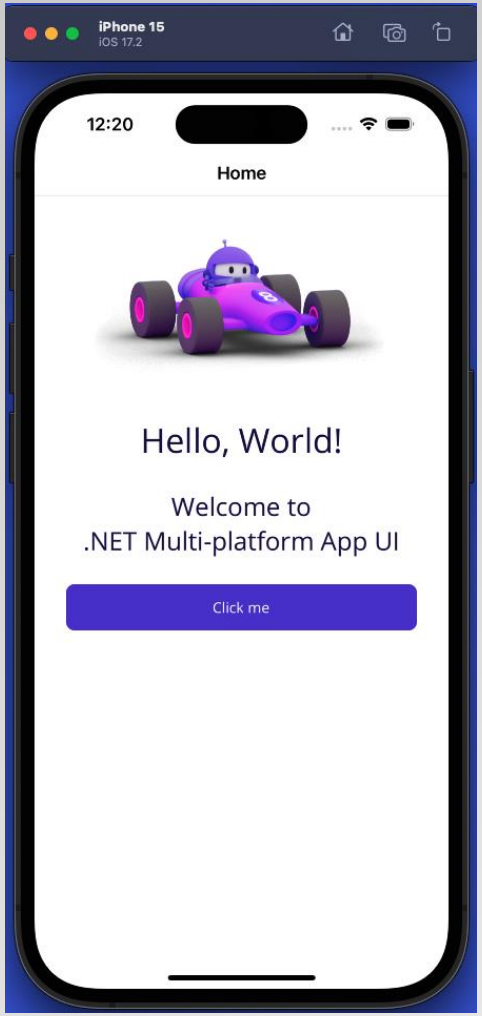
Windows App (WinUI 3)





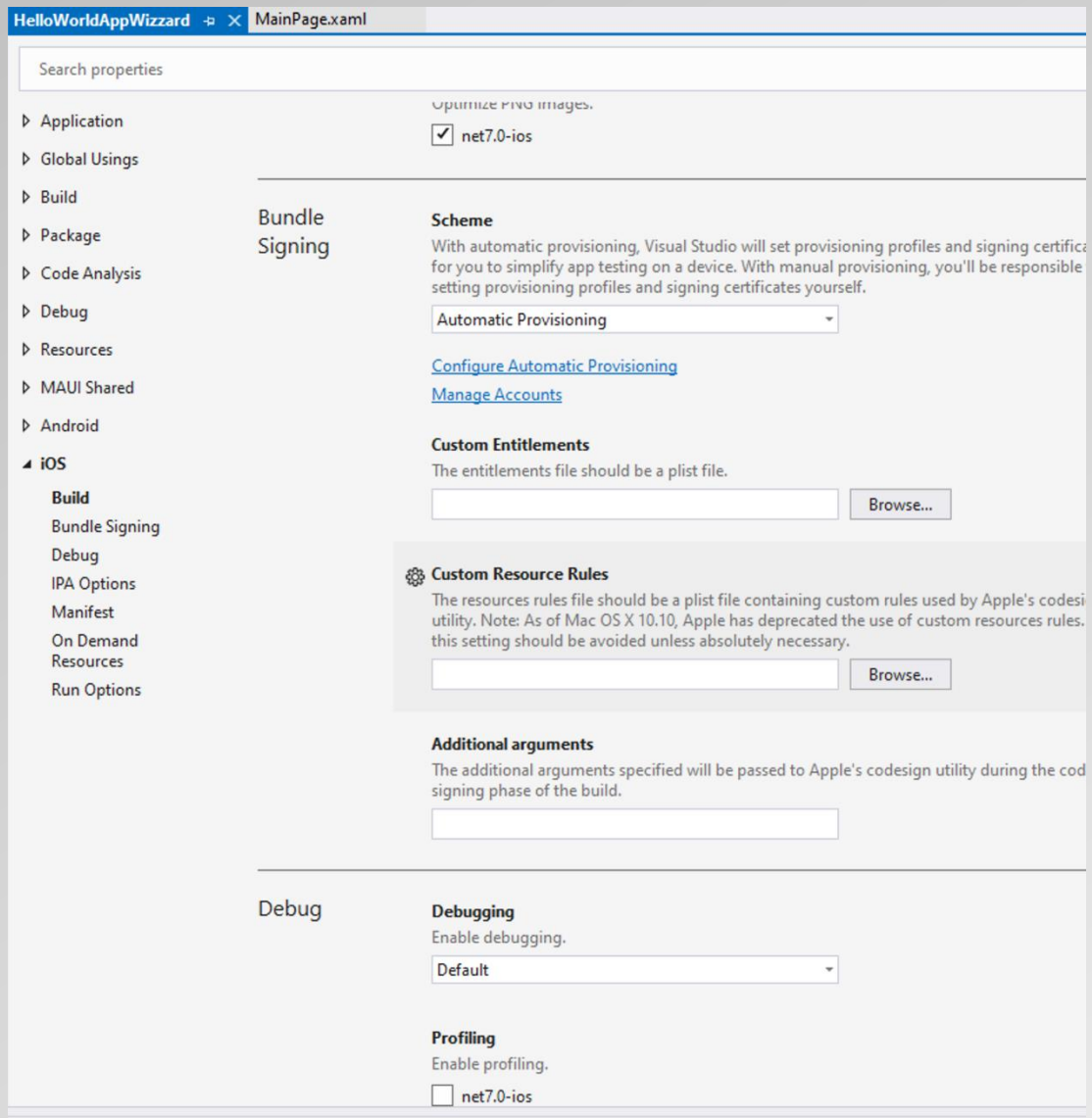
Simulator: iOS



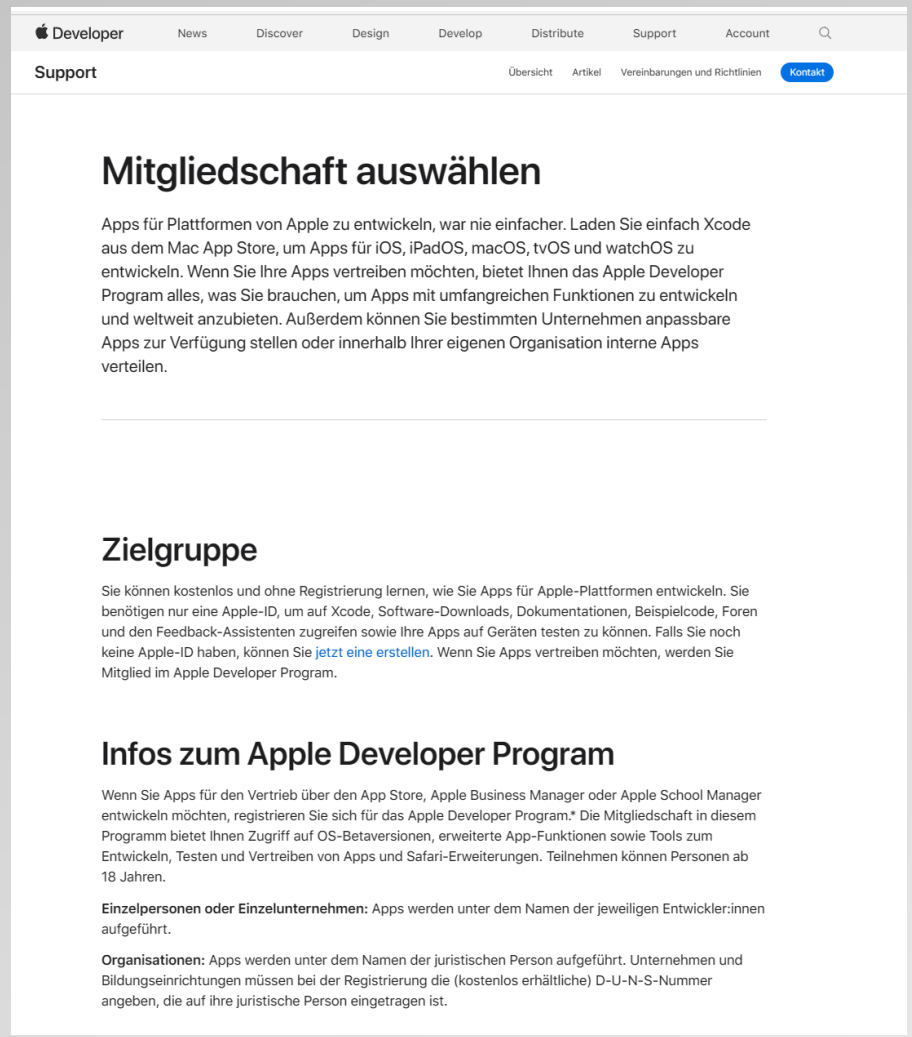


ppedv

Bereitstellungsoptionen für iOS prüfen



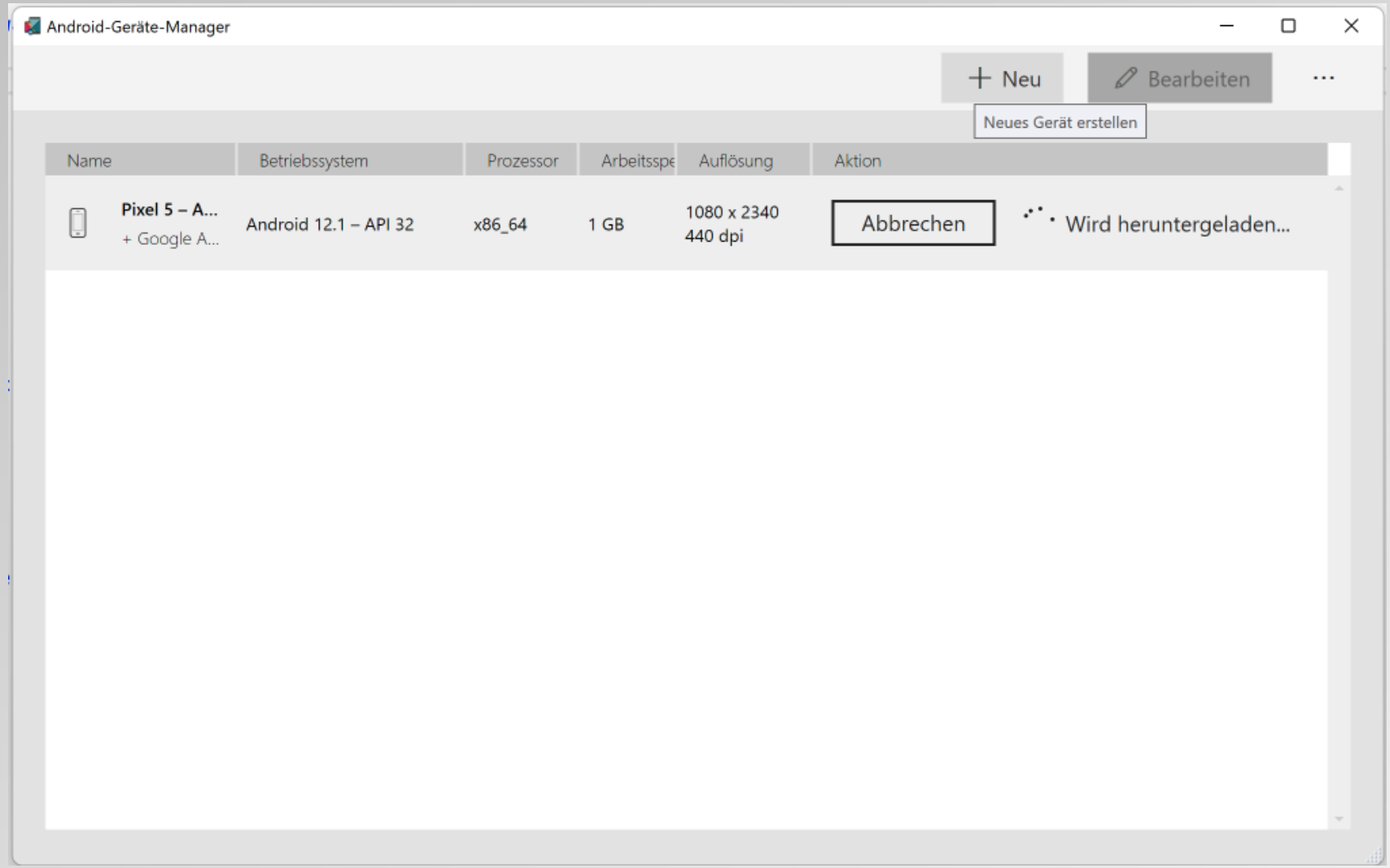
Apple Developer Programm für Signierung iOS



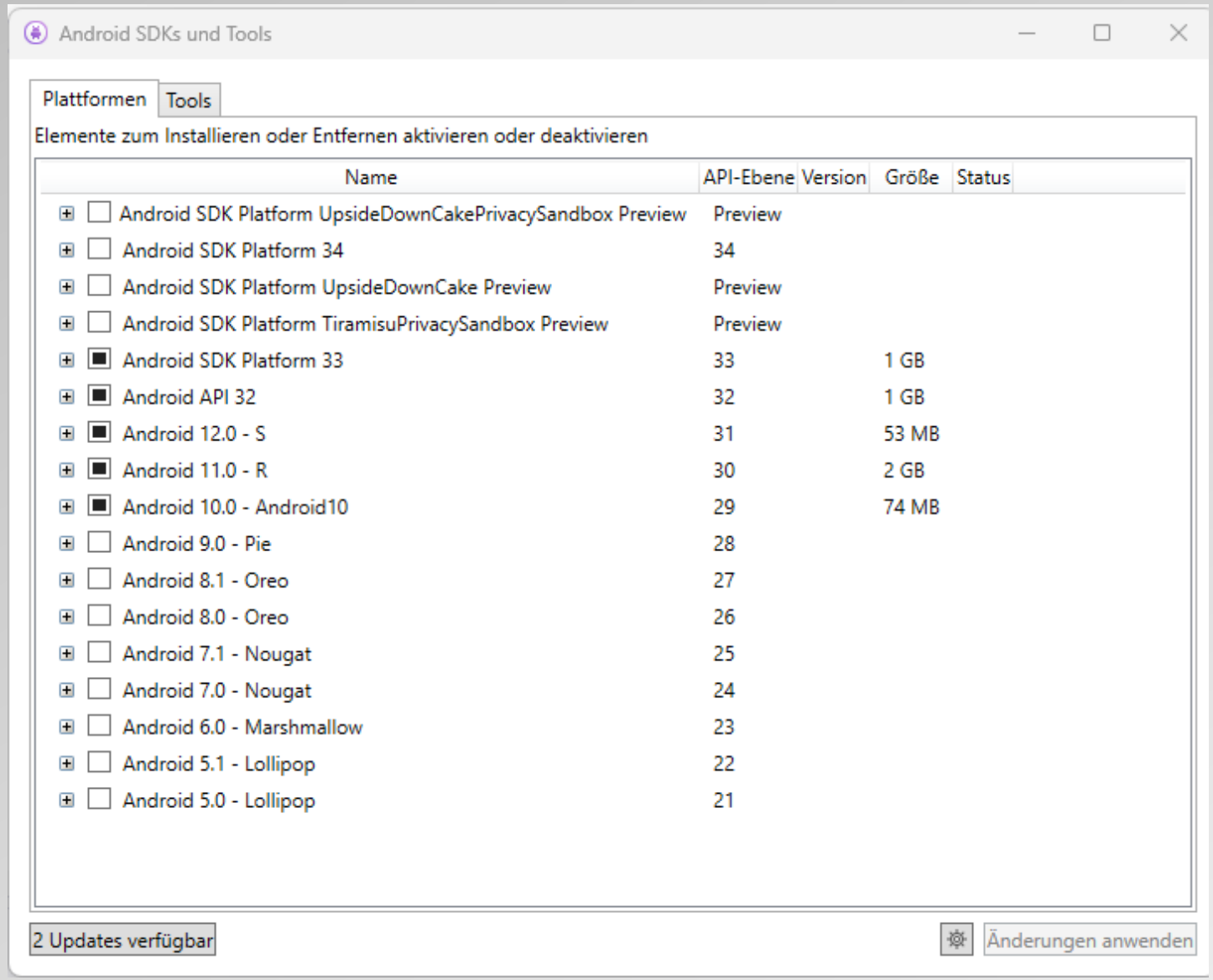
ppedv

Android Emulator (Windows)

- dieser funktioniert nicht (nur eingeschränkt) in einer VM (Parallels)
- besser: Emulator direkt auf Windows ausführen

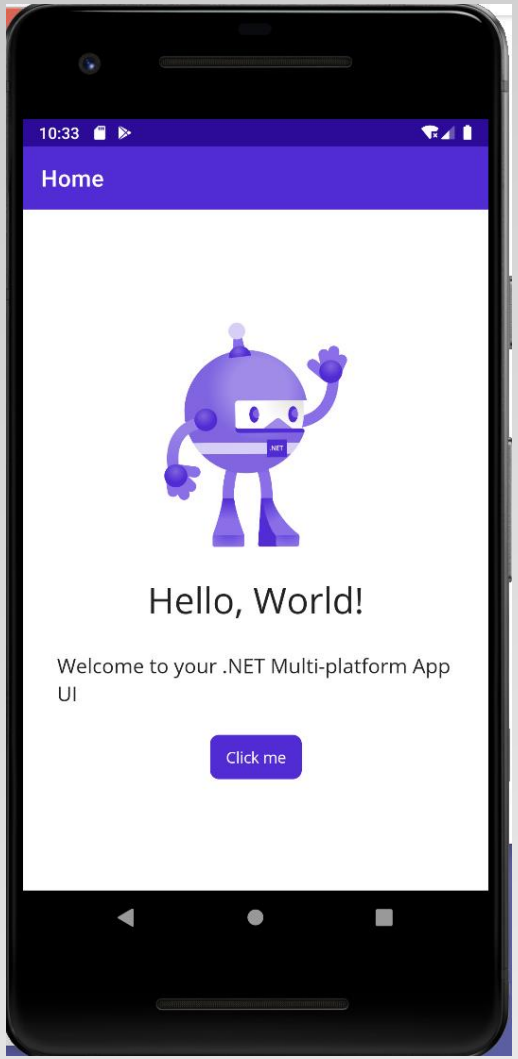


Android Emulator: SDKs und Tools



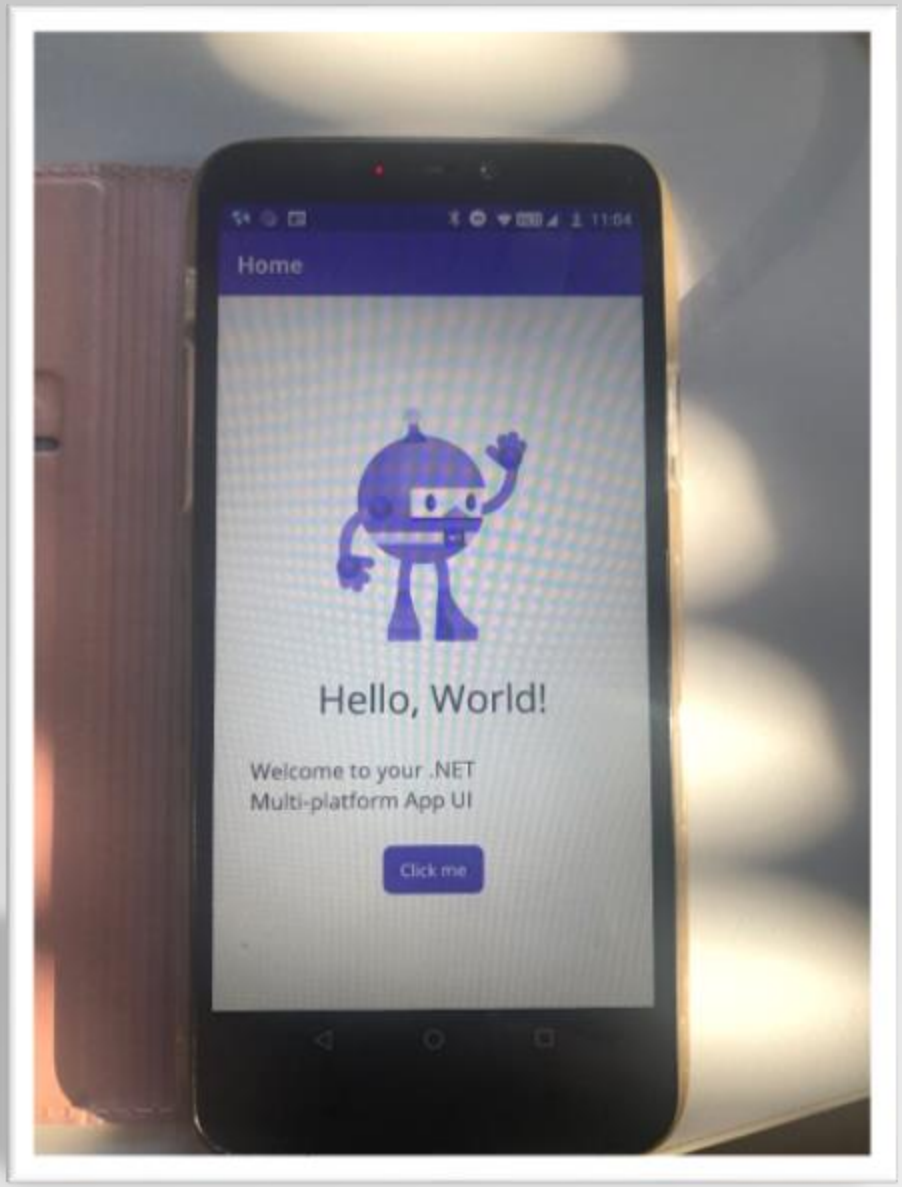
Emulator: Android (unter Windows)

haben Sie beim ersten Start Geduld…

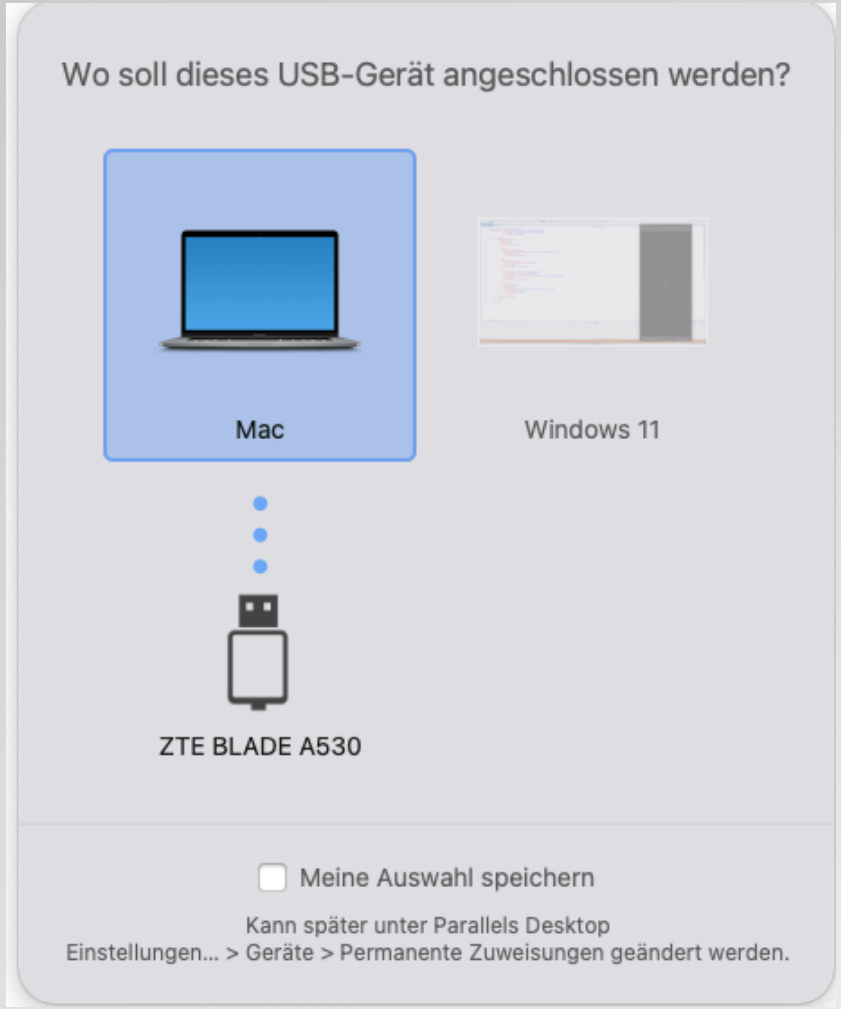


Emulator: Device

Anschluss über USB oder W-LAN

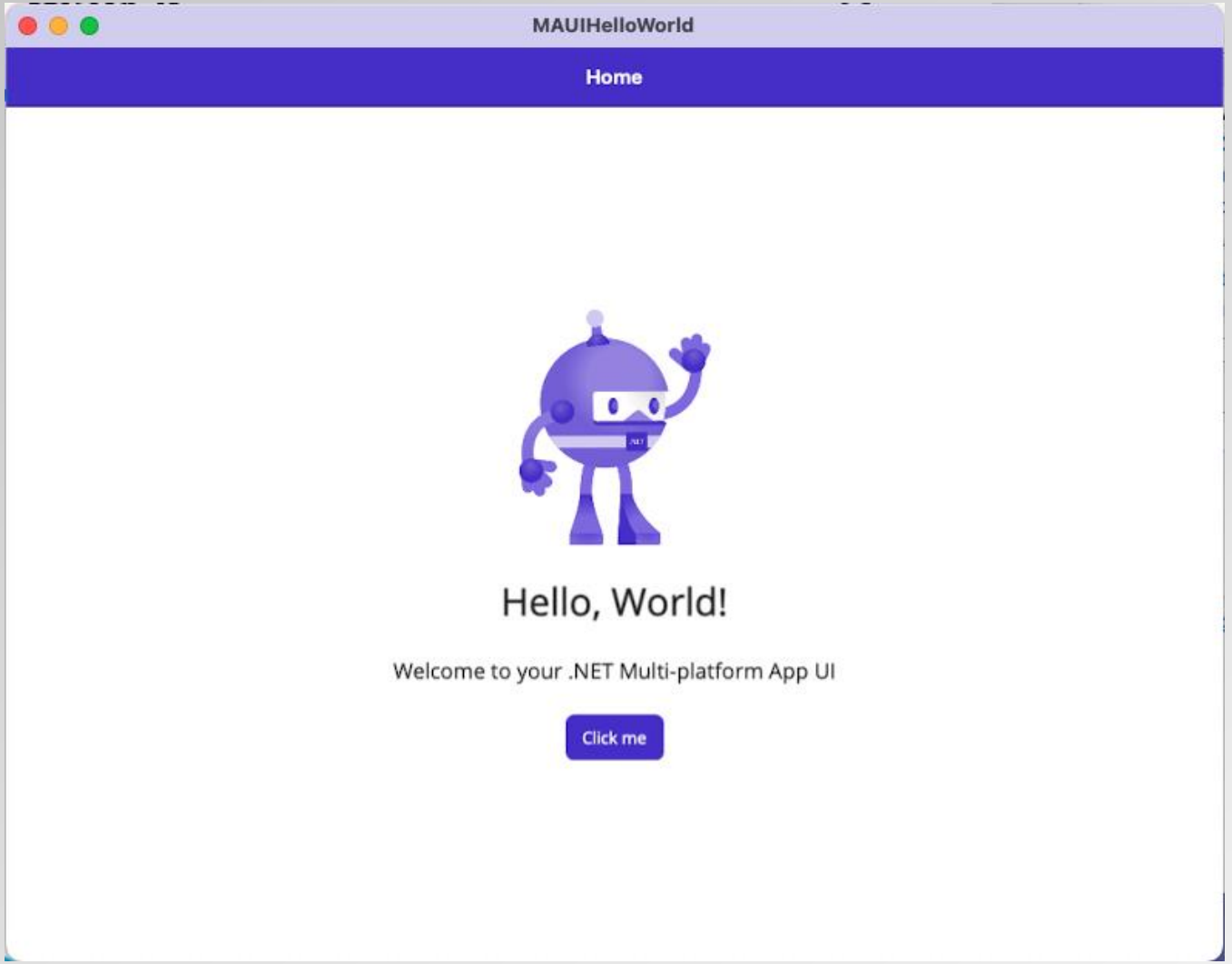


Device bei Nutzung einer virtuellen Maschine



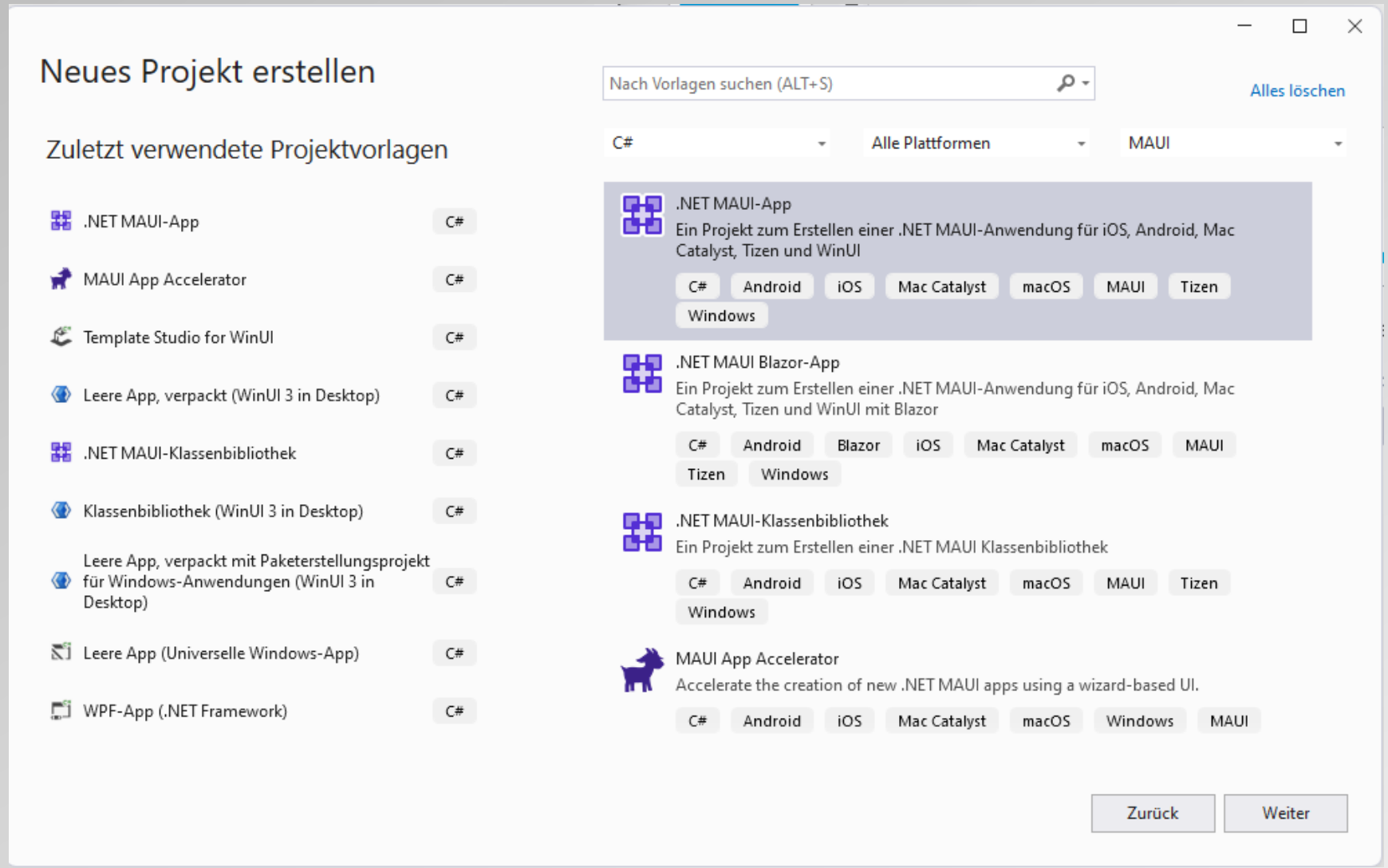
macOS (macOS Catalyst)

- macOS Catalyst ist eine vom kalifornischen Unternehmen Apple entwickelte Software, um ursprünglich für das mobile Betriebssystem iOS bzw. iPadOS programmierte Anwendungen auf Computer mit dem Betriebssystem macOS zu portieren.
- kein Debugging aus Visual Studio

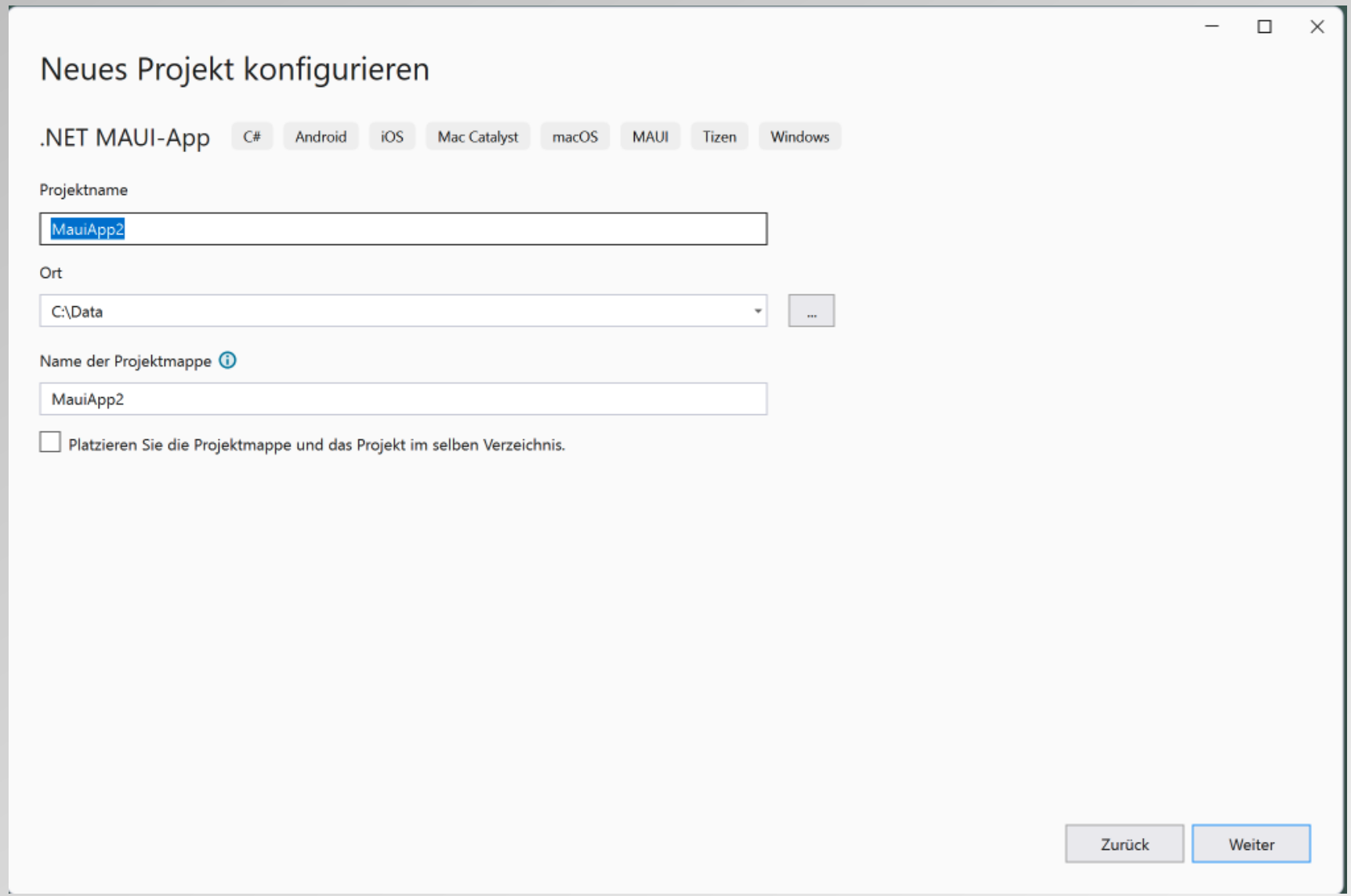


Eine erste App

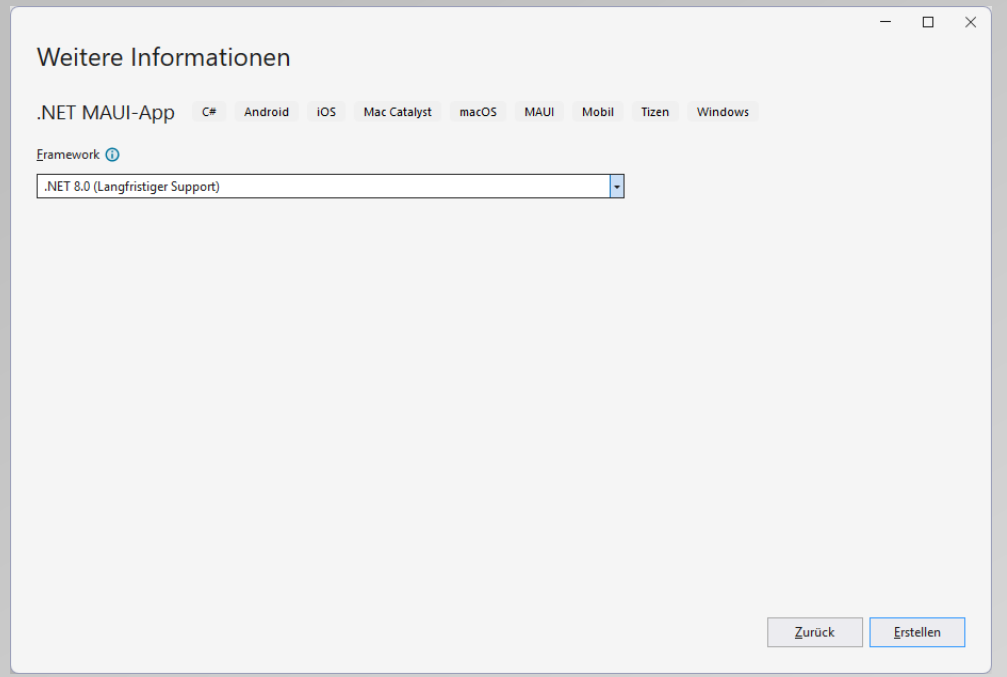
Neues Projekt: Visual Studio

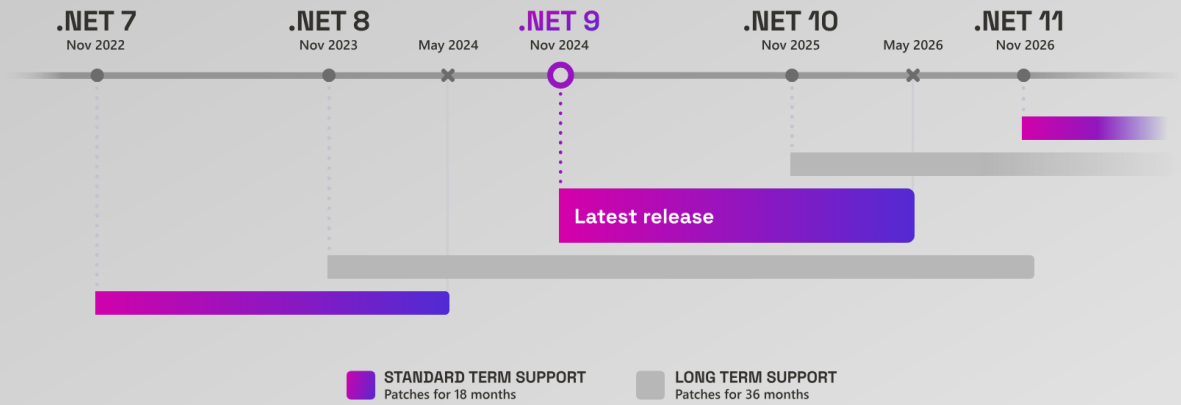


Neues Projekt: Projekteigenschaften



Neues Projekt: .NET-Version

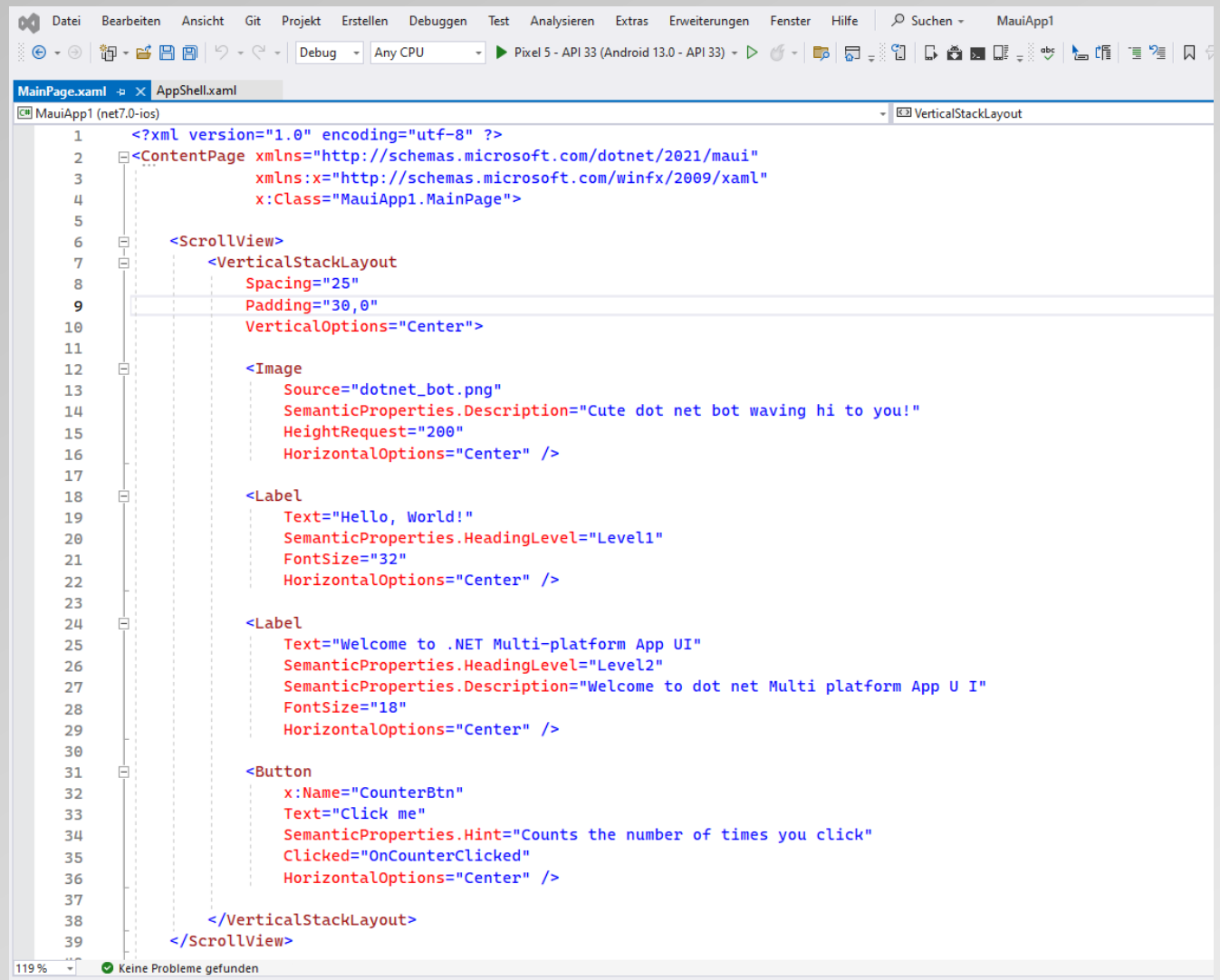




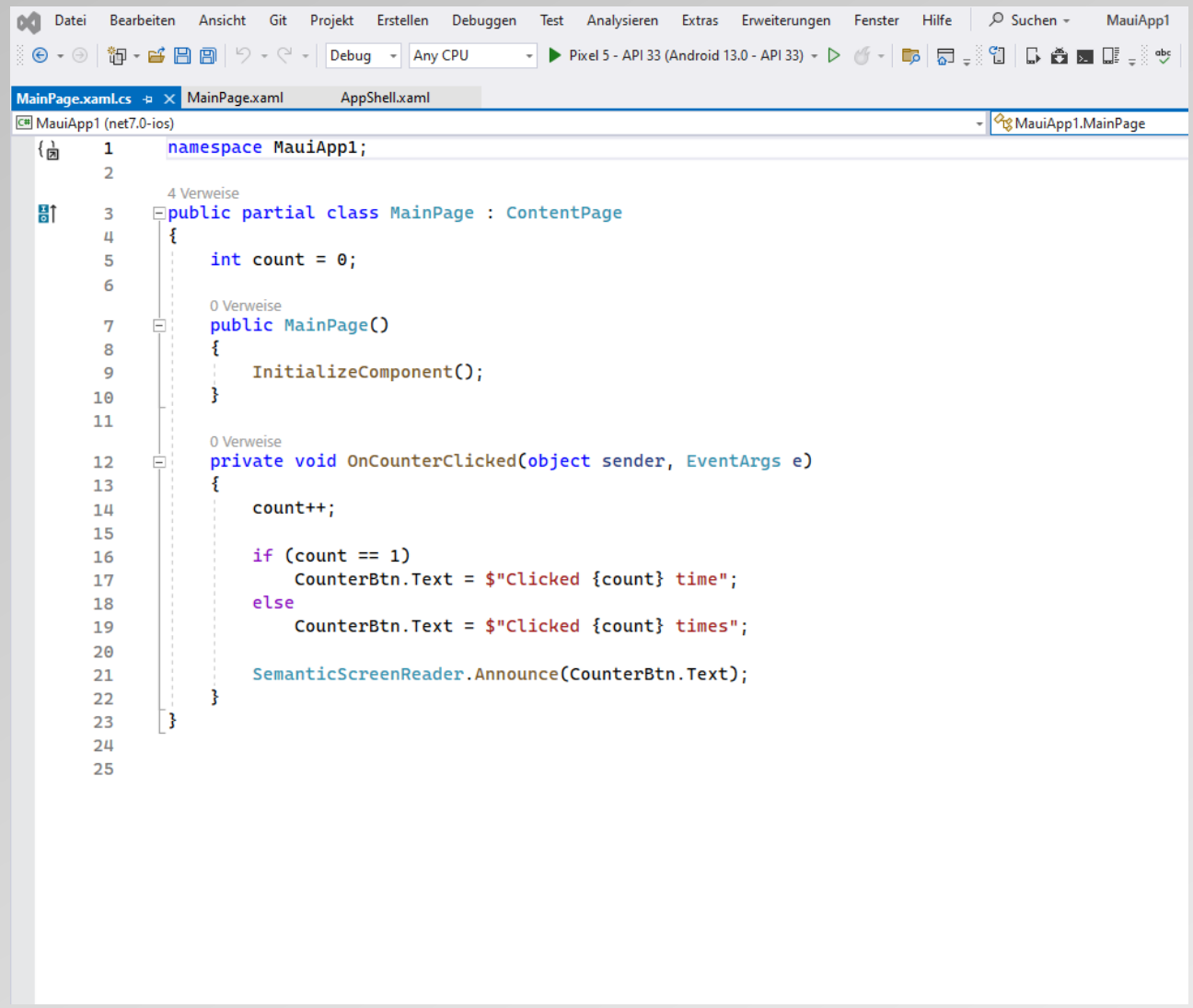
Version	Original release date	Latest patch version	Patch release date	Release type	Support phase	End of support
.NET 9	November 12, 2024	9.0.4	April 8, 2025	STS	Active	May 12, 2026
.NET 8	November 14, 2023	8.0.15	April 8, 2025	LTS	Active	November 10, 2026

Quelle: https://dotnet.microsoft.com/en-us/platform/support/policy

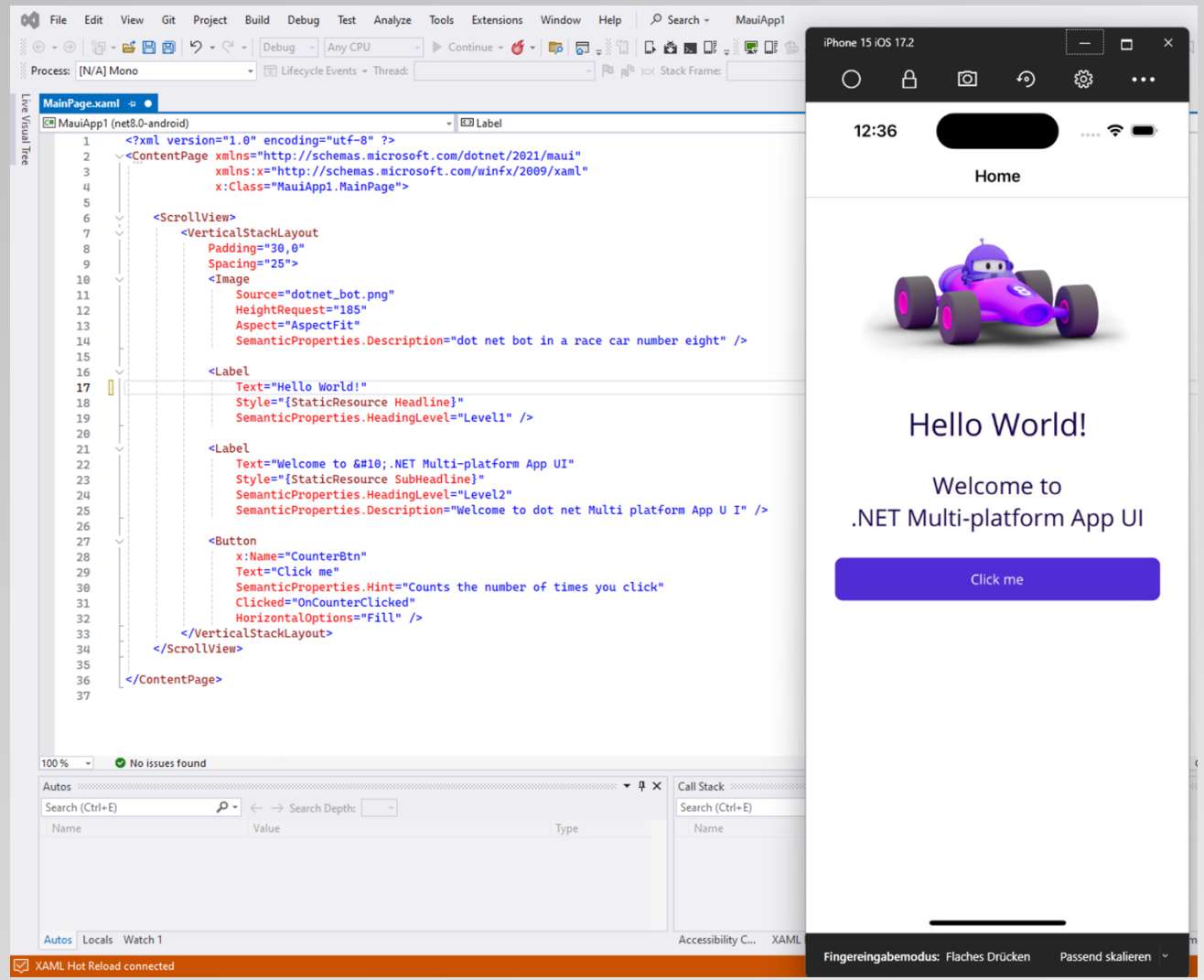
.NET MAUI-App in Visual Studio (XAML-Code)



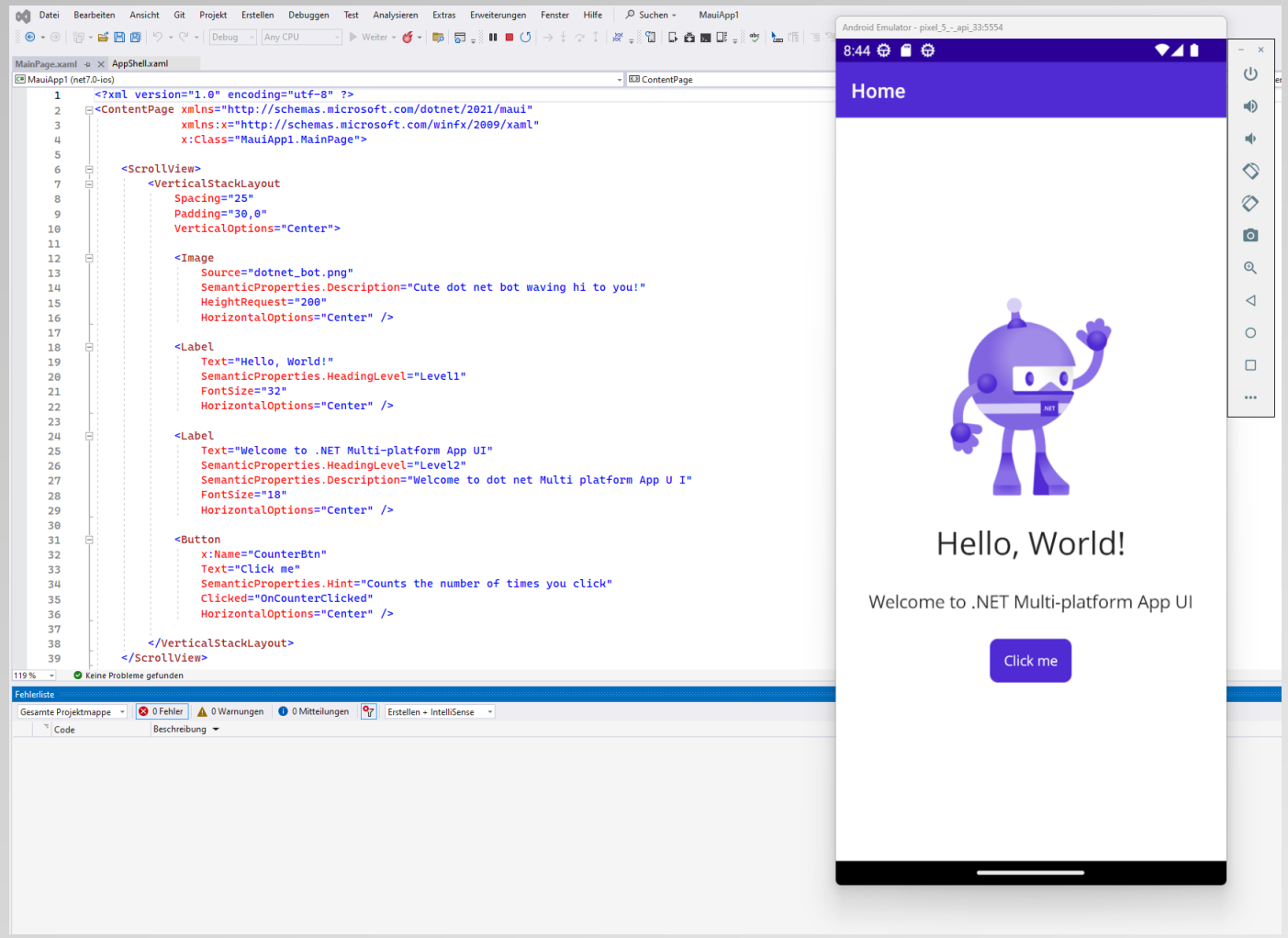
.NET MAUI-App in Visual Studio (C#-Code)



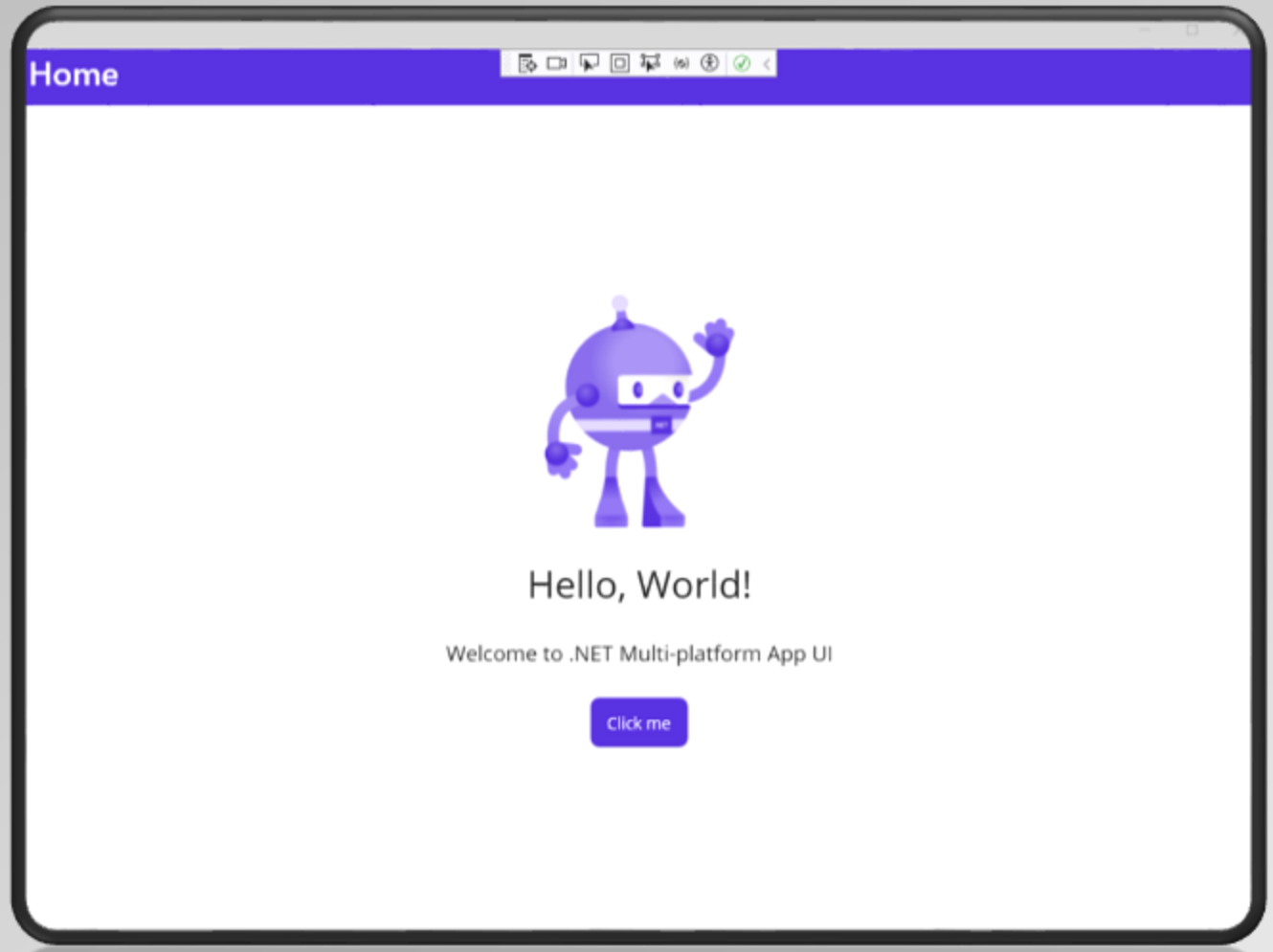
Erster Test: “Hello World” (iOS)



Erster Test: “Hello World” (Android)

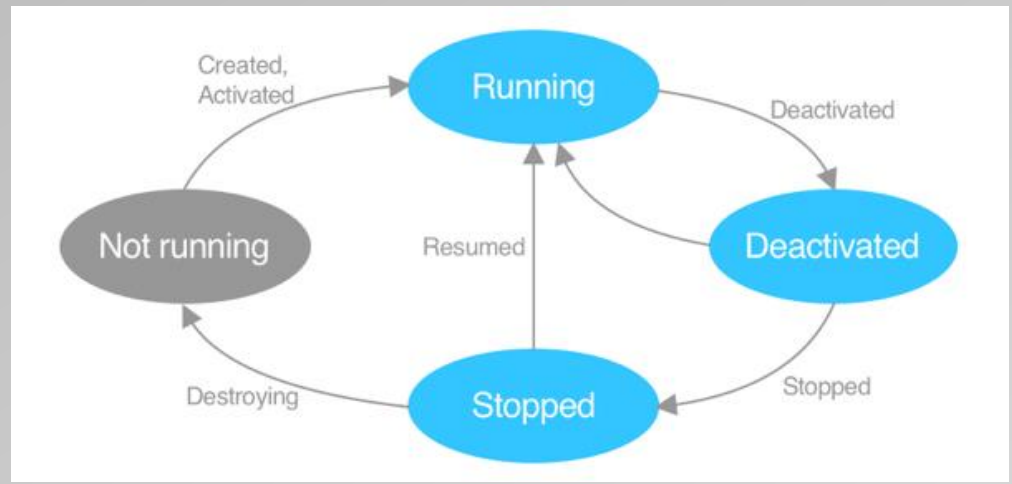


Erster Test: “Hello World” (Windows)



Beispiel:
„HelloWorld“

App-Lebenszyklus



Quelle: https://learn.microsoft.com/de-de/dotnet/maui/fundamentals/app-lifecycle?view=net-maui-9.0

Ereignis	BESCHREIBUNG	Auszuführende Aktion
`Created`	Dieses Ereignis wird ausgelöst, nachdem das systemeigene Fenster erstellt wurde. An diesem Punkt verfügt das plattformübergreifende Fenster über einen systemeigenen Fensterhandler, aber das Fenster ist möglicherweise noch nicht sichtbar.	
`Activated`	Dieses Ereignis wird ausgelöst, wenn das Fenster aktiviert wurde und das fokussierte Fenster ist oder wird.	
`Deactivated`	Dieses Ereignis wird ausgelöst, wenn das Fenster nicht mehr das fokussierte Fenster ist. Das Fenster ist jedoch möglicherweise weiterhin sichtbar.	
`Stopped`	Dieses Ereignis wird ausgelöst, wenn das Fenster nicht mehr sichtbar ist. Es gibt keine Garantie, dass eine App aus diesem Zustand fortgesetzt wird, da sie möglicherweise vom Betriebssystem beendet wird.	Trennen Sie alle Prozesse mit langer Ausführungsdauer, oder brechen Sie alle ausstehenden Anforderungen ab, die Geräteressourcen beanspruchen können.
`Resumed`	Dieses Ereignis wird ausgelöst, wenn eine App nach dem Beenden fortgesetzt wird. Dieses Ereignis wird beim ersten Starten der App nicht ausgelöst und kann nur ausgelöst werden, wenn das `Stopped` Ereignis zuvor ausgelöst wurde.	Abonnieren Sie alle erforderlichen Ereignisse, und aktualisieren Sie alle Inhalte, die sich auf der sichtbaren Seite befinden.
`Destroying`	Dieses Ereignis wird ausgelöst, wenn das systemeigene Fenster zerstört und die Zuordnung aufgehoben wird. Dasselbe plattformübergreifende Fenster kann für ein neues natives Fenster verwendet werden, wenn die App erneut geöffnet wird.	Entfernen Sie alle Ereignisabonnements, die Sie an das systemeigene Fenster angefügt haben.

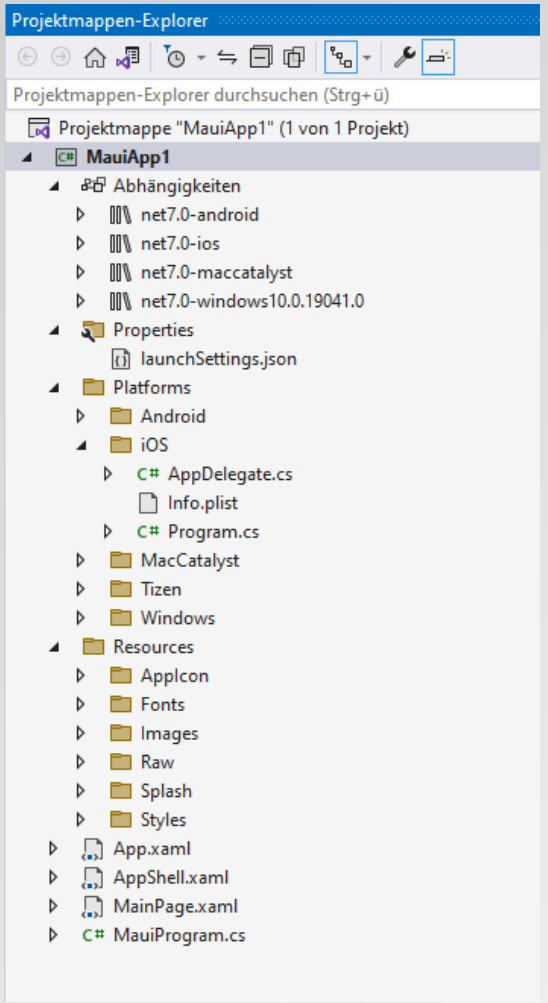
Aufgabe

1. Richten Sie Ihre Arbeitsumgebung für unseren Workshop so ein, dass Sie Apps für iOS und oder Android ausführen können.
2. Finden Sie mit Hilfe von Experimenten Ihre **individuelle** Arbeitsumgebung (Produktivität!)
3. Führen Sie die App auch als Desktop-Anwendung (Windows) aus.

Projektstruktur verstehen

MAUI: Multiplattform-Projekt

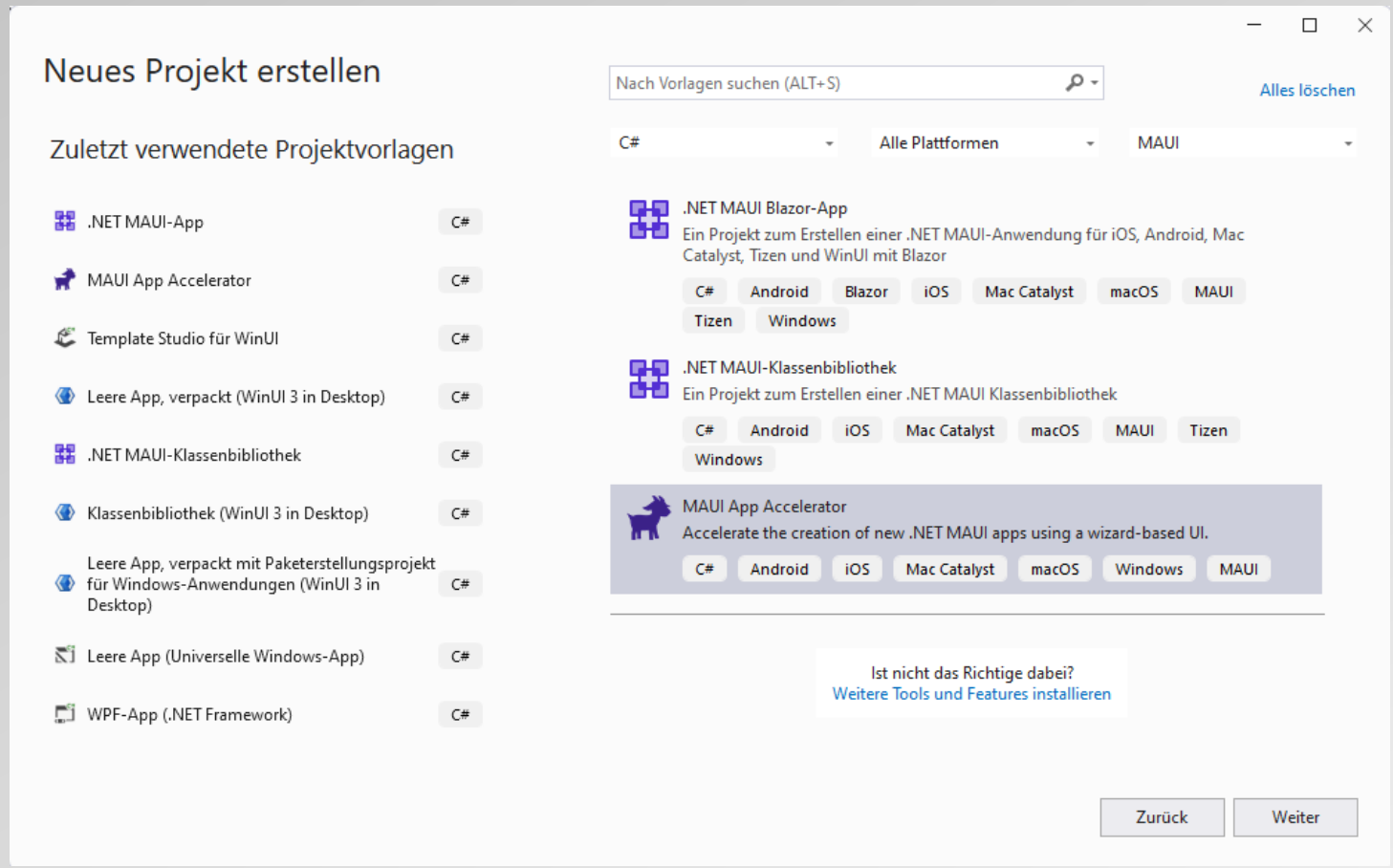
- es gibt für alle Zielsysteme nur ein einziges Projekt
- vereinfachte Auswahl des Debug-Ziels für die Ausführung der .NET MAUI-App
- freigegebene Ressourcendateien innerhalb des Projekts
- ein einzelnes App-Manifest, das den App-Titel, die -ID und die -Version angibt
- Zugriff auf plattformspezifische APIs und Tools bei Bedarf
- ein einzelner plattformübergreifender App-Einstiegspunkt



ppedv

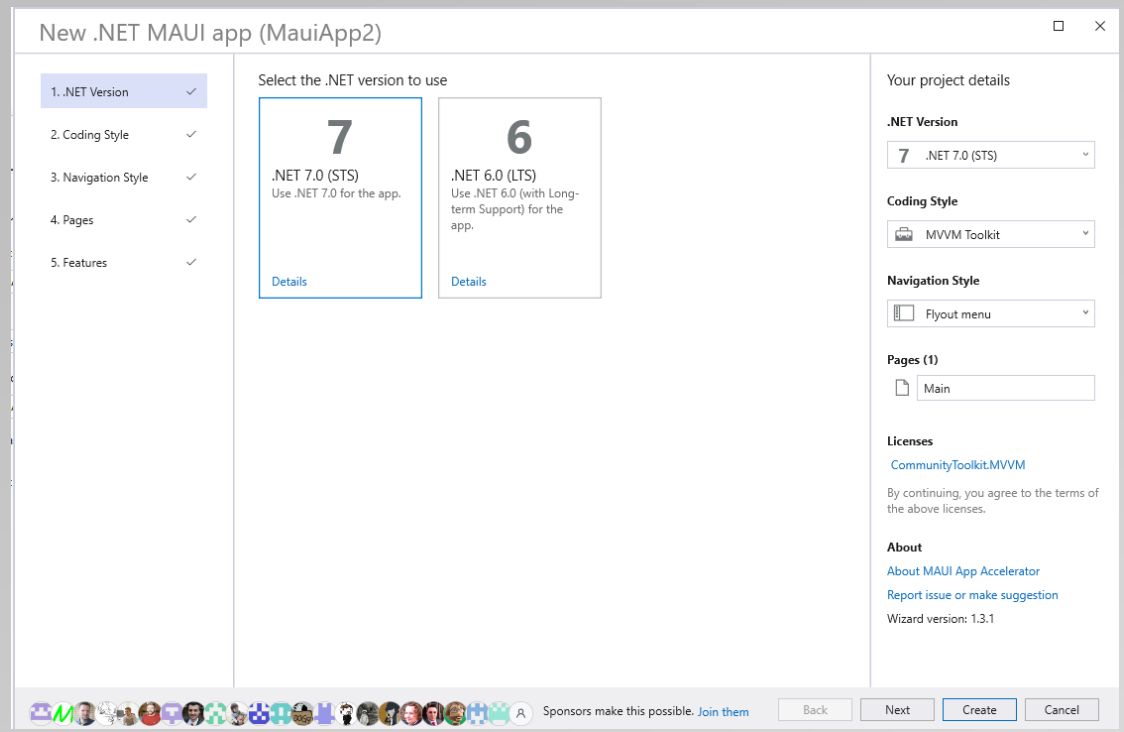
Tipp: MAUI App Accelerator

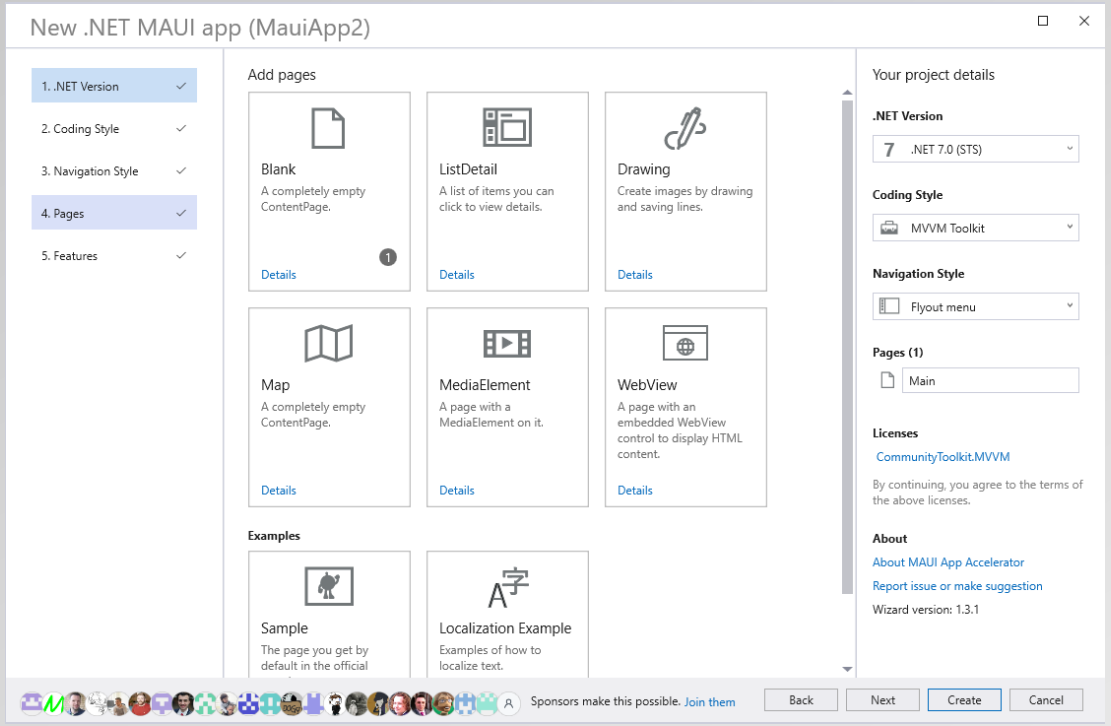
- Wizzard zum Erstellen einer .NET MAUI-App
- Installation als Extension in Visual Studio https://marketplace.visualstudio.com/items?itemName=MattLaceyLtd.MauiAppAccelerator
- generiert das App-Gerüst, inklusive Views und ViewModels



ppedv

Tipp: MAUI App Accelerator





Beispiel:
„HelloWorldAppWizzard "

Beispiel:
„ShellFlyoutSample "

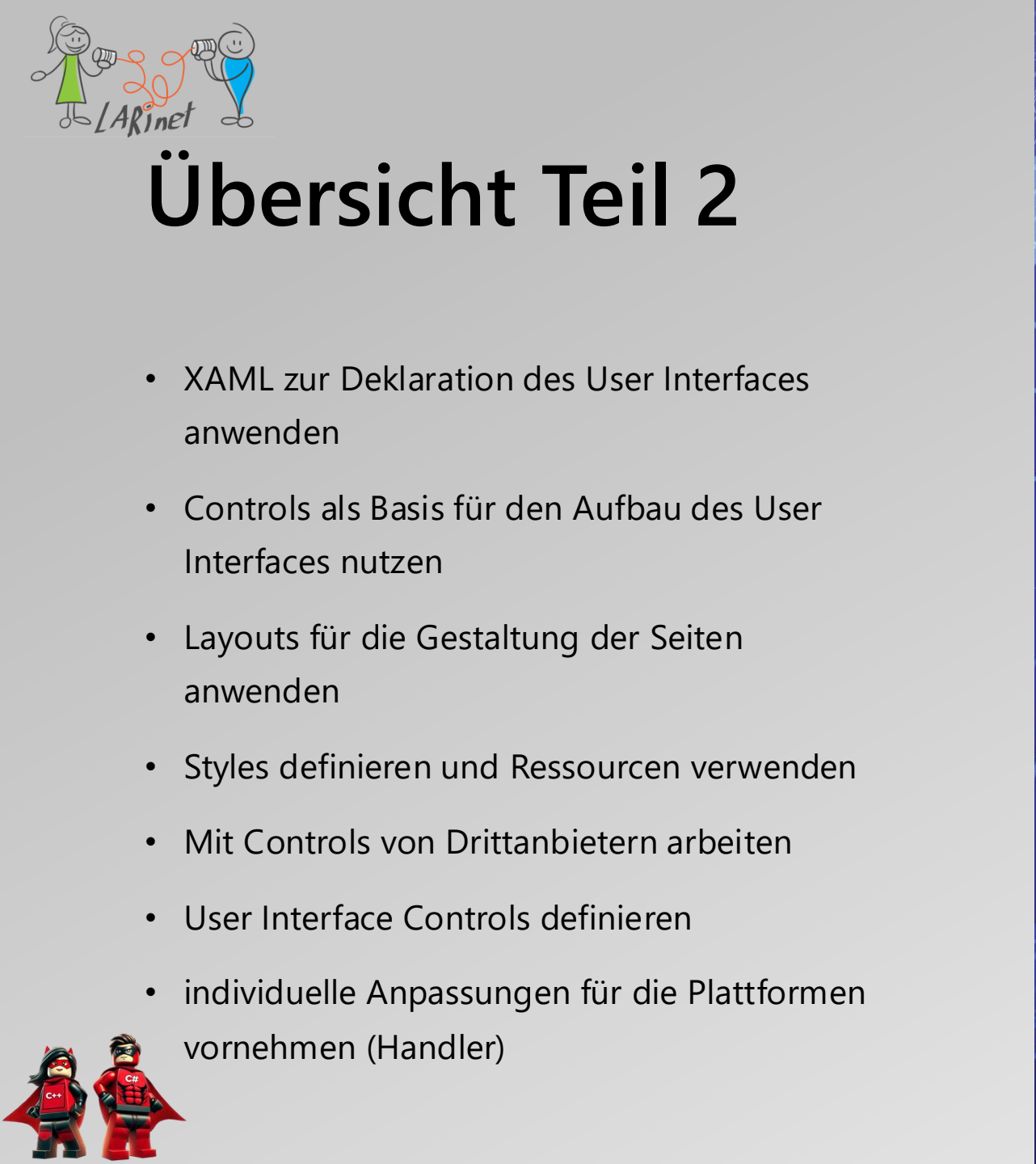
Beispiel:
„ShellTabBarSample "

Übungsaufgabe

1. Erstellen Sie eine erste .NET-MAUI-App.
2. Studieren Sie den Aufbau des Projektes.

Zwischenfazit

- Cross-Plattform-Entwicklung für Mobile und Desktop aus einer Quellcodebasis
- ein Projekt für alle Plattformen
- Entwicklung mit C#
- Visual Studio unter Windows als Entwicklungsumgebung
- deklarative Erstellung des User Interfaces
- Ergebnis: native App
- umfassende Toolunterstützung



Übersicht Teil 2

- XAML zur Deklaration des User Interfaces anwenden
- Controls als Basis für den Aufbau des User Interfaces nutzen
- Layouts für die Gestaltung der Seiten anwenden
- Styles definieren und Ressourcen verwenden
- Mit Controls von Drittanbietern arbeiten
- User Interface Controls definieren
- individuelle Anpassungen für die Plattformen vornehmen (Handler)

XAML zur Deklaration des User Interfaces

Was ist XAML?

- die **eXtensible Application Markup Language (XAML)** ist eine XML-basierte Sprache, die eine Alternative zum Programmieren von Code zum Instanziieren und Initialisieren von Objekten ist und diese Objekte in übergeordneten untergeordneten Hierarchien organisiert
- XAML ermöglicht Entwicklern, Benutzeroberflächen in .NET Multi-Platform App UI (.NET MAUI) -Apps mithilfe von Markup und nicht mit Code zu definieren

Anwendung von XAML

- XAML ist in einer .NET MAUI-App nicht erforderlich, aber es ist der empfohlene Ansatz für die Entwicklung Ihrer Benutzeroberfläche, da es häufig genauer, visuell kohärenter ist Toolunterstützung hat
- XAML eignet sich auch gut für die Verwendung mit dem Model-View-ViewModel (MVVM)-Muster, in dem XAML die Ansicht definiert, die mit ViewModel-Code verknüpft ist, indem XAML-basierte Datenbindungen verwendet werden

XAML

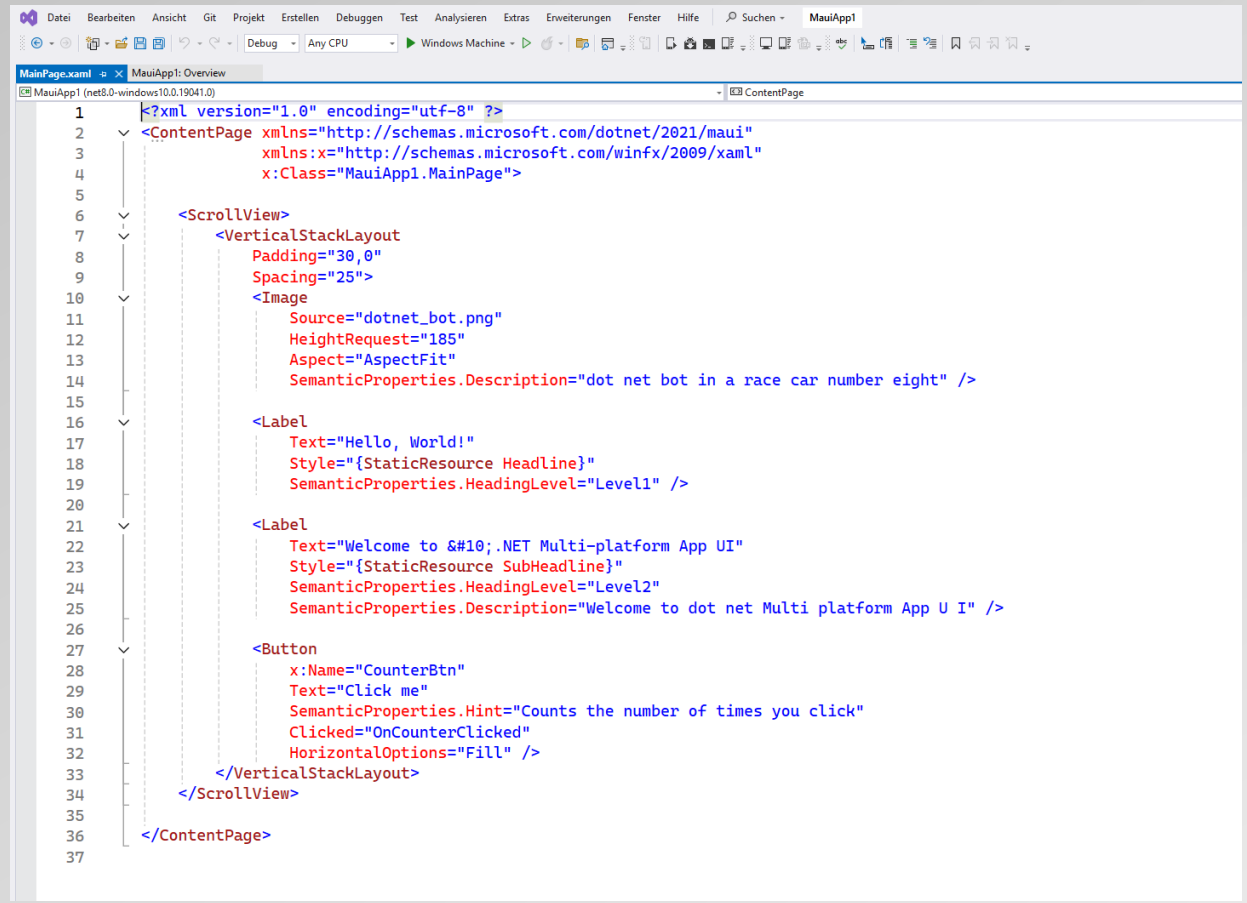
- kann keinen Code enthalten
- kann keine Schleifen für die wiederholte Verarbeitung enthalten
- kann keine bedingte Verarbeitung enthalten
- keine Klassen erzeugen, die **keinen parameterlosen Konstruktor** definieren

Wo ist der Designer?

- … es gibt keinen visuellen Designer zum Erstellen von XAML in .NET MAUI-Apps, d.h. alle XAML-Dateien müssen handgeschrieben sein
- aber Sie können XAML Hot Reload verwenden, um die Benutzeroberfläche während der Bearbeitung zu aktualisieren

Anatomie einer XAML-Datei

- XAML-Datei und Code-Behind-Datei bilden zusammen das Element des User Interfaces, zum Beispiel *MainPage.xaml* und *MainPage.xaml.cs*



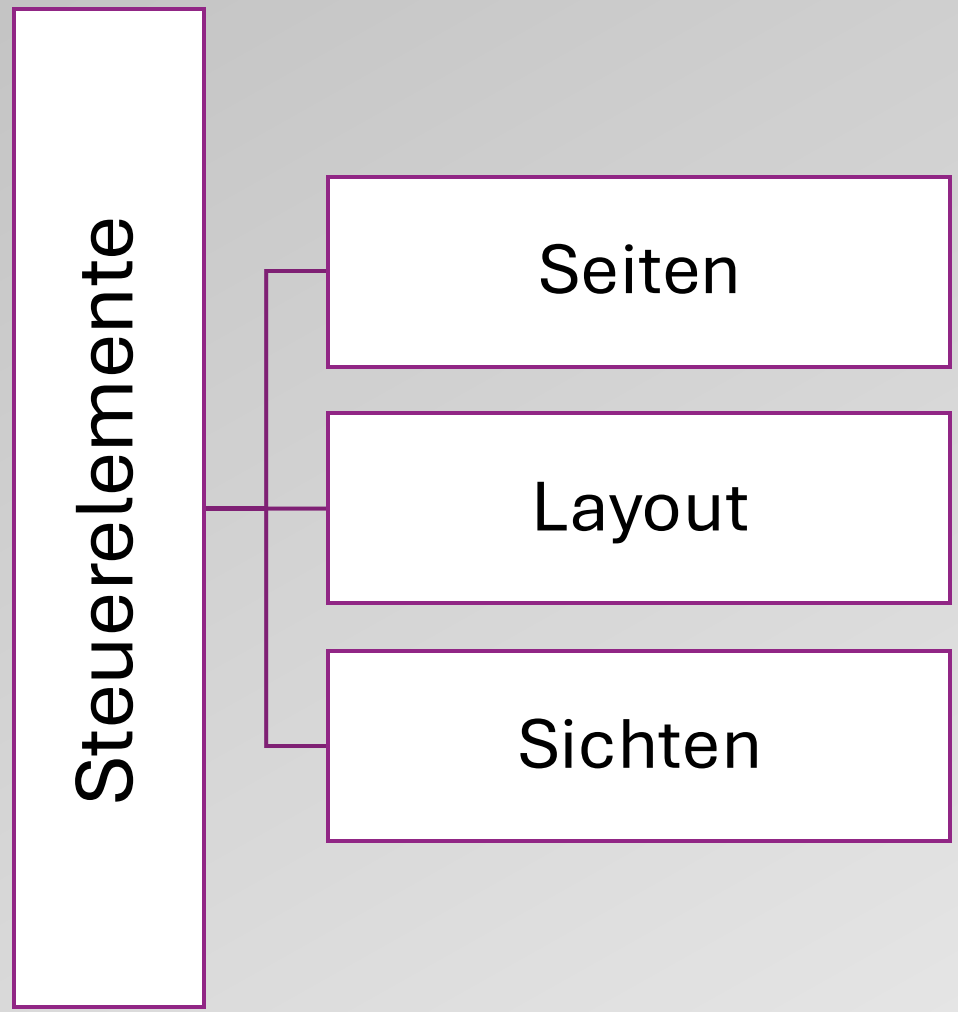
ppedv

Struktur einer XAML-Klasse


```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MyMauiApp.MainPage">
    ...
</ContentPage>
```

- die beiden XML-Namespacedeklarationen (*xmlns*) beziehen sich auf URIs auf microsoft.com. Es gibt jedoch keine Inhalte bei diesen URIs, und sie funktionieren grundsätzlich als Versionsbezeichner.
- die erste XML-Namespacedeklaration bedeutet, dass Tags, die innerhalb der XAML-Datei ohne Präfix definiert sind, auf Klassen in .NET MAUI verweisen
- die zweite Namespacedeklaration definiert ein Präfix von x. Dies wird für mehrere Elemente und Attribute verwendet, die sich in XAML selbst befinden und von anderen Implementierungen von XAML unterstützt werden.

Steuerelemente



Seiten

Seite	BESCHREIBUNG
`ContentPage`	`ContentPage` zeigt eine einzelne Ansicht an und ist der häufigste Seitentyp. Weitere Informationen finden Sie unter ContentPage.
`FlyoutPage`	`FlyoutPage` ist eine Seite, die zwei verwandte Seiten von Informationen verwaltet – eine Flyoutseite, die Elemente darstellt, und eine Detailseite, die Details zu Elementen auf der Flyoutseite darstellt. Weitere Informationen finden Sie unter FlyoutPage.
`NavigationPage`	`NavigationPage` bietet eine hierarchische Navigationserfahrung, in der Sie nach Bedarf durch Seiten navigieren, vorwärts und rückwärts navigieren können. Weitere Informationen finden Sie unter NavigationPage.
`TabbedPage`	`TabbedPage` besteht aus einer Reihe von Seiten, die von Registerkarten oben oder unten auf der Seite navigierbar sind, wobei jede Registerkarte den Seiteninhalt lädt. Weitere Informationen finden Sie unter TabbedPage.

Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/controls/

Layouts

Layout	Beschreibung
`AbsoluteLayout`	`AbsoluteLayout` positioniert untergeordnete Elemente an bestimmten Speicherorten relativ zu seinem übergeordneten Element. Weitere Informationen finden Sie unter AbsoluteLayout.
`BindableLayout`	`BindableLayout` ermöglicht eine beliebige Layoutklasse, seine Inhalte zu generieren, indem er an eine Auflistung von Elementen gebunden ist, wobei die Option zum Festlegen der Darstellung jedes Elements aktiviert ist. Weitere Informationen finden Sie unter BindableLayout.
`FlexLayout`	`FlexLayout` ermöglicht es den untergeordneten Elementen, mit unterschiedlichen Ausrichtungs- und Ausrichtungsoptionen gestapelt oder umgebrochen zu werden. `FlexLayout` basiert auf dem CSS Flexible Box Layout Modul, das als *Flexlayout* oder *Flex-Box* bezeichnet wird. Weitere Informationen finden Sie unter FlexLayout.
`Grid`	`Grid` positioniert seine untergeordneten Elemente in einem Raster von Zeilen und Spalten. Weitere Informationen finden Sie unter Grid.
`HorizontalStackLayout`	`HorizontalStackLayout` positioniert untergeordnete Elemente in einem horizontalen Stapel. Weitere Informationen finden Sie unter HorizontalStackLayout.
`StackLayout`	`StackLayout` positioniert untergeordnete Elemente entweder in einem vertikalen oder horizontalen Stapel. Weitere Informationen finden Sie unter StackLayout.
`VerticalStackLayout`	`VerticalStackLayout` positioniert untergeordnete Elemente in einem vertikalen Stapel. Weitere Informationen finden Sie unter VerticalStackLayout.

Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/controls/

ppedv

Controls (Sichten) als Basis für den Aufbau des UI

Sichten

Sicht	Beschreibung
`ActivityIndicator`	`ActivityIndicator` Verwendet eine Animation, um anzuzeigen, dass die App in einer langen Aktivität tätig ist, ohne dass der Fortschritt angegeben wird. Weitere Informationen finden Sie unter ActivityIndicator.
`BlazorWebView`	`BlazorWebView` Ermöglicht Ihnen das Hosten einer Blazor-Web-App in Ihrer .NET MAUI-App. Weitere Informationen finden Sie unter BlazorWebView.
`Border`	`Border` ist ein Containersteuerelement, das einen Rahmen, einen Hintergrund oder beides um ein anderes Steuerelement zeichnet. Weitere Informationen finden Sie unter Rahmen.
`BoxView`	`BoxView` zeichnet ein Rechteck oder ein Quadrat, eine angegebene Breite, Höhe und Farbe. Weitere Informationen finden Sie unter BoxView.
`Button`	`Button` zeigt Text an und reagiert auf einen Tippen oder Klicken, der eine App angibt, um eine Aufgabe auszuführen. Weitere Informationen finden Sie unter Schaltfläche.
`CarouselView`	`CarouselView` zeigt eine bildlaufbare Liste von Datenelementen an, in der Benutzer wischen, um durch die Sammlung zu navigieren. Weitere Informationen finden Sie unter CarouselView.
`CheckBox`	`CheckBox` ermöglicht Ihnen die Auswahl eines booleschen Werts mithilfe eines Schaltflächentyps, der entweder überprüft oder leer sein kann. Weitere Informationen finden Sie unter CheckBox.
`CollectionView`	`CollectionView` zeigt eine bildlaufbare Liste der ausgewählten Datenelemente an, wobei verschiedene Layoutspezifikationen verwendet werden. Weitere Informationen finden Sie unter CollectionView.
`ContentView`	`ContentView` ist ein Steuerelement, das die Erstellung benutzerdefinierter, wiederverwendbarer Steuerelemente ermöglicht. Weitere Informationen finden Sie unter ContentView.
`DatePicker`	`DatePicker` Ermöglicht ihnen die Auswahl eines Datums mit der Plattformdatumsauswahl. Weitere Informationen finden Sie unter DatePicker.
`Editor`	`Editor` Ermöglicht es Ihnen, mehrere Textzeilen einzugeben und zu bearbeiten. Weitere Informationen finden Sie im Editor.
`Ellipse`	`Ellipse` zeigt eine Ellipse oder einen Kreis an. Weitere Informationen finden Sie unter Ellipse.
`Entry`	`Entry` Ermöglicht ihnen, eine einzelne Textzeile einzugeben und zu bearbeiten. Weitere Informationen finden Sie unter "Eintrag".
`Frame`	`Frame` wird verwendet, um eine Ansicht oder ein Layout mit einem Rahmen umzuschließen, der mit Farbe, Schatten und anderen Optionen konfiguriert werden kann. Weitere Informationen finden Sie unter Frame.
`GraphicsView`	`GraphicsView` ist ein Grafikbereich, auf dem 2D-Grafiken mithilfe von Typen aus dem `Microsoft.Maui.Graphics` Namespace gezeichnet werden können. Weitere Informationen finden Sie unter GraphicsView.
`Image`	`Image` zeigt ein Bild an, das aus einer lokalen Datei, einem URI, einer eingebetteten Ressource oder einem Stream geladen werden kann. Weitere Informationen finden Sie unter Bild.
`ImageButton`	`ImageButton` zeigt ein Bild an und reagiert auf ein Tippen oder Klicken, das eine App angibt, um eine Aufgabe auszuführen. Weitere Informationen finden Sie unter ImageButton.
`IndicatorView`	`IndicatorView` zeigt Indikatoren an, die die Anzahl der Elemente in einem `CarouselView`. Weitere Informationen finden Sie unter "IndikatorView".
`Label`	`Label` Zeigt einzeiligen und mehrzeiligen Text an. Weitere Informationen finden Sie unter Bezeichnung.
`Line`	`Line` zeigt eine Linie von einem Startpunkt bis zu einem Endpunkt an. Weitere Informationen finden Sie unter Zeile.

Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/controls/

Sichten (Fortsetzung)

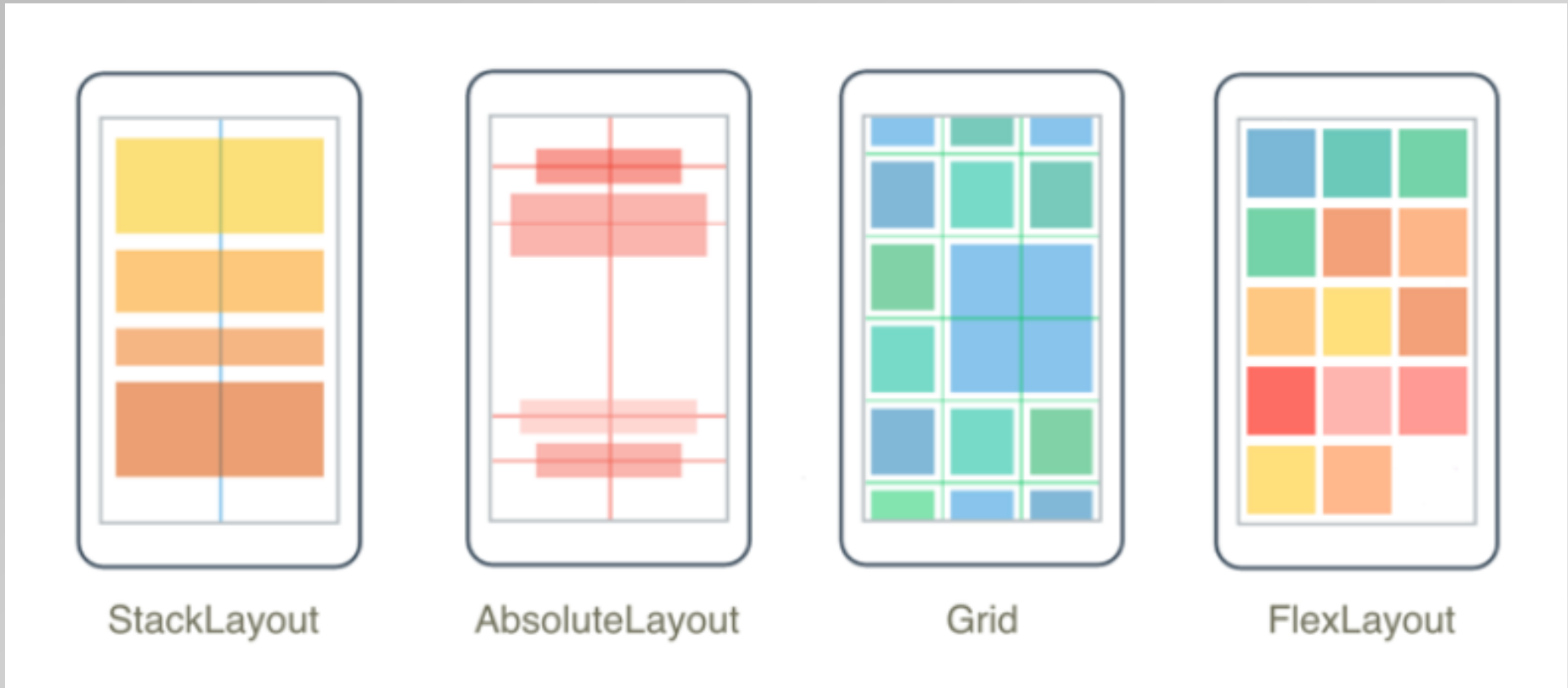
`ListView`	`ListView` zeigt eine bildlaufbare Liste der ausgewählten Datenelemente an. Weitere Informationen finden Sie unter ListView.
`Path`	`Path` Anzeigen von Kurven und komplexen Formen. Weitere Informationen finden Sie unter "Pfad".
`Picker`	`Picker` zeigt eine kurze Liste der Elemente an, aus denen ein Element ausgewählt werden kann. Weitere Informationen finden Sie unter Auswahl.
`Polygon`	`Polygon` zeigt ein Polygon an. Weitere Informationen finden Sie unter Polygon.
`Polyline`	`Polyline` zeigt eine Reihe verbundener gerader Linien an. Weitere Informationen finden Sie unter Polyline.
`ProgressBar`	`ProgressBar` Verwendet eine Animation, um anzuzeigen, dass die App durch eine lange Aktivität fortschreitet. Weitere Informationen finden Sie unter "ProgressBar".
`RadioButton`	`RadioButton` ist eine Art von Schaltfläche, die die Auswahl einer Option aus einem Satz zulässt. Weitere Informationen finden Sie unter RadioButton.
`Rectangle`	`Rectangle` zeigt ein Rechteck oder ein Quadrat an. Weitere Informationen finden Sie unter Rectangle.
`RefreshView`	`RefreshView` ist ein Containersteuerelement, das Pull-to-Refresh-Funktionen für scrollbare Inhalte bereitstellt. Weitere Informationen finden Sie unter RefreshView.
`RoundRectangle`	`RoundRectangle` zeigt ein Rechteck oder ein Quadrat mit abgerundeten Ecken an. Weitere Informationen finden Sie unter Rectangle.
`ScrollView`	`ScrollView` stellt einen Bildlauf des Inhalts bereit, der in der Regel ein Layout ist. Weitere Informationen finden Sie unter ScrollView.
`SearchBar`	`SearchBar` ist ein Benutzereingabesteuerelement, das zum Initiieren einer Suche verwendet wird. Weitere Informationen finden Sie unter SearchBar.
`Slider`	`Slider` ermöglicht es Ihnen, einen `double` Wert aus einem kontinuierlichen Bereich auszuwählen. Weitere Informationen finden Sie unter Slider.
`Stepper`	`Stepper` ermöglicht es Ihnen, einen `double` Wert aus einem Bereich inkrementeller Werte auszuwählen. Weitere Informationen finden Sie unter Stepper.
`SwipeView`	`SwipeView` ist ein Containersteuerelement, das ein Element des Inhalts umschließt, und stellt Kontextmenüelemente bereit, die durch eine Wischbewegung angezeigt werden. Weitere Informationen finden Sie unter SwipeView.
`Switch`	`Switch` ermöglicht es Ihnen, einen booleschen Wert mit einem Typ von Schaltfläche auszuwählen, der entweder ein- oder ausgeschaltet sein kann. Weitere Informationen finden Sie unter Switch.
`TableView`	`TableView` zeigt eine Tabelle mit bildlauffähigen Elementen an, die in Abschnitte gruppiert werden können. Weitere Informationen finden Sie unter TableView.
`TimePicker`	`TimePicker` ermöglicht ihnen die Auswahl einer Zeit mit der Plattformzeitauswahl. Weitere Informationen finden Sie unter TimePicker.
`WebView`	`WebView` Zeigt Webseiten oder lokale HTML-Inhalte an. Weitere Informationen finden Sie unter WebView.

Hinweis: weitere Steuerelemente werden von Drittanbietern zur Verfügung gestellt

Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/controls/

Layouts für die Gestaltung der Seiten anwenden

Layouts

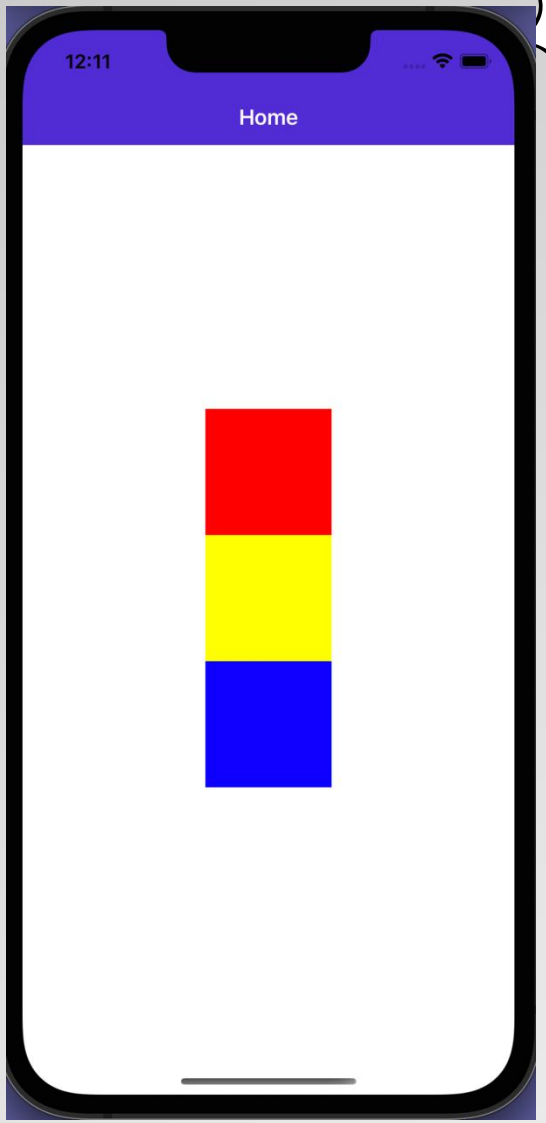


Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/layouts/

StackLayout

```
<?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="Layout.MainPage">
    <StackLayout VerticalOptions="Center" >
        <BoxView WidthRequest="100" HeightRequest="100" Color="Red" />
        <BoxView WidthRequest="100" HeightRequest="100" Color="Yellow" />
        <BoxView  WidthRequest="100" HeightRequest="100" Color="Blue" />
    </StackLayout>
</ContentPage>
```

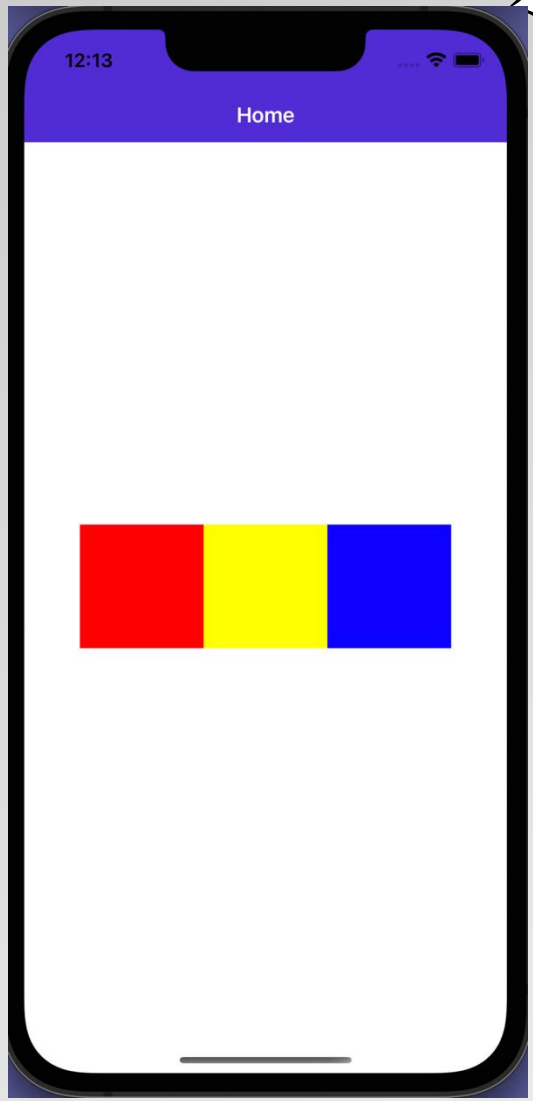
Beispiel: *„Layout “*



ppedv

StackLayout


```
<?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="Layout.MainPage">

    <StackLayout Orientation="Horizontal" HorizontalOptions="Center" >
        <BoxView WidthRequest="100" HeightRequest="100" Color="Red" />
        <BoxView WidthRequest="100" HeightRequest="100" Color="Yellow" />
        <BoxView  WidthRequest="100" HeightRequest="100" Color="Blue" />
    </StackLayout>
</ContentPage>
```

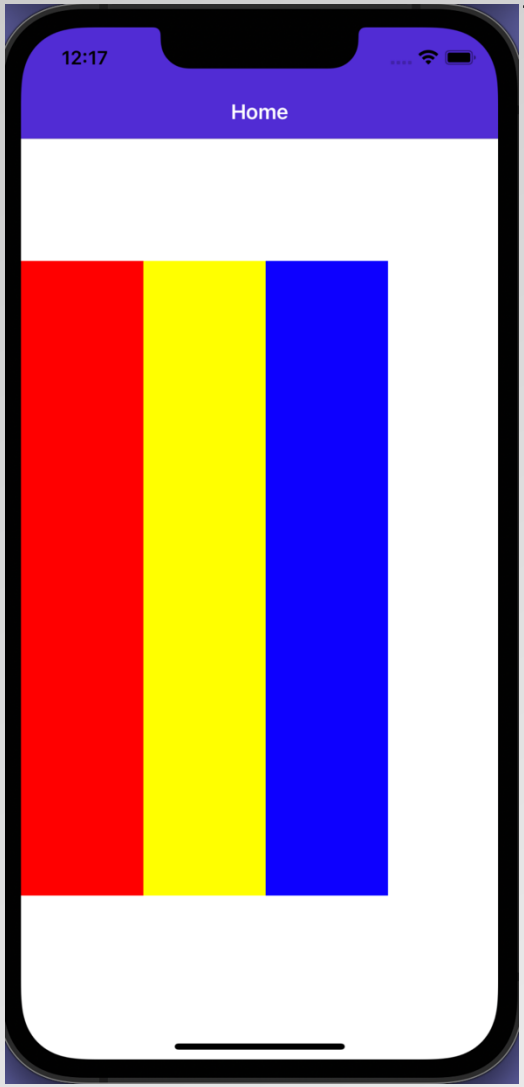


ppedv

HorizontalStackLayout/ VerticalStackLayout

```
<HorizontalStackLayout  Margin="0,100" HorizontalOptions="Start" >
        <BoxView  Color="Red" />
        <BoxView  Color="Yellow" />
        <BoxView   Color="Blue" />
</HorizontalStackLayout>
```

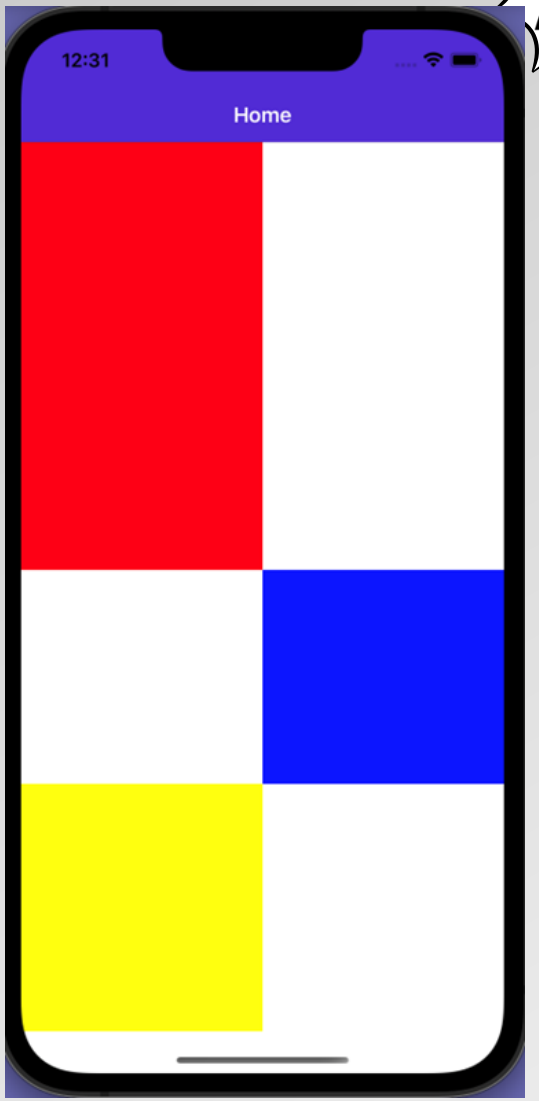
- ein **HorizontalStackLayout**/ **VerticalStackLayout** organisiert untergeordnete Ansichten in einem eindimensionalen horizontalen/ vertikalen Stapel
- **ist leistungsfähigere Alternative** zu einem *StackLayout*.



ppedv

Grid

```
<Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="2*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="200" />
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <BoxView  Color="Red" />
        <BoxView Grid.Row="2" Color="Yellow" />
        <BoxView Grid.Row="1" Grid.Column="1"  Color="Blue" />
</Grid>
```



ppedv

Grid-Beispiel

- in diesem Beispiel besteht kein Zweifel daran, ein Grid-Steuerelement zum Definieren des Layouts zu verwenden
- Sie können alle UI-Controls direkt in einer Zeile und Spalte platzieren
- einige Elemente werden über mehrere Spalten und Zeilen hinweg platziert

Grid-Hinweise

- das Grid Control bietet im Vergleich zu anderen Steuerelementen die beste Leistung
- es ist ideal für adaptives Design
- bevor wir beginnen, sollten wir Zeilen und Spalten zählen
- wir verwenden das *Grid-Layout* anstelle einer Hierarchie von *StackLayouts* (Performance)
- finden Sie einen guten Kompromiss zwischen der Anzahl der Spalten/ Zeilen und der Komplexität der Benutzeroberfläche, um die Leistung zu verbessern und die *Grid*-Komplexität zu reduzieren

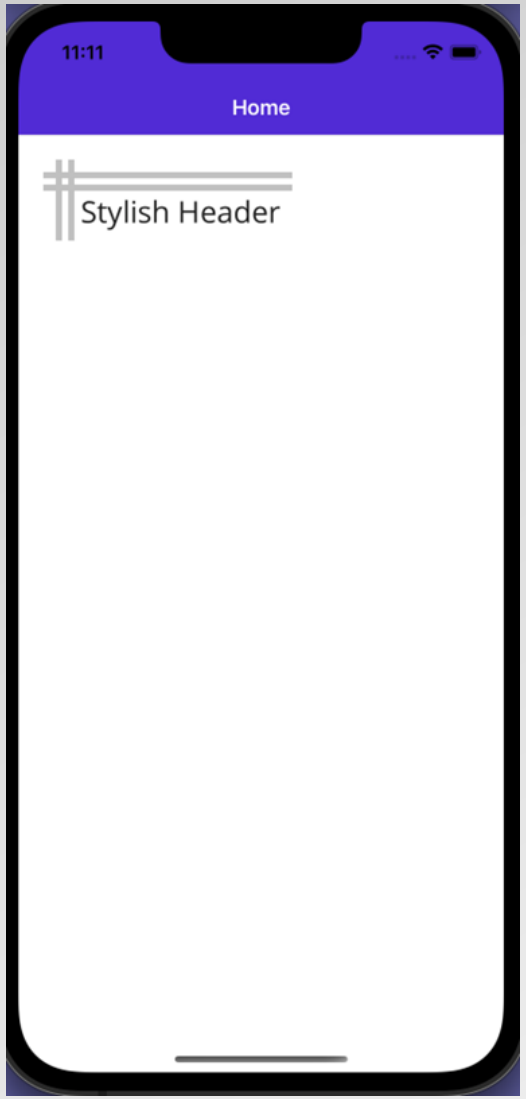
AbsoluteLayout

- ein *AbsoluteLayout* wird verwendet, um untergeordnete Elemente mithilfe expliziter Werte zu positionieren und zu vergrößern
- die Position wird durch die obere linke Ecke des untergeordneten Elements relativ zur oberen linken Ecke des *AbsoluteLayout* in geräteunabhängigen Einheiten angegeben
- *AbsoluteLayout* implementiert auch eine Funktion zur proportionalen Positionierung und Größenanpassung
- darüber hinaus ist *AbsoluteLayout* im Gegensatz zu einigen anderen Layoutklassen in der Lage, Kind-Elemente so zu positionieren, dass sie sich überlappen

Warnung: Die Verwendung von absoluten Werten für die Positionierung und Größenbestimmung von Kind-Elementen kann problematisch sein, da verschiedene Geräte unterschiedliche Bildschirmgrößen und Auflösungen haben.

AbsoluteLayout


```
<AbsoluteLayout Margin="20">
        <BoxView Color="Silver"
                 AbsoluteLayout.LayoutBounds="0, 10, 200, 5" />
        <BoxView Color="Silver"
                 AbsoluteLayout.LayoutBounds="0, 20, 200, 5" />
        <BoxView Color="Silver"
                 AbsoluteLayout.LayoutBounds="10, 0, 5, 65" />
        <BoxView Color="Silver"
                 AbsoluteLayout.LayoutBounds="20, 0, 5, 65" />
        <Label Text="Stylish Header"
               FontSize="24"
               AbsoluteLayout.LayoutBounds="30, 25" />
</AbsoluteLayout>
```

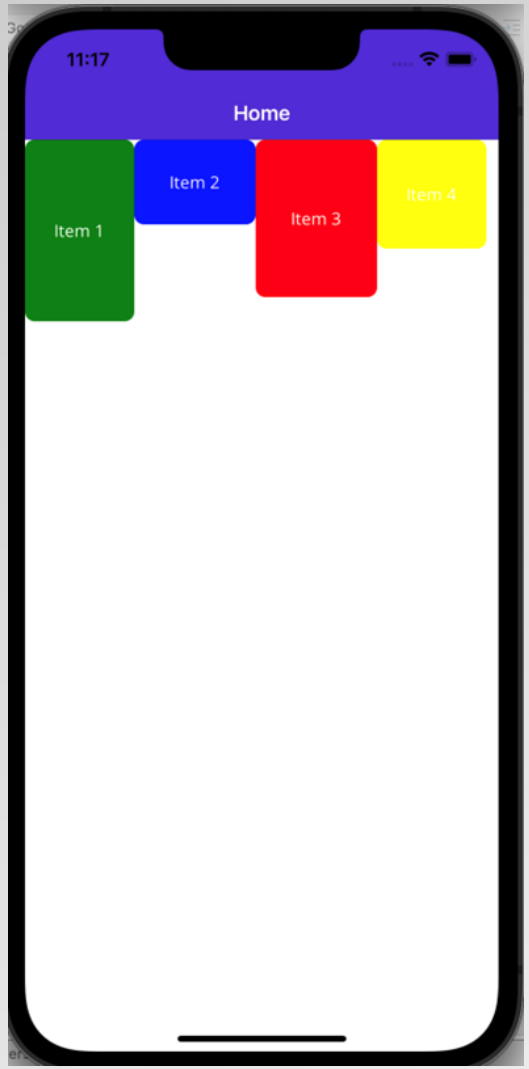


FlexLayout

- das *FlexLayout* basiert auf dem CSS Flexible Box Layout Module, bekannt als Flex-Layout oder Flex-Box. Es wird so genannt, weil es viele flexible Optionen enthält, um Kind-Elemente innerhalb des Layouts anzuordnen
- *FlexLayout* ähnelt dem *StackLayout* darin, dass es seine untergeordneten Elemente horizontal und vertikal in einem Stapel anordnen kann
- das *FlexLayout* ist jedoch auch in der Lage, seine untergeordneten Elemente zu umschließen, wenn zu viele vorhanden sind, um in eine einzelne Zeile oder Spalte zu passen, und bietet außerdem viele Optionen zur Ausrichtung, Ausrichtung und Anpassung an verschiedene Bildschirmgrößen

FlexLayout

```
FlexLayout Direction="Row">
        <Button
            BackgroundColor="Green"
            HeightRequest="150"
            Text="Item 1"
            WidthRequest="90" />
        <Button
            BackgroundColor="Blue"
            HeightRequest="70"
            Text="Item 2"
            WidthRequest="100" />
        <Button
            BackgroundColor="Red"
            HeightRequest="130"
            Text="Item 3"
            WidthRequest="100" />
        <Button
            BackgroundColor="Yellow"
            HeightRequest="90"
            Text="Item 4"
            WidthRequest="90" />
</FlexLayout>
```



ppedv

Best Practices – Layout

Schritte für ein besseres Layout:

1. Erstellen Sie einen visuellen Prototyp für die Struktur (nicht für das Design).
2. Wählen und stellen Sie die Hauptnavigation für die App ein.
3. Legen Sie das Grundlayout für jede Seite fest.
4. Verwenden Sie Platzhalter anstelle von Steuerelementen.
5. Frühe Tests auf Mobilgeräten mit Android und iOS
6. Design kommt später.

Styles definieren und Ressourcen verwenden

Ressourcen

- ein .NET MAUI *ResourceDictionary* ist ein Repository für Ressourcen, die von einer .NET MAUI-App verwendet werden
- typische Ressourcen, die in einem *ResourceDictionary* Formatvorlagen, Steuerelementvorlagen, Datenvorlagen, Konverter und Farben gespeichert werden

AppShell.xaml | Styles.xaml | MainPage.xaml | App.xaml

```xaml
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ShellFlyoutSample"
             x:Class="ShellFlyoutSample.App">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Resources/Colors.xaml" />
                <ResourceDictionary Source="Resources/Styles.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

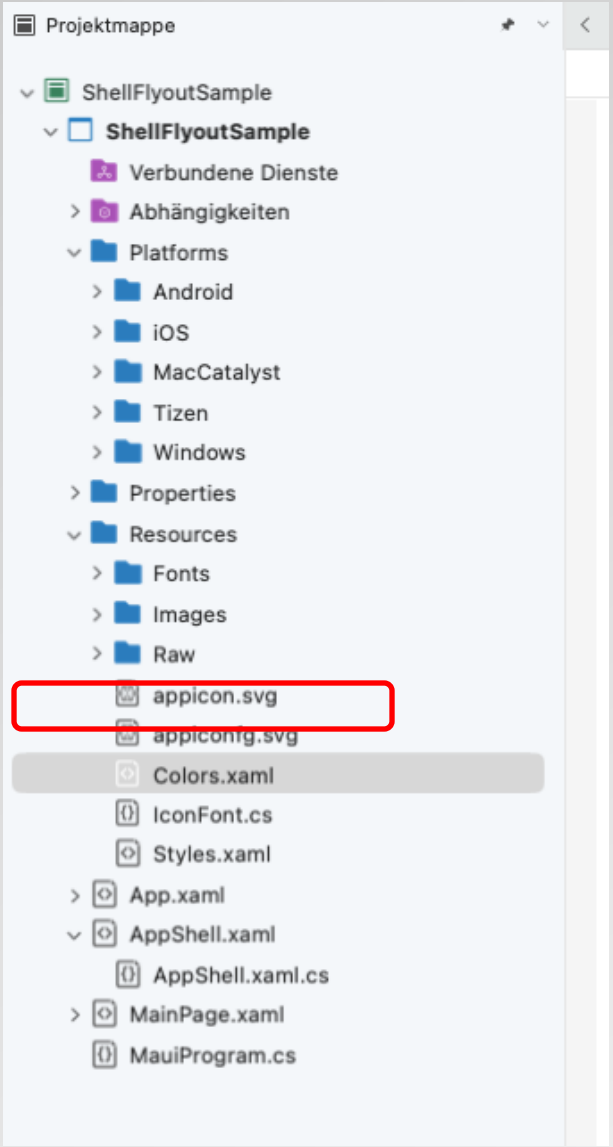
AppShell.xaml | Styles.xaml | MainPage.xaml | App.xaml | Colors.xaml

```xaml
<?xml version="1.0" encoding="UTF-8" ?>
<?xaml-comp compile="true" ?>
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

    <Color x:Key="Primary">#512BD4</Color>
    <Color x:Key="Light">#DFD8F7</Color>
    <Color x:Key="Dark">#2B0B98</Color>
    <Color x:Key="White">White</Color>
    <Color x:Key="Black">Black</Color>
    <Color x:Key="LightGray">#E5E5E1</Color>
    <Color x:Key="MidGray">#969696</Color>
    <Color x:Key="DarkGray">#505050</Color>
    <SolidColorBrush x:Key="PrimaryBrush" Color="{StaticResource Primary}"/>
    <SolidColorBrush x:Key="LightBrush" Color="{StaticResource Light}"/>
    <SolidColorBrush x:Key="DarkBrush" Color="{StaticResource Dark}"/>
    <SolidColorBrush x:Key="WhiteBrush" Color="{StaticResource White}"/>
    <SolidColorBrush x:Key="BlackBrush" Color="{StaticResource Black}"/>
    <SolidColorBrush x:Key="LightGrayBrush" Color="{StaticResource LightGray}"/>
    <SolidColorBrush x:Key="MidGrayBrush" Color="{StaticResource MidGray}"/>
    <SolidColorBrush x:Key="DarkGrayBrush" Color="{StaticResource DarkGray}"/>

    <Color x:Key="Yellow100Accent">#F7B548</Color>
    <Color x:Key="Yellow200Accent">#FFD590</Color>
    <Color x:Key="Yellow300Accent">#FFE5B9</Color>
    <Color x:Key="Cyan100Accent">#28C2D1</Color>
    <Color x:Key="Cyan200Accent">#7BDDEF</Color>
    <Color x:Key="Cyan300Accent">#C3F2F4</Color>
    <Color x:Key="Blue100Accent">#3E8EED</Color>
    <Color x:Key="Blue200Accent">#72ACF1</Color>
    <Color x:Key="Blue300Accent">#A7CBF6</Color>

</ResourceDictionary>
```

ppedv

Bilder – App-Symbol

- In NET MAUI-App-Projekt kann ein App-Symbol an einem einzelnen Speicherort in Ihrem App-Projekt angegeben werden. Zur Erstellungszeit kann dieses Symbol automatisch auf die richtige Auflösung für die Zielplattform und das Gerät angepasst und Ihrem App-Paket hinzugefügt werden.
- Dadurch wird vermieden, dass sie das App-Symbol auf Plattformbasis manuell duplizieren und benennen müssen. Standardmäßig werden Bitmap-Formate (nicht Vektor)-Bildformate von .NET MAUI nicht automatisch geändert.
- Ein .NET MAUI-App-Symbol kann jedes der standardmäßigen Plattformbildformate verwenden, einschließlich skalierbarer Vector Graphics (SVG)-Dateien.



Mit Controls von Drittanbietern arbeiten

UI-Controls von Drittanbietern

- mit Hilfe der Basis-Controls von .NET MAUI kann schon ein Großteil der Anforderungen an eine App umgesetzt werden
- dennoch gibt es Anforderungen an Funktion und Design, welche damit nicht realisiert werden können, beispielsweise ein DataGrid zu Anzeige von einer großen Zahl von Datensätzen aus einer Datenbank
- Entscheidungsgründe:
  - Funktion
  - unterstützte Plattformen (Mobile und Desktop?)
  - Lizenz und Kosten

Anbieter von Controls für .NET MAUI: https://www.syncfusion.com/

.NET MAUI DataViz & UI Controls

GRIDS
- DataGrid

VIEWER
- PDF Viewer PREVIEW

SLIDERS
- Slider
- Range Slider
- Range Selector

DATA VISUALIZATION
- Cartesian Charts
- Circular Charts
- Funnel Charts
- Pyramid Charts
- Radial Gauge
- Linear Gauge
- Maps
- Barcode Generator

MISCELLANEOUS
- Avatar View
- Effects View
- Shimmer

LAYOUT
- Backdrop
- ListView
- Popup
- Text Input Layout
- Expander PREVIEW
- Accordion PREVIEW

DOCUMENT PROCESSING LIBRARIES
- Excel
- PDF
- Word
- PowerPoint

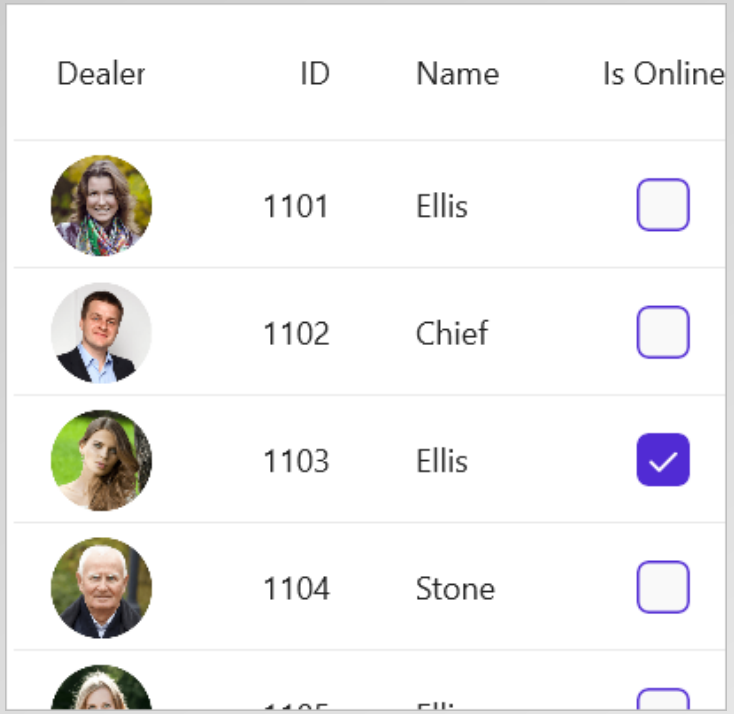
CALENDARS
- Scheduler
- Calendar

BUTTONS
- Chips PREVIEW

NOTIFICATION
- Badge View
- Busy Indicator
- Progress Bar

NAVIGATION
- Tab View

EDITORS
- Autocomplete
- ComboBox
- DataForm
- Masked Entry
- Numeric Entry PREVIEW
- Signature Pad
- Rating
- Image Editor PREVIEW



Quelle: https://www.syncfusion.com/maui-controls/maui-datagrid

ppedv

Anbieter von Controls für .NET MAUI https://www.telerik.com

Explore our .NET MAUI Controls:

DATA CONTROLS
- TreeView NEW
- ListView
- ItemsControl
- DataGrid UPDATED
- DataForm UPDATED

DATA VISUALIZATION
- Rating
- Map
- Gauge
- Barcode

NAVIGATION & LAYOUT
- WrapLayout
- Toolbar
- TabView
- SignaturePad
- SideDrawer
- Expander
- DockLayout
- Accordion

CHARTS
- Spline Chart
- Spline Area Chart
- Scatter Spline Chart
- Scatter Spline Area Chart
- Scatter Point Chart
- Scatter Line Chart
- Scatter Area Chart
- Pie Chart
- OHLC Chart
- Line Chart
- Financial Chart
- Donut Chart
- Charts Overview
- Candlestick Chart
- Bar Chart
- Area Chart

EDITORS
- TimeSpanPicker
- TimePicker
- TemplatedPicker
- RichTextEditor NEW
- NumericInput
- MaskedEntry
- ListPicker
- ImageEditor
- Entry
- DateTimePicker
- DatePicker
- ComboBox
- AutoComplete

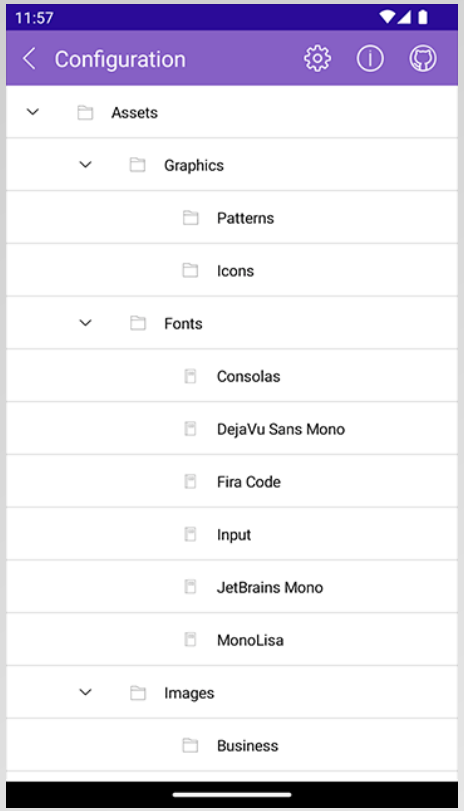
BUTTONS
- Segmented Control
- CheckBox
- Button

INTERACTIVITY & UX
- SlideView NEW
- ProgressBar
- Popup
- Path
- Chat (Conversational UI) NEW
- BusyIndicator
- Border
- BadgeView

PDF VIEWER
- PDF Viewer NEW

CALENDAR
- Calendar NEW

DOCUMENT PROCESSING
- Zip Library
- WordProcessing
- SpreadStreamProcessing UPDATED
- SpreadProcessing
- PDFProcessing UPDATED



Quelle: https://www.telerik.com/maui-ui

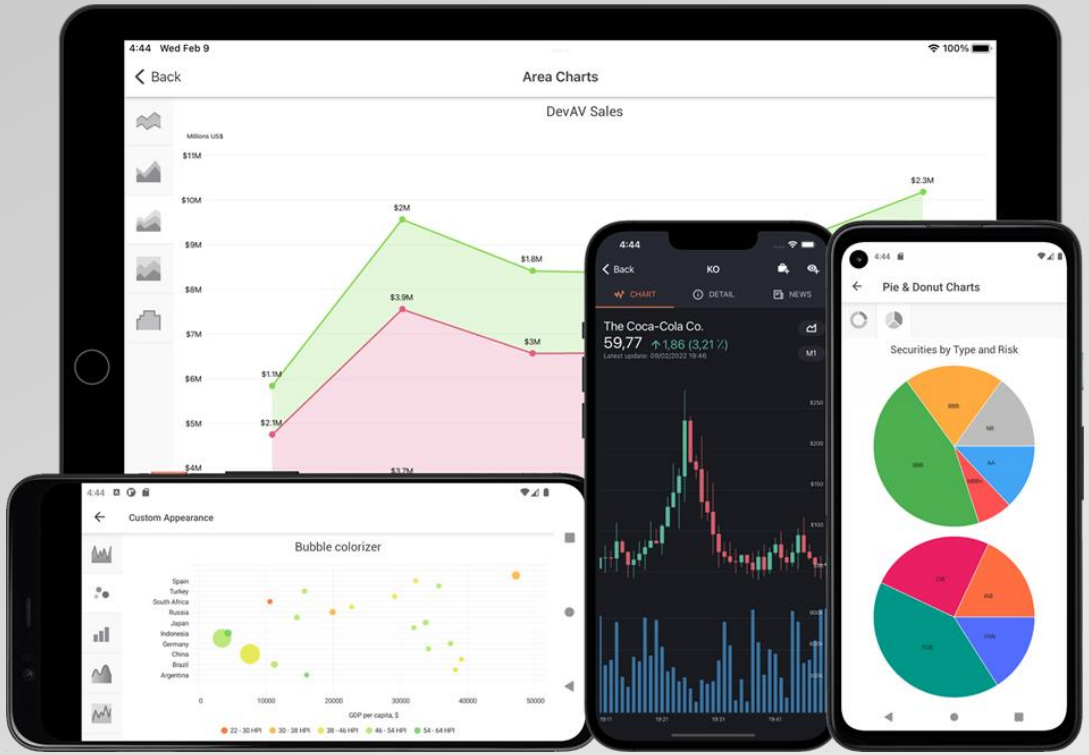
ppedv

Anbieter von Controls für .NET MAUI https://www.devexpress.com/

Controls

DevExpress provides the following .NET MAUI controls:

- Data Grid
- Data Editors
- Data Form
- Collection View
- Charts
- Tab View
- Scheduler
- Popup
- Bottom Sheet
- Form Items
- Shimmer View



Quelle: https://www.telerik.com/maui-ui

ppedv

UI-Controls von Drittanbietern: Integration

NuGet-Package suchen und einbinden

Initialisierung und Setup der Library in der App

Einbindung des Controls im XAML-Code

UI Controls definieren

User Interface-Controls definieren

- Mithilfe von .NET MAUI-Steuerelementvorlagen können Sie die visuelle Struktur von *ContentView* abgeleiteten benutzerdefinierten Steuerelementen definieren.
- Die Steuerelementvorlage kann anschließend in einer anderen *ContentView* genutzt werden
- Die Basisklasse ist die Klasse *ContentView*.

UI-Controls von Drittanbietern

Definition des Steuerelements (XAML und C#)

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ControlTemplateDemos.Controls.CardViewUI"
             x:Name="this">
    <Frame BindingContext="{x:Reference this}"
           ...>
        <Grid>
        ....
        </Grid>
 </Frame>
</ContentView>
```

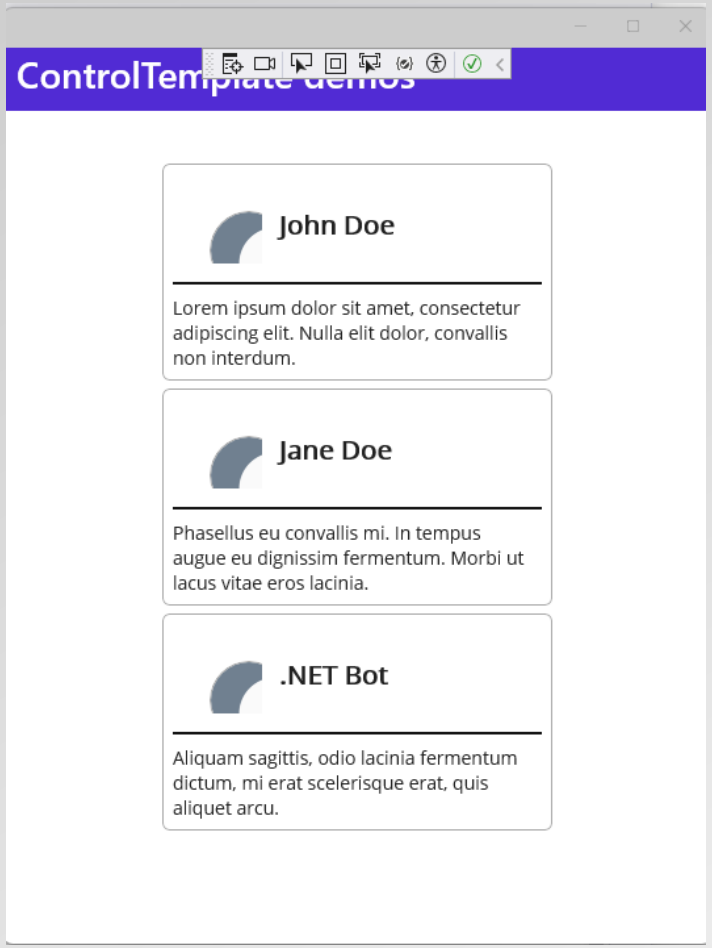
```
public partial class CardViewUI : ContentView
  {
     public static readonly BindableProperty CardTitleProperty = BindableProperty.Create(nameof(CardTitle), typeof(string), typeof(CardViewUI), string.Empty);


     public string CardTitle
     {
        get => (string)GetValue(CardTitleProperty);
        set => SetValue(CardTitleProperty, value);
     }
     ...
```

UI-Controls von Drittanbietern

Verwendung des Steuerelements (XAML)

```
<ContentPage
    x:Class="ControlTemplateDemos.MainPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:controls="clr-namespace:ControlTemplateDemos.Controls"
    xmlns:local="clr-namespace:ControlTemplateDemos"
    Title="ControlTemplate demos"
    Padding="10">
    <StackLayout Margin="30">
        <controls:CardViewUI
            BorderColor="DarkGray"
            CardDescription="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla elit dolor, convallis non interdum."
            CardTitle="John Doe"
            IconBackgroundColor="SlateGray"
            IconImageSource="user.png" />
        <controls:CardViewUI
            BorderColor="DarkGray"
            CardDescription="Phasellus eu convallis mi. In tempus augue eu dignissim fermentum. Morbi ut lacus vitae eros lacinia."
            CardTitle="Jane Doe"
            IconBackgroundColor="SlateGray"
            IconImageSource="user.png" />
        ...
    </StackLayout>
</ContentPage>
```

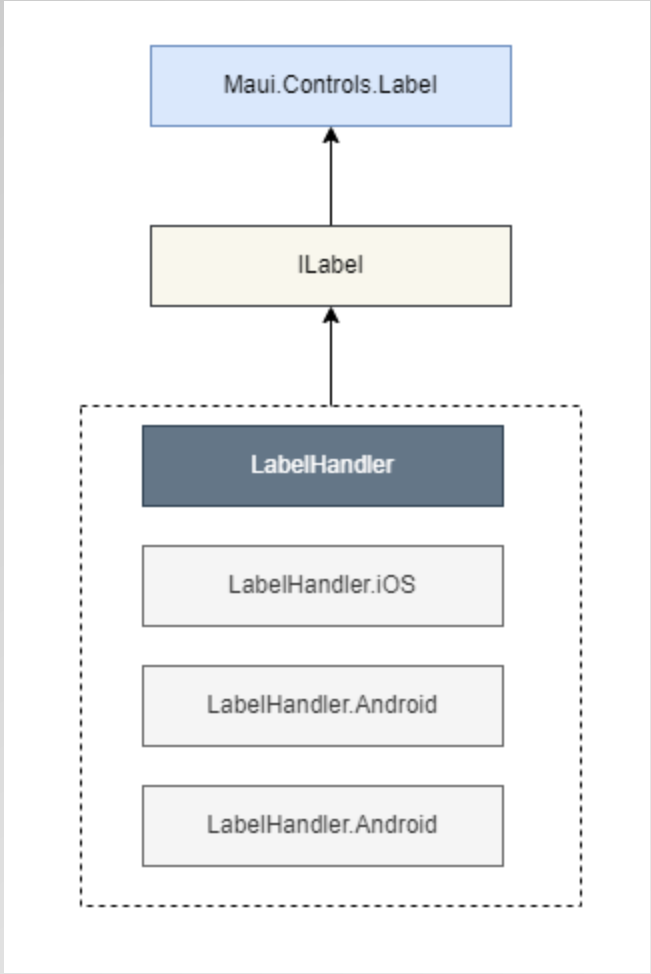
Beispiel:
„ControlTemplateDemos“



Individuelle Anpassungen für die Plattformen vornehmen (Handler)

Plattformspezifische Anpassungen

- wir müssen im Quellcode anhand von Compiler-Direktiven zwischen den Plattformen unterscheiden
- dazu „klinken“ wir uns direkt in den Quellcode der betreffenden Seite oder der gesamten App ein und passen den Handler an



Plattformspezifische Anpassungen

Funktionsweise


```
public MainPage()
{
    InitializeComponent();
    Microsoft.Maui.Handlers.EntryHandler.EntryMapper.AppendToMapping(nameof(IView.Background), (h, v) =>
    {
            #if __ANDROID__
                    h.NativeView.SetBackgroundColor(Colors.Red.ToNative());
            #elif __IOS__
                    h.NativeView.BackgroundColor = Colors.Green.ToNative();
            #endif
    });
}
```

mittels:

#if __ANDROID__

und

#elif __IOS__

wird zwischen Android und iOS unterschieden

Zwischenfazit

- das User Interface wird über alle Plattformen hinweg in XAML definiert
- die Basiselemente sind Seiten, Sichten (Controls) und Layout
- Layout-Container als Ausgangspunkt für eine responsive Gestaltung
- zentrale Ablage von Ressourcen, beispielsweise Bilder in der App
- Plattformanpassungen sind möglich
- das User Interface kann direkt an die Logik gebunden werden (Datenbindung)

Übersicht Teil 3

- Das Entwurfsmuster MVVM kennenlernen und anwenden
- Datenbindung verstehen, definieren und einsetzen
- Commands definieren und verwenden
- Einsatz eines MVVM-Frameworks
- Master-/ Detail-View mit Datenbindung

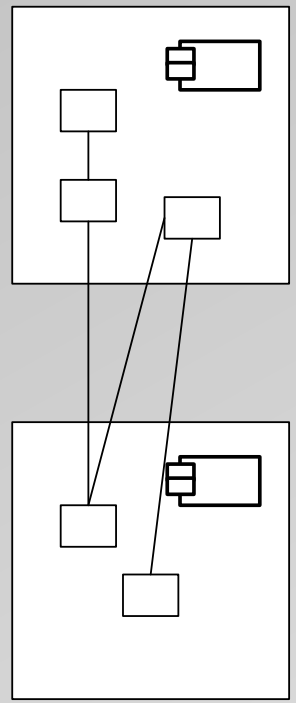
Das Entwurfsmuster MVVM kennenlernen und anwenden

Bindung und Kopplung

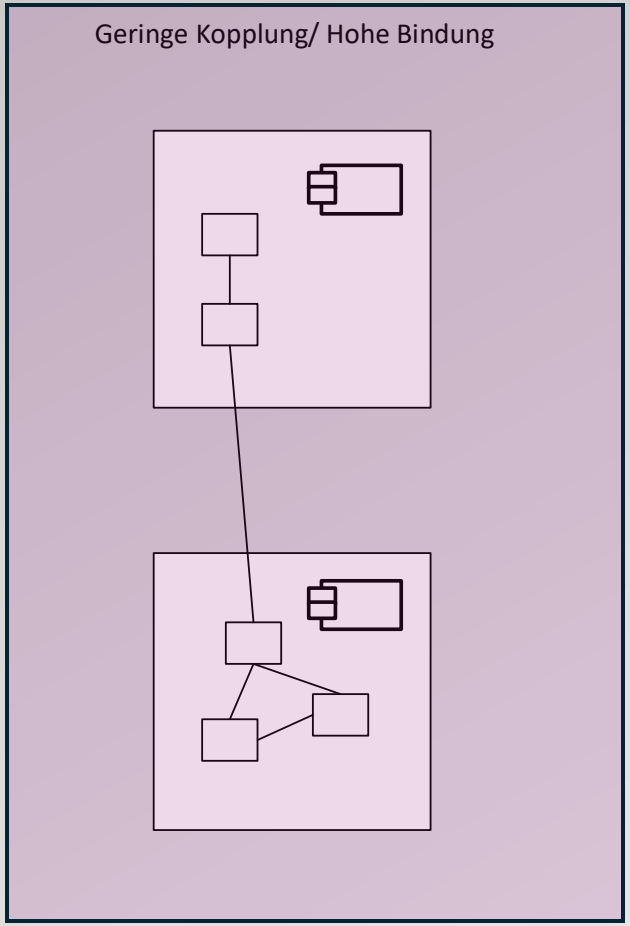
- Die Bindung (*Cohesion*) innerhalb einer Systemkomponente und die Kopplung (*Coupling*) der Systemkomponenten untereinander bestimmen die Struktur eines Softwaresystems.
- Dabei sollen die Beziehungen zwischen den Elementen der Systemkomponenten möglichst intensiv sein.
- Dagegen sollte die Kopplung der Systemkomponenten untereinander möglichst gering sein.
- Mit anderen Worten: Alle Funktionen innerhalb eines Moduls sollen der Erfüllung einer Aufgabe dienen. Unterschiedliche Module sollen dagegen unterschiedliche Aufgaben haben.
- Mit einer hohen Bindung innerhalb der Komponenten und einer losen Kopplung zwischen den Komponenten schafft man die Voraussetzungen für eine Wiederverwendung der Komponenten und für eine leichtere Wartung des Gesamtsystems.
- Beispielsweise ist es nur so möglich, eine Komponente gegen eine neuere Version mit verbesserter Leistung auszutauschen, ohne dass weitere Änderungen am Gesamtsystem durchgeführt werden müssen. Ebenso können Komponenten eines Systems in einem anderen System ohne Anpassungen wiederverwendet werden.
- Beispiel: Die Teilsysteme Kunden- und Artikelverwaltung sind unabhängig voneinander und nur über das Bestellmanagement verbunden. Möchte man Änderungen an der Kundenverwaltung durchführen, beispielsweise weil es notwendig ist, neue Datenfelder aufzunehmen, hat das keine Auswirkungen auf den Rest des Systems

Hohe Kopplung/geringe Bindung

Hohe Kopplung/ Geringe Bindung

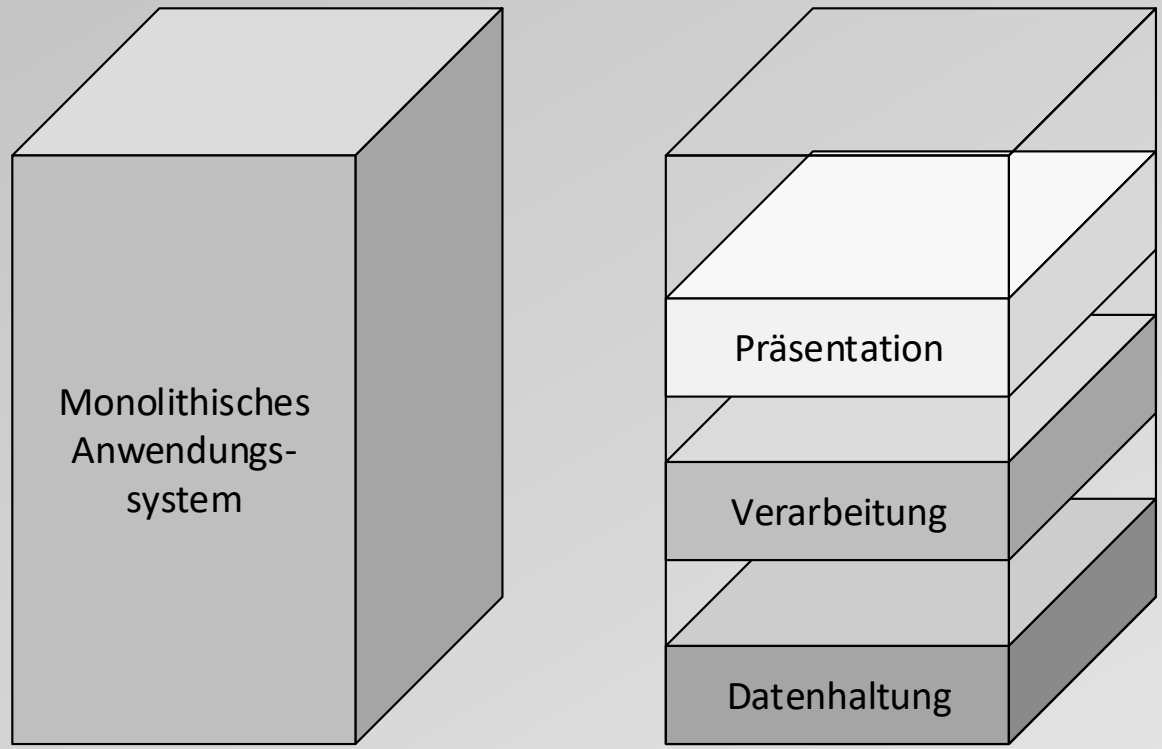


Geringe Kopplung/ Hohe Bindung



ppedv

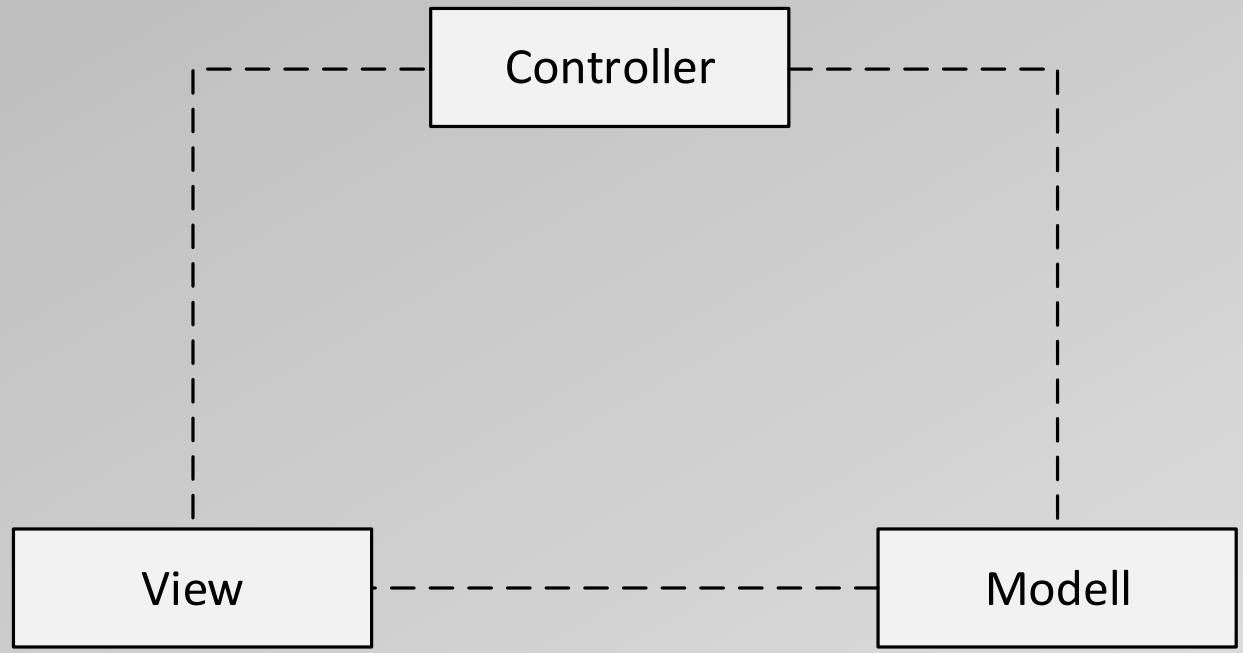
Monolith versus Schichtenarchitektur



Model-View-Controller (MVC)-Muster

- Model-View-Controller-Muster (MVC-Muster) beschäftigt sich mit der Schichtentrennung der Applikation.
- es geht um die Bereitstellung der Businesslogik und ihre Interaktion mit der Benutzeroberfläche.
- dazu trennt das MVC-Muster die Schichten Datenhaltung (*Model*), Steuerung (*Controller*) und Präsentation (*View*) voneinander.

Das Grundprinzip des MVC-Entwurfsmusters

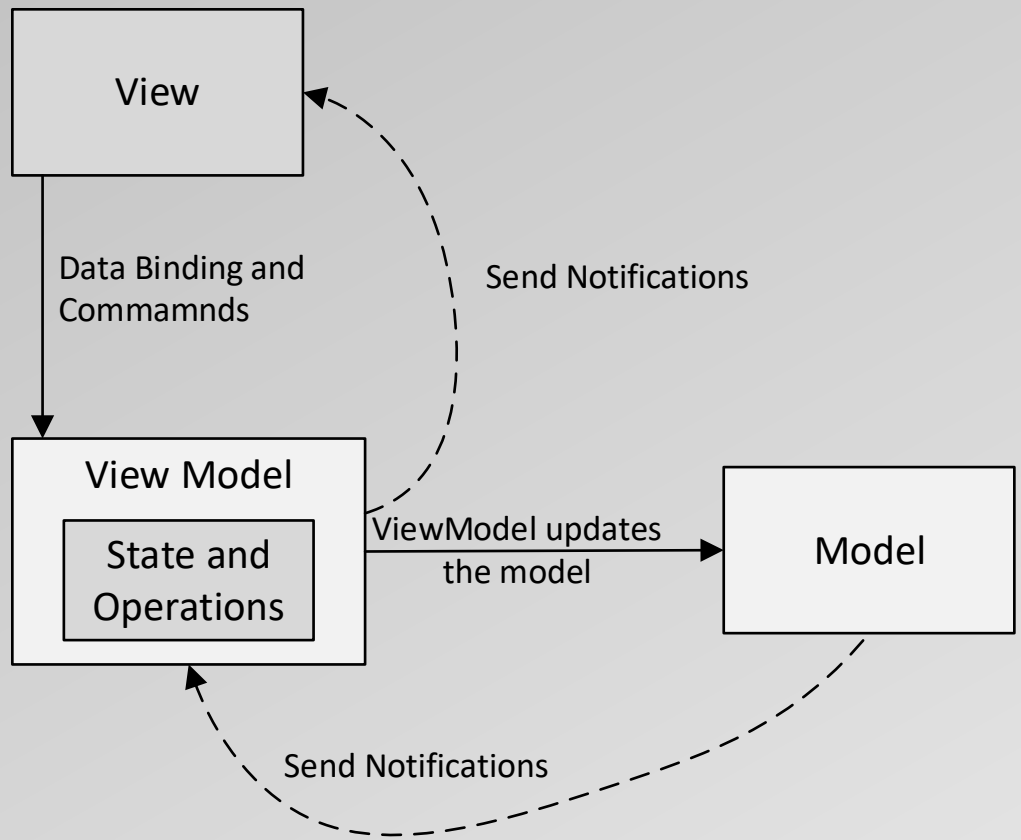


- *Model*: steht für die Daten der Anwendung, unter anderem ist es für deren Speicherung verantwortlich.
- *View*: hat die Aufgabe, die Daten des Models darzustellen und Benutzereingaben zu verarbeiten.
- *Controller*: verbindet das Model mit der View. Er organisiert den Datenaustausch zwischen beiden Elementen

Das Model View ViewModel (MVVM)-Muster

- Wird bei der Bindung der Benutzeroberfläche an die Programlogik im Umfeld der Programmierung mithilfe des .NET-Frameworks verwendet.
- Es kann u.a. in folgenden Technologien angewendet werden:
  - Desktop-Applikationen auf der Basis von *Windows Presentation Foundation (WPF)*
  - Desktop-Applikationen auf der Basis von *WinUI 3*
  - Apps auf der Basis der Universal Windows Platform (*UWP*)
  - plattformübergreifende Apps für iOS, Android und Windows auf der Basis von *MAUI*

Das MVVM-Muster



Quelle: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN

MVVM: View

- Die Benutzeroberfläche wird deklarativ erstellt.
- Dazu wird die Beschreibungssprache XAML (Extensible Application Markup Language) verwendet. XAML basiert auf XML.
- XAML wird eingesetzt, um die grafischen Elemente der Benutzeroberfläche, Verhaltensweisen, Animationen, Transformationen, Farbverläufe und vieles mehr zu definieren.
- XAML ist rein deklarativ, d. h., die eigentliche Programmlogik kann damit nicht dargestellt werden.

MVVM: Model

- Hierbei handelt sich um die Schicht zur Datenhaltung im Programmcode.
- Es kann sich dabei um Klassen, Strukturen oder eine Verbindung zu einer physischen Datenbank handeln, das spielt keine Rolle. Wichtig ist nur: Das Model verkörpert die Daten, die durch die Applikation bzw. App verarbeitet werden.
- Die Daten werden dabei über die *View* angezeigt, bzw. die *View* stellt eine Form der Benutzerinteraktion bereit, die ein Bearbeiten der Daten erlaubt.
- Zwischen *View* und *Model* soll jedoch keine direkte Verbindung bestehen.
- Nur dann ist es möglich, die einzelnen Schichten unabhängig voneinander zu verwalten und weiterzuentwickeln.
- Die Verbindung zwischen *View* und *Model* wird durch das *ViewModel* sichergestellt.

MVVM: ViewModel

- Zwischen der Datenschicht (*Model*) und der View (*UI*) ist das *ViewModel* angesiedelt.
- Die *View* übermittelt die Daten an das *ViewModel*, und das *ViewModel* interagiert wiederum mit dem Model.
- Das *ViewModel* registriert, wenn sich die Daten ändern, und leitet diese Änderungen an die registrierten *Views* weiter.
- Mehrere Views können an ein ViewModel gebunden werden – Realisierung unterschiedlicher User Interfaces..

MVVM: Umsetzung

- Für die Kommunikation zwischen den einzelnen Schichten stehen bestimmte technologische Instrumente zur Verfügung.
- Die *View* wird über *Data Binding* an das *ViewModel* gebunden.
- Dazu wird der *View* bekannt geben, in welcher Klasse sie die zugehörigen Informationen findet-
- Das erfolgt über die Definition des *DataContext*.
- Die einzelnen Eigenschaften der Controls werden dann an die öffentlichen Eigenschaften der Klassen des *ViewModel* gekoppelt.
- Ändern sich die Werte dieser Eigenschaften, benachrichtigt das *ViewModel* die *View* über die Datenänderung, so dass diese die aktuellen Werte erhält.
- Bestimmte Aktionen der *View*, wie zum Beispiel das Drücken eines Buttons durch den Nutzer, werden mithilfe sogenannter *Commands* an das *ViewModel* weitergeleitet.
- Zwischen *ViewModel* und *Model* besteht eine ähnliche Systematik zum Datenaustausch.

MVVM: Vorteile

- Das MVVM-Entwurfsmuster führt bei korrekter Anwendung zu einer nahezu vollständigen Entkopplung der Schichten untereinander.
- Das *Model* ist unabhängig von der *View* und auch vom *ViewModel*, d. h., das *Model* kennt beide Schichten nicht.
- Das *ViewModel* kennt sein *Model*, aber nicht die *Views*.
- Diese Trennung hat den Vorteil, dass man beispielsweise das UI einfach austauschen kann, ohne *Model* oder *ViewModel* anpassen zu müssen.
- Im Übrigen ist es möglich, dass ein *ViewModel* über mehrere *Views* verfügt.
- Auf diese Weise können die Daten – je nach Darstellungszweck – unterschiedlich repräsentiert werden.

Datenbindung verstehen, definieren und einsetzen

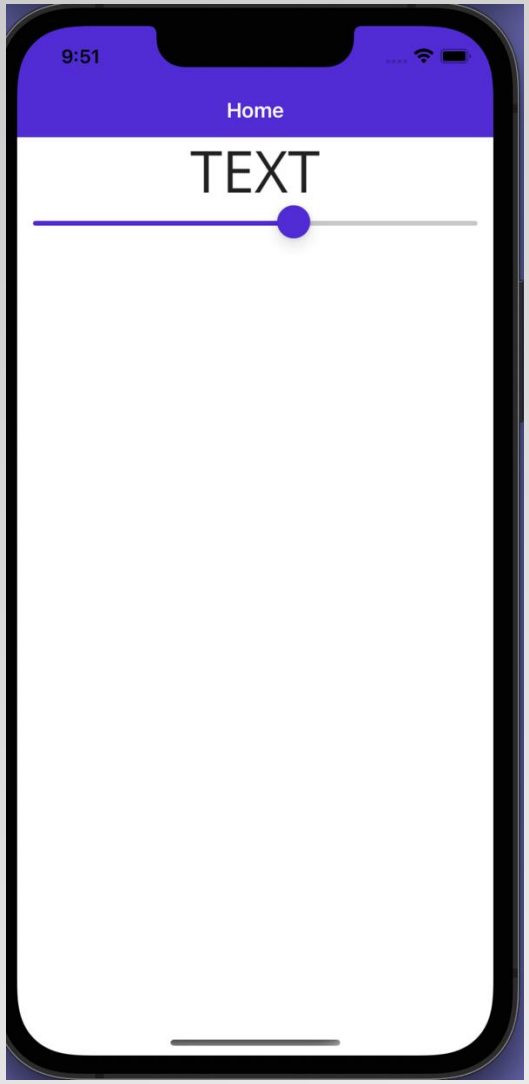
Datenbindung

- die Datenbindung ist eine Technik der Verknüpfung von Eigenschaften von zwei Objekten, so dass Änderungen in einer Eigenschaft automatisch in der anderen Eigenschaft wiedergegeben werden
- eines der beiden Objekte, das an einer Datenbindung beteiligt ist, ist immer ein von *View* abgeleitetes Element und ist Teil der visuellen Oberfläche einer Seite. Das andere Objekt ist eines der folgenden:
  - Ein weiteres *View-Derivat*, meist auf der gleichen Seite
  - Ein Objekt in einer Code-Datei

Datenbindung – Beispiel

```
<?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="DataBinding.MainPage">
    <StackLayout Padding="10, 0">
        <Label Text="TEXT"
               FontSize="40"
               HorizontalOptions="Center"
               VerticalOptions="Center"
               Scale="{Binding Source={x:Reference slider},
                               Path=Value}" />

        <Slider x:Name="slider"
                Minimum="0"
                Maximum="2"
                Value="1"
                VerticalOptions="Center" />
    </StackLayout>
</ContentPage>
```

Beispiel:
„DataBinding“



Bindungen mit einem Bindungskontext


```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.BasicXamlBindingPage"
             Title="Basic XAML Binding">
    <StackLayout Padding="10, 0">
        <Label Text="TEXT"
               FontSize="80"
               HorizontalOptions="Center"
               VerticalOptions="Center"
               BindingContext="{x:Reference Name=slider}"
               Rotation="{Binding Path=Value}" />

        <Slider x:Name="slider"
                Maximum="360"
                VerticalOptions="Center" />
    </StackLayout>
</ContentPage>
```

- Die Markuperweiterung *x:Reference* ist erforderlich, um auf das Quellobjekt zu verweisen – hier auf den *Slider* mit dem Namen *slider*.
- Die Markuperweiterung *Binding* verknüpft die *Rotation*-Eigenschaft des *Labels* mit der *Value*-Eigenschaft des *Sliders*.

Beispiele:
„DataBinding2/
BasicXamlBindingPage
BasicCodeBindingPage“

Bindungen ohne Bindungskontext


```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.AlternativeXamlBindingPage"
             Title="Alternative XAML Binding">
    <StackLayout Padding="10, 0">
        <Label Text="TEXT"
               FontSize="40"
               HorizontalOptions="Center"
               VerticalOptions="Center"
               Scale="{Binding Source={x:Reference slider},
                               Path=Value}" />

        <Slider x:Name="slider"
                Minimum="-2"
                Maximum="2"
                VerticalOptions="Center" />
    </StackLayout>
</ContentPage>
```

In diesem Beispiel verfügt die Binding über zwei festgelegte Eigenschaften, *Source* und *Path*, getrennt durch ein Komma.

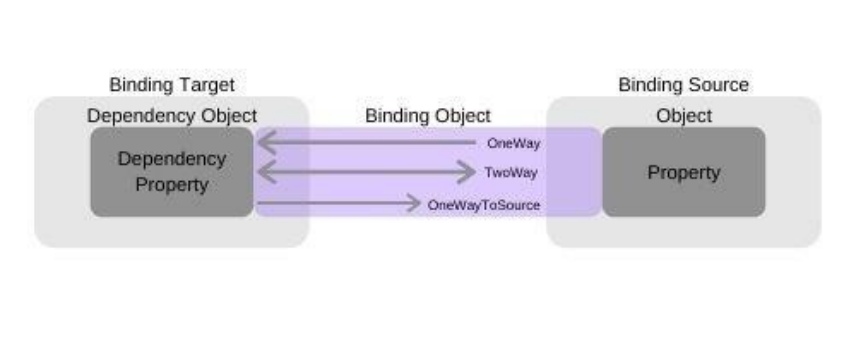
Die Eigenschaft *Source* wird auf eine eingebettete *x:Reference*-Markuperweiterung festgelegt.

Wenn beide Eigenschaften festgelegt sind, hat die *Source*-Eigenschaft des Binding-Objekts Vorrang vor der *BindingContext*-Eigenschaft.

Wir können auf diese Weise einen übergeordneten DataContext „überschreiben“.

Beispiele:
„DataBinding2/
AlternativeCodeBindingPage
AlternativeXamlBindingPage“

Datenbindung – Bindungsmodus



- Jede .NET MAUI-Bindungseigenschaft verfügt über einen Standardbindungsmodus, der festgelegt wird, wenn die bindungsfähige Eigenschaft erstellt wird und die über die *DefaultBindingMode* Eigenschaft des *BindableProperty* Objekts verfügbar ist.
- Dieser Standardbindungsmodus gibt den geltenden Modus an, wenn diese Eigenschaft das Ziel einer Datenbindung ist.
- Der Standardbindungsmodus für die meisten Eigenschaften ist OneWay.
- Wenn der Standardbindungsmodus für die Zieleigenschaft nicht für eine bestimmte Datenbindung geeignet ist, können Sie diesen überschreiben, indem Sie die *Mode*-Eigenschaft von Binding auf eines der Member der *BindingMode* Enumeration festlegen:
  - *Default*
  - *TwoWay*: Daten gehen zwischen Quelle und Ziel in beide Richtungen
  - *OneWay*: Daten werden von der Quelle zum Ziel geleitet
  - *OneWayToSource*: Daten werden von Ziel zu Quelle geleitet
  - *OneTime*: Daten werden von der Quelle zum Ziel geleitet, aber nur, wenn sich der *BindingContext* ändert

Quelle: https://learn.microsoft.com/de-de/dotnet/maui/fundamentals/data-binding/binding-mode

Datenbindung – Zeichenfolgenformatierung


```
<Slider x:Name="slider" />
<Label Text="{Binding Source={x:Reference slider},
                      Path=Value,
                      StringFormat='The slider value is {0:F2}'}" />
```

Beispiel:
„DataBinding2/ StringFormattingPage“

- Die Formatierung von Zeichenfolgen kann auch mit Datenbindungen erreicht werden, indem die *StringFormat*-Eigenschaft von Binding auf eine .NET-Formatierungszeichenfolge mit einem Platzhalter festgelegt wird.
- In diesem Beispiel bewirkt die Formatierungsspezifikation von *F2* , dass der Wert mit zwei Dezimalstellen angezeigt wird.

Datenbindung – Bindungspfad (Untereigenschaften)


```
{Binding Source={x:Reference timePicker},
         Path=Time.TotalSeconds}
```

Beispiel:
„DataBinding2/ PathVariationsPage“

- Es kann die *Path*-Eigenschaft der Binding Klasse auf eine einzelne Eigenschaft, auf eine Untereigenschaft oder auf ein Element einer Auflistung festgelegt werden.
- Die Eigenschaften *Time* und *TotalSeconds* sind durch einen Punkt miteinander verbunden.

Datenbindung – Bindungspfad (Indexer)


```
<Label Text="{Binding Source={x:Static
globe:CultureInfo.CurrentCulture},
       Path=DateTimeFormat.DayNames[3],
       StringFormat='The middle day of the week is {0}'}" />
```

- Die Quelle ist auf die statische *CultureInfo.CurrentCulture*-Eigenschaft festgelegt, bei der es sich um ein Objekt vom Typ *CultureInfo* handelt.
- Diese Klasse definiert eine Eigenschaft mit dem Namen *DateTimeFormat* vom Typ *DateTimeFormatInfo*, die eine *DayNames*-Collection enthält. Der Index wählt das vierte Element aus.

Beispiel:
„DataBinding2/PathVariationsPage“

Datenbindung – Bindungswertkonverter

- Sie möchten eine Datenbindung definieren, bei der die Quelleigenschaft vom Typ *int*, die Zieleigenschaft jedoch vom Typ *bool* ist.
- Sie möchten, dass diese Datenbindung einen *false*-Wert erzeugt, wenn die Quelle gleich *0* ist und andernfalls *true*.
- Der Konverter ist beispielsweise in C# in einer Hilfsklasse zu definieren.
- Datenkonverter können zwischen unterschiedlichsten Datentypen erstellt werden.

Datenbindung – Bindungswertkonverter: Konverter


```
public class IntToBoolConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)value != 0;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (bool)value ? 1 : 0;
    }
}
```

Beispiel:
„DataBinding2/IntToBoolConverter“

Datenbindung – Bindungswertkonverter: User Interface (XAML)

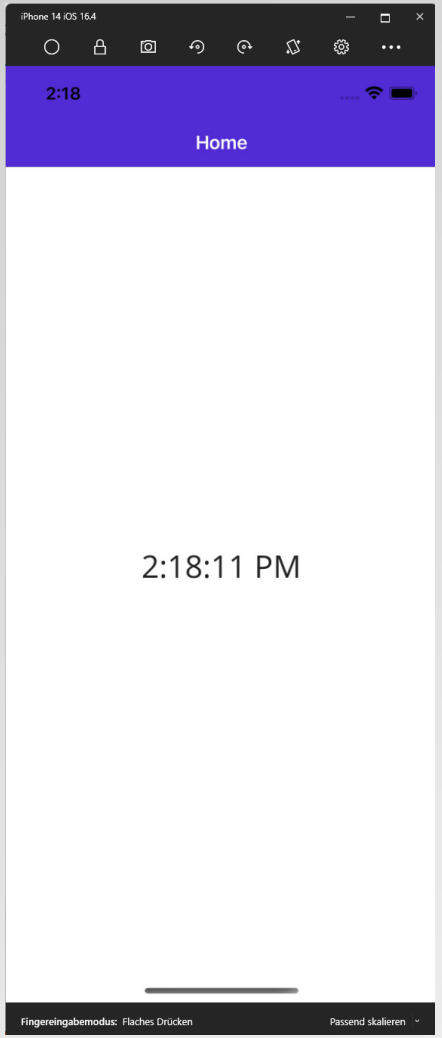

```
 <ContentPage.Resources>
        <local:IntToBoolConverter x:Key="intToBool" />
    </ContentPage.Resources>
<Entry x:Name="entry1"
               Text=""
               Placeholder="enter search term"
               VerticalOptions="Center" />
        <Button Text="Search"
                HorizontalOptions="Center"
                VerticalOptions="Center"
                IsEnabled="{Binding Source={x:Reference entry1},
                                    Path=Text.Length,
                                    Converter={StaticResource intToBool}}" />
```

Beispiel:
„DataBinding2/ EnableButtonsPage“

MVVM: Beispiel

View:

```
<?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"
             xmlns:local="clr-namespace:MVVM"
             x:Class="MVVM.MainPage">
    <ContentPage.BindingContext>
        <local:ClockViewModel/>
    </ContentPage.BindingContext>
    <VerticalStackLayout VerticalOptions="Center">
        <Label Text="{Binding DateTime, StringFormat='{0:T}'}"
           FontSize="28"
           HorizontalOptions="Center"
           VerticalOptions="Center" />
    </VerticalStackLayout>
</ContentPage>
```



ppedv

MVVM: Beispiel

ViewModel:

```
class ClockViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    private DateTime _dateTime;
    private Timer _timer;

    public DateTime DateTime
    {
        get => _dateTime;
        set
        {
            if (_dateTime != value)
            {
                _dateTime = value;
                OnPropertyChanged(); //
            }
        }
    }

    …
```

```
public ClockViewModel()
    {
        this.DateTime = DateTime.Now;
        _timer = new Timer(new TimerCallback((s) => this.DateTime = DateTime.Now),
                           null, TimeSpan.Zero, TimeSpan.FromSeconds(1));
    }

    ~ClockViewModel() =>
        _timer.Dispose();

    public void OnPropertyChanged([CallerMemberName] string name = "") =>
?.Invoke(this, new PropertyChangedEventArgs(name));
}
```

Beispiel:
„MVVM“

ppedv

MVVM: Hinweise

- *Zentrale Basisklasse*: Man sollte eine zentrale abstrakte Basisklasse für alle *ViewModel*-Klassen anlegen, welche das Interface *PropertyChanged* implementiert.
- *Eindeutige Rollenklärung*: Ordnen Sie Verantwortlichkeiten eindeutig und durchgehend den Schichten Ihrer App zu. Welche Bestandteile gehören in die *View*, welche in das *ViewModel* und welche sind Gegenstand des *Models*?
- *Kleine, spezialisierte ViewModels*: Vermeiden Sie es große und komplexe *ViewModels* zu erstellen. Teilen Sie sie stattdessen in kleinere, spezialisierte *ViewModel*-Klassen auf, die jeweils für einen bestimmten Teil der Benutzeroberfläche verantwortlich sind.
- *Vermeiden von Code-Behind*: Halten Sie die Code-Behind-Logik in XAML-Dateien so gering wie möglich. Die meiste Logik sollte im *ViewModel* platziert werden, um eine saubere Trennung zu gewährleisten.

MVVM: Hinweise

- *Vermeiden von Geschäftslogik in der View:* Stellen Sie sicher, dass die View nur für die Anzeige von Daten verantwortlich ist. Alle geschäftsbezogenen Entscheidungen und Berechnungen sollten im ViewModel erfolgen.
- *Konsistente Benennung*: Verwenden Sie konsistente Namenskonventionen für Ihre Klassen, Methoden und Eigenschaften, um die Lesbarkeit und das Verständnis des Codes zu erleichtern.
- *Kontinuierliches Refactoring*: Führen Sie regelmäßig ein Refactoring durch, um Ihren Code sauber und wartbar zu halten.

Commands definieren und verwenden

Commands

- Oft hat eine App Anforderungen, die über Eigenschaftsbindungen hinausgehen, indem der Benutzer Befehle initiieren muss, die sich auf Aktionen im *ViewModel* auswirken (Interaktivität).
- Diese Befehle werden in der Regel ausgeführt, indem Sie auf Schaltflächen klicken oder auf den Bildschirm tippen.
- Erste Lösungsidee: Diese Befehle werden in der Code-Behind-Datei in einem Handler für das *Clicked*-Ereignis von Buttons oder das *Tapped*-Ereignis verarbeitet.
- Nachteil: Diese Vorgehensweise ist nicht „kompatibel“ mit dem MVVM-Muster (Trennung der Schichten, Architekturmodell).

Commands in .NET MAUI

- Um eine Datenbindung zwischen einem *Button* und einem *ViewModel* zuzulassen, definiert der *Button* zwei Eigenschaften:
  - *Command* vom Typ *System.Windows.Input.Icommand*
  - *CommandParameter* vom Typ *Object*
- Um die Befehlsschnittstelle zu verwenden, definieren Sie eine Datenbindung, die auf die *Command*-Eigenschaft von *Button* abzielt, bei der die Quelle eine Eigenschaft im *ViewModel* vom Typ *Icommand* ist.
- Das *ViewModel* enthält Code, der dieser *ICommand* Eigenschaft zugeordnet ist und ausgeführt wird, wenn auf die Schaltfläche geklickt wird.
- Sie können die *CommandParameter*-Eigenschaft auf beliebige Daten festlegen, um zwischen mehreren Schaltflächen zu unterscheiden, wenn alle an dieselbe *Icommand*-Eigenschaft im *ViewModel* gebunden sind.
- Viele andere Controls außer Buttons definieren die Member für *Command* und *CommandParameter*.

Einsatz von Commands

ViewModel:

```
public interface ICommand
{
    public void Execute (Object parameter);
    public bool CanExecute (Object parameter);
    public event EventHandler CanExecuteChanged;
}

public ICommand MyCommand { private set; get; }
```

View

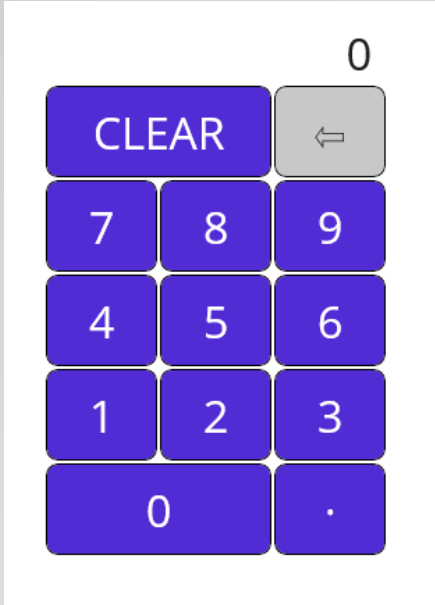
```
<Button Text="Execute command"
        Command="{Binding MyCommand}" />
```

- Wenn der Benutzer das *Button*-Element anklickt, ruft *Button* die *Execute*-Methode auf.
- Wenn *CanExecute false* zurückgibt, deaktiviert der *Button* sich selbst. Das bedeutet, dass der entsprechende Befehl aktuell nicht verfügbar ist.
- Das *Button*-Element aktiviert sich selbst, wenn *CanExecute true* zurückgibt.

Commands: Beispiel

View

```
<Button Text="CLEAR"
        Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"
        Command="{Binding ClearCommand}" />
<Button Text="&#x21E6;"
        Grid.Row="1" Grid.Column="2"
        Command="{Binding BackspaceCommand}" />
<Button Text="7"
        Grid.Row="2" Grid.Column="0"
        Command="{Binding DigitCommand}"
        CommandParameter="7" />
<Button Text="8"
        Grid.Row="2" Grid.Column="1"
        Command="{Binding DigitCommand}"
        CommandParameter="8" />
```



Beispiel:
„DataBinding2/DecimalKeypad“

Commands: Beispiel

ViewModel

```
public ICommand ClearCommand { private set; get; }
public ICommand DigitCommand { private set; get; }

ClearCommand = new Command(
    execute: () =>
    {
        Entry = "0";
        RefreshCanExecutes();
    });
```

```
DigitCommand = new Command<string>(
    execute: (string arg) =>
    {
        Entry += arg;
        if (Entry.StartsWith("0") && !Entry.StartsWith("0."))
        {
            Entry = Entry.Substring(1);
        }
        RefreshCanExecutes();
    },
    canExecute: (string arg) =>
    {
        return !(arg == "." && Entry.Contains("."));
    });
```

ppedv

Einsatz eines MVVM-Frameworks

Community-Toolkit für .NET MAUI

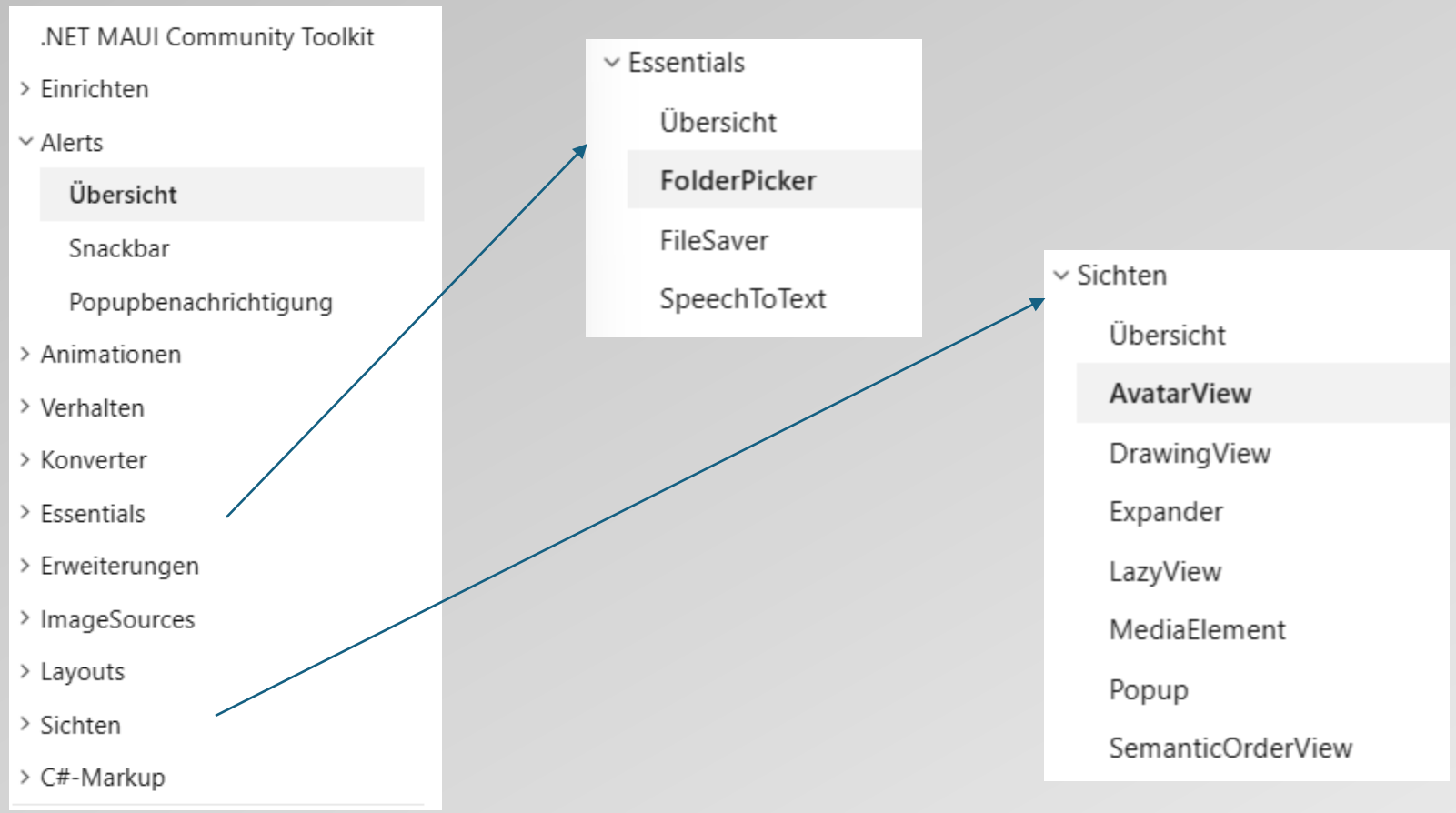
- Das *.NET MAUI Community Toolkit* ist eine Sammlung wiederverwendbarer Elemente für die Anwendungsentwicklung mit MAUI, einschließlich Animationen, Verhaltensweisen, Konvertern, Effekten und Hilfsprogrammen.
- Es vereinfacht und veranschaulicht allgemeine Entwickleraufgaben beim Erstellen von iOS-, Android-, macOS- und WinUI-Anwendungen mit MAUI.
- Das MAUI Community Toolkit ist als eine Reihe von NuGet-Paketen für neue oder vorhandene MAUI-Projekte verfügbar.

Quelle: https://learn.microsoft.com/de-de/dotnet/communitytoolkit/maui/

Community-Toolkit für .NET MAUI: Unterstützte Versionen

- Android 5.0 (API 21) oder höher
- iOS 10 oder höher
- macOS 10.15 oder höher mit Mac Catalyst
- Windows 11 und Windows 10 Version 1809 oder höher mithilfe der Windows UI Library (WinUI) 3
- Tizen 7.0 oder höher

Bestandteile des Community-Toolkit für .NET MAUI



Community-Toolkit für .NET MAUI: Installation

- Fügen Sie das entsprechende NuGet-Package der App hinzu:
  - Paketname: *CommunityToolkit.Maui*
  - Paket-URL: *https://www.nuget.org/packages/CommunityToolkit.Maui*
- Initialisieren des Pakets:

  using CommunityToolkit.Maui
- Um das Toolkit ordnungsgemäß zu verwenden, muss die *UseMauiCommunityToolkit*-Methode für die *MauiAppBuilder*-Klasse aufgerufen werden.
- Das geschieht in der Datei *MauiProgram.cs* (Bootstrapping:

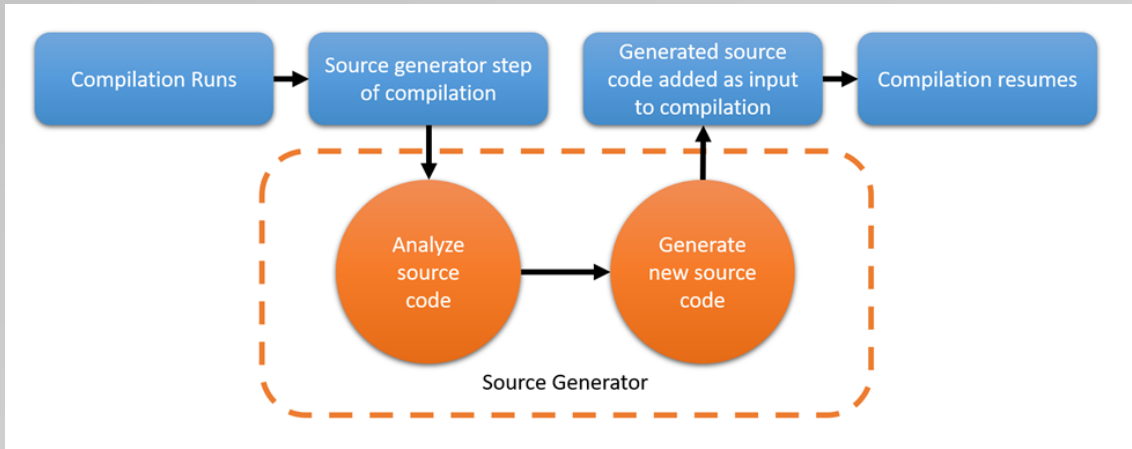
  var builder = MauiApp.CreateBuilder();

  builder.UseMauiApp<App>();

  builder.UseMauiCommunityToolkit();

Quelle: https://learn.microsoft.com/de-de/dotnet/communitytoolkit/maui/get-started?tabs=CommunityToolkitMaui#adding-the-nuget-packages

Community-Toolkit für .NET MAUI: Quellgeneratoren



- Ab Version 8.0 enthält das MVVM-Toolkit Roslyn basierte Quellgeneratoren, die den Aufwand beim Schreiben von Code mithilfe der MVVM-Architektur erheblich reduzieren
- Teile des Quellcodes für das *ViewModel* werden automatisch generiert und auch unmittelbar kompiliert .

Quellcode: https://learn.microsoft.com/de-de/dotnet/communitytoolkit/mvvm/generators/overview

Community-Toolkit für .NET MAUI: Quellgeneratoren – Voraussetzung

- Die Quellcodegeneratoren werden durch Annotierung der Eigenschaften und Methoden „aktiviert“.
- Damit das funktioniert, müssen sich annotierte Felder in einer *partiellen Klasse* mit der erforderlichen *INotifyPropertyChanged* Infrastruktur befinden.
- Konkret leiten wir von der Klasse *ObservableRecipient* ab (Bestandteil des Toolkits).

Quellcode: https://learn.microsoft.com/de-de/dotnet/communitytoolkit/mvvm/generators/overview

Klasse: ObservableRecipient

- Der *ObservableRecipient*-Typ soll als Basis für *ViewModel*-Klassen verwendet werden.
- Die Klasse stellt eine *IsActive*-Eigenschaft bereit, um das *ViewModel* zu aktivieren/ deaktivieren.
- In diesem Zusammenhang bedeutet „aktivieren", dass es mit dem „Hören“ von registrierten Nachrichten beginnt.
- Beispiel:

```
public class MyViewModel : ObservableRecipient {

    ….

}
```

Community-Toolkit für .NET MAUI: Quellgeneratoren – ObservableProperty-Attribut

- übliche Schreibweise im ViewModel:

```
private string? name;

public string? Name
{
    get => name;
    set => SetProperty(ref name, value);
}
```

- nun wird lediglich ein annotieres Feld verwendet:

```
[ObservableProperty]
private string? name;
```

- Im XAML-Code wird die Eigenschaft *Name* gebunden, d.h. der Compiler generiert diese automatisch noch zur Entwurfszeit

Community-Toolkit für .NET MAUI: Quellgeneratoren – ObservableProperty-Attribut Extra Logik im Setter

- übliche Schreibweise im ViewModel:

```
private bool _numberOfCopys;
public int NumberOfCopys
{
    get => _ numberOfCopys
    set
    {
       SetField(ref _ numberOfCopys, value);
       // weitere Methodenaufrufe oder anderer Quellcode
    }
}
```

- Wie „kommen“ wir an den Setter bei automatisch generierten Properties?

Community-Toolkit für .NET MAUI: Quellgeneratoren – ObservableProperty-Attribut Extra Logik im Setter

- Wir können wie folgt vorgehen:

```
[ObservableProperty]
private int _numberOfCopys;

partial void OnNumberOfCopysChanging(int value)
{
    Console.WriteLine($"Property {nameof(NumberOfCopys)} is about to change. Current value: {NumberOfCopys}, new
      value: {value}");
     // weitere Methodenaufrufe oder anderer Quellcode
}

partial void OnNumberOfCopysChanged(int value)
{
    Console.WriteLine($"Property {nameof(NumberOfCopys)} is has changed. Current value: {NumberOfCopys}, new
      value: {value}");
    // weitere Methodenaufrufe oder anderer Quellcode
}
```

- Das bedeutet, dass wir uns in den Setter einbinden können, indem wir Implementierungskörper für die Methoden *OnNumberOfCopysChanging* und *OnNumberOfCopysChanging* bereitstellen, um die gleiche Funktionalität zu erreichen, die wir ohne die Quellgeneratoren erreicht haben.

Quelle: https://blog.ewers-peters.de/mvvm-source-generators-advanced-scenarios

Community-Toolkit für .NET MAUI: Quellgeneratoren – RelayCommand-Attribut

- Übliche Schreibweise im ViewModel:

```
private void SayHello()
{
    Console.WriteLine("Hello");
}

private ICommand? sayHelloCommand;

public ICommand SayHelloCommand => sayHelloCommand ??= new
RelayCommand(SayHello);
```

- nun wird lediglich eine annotiere Methode verwendet:

```
[RelayCommand]
private void SayHello()
{
    Console.WriteLine("Hello");
};
```

- Im XAML-Code wird die Methode *SayHelloCommand* gebunden, d.h. der Compiler generiert diese automatisch bereits zur Entwurfszeit.

ppedv

Community-Toolkit für .NET MAUI: Quellgeneratoren – RelayCommand-Attribut mit Parameter

- Nun wird eine annotiere Methode verwendet:

```
[RelayCommand]
private void GreetUser(User user)
{
    Console.WriteLine($"Hello {user.Name}!");
}
```

- Im XAML-Code wird die Methode *SayHelloCommand* und das Attribut *Parameter* gebunden, d.h. der Compiler generiert diese automatisch zur Entwurfszeit.

- Generiert wird folgender Quellcode:

```
private RelayCommand<User>? greetUserCommand;

public IRelayCommand<User> GreetUserCommand => greetUserCommand ??= new RelayCommand<User>(GreetUser);
```

Community-Toolkit für .NET MAUI: Quellgeneratoren – RelayCommand-Attribut (Asynchrone Befehle)

- Die *[RelayCommand]* Annotierung unterstützt auch das Generieren asynchroner Methoden.
- Diese werden automatisch erstellt, wenn eine Methode einen *Task*-Typ zurückgibt.

```
[RelayCommand]
private async Task GreetUserAsync()
{
    User user = await userService.GetCurrentUserAsync();

    Console.WriteLine($"Hello {user.Name}!");
}
```

- Generiert wird folgender Quellcode:

```
private AsyncRelayCommand? greetUserCommand;

public IAsyncRelayCommand GreetUserCommand => greetUserCommand ??= new AsyncRelayCommand(GreetUserAsync);
```

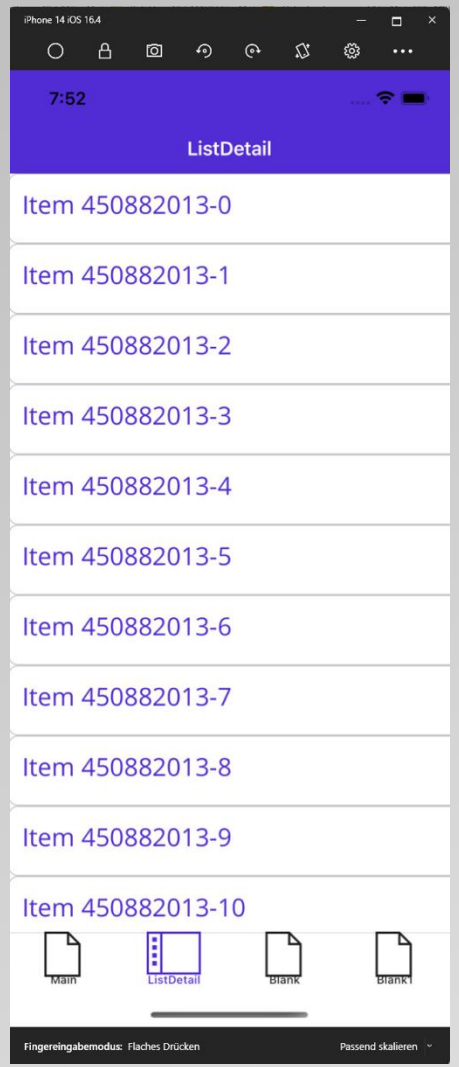
Community-Toolkit für .NET MAUI: Quellgeneratoren – RelayCommand-Attribut (Aktivieren/Deaktivieren von Befehlen)

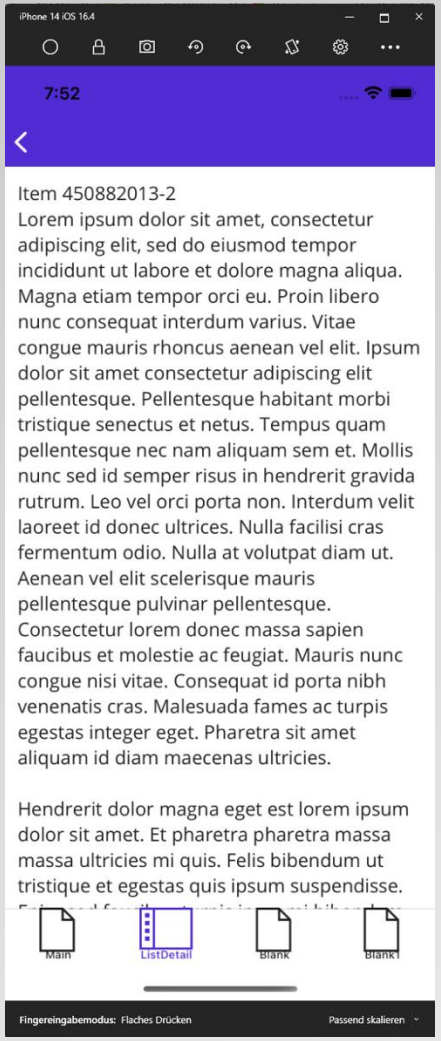
- Häufig ist es nützlich, Befehle zu deaktivieren und wieder zu aktivieren.
- Um dies zu unterstützen, macht das *RelayCommand*-Attribut die *CanExecute*-Eigenschaft verfügbar, die verwendet werden kann, um eine Zieleigenschaft oder -methode anzugeben, mit der ausgewertet werden kann, ob ein Befehl ausgeführt werden kann.
- Der dafür notwendige Quellcode lautet:

```
[RelayCommand(CanExecute = nameof(CanGreetUser))]
private void GreetUser(User? user)
{
    Console.WriteLine($"Hello {user!.Name}!");
}

private bool CanGreetUser(User? user)
{
    return user is not null;
}
```

Master-/ Detail-View mit Datenbindung

MVVM: Master/ Detail-Binding





MVVM: Master/ Detail-Bindung

MasterView

Beispiel:
„MVVM2“

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
                            ...
                            x:DataType="vm:ListDetailViewModel">
<CollectionView ItemsSource="{Binding Items}" RemainingItemsThreshold="10"
RemainingItemsThresholdReachedCommand="{Binding LoadMoreCommand}">
        <CollectionView.ItemTemplate>
                <DataTemplate x:DataType="m:SampleItem">
                        <Frame Margin="4" Padding="12">
                                <Frame.GestureRecognizers>
                                        <TapGestureRecognizer Command="{Binding Source={RelativeSource
                                            AncestorType={x:Type vm:ListDetailViewModel}},
                                            Path=GoToDetailsCommand}" CommandParameter="{Binding .}" />
                                </Frame.GestureRecognizers>
                                <Label Text="{Binding Title}" FontSize="Large" TextColor="{AppThemeBinding
                                    Light={StaticResource Primary}, Dark={StaticResource Black}}" />
                        </Frame>
                </DataTemplate>
        </CollectionView.ItemTemplate>
</CollectionView>
```

MVVM: Master/ Detail-Bindung

MasterViewModel

```
public partial class ListDetailViewModel : BaseViewModel
{
            ...

            [ObservableProperty]
            ObservableCollection<SampleItem> items;

            ...

            [RelayCommand]
            private async void GoToDetails(SampleItem item)
            {
                        await Shell.Current.GoToAsync(nameof(ListDetailDetailPage), true, new Dictionary<string, object>
                        {
                                    { "Item", item }
                        });
            }
}
```

MVVM: Master/ Detail-Bindung

Model

```
public class SampleItem
{
      public string Title { get; set; }

      public string Description { get; set; }
}
```

MVVM: Master/ Detail-Bindung

DetailView

```
<ContentPage …
        x:DataType="vm:ListDetailDetailViewModel">
        <ScrollView>
                <VerticalStackLayout Margin="12">
                        <Label Text="{Binding Item.Title}" FontSize="Header" />
                        <Label Text="{Binding Item.Description}" FontSize="Default" />
                </VerticalStackLayout>
        </ScrollView>
</ContentPage>
```

MVVM: Master/ Detail-Bindung

DetailViewModel

```
public partial class ListDetailDetailViewModel : BaseViewModel
{
        [ObservableProperty]
        SampleItem item;
}
```

ppedv

Zwischenfazit

- Das MVVM-Muster (Model-View-ViewModel) ist ein Designmuster, das in der Softwareentwicklung häufig verwendet wird, um die Struktur und Organisation von Code in einer Weise zu gestalten, welches die Trennung von Benutzeroberfläche (View), Geschäftslogik (ViewModel) und Datenmodell (Model) fördert.
- In .NET MAUI wird das MVVM-Muster häufig eingesetzt, um robuste, wartbare und skalierbare Anwendungen zu erstellen.
- Das MVVM-Muster in .NET MAUI umfasst folgende Aspekte:
  - Model: Das Model repräsentiert die Daten und Geschäftslogik der Anwendung. Es handelt sich um Klassen, die den Zustand und die Funktionalität der Anwendung darstellen. Das können beispielsweise Datenbankzugriffe, API-Anfragen oder andere Datenverarbeitungsaufgaben sein.
  - View: Die View ist für die Darstellung der Benutzeroberfläche verantwortlich. Diese besteht aus XAML-Dateien für das Layout und die Steuerelemente.
  - ViewModel: Das ViewModel fungiert als Bindeglied zwischen der View und dem Model. Es enthält die Präsentationslogik, die für die Verarbeitung von Benutzerinteraktionen und ist für die Vorbereitung der Daten für die Anzeige in der View verantwortlich. Das ViewModel sollte unabhängig von der konkreten Benutzeroberfläche und somit auch leicht testbar sein.
- Datenbindung: In .NET MAUI können Datenbindungsausdrücke in XAML verwendet werden, um Daten zwischen der View und dem ViewModel bidirektional zu synchronisieren. Dadurch wird vermieden, dass die View und das ViewModel direkt miteinander interagieren müssen.
- Commands: In MVVM werden Befehle verwendet, um Aktionen aus der Benutzeroberfläche an das ViewModel weiterzuleiten. Befehle ermöglichen die Abstraktion von Benutzerinteraktionen und können beispielsweise verwendet werden, um auf Klicks auf Schaltflächen zu reagieren.

Übersicht Teil 4

- Plattformintegration
- Benutzerdefinierte Steuerelemente (Handler)
- Plattformcode schreiben und Systemfunktionen verfügbar machen

Plattformintegration

Plattformintegration

- Jede Plattform, welche NET MAUI unterstützt, bietet einzigartige Betriebssystem- und Plattform-APIs, auf die Sie von C# zugreifen können.
- .NET MAUI bietet plattformübergreifende APIs für den Zugriff auf viele dieser Plattformfunktionen
- Zugriff auf Sensoren
- Zugriff auf Informationen über das Gerät, auf dem eine App ausgeführt wird
- Überprüfung der Netzwerkverbindung
- Speichern von Daten

https://learn.microsoft.com/de-de/dotnet/maui/platform-integration/?view=net-maui-9.0

Plattformintegration: Übersicht

- Anwendungsmodell
- Kommunikation
- Gerätefunktionen
- Medien
- Freigabe
- Speicher
- Access-Plattform-APIs (je nach Zielsystem)

Anwendungsmodell

Funktionalität	BESCHREIBUNG
App-Aktionen	Mit der `AppActions` Klasse können Sie App-Verknüpfungen erstellen und beantworten, die zusätzliche Möglichkeiten zum Starten Ihrer App bieten. Weitere Informationen finden Sie unter App-Aktionen.
App-Informationen	Die `AppInfo` Klasse bietet Zugriff auf grundlegende App-Informationen, die den App-Namen und die Version sowie das aktuelle aktive Design für das Gerät enthalten. Weitere Informationen finden Sie unter App-Informationen.
Browser	Die `Browser` Klasse ermöglicht es einer App, einen Weblink in einem In-App-Browser oder im Systembrowser zu öffnen. Weitere Informationen finden Sie im Browser.
Startprogramm	Die `Launcher` Klasse ermöglicht es einer App, einen URI zu öffnen, und wird häufig verwendet, wenn eine Deep-Verknüpfung mit den benutzerdefinierten URI-Schemas einer anderen App hergestellt wird. Weitere Informationen finden Sie unter Launcher.
Hauptthread	Mit der `MainThread` Klasse können Sie Code im UI-Thread ausführen. Weitere Informationen finden Sie im Hauptthread.
Maps	Mit der `Map` Klasse kann eine App die Systemzuordnungs-App zu einem bestimmten Ort oder Ortszeichen öffnen. Weitere Informationen finden Sie unter "Karten".
Berechtigungen	Mit der `Permissions` Klasse können Sie berechtigungen zur Laufzeit überprüfen und anfordern. Weitere Informationen finden Sie unter Berechtigungen.
Versionsverfolgung	Mit der `VersionTracking` Klasse können Sie die Version und Buildnummern der App überprüfen und ermitteln, ob die App zum ersten Mal gestartet wurde. Weitere Informationen finden Sie unter Versionsnachverfolgung.

Quelle: https://learn.microsoft.com/de-de/dotnet/maui/platform-integration/?view=net-maui-9.0

Kommunikation

Funktionalität	BESCHREIBUNG
Kontakte	Die `Contacts` Klasse ermöglicht es einer App, einen Kontakt auszuwählen und Informationen darüber zu lesen. Weitere Informationen finden Sie unter "Kontakte".
Email	Mit der `Email` Klasse können Sie die Standard-E-Mail-App öffnen und eine neue E-Mail mit den angegebenen Empfängern, Betreff und Text erstellen. Weitere Informationen finden Sie unter E-Mail.
Netzwerk	Mit `Connectivity` der `Microsoft.Maui.Networking` Klasse im Namespace können Sie die Netzwerkbarrierefreiheit des Geräts überprüfen, auf dem Ihre App ausgeführt wird. Weitere Informationen finden Sie unter Connectivity.
Wählhilfe	Mit der `PhoneDialer` Klasse kann eine App eine Telefonnummer in der Einwahl öffnen. Weitere Informationen finden Sie unter Telefonwählhilfe.
SMS (Messaging)	Die `Sms` Klasse kann verwendet werden, um die Standard-SMS-App zu öffnen und sie mit einem Empfänger und einer Nachricht vorzuladen. Weitere Informationen finden Sie unter SMS.
Webauthentifikator	Mit `WebAuthenticator` der `Microsoft.Maui.Authentication` Klasse im Namespace können Sie einen browserbasierten Authentifizierungsfluss starten, der auf einen Rückruf auf eine bestimmte URL wartet, die für die App registriert ist. Weitere Informationen finden Sie unter Webauthentifikator.

https://learn.microsoft.com/de-de/dotnet/maui/platform-integration/?view=net-maui-9.0

Gerätefunktionen

Funktionalität	BESCHREIBUNG
Akku	Mit der `Battery` Klasse kann eine App die Akkuinformationen des Geräts überprüfen und den Akku auf Änderungen überwachen. Weitere Informationen finden Sie unter Akku.
Gerätanzeige	Mit der `DeviceDisplay` Klasse kann eine App Informationen zu den Bildschirmmetriken des Geräts lesen. Weitere Informationen finden Sie unter "Geräteanzeige".
Geräteinformationen	Die `DeviceInfo` Klasse ermöglicht es einer App, Informationen über das Gerät zu lesen, auf dem die App ausgeführt wird. Weitere Informationen finden Sie unter Geräteinformationen.
Gerätesensoren	Typen im `Microsoft.Maui.Devices.Sensors` Namespace bieten Zugriff auf den Beschleunigungsmesser, Barometer, Kompass, Gyroskop, Magnetometer und Ausrichtungssensor des Geräts. Weitere Informationen finden Sie unter Gerätesensoren.
Taschenlampe	Die `FlashLight` Klasse kann den Kamera-Blitz des Geräts ein- und ausschalten, um eine Taschenlampe zu emulieren. Weitere Informationen finden Sie unter Flashlight.
Geocodierung	Die `Geocoding` Klasse im `Microsoft.Maui.Devices.Sensors` Namespace stellt APIs bereit, um eine Ortsmarkierung zu einer Positionskoordinate zu geokoordinaten zu codieren und eine Koordinate in eine Ortsmarkierung umzudrehen. Weitere Informationen finden Sie unter Geocodierung.
Geolocation	Die `Geolocation` Klasse im `Microsoft.Maui.Devices.Sensors` Namespace stellt APIs bereit, um die aktuellen Geolocation-Koordinaten des Geräts abzurufen. Weitere Informationen finden Sie unter Geolocation.
Haptisches Feedback	Das `HapticFeedback` haptische Feedback des Klassensteuerelements auf einem Gerät, das in der Regel als ein sanftes Vibrationsgefühl manifestiert wird, um dem Benutzer eine Antwort zu geben. Weitere Informationen finden Sie unter Haptisches Feedback.
Vibration	Mit der `Vibration` Klasse können Sie die Schwingungsfunktionen für eine gewünschte Zeit starten und beenden. Weitere Informationen finden Sie unter Vibration.

https://learn.microsoft.com/de-de/dotnet/maui/platform-integration/?view=net-maui-9.0

ppedv

Medien

Funktionalität	BESCHREIBUNG
Medienauswahl	Mit der `MediaPicker` Klasse können Sie den Benutzer auffordern, ein Foto oder Video auf dem Gerät zu wählen oder aufzunehmen. Weitere Informationen finden Sie unter "Medienauswahl".
Screenshot	Mit der `Screenshot` Klasse können Sie den aktuellen angezeigten Bildschirm der App erfassen. Weitere Informationen finden Sie im Screenshot.
Text-zu-Sprache	Mit der `TextToSpeech` Klasse kann eine App die integrierten Text-zu-Sprache-Engines verwenden, um Text vom Gerät zu sprechen. Weitere Informationen finden Sie unter Text-zu-Sprache.
Einheitenkonverter	Die `UnitConverters` Klasse stellt Einheitenkonverter bereit, die Ihnen beim Konvertieren von einer Maßeinheit in eine andere helfen. Weitere Informationen finden Sie unter Unit Converters.

https://learn.microsoft.com/de-de/dotnet/maui/platform-integration/?view=net-maui-9.0

Freigabe

Funktionalität	BESCHREIBUNG
Zwischenablage	Die `Clipboard` Klasse ermöglicht es einer App, Text in die Systemablage zu kopieren und einzufügen. Weitere Informationen finden Sie unter Zwischenablage.
Freigeben von Dateien und Text	Die `Share` Klasse stellt eine API zum Senden von Daten bereit, z. B. Text oder Weblinks, an die Freigabefunktion des Geräts. Weitere Informationen finden Sie unter "Freigeben".

https://learn.microsoft.com/de-de/dotnet/maui/platform-integration/?view=net-maui-9.0

Speicher

Funktionalität	BESCHREIBUNG
Dateiauswahl	Mit der `FilePicker` Klasse können Sie den Benutzer auffordern, eine oder mehrere Dateien vom Gerät aus zu wählen. Weitere Informationen finden Sie unter "Dateiauswahl".
Hilfselemente des Dateisystems	Die `FileSystem` Klasse stellt Hilfsmethoden bereit, die auf den Cache- und Datenordner der App zugreifen und auf Dateien zugreifen können, die im App-Paket gespeichert sind. Weitere Informationen finden Sie unter Dateisystemhilfen.
Einstellungen	Die Klasse hilft beim Speichern von `Preferences` App-Einstellungen in einem Schlüssel-/Wertspeicher. Weitere Informationen finden Sie unter "Einstellungen".
Schützen von Speicher	Die `SecureStorage` Klasse hilft, einfache Schlüssel-/Wertpaare sicher zu speichern. Weitere Informationen finden Sie unter Secure Storage.

https://learn.microsoft.com/de-de/dotnet/maui/platform-integration/?view=net-maui-9.0

Vorgehensweise bei der Nutzung von Systemfunktionen

das Feature wird direkt durch .NET MAUI unterstützt

- Welche Zielsysteme werden unterstützt?
- Welche Voraussetzungen sind auf den Zielsystemen herzustellen?
  - systemspezifisches Setup
  - Systemberechtigungen anfordern
- Plattformunabhängigen Code schreiben

Feature wird durch eine Bibliothek (Drittanbieter) bereitgestellt

- Installation über NuGet
- Welche Zielsysteme werden unterstützt?
- Welche Voraussetzungen sind auf den Zielsystemen herzustellen?
  - systemspezifisches Setup
  - Systemberechtigungen anfordern
- Plattformunabhängigen Code schreiben

Plattformspezifischen Code für die gewünschten Zielsysteme schreiben

Beispiel: Bildschirminformationen

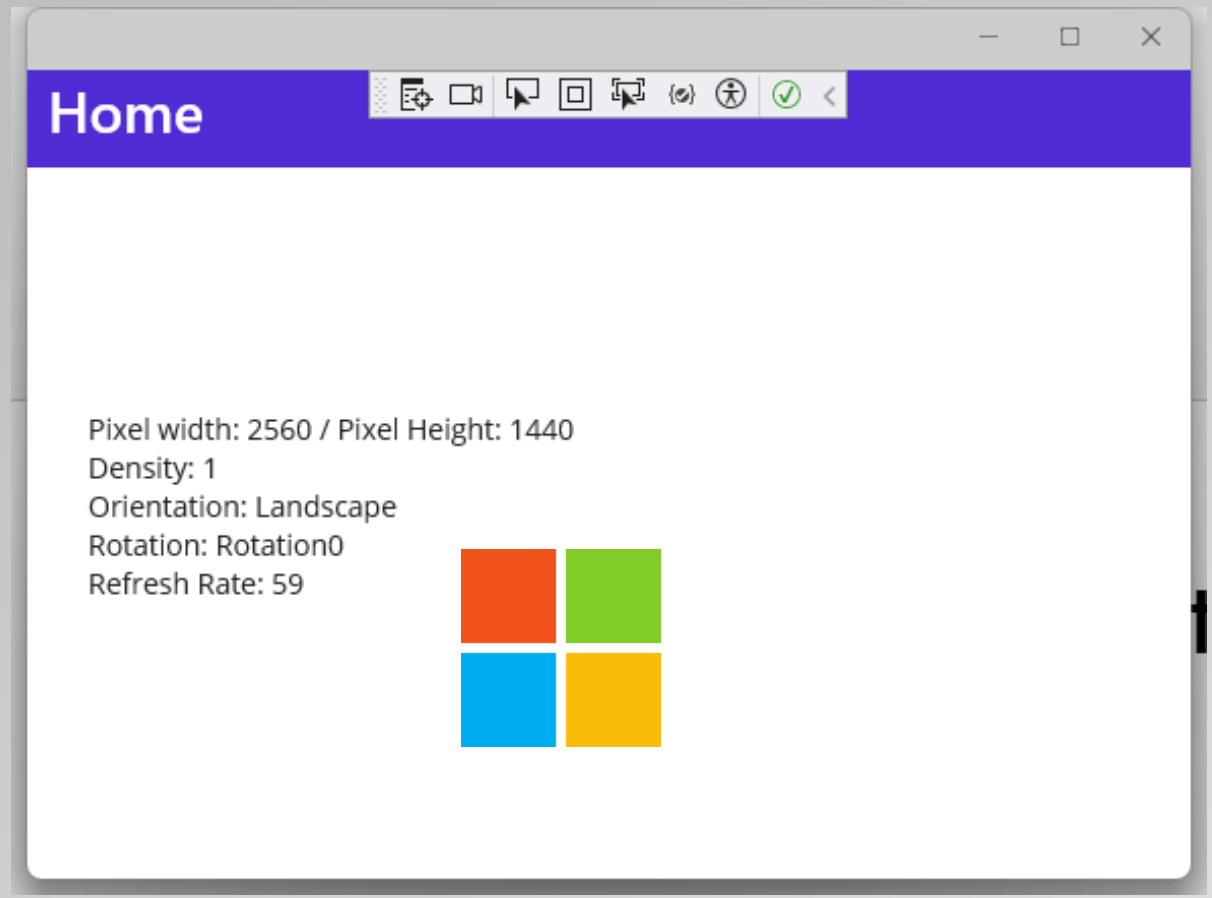
- Die *MainDisplayInfo*-Eigenschaft gibt Informationen zum Bildschirm und zur Ausrichtung zurück
- Die *IDeviceDisplay*-Schnittstelle stellt auch das *MainDisplayInfoChanged*-Ereignis bereit, das ausgelöst wird, wenn sich eine Bildschirmmetrik ändert, z.B. wenn sich die Geräteausrichtung von *DisplayOrientation.Landscape* zu *DisplayOrientation.Portrait* ändert.

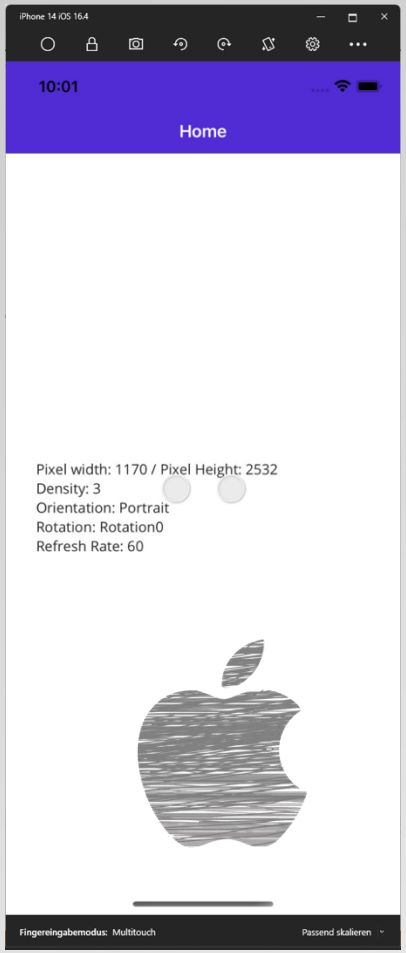
```
private void ReadDeviceDisplay()
{
    System.Text.StringBuilder sb = new System.Text.StringBuilder();

    sb.AppendLine($"Pixel width: {DeviceDisplay.Current.MainDisplayInfo.Width} / Pixel Height: {DeviceDisplay.Current.MainDisplayInfo.Height}");
    sb.AppendLine($"Density: {DeviceDisplay.Current.MainDisplayInfo.Density}");
    sb.AppendLine($"Orientation: {DeviceDisplay.Current.MainDisplayInfo.Orientation}");
    sb.AppendLine($"Rotation: {DeviceDisplay.Current.MainDisplayInfo.Rotation}");
    sb.AppendLine($"Refresh Rate: {DeviceDisplay.Current.MainDisplayInfo.RefreshRate}");

    DisplayDetailsLabel.Text = sb.ToString();
}
```

Beispiel:
„MAUIDevices“

ppedv

Beispiel: Bildschirminformationen





Beispiel: Geräteinformationen

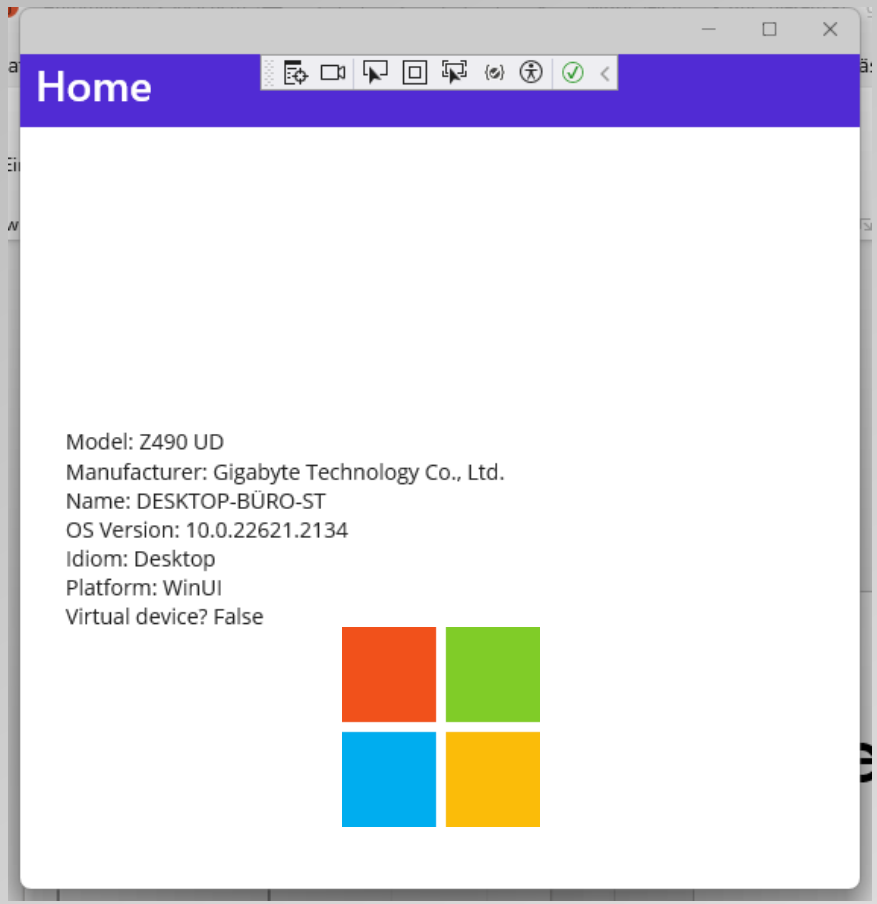
- Die *IDeviceInfo* Schnittstelle bietet viele Eigenschaften, die das Gerät beschreiben:

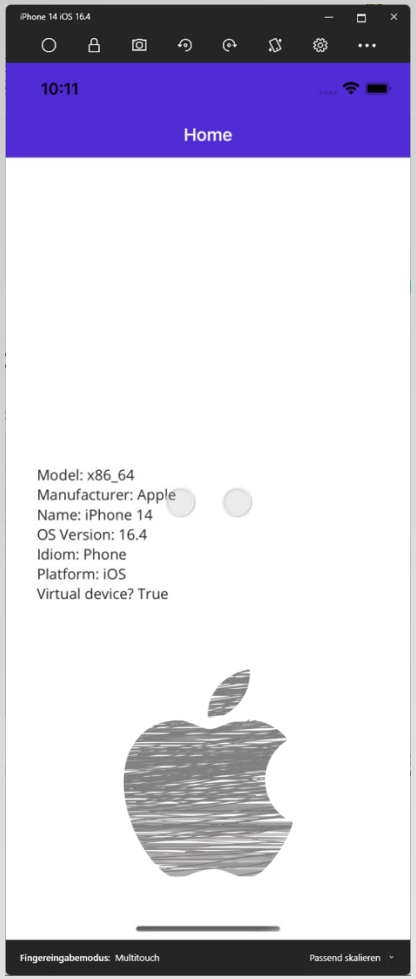
```
private void ReadDeviceInfo()
{
    System.Text.StringBuilder sb = new System.Text.StringBuilder();

    sb.AppendLine($"Model: {DeviceInfo.Current.Model}");
    sb.AppendLine($"Manufacturer: {DeviceInfo.Current.Manufacturer}");
    sb.AppendLine($"Name: {DeviceInfo.Current.Name}");
    sb.AppendLine($"OS Version: {DeviceInfo.Current.VersionString}");
    sb.AppendLine($"Idiom: {DeviceInfo.Current.Idiom}");
    sb.AppendLine($"Platform: {DeviceInfo.Current.Platform}");

    bool isVirtual = DeviceInfo.Current.DeviceType switch
    {
        DeviceType.Physical => false,
        DeviceType.Virtual => true,
        _ => false
    };

    sb.AppendLine($"Virtual device? {isVirtual}");

    DisplayDeviceLabel.Text = sb.ToString();
}}
```

Beispiel:
„MAUIDevices“

ppedv

Beispiel: Geräteinformationen





Beispiel: Geolocation

- Die Standardimplementierung der *Igeolocation*-Schnittstelle ist über die *Geolocation.Default*-Eigenschaft verfügbar.
- Sowohl die Schnittstelle *Geolocation* als auch die *IGeolocationMicrosoft.Maui.Devices.Sensors*-Klasse sind im Namespace enthalten.
- Es müssen unterschiedliche Systemberechtigungen je nach Zielsystem angefordert werden.

Quelle: https://pixabay.com/de/images/search/geolocation/

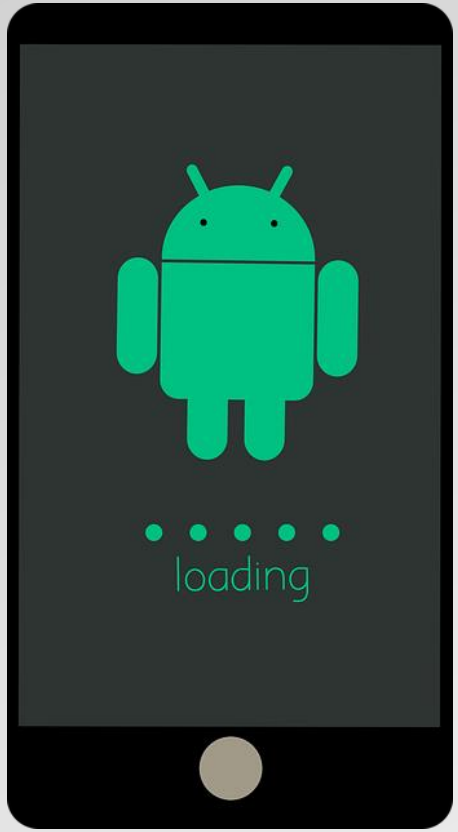
Beispiel: Geolocation/ Android

- Grobe oder feine Standortberechtigungen müssen angegeben werden.
- Hardwarefeatures anfordern:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-feature android:name="android.hardware.location" android:required="false" />
<uses-feature android:name="android.hardware.location.gps" android:required="false" />
<uses-feature android:name="android.hardware.location.network" android:required="false" />
```

- Ab Android 10 – Q (API-Ebene 29 oder höher) zusätzlich:

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



Quelle: https://pixabay.com/de/vectors/android-betriebssystem-neustart-2995824/

Beispiel: Geolocation/ iOS

- in den Dateien *Platforms/iOS/Info.plist* und *Platforms/MacCatalyst/Info.plist* die folgenden Schlüssel und Werte hinzu:

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>Fill in a reason why your app needs access to location.</string>
```

- wenn Sie die vollständige Genauigkeit mit der *GeolocationRequest.RequestFullAccuracy*-Eigenschaft anfordern möchten, fügen Sie diesen Schlüssel zu den Dateien *Platforms/iOS/Info.plist* und *Platforms/MacCatalyst/Info.plist* hinzu:

```
<key>NSLocationTemporaryUsageDescriptionDictionary</key>
<array>
  <dict>
    <key>TemporaryFullAccuracyUsageDescription</key>
    <string>Fill in a reason why your app needs full accuracy</string>
  </dict>
</array>
```

Beispiel: Geolocation/ Windows

Es ist kein Setup erforderlich ☺

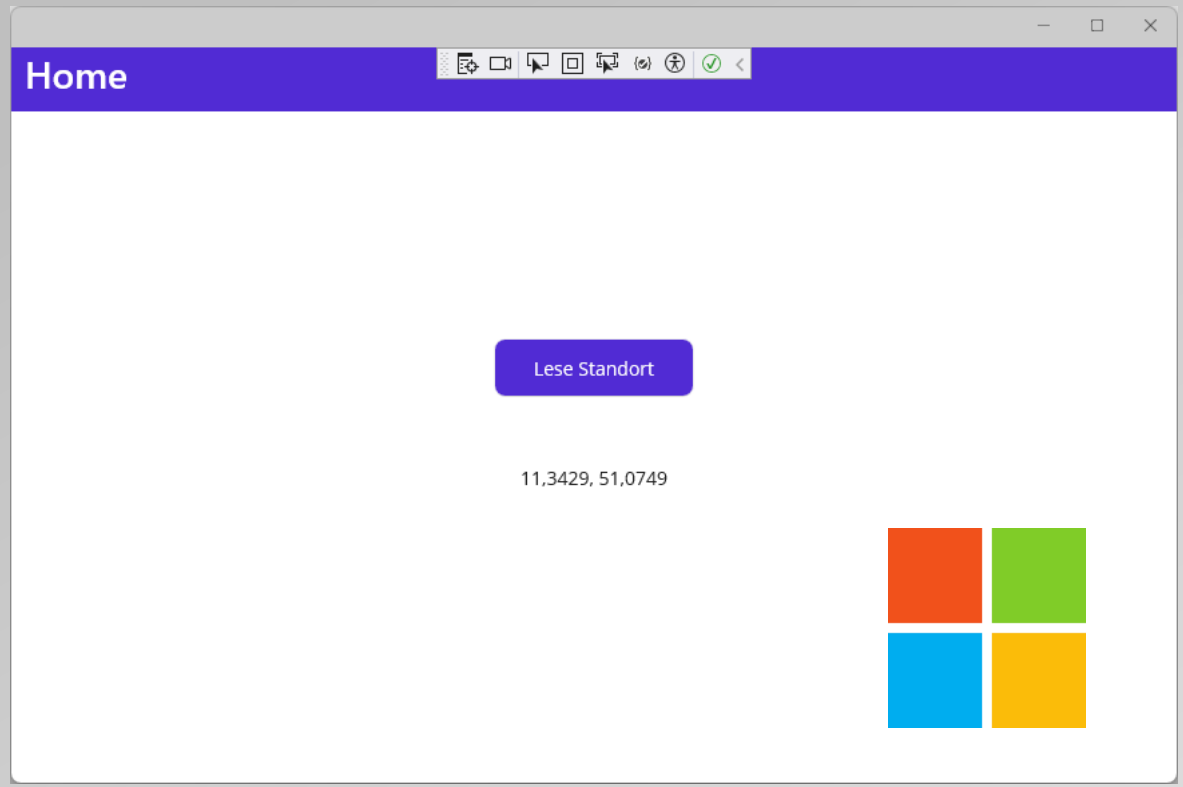
Beispiel: Geolocation
Abrufen der aktuellen Position

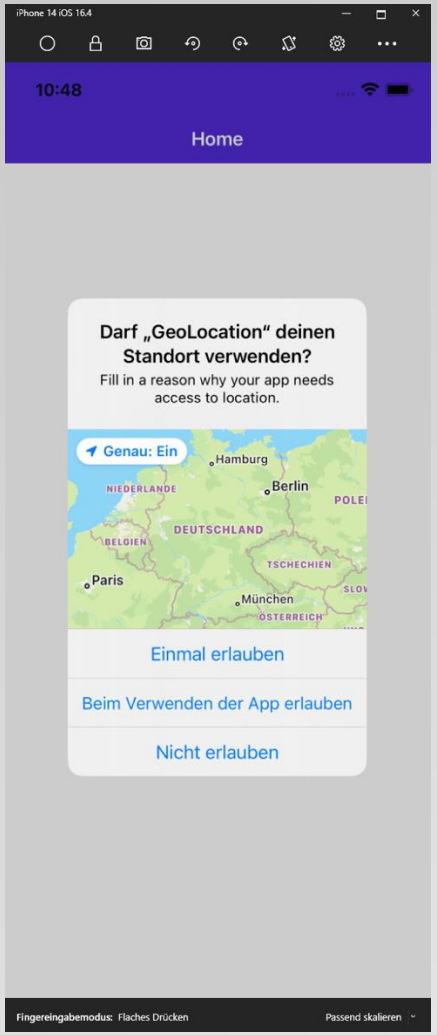
```
public async Task GetCurrentLocation()
{
    try
    {
        _isCheckingLocation = true;

        GeolocationRequest request = new GeolocationRequest(GeolocationAccuracy.Medium, TimeSpan.FromSeconds(10));

        _cancelTokenSource = new CancellationTokenSource();

      location = await Geolocation.Default.GetLocationAsync(request, _cancelTokenSource.Token);

    }
    catch (Exception ex)
    {
        // Unable to get location
    }
    finally
    {
        _isCheckingLocation = false;
    }
}
```

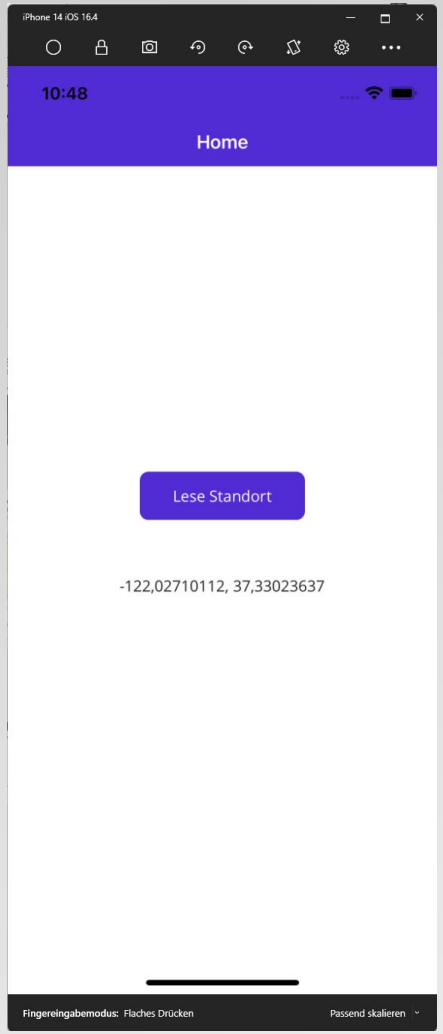
Beispiel:
„GeoLocation“

ppedv

Beispiel: Geolocation





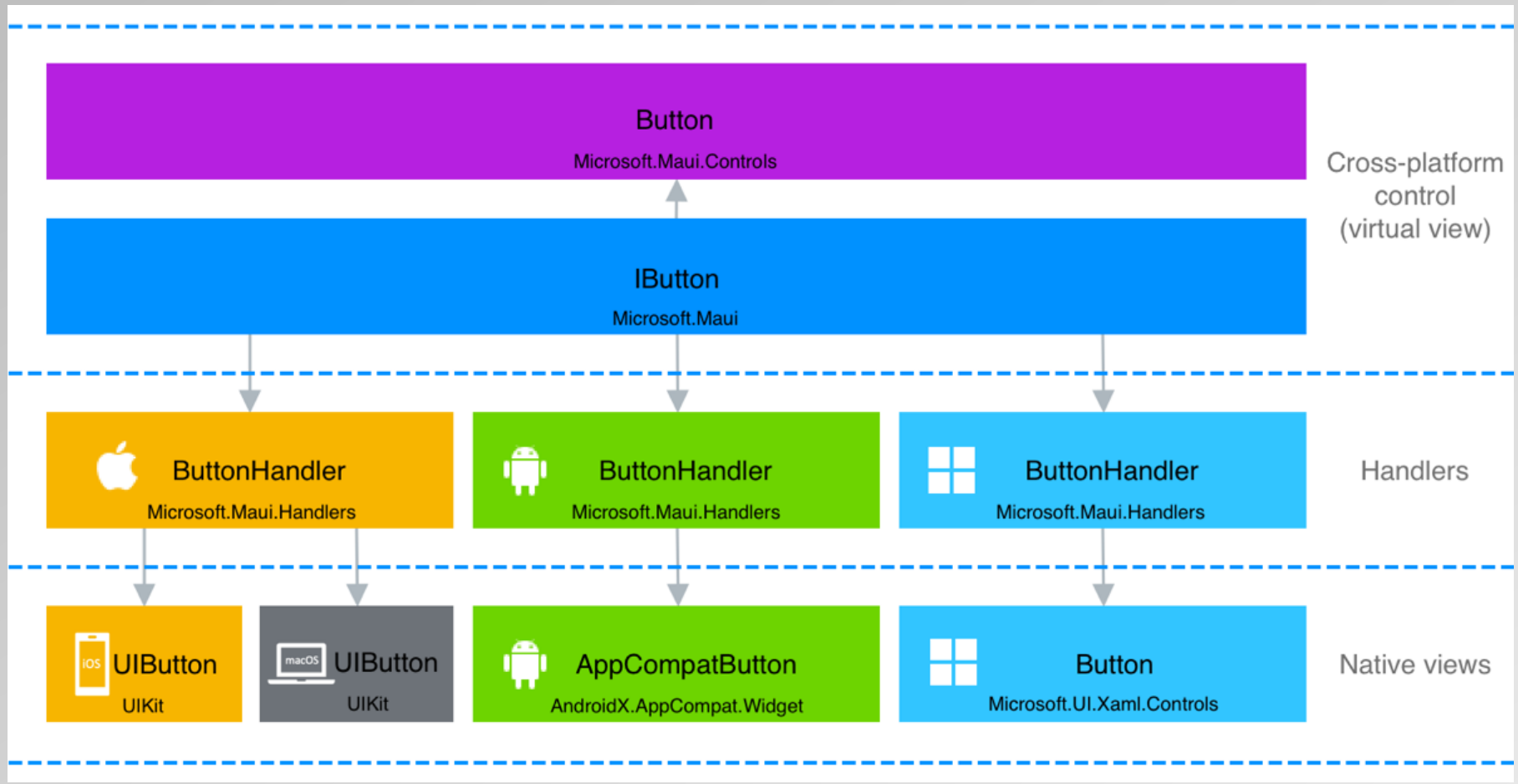


Benutzerdefinierte Steuerelemente (Handler)

Funktionsweise Steuerelemente

- .NET MAUI bietet eine Sammlung plattformübergreifender Steuerelemente, die zum Anzeigen von Daten, Initiieren von Aktionen usw. verwendet werden
- jedes Steuerelement verfügt über eine Schnittstellendarstellung, die das Steuerelement abstrahiert
- plattformübergreifende Steuerelemente, die diese Schnittstellen implementieren, werden als virtuelle Ansichten bezeichnet
- Handler ordnen diese virtuellen Ansichten Steuerelementen auf jeder Plattform zu, die als native Ansichten bezeichnet werden

Handler



Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/handlers/

Anpassen der Handler

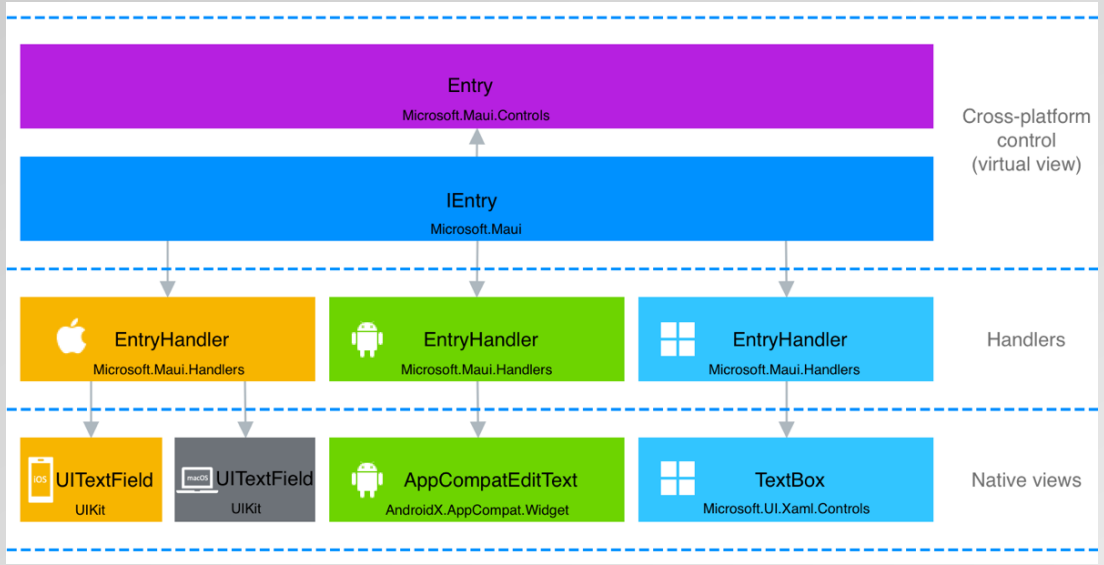
Die Anpassung, welche die nativen Ansichten für das plattformübergreifende Steuerelement ändert, wird erreicht, indem der Mapper für einen Handler mit einer der folgenden Methoden geändert wird:

- *PrependToMapping*, wodurch der Mapper für einen Handler geändert wird, bevor die .NET MAUI-Steuerelementzuordnungen angewendet wurden.
- *ModifyMapping*, wodurch eine vorhandene Zuordnung geändert wird.
- *AppendToMapping*, wodurch der Mapper für einen Handler geändert wird, nachdem die .NET MAUI-Steuerelementzuordnungen angewendet wurden.

Handler: Beispiel - Entry

Die .NET MAUI-Ansicht *Entry* ist ein einzeiliges Texteingabesteuerelement, das die *IEntry*-Schnittstelle implementiert. Ein *EntryHandler* ordnet die Entry Ansicht den folgenden nativen Ansichten für jede Plattform zu:

- iOS/ Mac Catalyst: *UITextField*
- Android: *AppCompatEditText*
- Windows: *TextBox*



Quelle: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/handlers/customize

ppedv

Handler: Beispiel – Entry

- Ein Handler wirkt sich auf alle Steuerelemente eines Typs aus, sobald der Handler „aktiviert“ wurde.
- Alle *Entry*-Elemente werden in der .NET MAUI-App angepasst:

```
 void ModifyEntry()
    {
        Microsoft.Maui.Handlers.EntryHandler.Mapper.AppendToMapping("MyCustomization", (handler, view) =>
        {
#if ANDROID
            handler.PlatformView.SetSelectAllOnFocus(true);
#elif IOS || MACCATALYST
            handler.PlatformView.EditingDidBegin += (s, e) =>
            {
                handler.PlatformView.PerformSelector(new ObjCRuntime.Selector("selectAll"), null, 0.0f);
            };
#elif WINDOWS
            handler.PlatformView.GotFocus += (s, e) =>
            {
                handler.PlatformView.SelectAll();
            };
#endif
        });
    }
```

Beispiel:
„CustomizeHandlersDemo“
(Microsoft)

Handler: Beispiel - Entry

- Handler für bestimmte Steuerelementinstanzen können durch Unterklassen des Steuerelements und dann durch Ändern des Handlers für den Basissteuerelementtyp erstellt werden.
- Eigenes Steuerelement ableiten:

```
namespace CustomizeHandlersDemo.Controls
{
    internal class MyEntry : Entry
    {

    }
}
```

ppedv

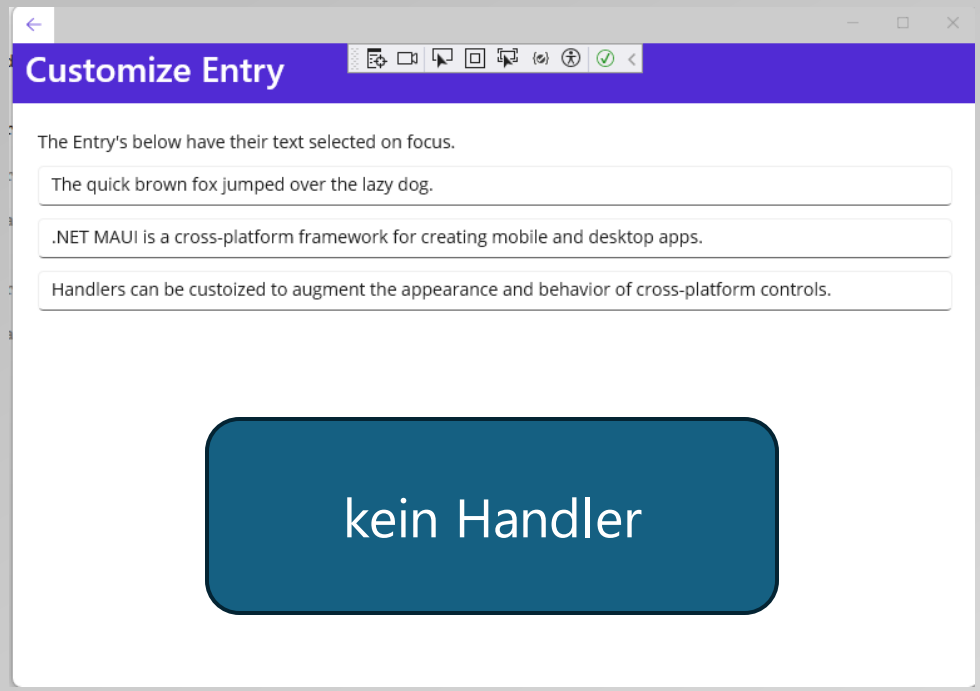
Handler: Beispiel - Entry

- Handler für die abgeleitete Klasse definieren:

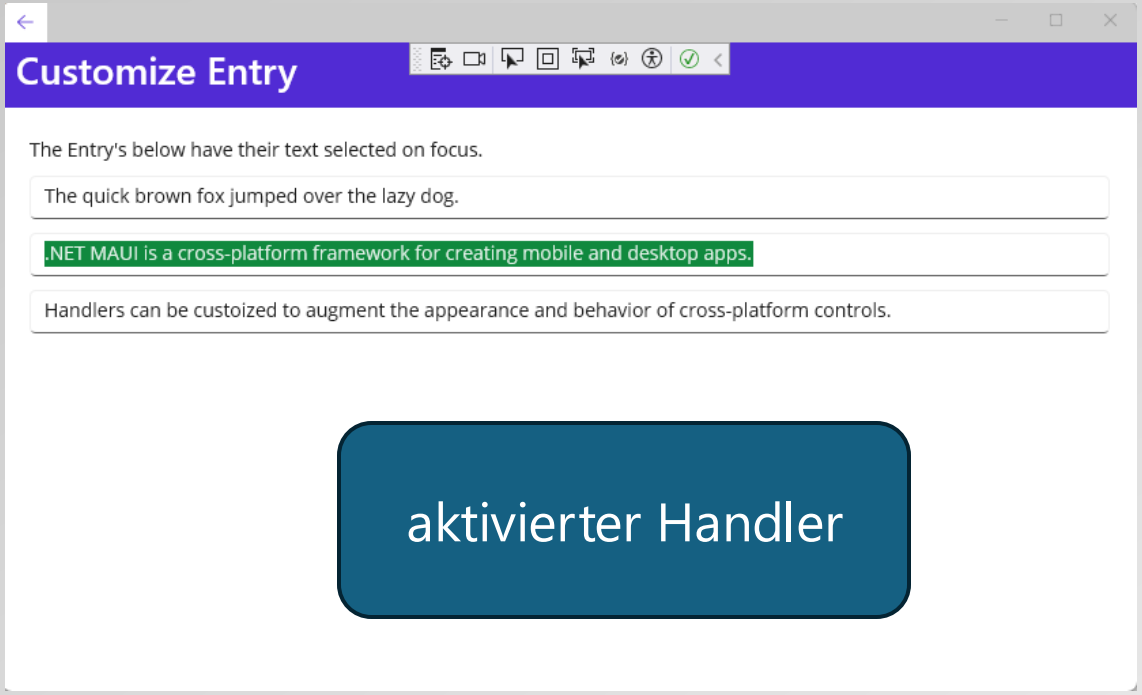
```
Microsoft.Maui.Handlers.EntryHandler.Mapper.AppendToMapping("MyCustomization", (handler, view) =>
{
    if (view is MyEntry)
    {
#if ANDROID
        handler.PlatformView.SetSelectAllOnFocus(true);
#elif IOS || MACCATALYST
        handler.PlatformView.EditingDidBegin += (s, e) =>
        {
            handler.PlatformView.PerformSelector(new ObjCRuntime.Selector("selectAll"), null, 0.0f);
        };
#elif WINDOWS
        handler.PlatformView.GotFocus += (s, e) =>
        {
            handler.PlatformView.SelectAll();
        };
#endif
    }
});
```

Beispiel:
„CustomizeHandlersDemo“
(Microsoft)

Handler: Beispiel - Entry



kein Handler



aktivierter Handler

ppedv

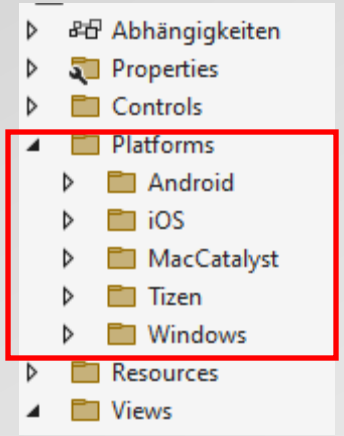
Plattformcode schreiben und Systemfunktionen verfügbar machen

Plattformcode

- In Situationen, in denen .NET MAUI keine APIs für den Zugriff auf bestimmte Plattform-APIs bereitstellt, können Sie Ihren **eigenen Code schreiben**, um auf die erforderlichen Plattform-APIs zuzugreifen.
- Dies **erfordert Kenntnisse der iOS- und MacCatalyst-APIs von Apple, Googles Android-APIs und Microsofts Windows App SDK APIs**.
- Umsetzung: mittels bedingter Kompilierung oder Teilklassen.

Plattformcode: Teilklassen

- Ein .NET MAUI-App-Projekt enthält einen Ordner „*Platforms*" mit je einem Unterordner, der eine Plattform darstellt.
- Die Ordner für jede Zielplattform enthalten plattformspezifischen Code, der die App auf jeder Plattform startet, sowie alle zusätzlichen Plattformcodes, die Sie hinzufügen.
- Zum Build-Zeitpunkt enthält das Build-System nur den Code aus dem betreffenden Ordner für diese bestimmte Plattform.
- Wenn Sie beispielsweise für Android eine App erstellen, werden die Dateien im Android-Ordner in das App-Paket integriert, aber nicht in die anderen Zielsysteme.
- Das Feature wird als Multi-Targeting bezeichnet.



Plattformcode: Vorgehen

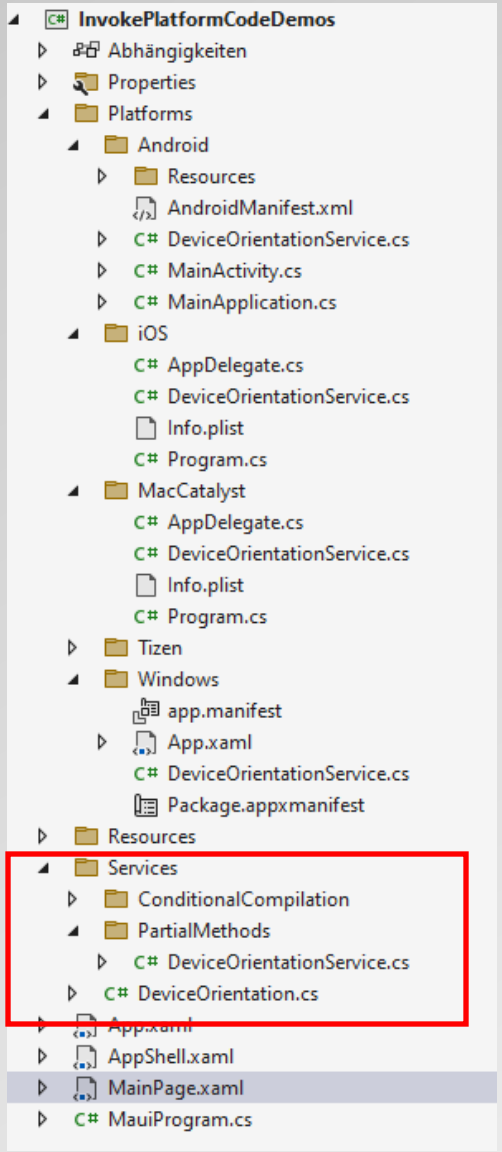
1. Definieren Sie die plattformübergreifende API als **Teilklasse**, die Methodensignaturen für alle Vorgänge definiert, die Sie auf jeder Plattform aufrufen möchten.
2. Implementieren Sie die plattformübergreifende API pro Plattform, indem Sie **dieselbe Teilklasse** und die gleichen Methodensignaturen definieren und auch die Methodenimplementierungen bereitstellen.
3. Rufen Sie die plattformübergreifende API auf, indem Sie eine Instanz der Teilklasse erstellen und ihre Methoden aufrufen.

Plattformcode: Beispiel

Definieren der plattformübergreifenden API:

```
{
    public enum DeviceOrientation
    {
        Undefined,
        Landscape,
        Portrait
    }
}

namespace InvokePlatformCodeDemos.Services.PartialMethods
{
    public partial class DeviceOrientationService
    {
        public partial DeviceOrientation GetOrientation();
    }
}
```

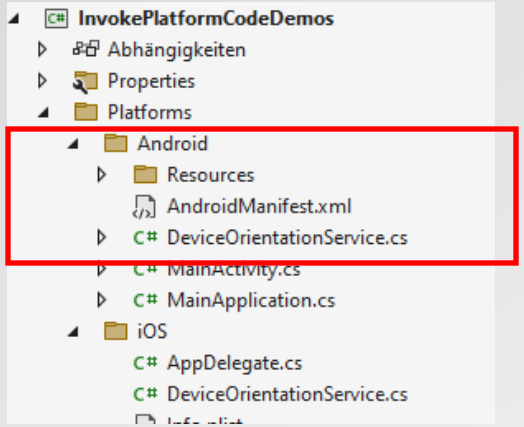
Beispiel:
„InvokePlatformCodeDemos“
(Microsoft)



Plattformcode: Beispiel

Implementierung für Android:

```
public partial DeviceOrientation GetOrientation()
{
    IWindowManager windowManager =
        Android.App.Application.Context.GetSystemService(Context.WindowService).JavaCast<IWindowManager>();
    SurfaceOrientation orientation = windowManager.DefaultDisplay.Rotation;
    bool isLandscape = orientation == SurfaceOrientation.Rotation90 || orientation == SurfaceOrientation.Rotation270;
    return isLandscape ? DeviceOrientation.Landscape : DeviceOrientation.Portrait;
}
```

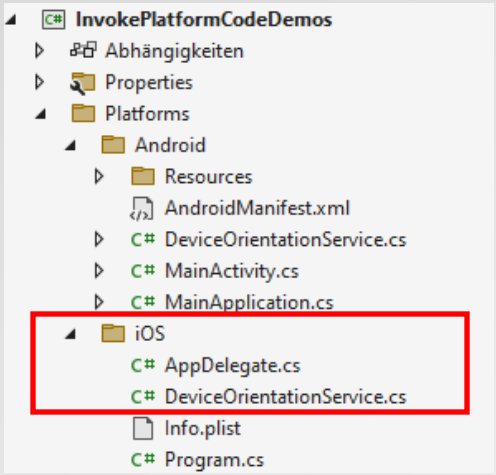


ppedv

Plattformcode: Beispiel

Implementierung für iOS:

```
public partial DeviceOrientation GetOrientation()
    {
        UIInterfaceOrientation orientation = UIApplication.SharedApplication.StatusBarOrientation;
        bool isPortrait = orientation == UIInterfaceOrientation.Portrait || orientation ==
            UIInterfaceOrientation.PortraitUpsideDown;
        return isPortrait ? DeviceOrientation.Portrait : DeviceOrientation.Landscape;
    }
```

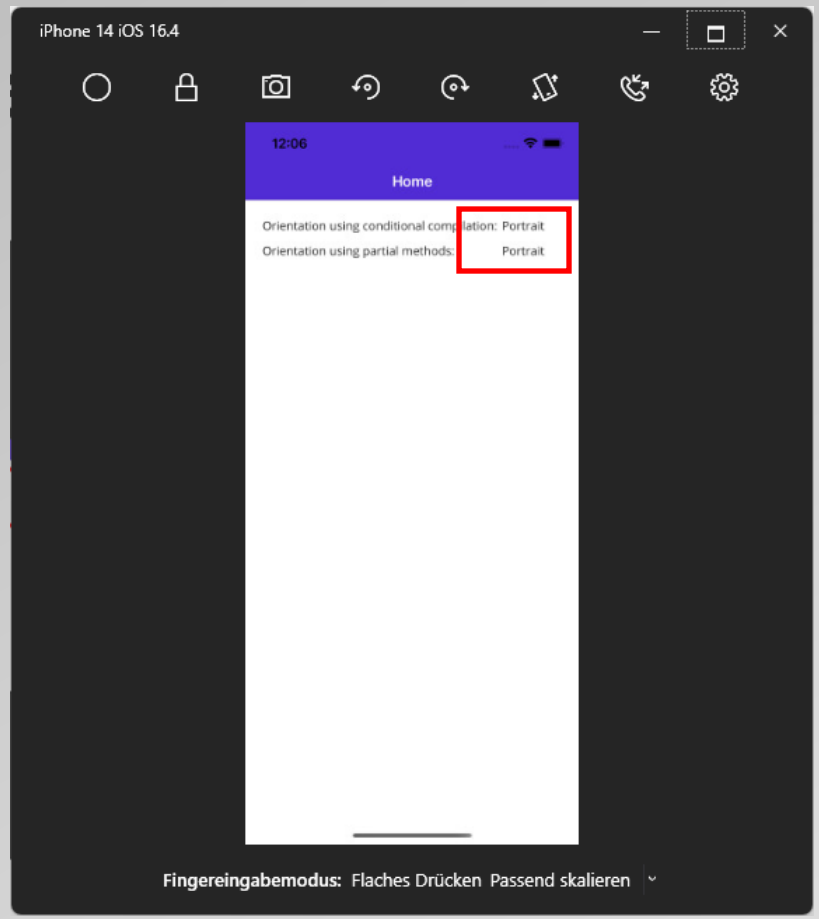


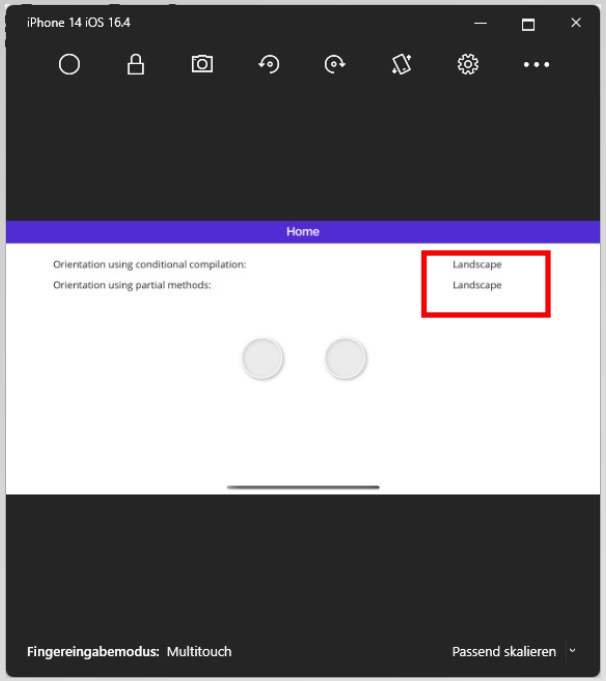
Plattformcode: Beispiel

Aufrufen der plattformübergreifenden API:

```
using InvokePlatformCodeDemos.Services;
using InvokePlatformCodeDemos.Services.PartialMethods;
...

DeviceOrientationService deviceOrientationService = new DeviceOrientationService();
DeviceOrientation orientation = deviceOrientationService.GetOrientation();
```

Plattformcode: Beispiel (iOS)





Plattformcode: Hinweise

- Plattformimplementierungen müssen sich im gleichen Namespace und derselben Klasse befinden, in der die plattformübergreifende API definiert wurde.
- Die Definition ist auch nur für einige Systeme möglich, beispielsweise für iOS und Android (Mobile Apps).
- Auf diese Weise kann plattformspezifische Hardware, wie Sensoren usw. verwendet werden.
- Eine .NET MAUI-App kann damit auf die APIs der Zielsysteme zugreifen.

Zwischenfazit

- .NET MAUI bietet ein umfassendes API, um systemunabhängige Funktionen wie Sensoren oder Gerätefunktionen zu nutzen.
- Über das Schreiben von plattformspezifischem Code kann man native Systemfunktionen auf jeder Zielplattform aufrufen.
- Dazu muss man mit dem API des Zielsystems interagieren.
- Ebenso ist es möglich eigene Steuerelemente für die UI-Gestaltung zu definieren oder vorhandene Steuerelemente für jede Zielplattform individuell anzupassen.
- Dieses erfolgt durch die Definition bzw. Modifikation von Handlern.

.NET MAUI 9 Features

- Neue Steuerelemente
  - HybridWebView
  - TitleBar (Windows)
- Steuerelementverbesserungen
- Fehlerbereinigungen – und dadurch bessere Stabilität
- Neue Projektvorlage
- Open Source-Toolkit von Syncfusion (Chart-Controls, Tab-Control, visuelle Effekte)

https://learn.microsoft.com/de-de/dotnet/maui/whats-new/dotnet-9?view=net-maui-9.0

Fazit

- .NET MAUI ist ein Framework für die plattformübergreifende App-Entwicklung, das von Microsoft entwickelt wurde.
- Es ermöglicht native Anwendungen für verschiedene Betriebssysteme wie Android, iOS, macOS und Windows mit gemeinsamem Code zu erstellen.
- Die wichtigsten Features sind:
  - *Plattformübergreifendes Framework*: NET MAUI wurde entwickelt, um das Erstellung von Anwendungen für verschiedene Plattformen mit einem einzigen Codebasis zu ermöglichen.
  - *XAML-basierte UI*: Die Benutzeroberfläche wird mit XAML entworfen, was eine deklarative Beschreibung der Benutzeroberfläche ermöglicht.
  - *Wiederverwendbarer Code:* Ein Großteil des Codes kann zwischen verschiedenen Plattformen wiederverwendet werden, was die Entwicklungseffizienz steigert.
  - *Plattformspezifische APIs*: NET MAUI bietet Zugriff auf plattformspezifische APIs, so dass Entwickler Funktionen nutzen können, die auf einer bestimmten Plattform verfügbar sind.
  - *Unterstützung für MVVM:* Das Framework fördert das MVVM (Model-View-ViewModel)-Muster, um die Trennung von Geschäftslogik und Benutzeroberfläche zu ermöglichen.
  - *Anpassbare Benutzeroberfläche:* Trotz der gemeinsamen Codebasis können Entwickler die Benutzeroberfläche an die spezifischen Designrichtlinien der Zielplattformen anpassen.
  - *Kontinuierliche Weiterentwicklung*: NET MAUI wird aktiv von Microsoft weiterentwickelt, um neue Funktionen hinzuzufügen, Fehler zu beheben und mit den neuesten Technologietrends Schritt zu halten.

Fazit

Insgesamt bietet NET MAUI eine leistungsstarke Möglichkeit, plattformübergreifende native Apps zu entwickeln.

Es ist eine geeignete Wahl für Entwickler, die ihre Apps auf verschiedenen Plattformen bereitstellen möchten.

Foliensatz und Quellcodebeispiele: über die Konferenzwebseite oder auf https://larinet.com

Fragen?

Vielen Dank!
Ich freue mich auf Feedback!