**Belegarbeit**

**Internet-Technologien**

Native App-Entwicklung mit Xamarin in Visual Studio

**Vorgelegt am:** \_\_.\_\_.2023

**Von:** Elias Kunze, Pascal Köppel

**Matrikelnummern:** 4004399, 4004454

**Studiengang:** Technische Informatik

**Modul:** Internet-Technologien

**Modulcode:** 4TI-INT-40

**Seminargruppe:** TI21

**Gutachter:** Dr. Mathias Sporer (Staatl. Studienakademie Glauchau)

Inhaltsverzeichnis

[Inhaltsverzeichnis II](#_Toc137807260)

[1 Zielführung 3](#_Toc137807261)

[2 theoretische Grundlagen 3](#_Toc137807262)

[**2.1** **Cross-Platform-App** 3](#_Toc137807263)

[**2.2** **Xamarin** 4](#_Toc137807264)

[**2.3** **Application Programming Interface** 4](#_Toc137807265)

[3 praktische Grundlagen 5](#_Toc137807266)

[**3.1** **Einrichten von Visual Studio** 5](#_Toc137807267)

[**3.2** **Verwendung von Xamarin** 10](#_Toc137807268)

[**3.2.1** **Anlegen einer neues Klasse** 10](#_Toc137807269)

[**3.2.2** **Zugriff auf Gerätehardware und -software** 10](#_Toc137807270)

[**3.3** **Veröffentlichung von nativen Apps** 14](#_Toc137807271)

[4 Aufgaben 14](#_Toc137807272)

[**4.1** **geführte Aufgabe** 14](#_Toc137807273)

[**4.2** **selbstständige Aufgabe** 15](#_Toc137807274)

[Quellenverzeichnis 16](#_Toc137807275)

# **Zielführung**

Ziel der Belegarbeit ist eine Übungsanleitung zur Verwendung von Xamarin in Visual Studio. Diese Open-Source-Plattform ermöglicht das Entwickeln von nativen iOS-, Android- oder Windows-Applikationen. Die entsprechenden Anwendungen können betriebssystemunabhängig entwickelt werden und werden in nativen Paketen kompiliert. Ein geführtes Beispiel soll zu einem besseren Verständnis beitragen und das Lösen der eigenständigen Aufgabe ermöglichen.

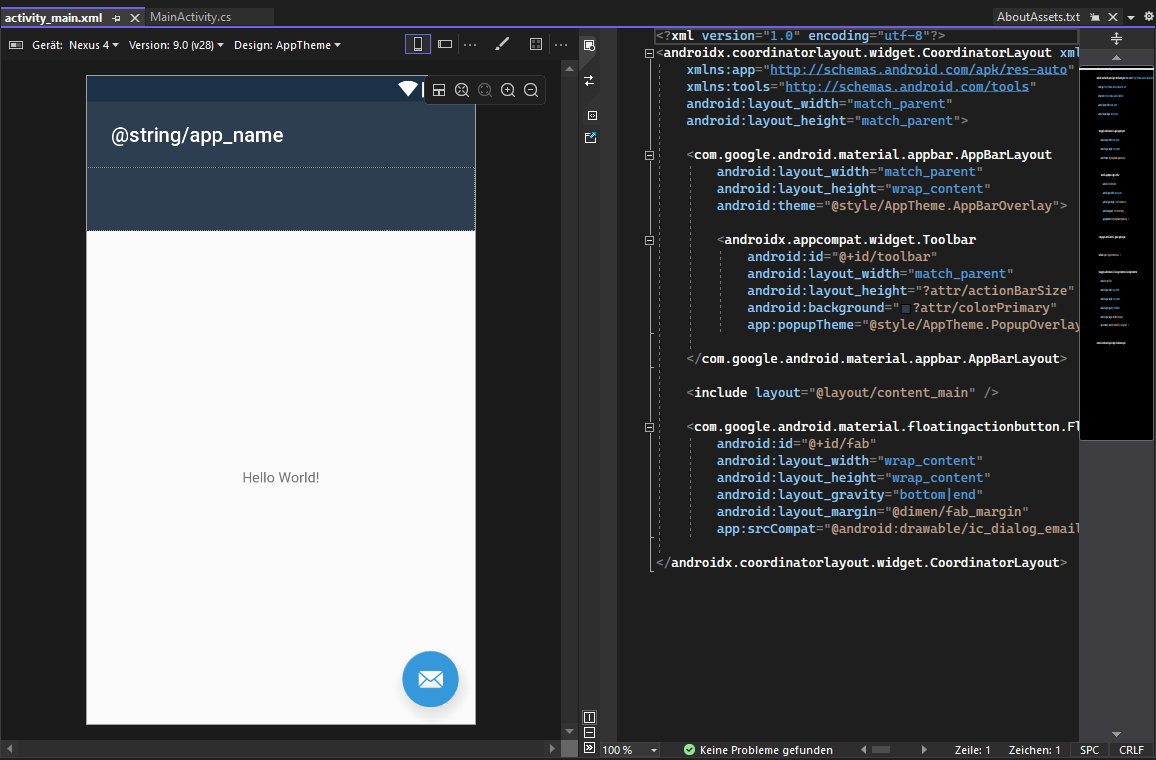
# **theoretische Grundlagen**

## **Cross-Plattform-App**

Da im Rahmen dieser Belegarbeit eine App entwickelt wird, welche sowohl für IOS als auch Android benutzbar ist, bietet sich eine Crossapplikation (Cross-App) an. Hierbei handelt es sich um eine App, welche aus nur einem Programm Code besteht und mithilfe eines Frameworks in die jeweilige Systemsprache kompiliert wird. Neben Cross-Plattform-Apps gibt es noch Web-Apps, Hybrid-Apps und native Apps. Letzteres wird beim Entwickeln genau auf das Betriebssystem angepasst. Was dazu führt, dass native App die Leistungsfähigsten sind und auch auf alle Features des Endgerätes zugreifen können. Bei einer Web-App handelt es sich um eine App die über den Webserver geladen und im Webbrowser ausgeführt wird. Eine Mischung aus diesen beiden Arten bildet die Hybrid-App. Sie greift zum Teil auf das Betriebssystem des Endgerätes zu, aber nutz auch Web-Anwendungen.

Bei einer Cross-App kann 90 % des Codes für jedes Betriebssystem genutzt werden. So muss nur ein kleiner Teil der Anwendung auf die Betriebssysteme angepasst werden. Dadurch ist diese Variante einer App sehr kosten günstig und zeitsparend. Ein weiterer Vorteil ist, dass eine Cross-App sich augenscheinlich nicht von einer Nativen-App, welche direkt für das Betriebssystem programmiert wird, unterscheidet. Jedoch kann diese Art von App nicht auf alle Funktionen des Endgerätes zugreifen, wie es beispielsweise eine Native-App kann. Außerdem ist die App durch die Kompilierung größer und hat somit leichte Performanceeinbuße, welche der Anwender jedoch kaum bemerkt. Jedoch muss beachtet werden, dass eine Cross-App von Betriebssystem-Updates nicht verschont bleibt. Das bedeutet, dass die Frameworks immer wieder an die neuen Verhältnisse bzw. Bedingungen des Betriebssystems angepasst werden müssen.

## **Xamarin**

Um eine Cross-Plattform-App zu entwickeln, benötigt zuerst ein Framework, um die App für mehrere Betriebssysteme zugänglich zu machen. Das Framework, welches im Rahmen dieser App Entwicklung verwendet wird, ist Xamarin. Xamarin unterstützt hauptsächlich C# und .Net und bietet SDKs für Android und IOS. Mit Software Development Kits (SDKs) können Anwendungen für spezifische Plattformen erstellt werden. Diese werden meist vom Hersteller der Plattform oder des Betriebssystems bereitgestellt. Zu einem SDK gehören Compiler, Debugger und APIs. Weiterhein kann ein SDK eine Dokumentation, Editoren, Bibliotheken und Netzwerkprotokolle enthalten. Weiterhin beinhaltet Xamarin Xamarin.Forms mit welchen Oberflächen durch Beschreibungssprache erstellt werden können. Xamarin.Forms nimmt normal UI-Elemente und wandelt sie in die Entsprechenden Elemente des nativen Betriebssystems um. Das spart einen Großteil an Entwicklungszeit, da die Benutzeroberfläche nicht mehr für jedes Betriebssystem einzeln erstellt werden muss. Durch die Komplexität ist Xamarin jedoch sehr Fehleranfällig und muss bei jedem Betriebssystem Update geprüft werden.

## **Application Programming Interface**

Benötigt die native App eine Datenbank, um auf dessen Daten zuzugreifen oder Daten abzuspeichern, wird ein Kommunikationsprogramm für den Austausch zwischen App und Datenbank benötigt. Hierfür wird hauptsächlich eine API verwendet, welche den Zugangspunkt für den Server regelt. Eine API besitz einen Satz von Befehlen, Protokollen, Funktionen und Objekten, welche von Programmieren verwendet, werden können, um Software zu erstellen oder um mit einem externen System zu interagieren. Weiterhin stellt die API-Standardbefehle für die allgemeinen Operationen zu Verfügung und ermöglicht die Kommunikation zwischen Anwendungen. Es gibt verschiedene Arten von APIs, welche für verschiedene zwecke genutzt werden können. Eine Internal API bleibt für externe Nutzer verborgen und ist nur für interne Nutzung. Die Customer API ist für Geschäftspartner bestimmt, das heißt sie ist eingeschränkt verfügbar und ist über öffentliche API-Entwicklerportale zugänglich. Die Open Source API ist die am häufigsten verwendete API. Diese API wird im Internet veröffentlicht und ist für die Softwareentwickler öffentlich zugänglich. Durch eine Open Source API bleibt die App Struktur flexibel, da diese immer wieder angepasst und verändert werden kann. Neben den verschiedenen Arten von APIs gibt es noch verschiedene API-Protokolle, mit welchen der Informationsaustausch erleichtert wird. Eine Art dieser Protokolle ist das SOAP (Simple Object Access Protocol). Dies Protokoll basiert auf XML und mit ihm kann ein User über SMTP und HTTP Daten senden und empfangen. Eine weitere Art eines API-Protokolls ist das REST-Protokoll. Jedoch handelt es sich hierbei eher um einen Satz der Web-API-Architektur. Das bedeutet das es keine offiziellen Standards gibt. Damit die API eine REST-API ist müssen die Schnittstellen bestimmte architektonische Einschränkungen beinhalten.

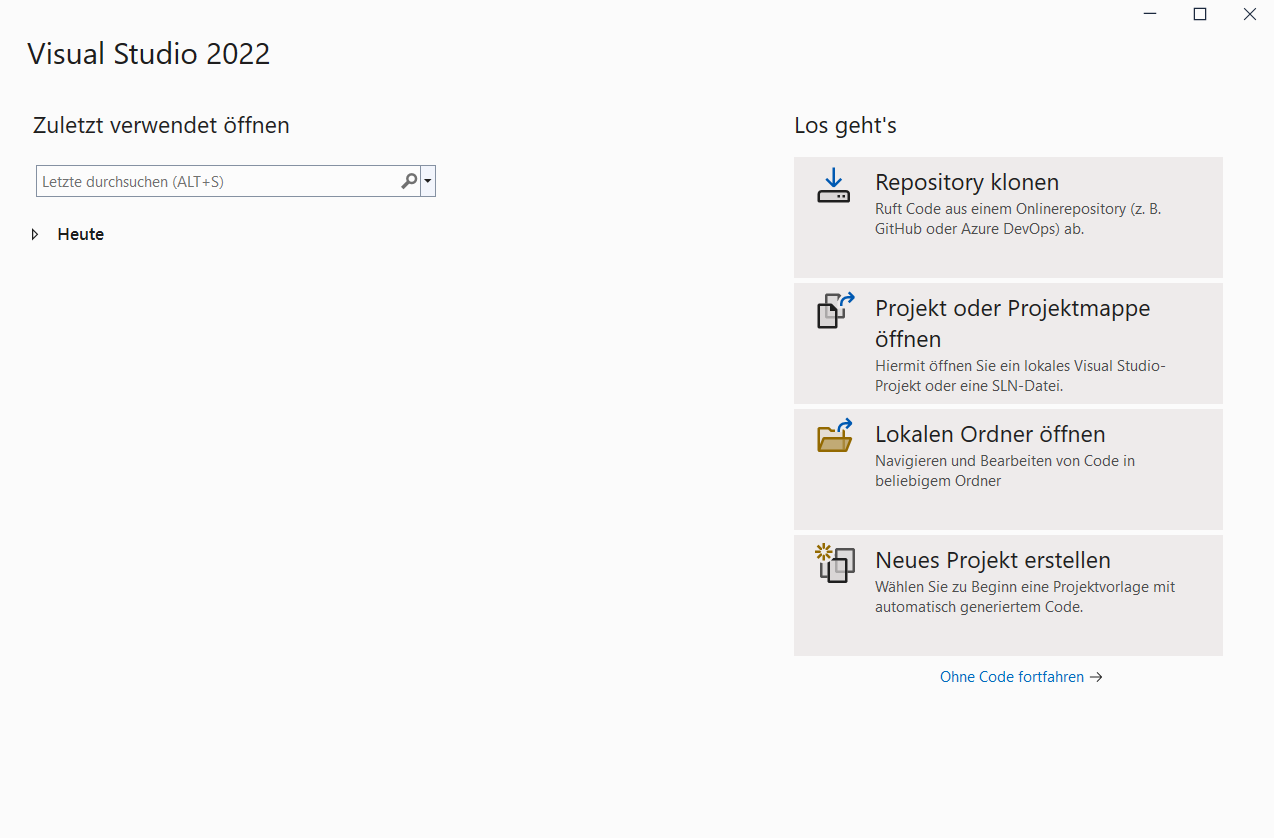
Die Funktionsweise einer API lässt sich wie folgt beschreiben. Zuerst wird über eine Clientanwendung ein Aufruf der API gestartet, um beispielsweise Informationen abzurufen. Hierbei leitet der URI die Anfrage an den Webserver weiter. Nach dem die Anfrage eingegangen ist ruft die API den Webserver oder ein externes Programm auf, um die Informationen zu holen. Der Server sendet die Informationen an die API, welche diese dann auf die ursprüngliche Anwendung überträgt. Hierbei ist zu beachten, dass die Anfrage und Antwort immer über die API laufen aber sich die Datenübertragung je nach Web-Service unterscheiden kann.

# **praktische Grundlagen**

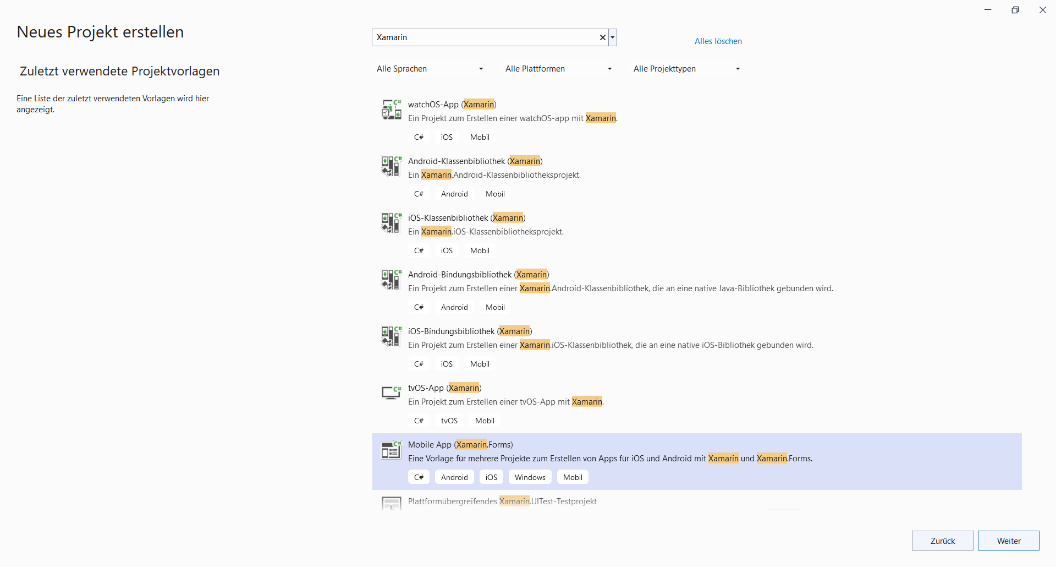
## **Einrichten von Visual Studio**

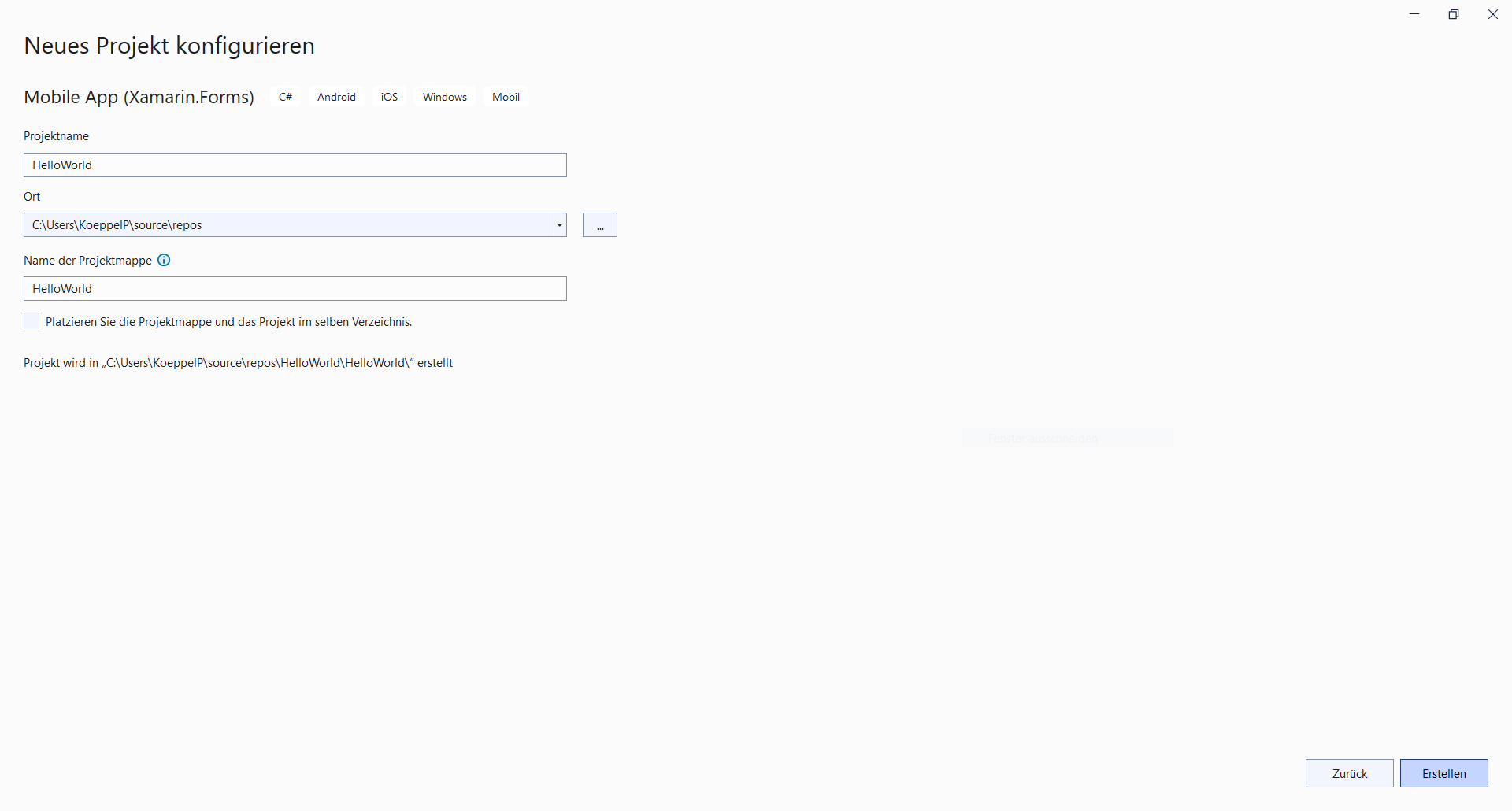
Um Xamarin für die Entwicklung einer nativen App zu verwenden, ist die Installation des Workloads *.NET Multi‑Plattform App‑Benutzeroberflächenentwicklung* notwendig. Zusätzlich muss in den Installationsdetails als optionales Paket ausgewählt und heruntergeladen werden.

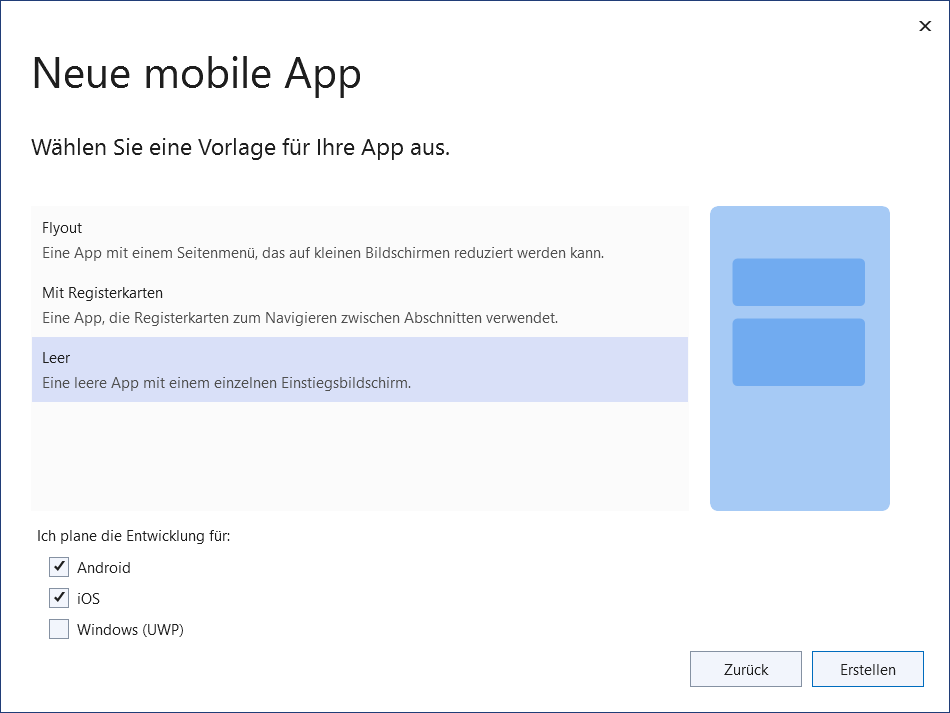
Nach Abschluss des Installationsvorgangs kann Visual Studio gestartet und ein erstes Startprojekt angelegt werden. Zur Erstellung ist *Neues Projekt erstellen* auszuwählen.

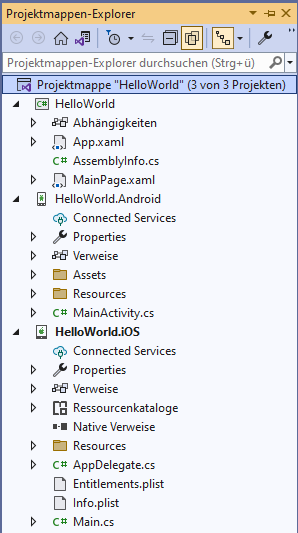


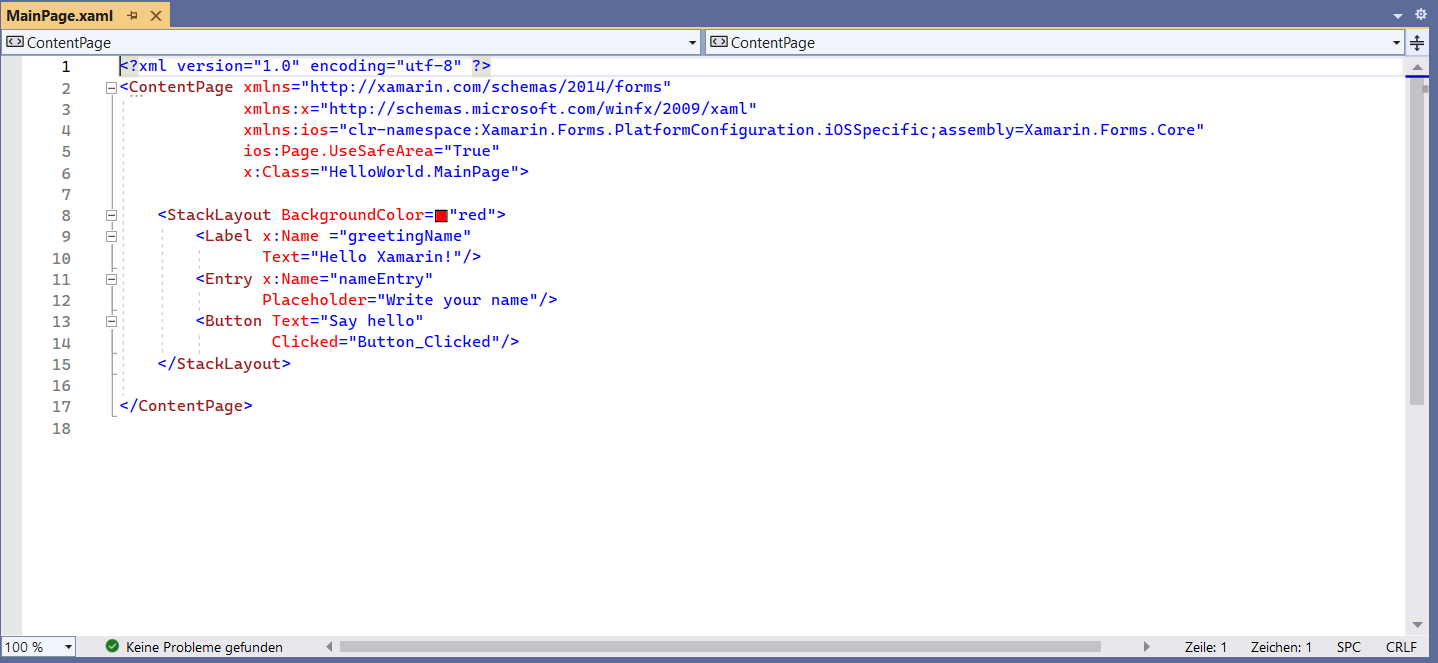
Um in die passende Vorlage zu finden, ist in die Suchleiste *Xamarin* einzugeben. Als Ergebnis werden verschiedene Entwürfe für Plattformen wie Android und iOS und unterschiedliche Geräte wie Handys, Uhren und TV-Geräte vorgeschlagen. Für die Entwicklung einer nativen App ist die Verwendung von *Mobile App (Xamarin.Forms)* passend.

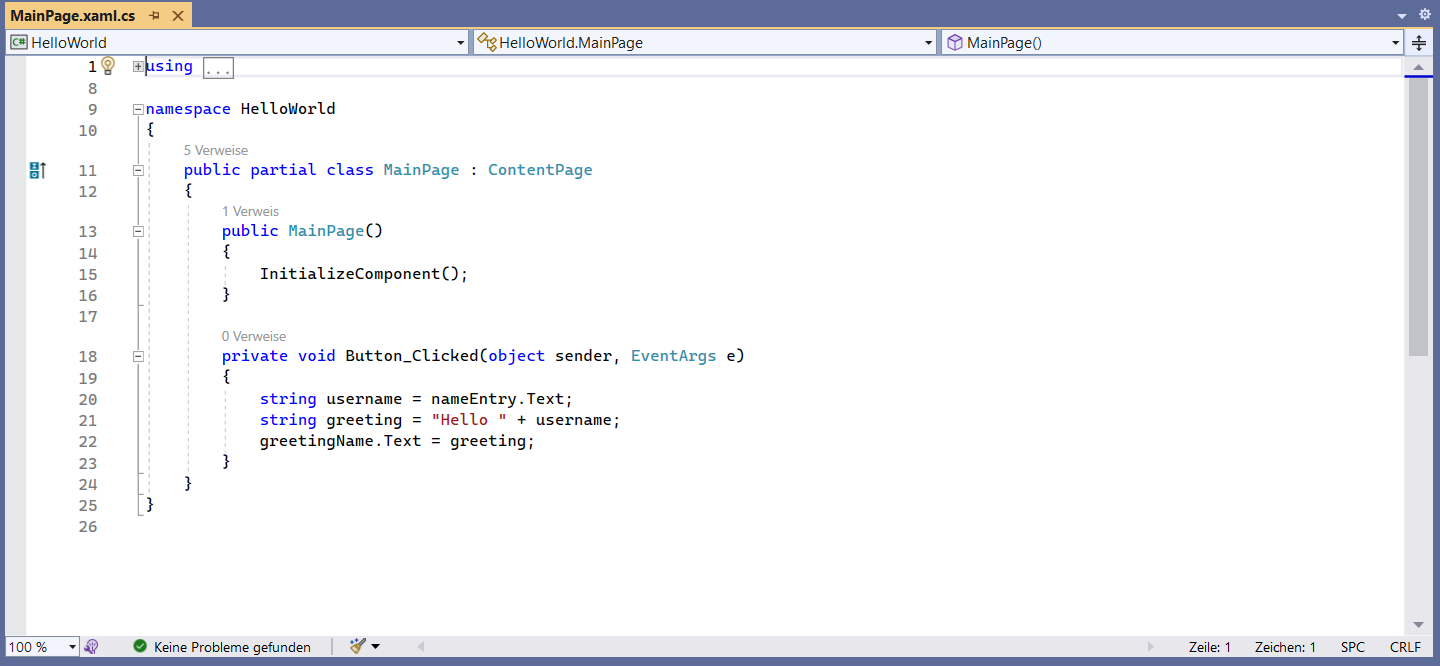


Als simpler Projektname für die Starter‑App ist *HelloWorld* angebracht. Im nächsten Schritt ist eine Vorlage für die Anwendung auszuwählen. In diesem Testfall ist *Leer* zu wählen, um das Projekt so einfach wie möglich zu erstellen. Da eine Android und iOS‑Applikation zu erstellen ist, sind diese beiden Fälle in den unteren Kästchen anzuhaken.

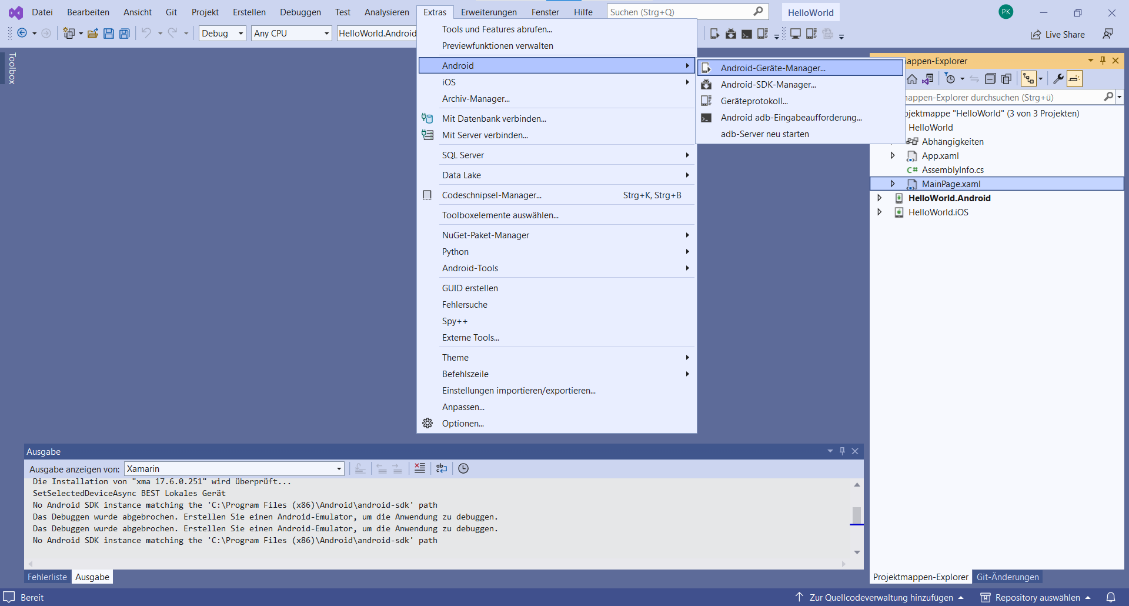


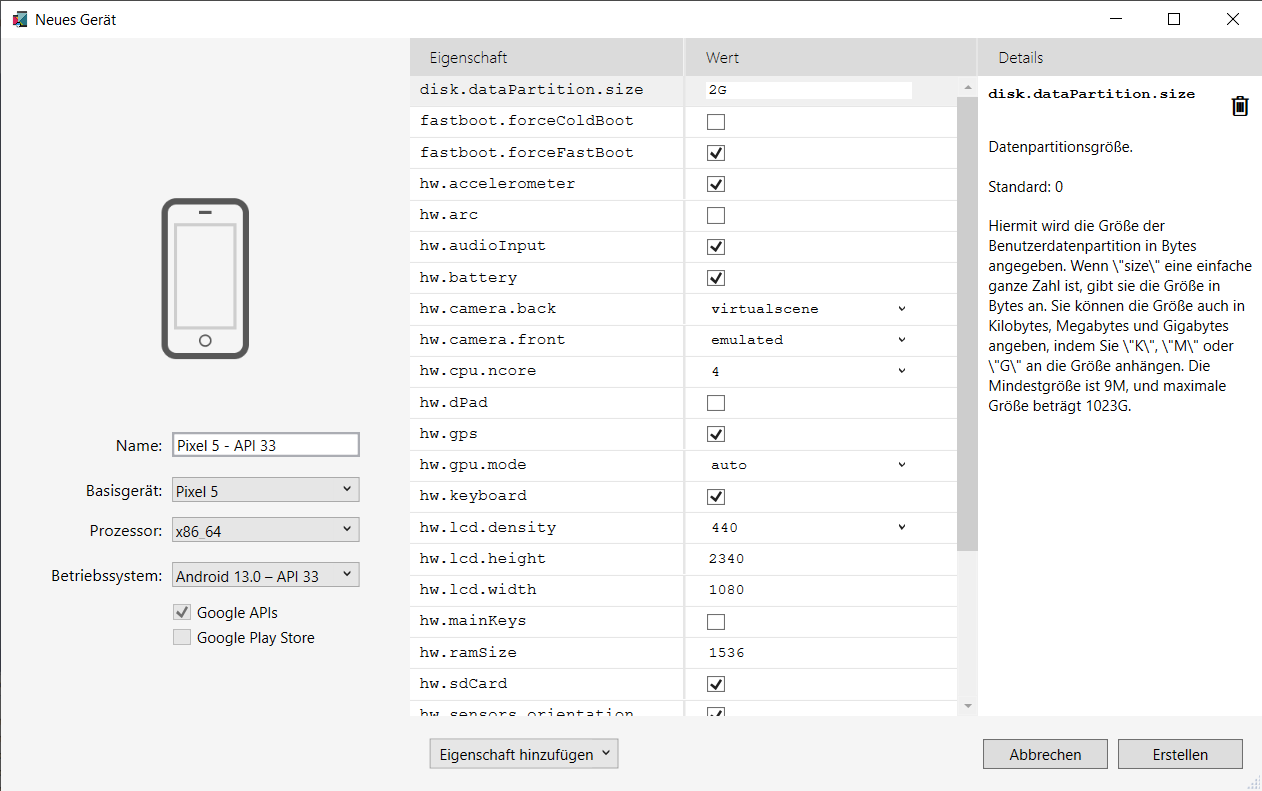
Die erstellte Projektmappe besteht aus drei einzelnen Projekten. In dem Projekt ohne iOS oder Android Erweiterung, werden alle Funktionalitäten programmiert und das Nutzerinterface erstellt. In diesem Projekt wird der Großteil der Anwendung implementiert. Der Aufbau Projekte ist identisch zu dem Aufbau bei der expliziten Appentwicklung. Der einzige Unterschied ist, dass ausschließlich C# Dateien zum Einsatz kommen. Innerhalb des Hauptprojekt ist die Klasse *App* zu finden, welche die Dateien *App.xaml* und *App.xaml.cs* enthält. *App.xaml.cs* enthält die gesamte Funktionalität des Projektes. In ihrem Konstruktor wird das Programm erstellt. Dieser Konstruktor wird in den Hauptklassen der jeweiligen Plattformprojekte aufgerufen. Der Aufruf ist möglich, da das Hauptprojekt in den Verweisen referenziert wurde. Der Programmcode des Konstruktors erzeugt ein neues Objekt der Klasse *MainPage*, welche ebenfalls in die Dateien *MainPage.xaml* für die Nutzeroberfläche und *MainPage.xaml.cs* für die Logik unterteilt ist.

Für ein einfaches *HelloWorld*-Programm wird der Quelltext der *MainPage*-Klasse verändert. Es muss der Bereich innerhalb des *<StackLayout></StackLayout>* geleert werden. Ziel ist es durch das Betätigen eines Buttons den in ein Textfeld eingegebene Namen auf einem Label auszugeben. Um die benötigten Elemente einzufügen kann unter Ansicht die Toolbox verwendet werden. Des Weiteren muss eine SafeArea für iOS angelegt werden. Um die Größe der Applikation anzupassen. Zu beachten ist, dass die Bestandteile die folgenden Eigenschaften enthalten:

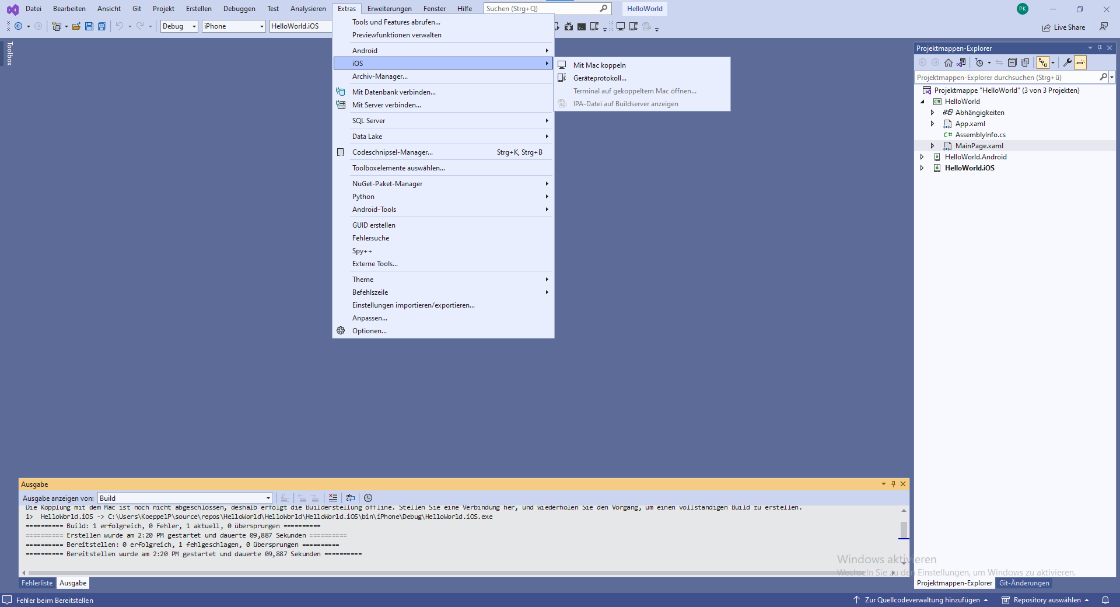
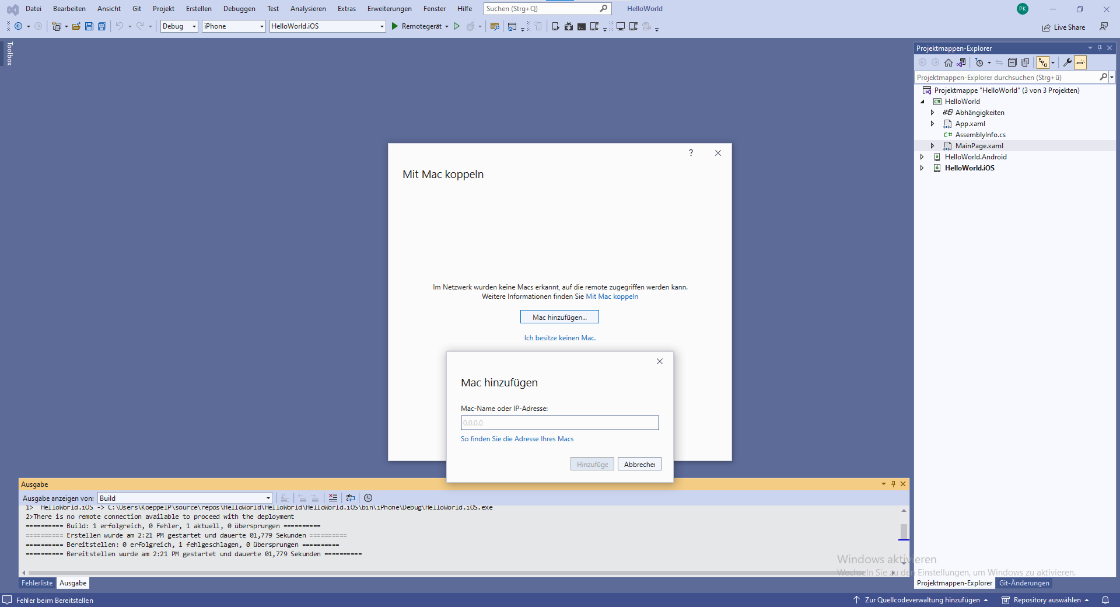
Durch das Einfügen der *Clicked*-Eigenschaft wird innerhalb der *MainPage.xaml.cs* Datei eine Methode angelegt, welche beim Aufruf des Klickereignisses durchlaufen wird. Innerhalb dieser Klasse wird der folgende Quelltext verwendet, um den eigetragenen Namen zu holen, zu modifizieren und erneut auszugeben.

Bei der ersten Ausführung des Quelltextes müssen für iOS und Android einige Vorbereitungen getroffen werden. Um das Android-Projekt zu testen, muss im *Android Device Manager* ein neuer Emulator angelegt werden. Dieser kann über **Extras -> Android -> Android-Geräte-Manager** geöffnet werden.



In dem geöffneten Fenster kann nun ein neues Gerät erstellt werden. Der Manager ermöglicht das Arbeiten mit verschiedenen Geräten mit unterschiedlichen Prozessoren und Betriebssystemen. Der Nutzer hat die Möglichkeit das Smartphone nach seinen eigenen Vorstellungen anzupassen. Zu testzwecken sind die Standarteinstellungen des Pixel 5 mit einem x86\_64 Prozessor und Android 13.0 ausreichend. Nach dem Erstellen dieses Emulators ist es möglich die App darauf zu starten.

Das Testen des iOS-Projektes ist über zwei Wege möglich. Für die erste Option mit einem Remotegerät ist ein Mac notwendig. Dieser wird mittels IP-Adresse im eigenen Netzwerk mit Visual Studio verbunden. Dies ist über **Extras -> Optionen… -> Xamarin -> iOS-Einstellungen** möglich. Optional kann der Nutzer unter *Simulator remote auf Windows* auswählen, ob das Testfenster auf dem Mac oder in Visual Studio zu sehen ist. Dann kann beim Start des Tests eines der vorgegebenen Geräte ausgewählt werden.



Option zwei ist das Testen über ein lokales Gerät. Voraussetzungen für diese Variante sind ein iPhone, iTunes auf dem PC und einen Apple Developer Account, welcher im Jahr 99 € kostet. Nach der Anmeldung mit der Apple ID, muss das Smartphone über USB mit dem PC verbunden werden und kann dann als Testgerät verwendet werden.

## **Verwendung von Xamarin**

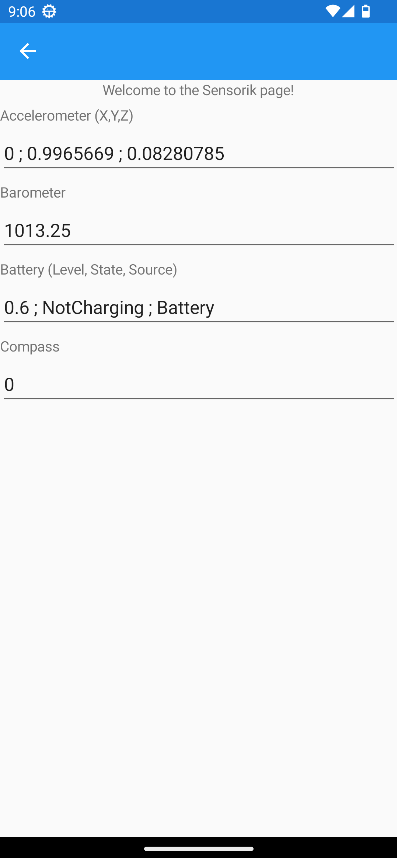
### **Anlegen einer neuen Klasse**

Xamarin.Forms besitzt bereits mehrere Vorlagen zur Erstellung von neuen Dateien. Es gilt die dabei die Unterscheidung zwischen zwei verschiedenen Vorlagetypen. Bei den Elementen ohne (C#) werden die XAML-Datei für das Format und die C#-Datei für die Logik erstellt. Bei den anderen Vorlagen wird ausschließlich die C#-Datei erstellt. Des Weiteren Unterscheidet Xamarin zwischen Inhaltsseite, Listenansichtsseite und einer Registerkartenseite. Die Inhaltsseite ist eine Standardseite ohne eine besondere Formatierung, in welcher der Inhalt simpel wieder gegeben wird. Bei einer Listenansichtsseite handelt es sich um eine Ansicht, welche eine ListView zum Anzeigen von verschiedenen Daten beinhaltet. Die Registerkartenseiten dienen zur Navigation zwischen verschiedenen Seiten. Sie besitzen einen Navigationskopf und enthalten untergeordnete Seiten, durch welche navigiert werden kann.

### **Zugriff auf Gerätehardware und -software**

Um auf dem Endgerät mit anderen Applikationen und Sensoren zu kommunizieren, existieren eine Vielzahl von verschiedenen NuGet-Paketen. Diese Bibliotheken enthalten Klassen, welche ein bestimmtes Problem lösen und dem Entwickler die Arbeit erleichtern. Sie sind in den meisten Fällen kostenlos und können einfach über **Projekt -> NuGet-Pakete verwalten…** im jeweiligen Projekt installiert werden. Eine wichtige Bibliothek für den Zugriff auf externe Daten ist Xamarin.Essentials. Dieses Paket von Microsoft enthält mehrere essenzielle APIs für die Applikation und wird regelmäßig angepasst und aktualisiert. Das Paket wird bereits bei der Erstellung des Programms installiert. Es sollte aber geprüft werden, ob bereits die neuste Version mitgeliefert wurde oder gegebenenfalls eine aktuellere Variante nachinstalliert werden muss. Den genauen Einsatz dieses Paketes soll an einigen Beispielen demonstriert werden. Für einige dieser Beispiele müssen separat Zugriffsrechte erteilt werden. Für Android werden diese unter **HelloWorldProject.Android ‑> Properties ‑> AndroidManifest.xml** festgelegt. Für das iOS-Projekt muss die Datei **Info.plist** überarbeitet werden. Der Programmcode für die nachfolgenden Beispiele ist im Anhang … aufgelistet.

Im ersten Teil soll der Zugriff auf einige Sensoren und Aktoren des Gerätes beschrieben werden. Hierfür wird ein neues Inhalts-Element mit der Bezeichnung *SensorPage.xaml* innerhalb des HelloWorldProject hinzugefügt. Um einen Zugriff auf diese Klasse zu haben, wird in der Klasse *MainPage.xaml* zu dieser per Button-Druck navigiert. Um diese Navigierung zur ermöglichen muss in der Hauptdatei *App.xaml.cs* der Zugriff auf die MainPage-Klasse abgeändert werden. Anstatt mit *MainPage = new MainPage()* diese direkt zu erzeugen, soll sie nun über eine NavigationPage welche als Wurzel das die neu erzeugte MainPage enthält, erzeugt werden. Im Quellcode findet dieser Zugriff wie folgt statt: *MainPage = new NavigationPage(new MainPage())*

Die Oberfläche der neu erzeugten Klasse SensorPage beinhaltet ausschließlich lesbare Textfelder, welche die Werte der Sensoren ausgeben und Beschriftungen für diese Textfelder.

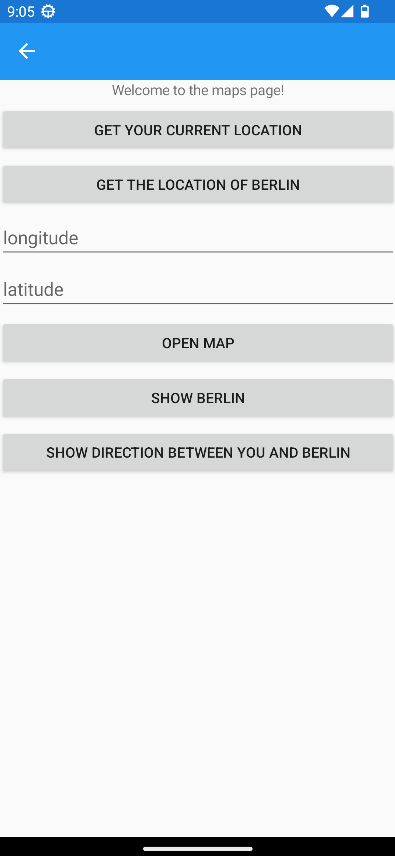
In der C#-Datei erfolgt der Zugriff auf die einzelnen Sensoren und Aktoren. In diesem Beispiel wird auf den Beschleunigungsmesser, das Barometer, die Batterie, den Kompass und den Erschütterungssensor zugegriffen. Als Aktor wird eine Vibration ausgeführt. Der Programmablauf für die einzelnen Sensoren ist analog. Es wird abgefragt, ob sich die Werte aktualisiert haben und dann die Inhalte der Textfelder aktualisiert. Die Vibration findet zwei Sekunden lang statt, wenn das Gerät geschüttelt wird. Für alle diese Zugriffe müssen die folgenden Berechtigungen für iOS und Android vergeben werden.

Ein weiteres Beispiel ist die Verwendung der Kamera. Auch hierfür wird eine neue Klasse mit dem Namen *CameraPage.xaml* erstellt. Diese lässt sich erneut von einem Button auf der MainPage aufrufen. Die Seite besteht aus vier Elementen. Einem Label mit einer kurzen Willkommensnachricht, einem Button zum Öffnen der Galerie, um ein Bild auszuwählen, ein Button zum Öffnen der Kamera, um ein Bild aufzunehmen und einem Image, um das jeweilige Bild anzuzeigen.

Die Zugriffe auf Kamera und Galerie erfolgen über die MediaPicker Klasse von Xamarin.Forms in einer asynchronen Methode. Dies ist notwendig, um bei Aufruf der Klasse mit await den aktuellen Thread nicht zu blockieren und die Task weiter auszuführen. Auch für dieses Beispiel müssen die folgenden Berechtigungen vergeben werden.



Ein weiteres Beispiel beschäftigt sich mit dem Umgang mit Karten in Xamarin.Forms in der neu erzeugten Klasse *MapsPage.xaml*. Die Seite besteht aus einem Willkommenslabel, zwei Textfeldern zur Ausgabe und Eingabe des gewählten Länge‑ und Breitengrades, einem Button, der die aktuelle Position ermittelt, einem Button, der die Koordinaten von Berlin anzeigt und mehreren Buttons, welche die ausgewählte Position auf der Karte anzeigen, Berlin auf der Karte anzeigen oder die Route zwischen dem aktuellen Standort und Berlin ausgibt.

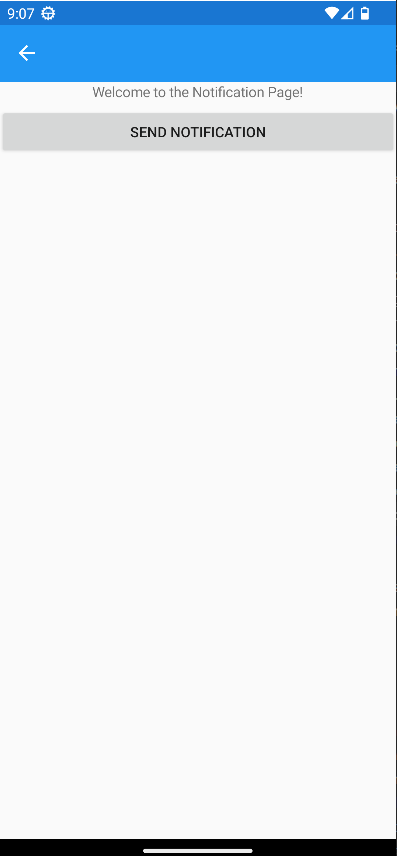


Die Karte wird in allen Fällen über die Funktion *OpenMapsAsync()* geöffnet. Der Quellcode zum Erzeugen und Ausgeben der Positionen ist im Anhang angeführt und genauer beschrieben. Auch für diese Funktionen müssen die folgenden Berechtigungen vergeben werden.

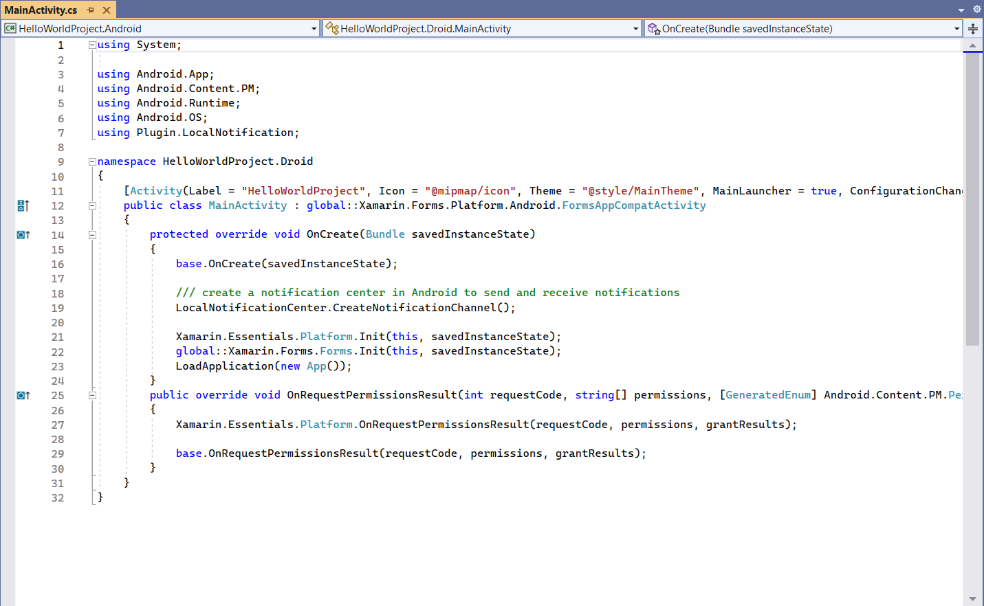


Xamarin Essentials deckt einen großen Teil der Funktionalitäten ab. Es ist beispielsweise mit diesem Paket auch möglich, eine SMS zu versenden, Text in Sprache auszugeben, Zugriff auf Kontakte und Zwischenablage zu haben und weitere Funktionen. Mit diesem Paket, ist allerdings nicht alles möglich. Beispielsweise kann mit Xamarin Essentials keine Datenbank erstellt und auch nicht darauf zugegriffen werden. Hierfür müssen in NuGet andere Pakete wie sqlite-net-pcl nachinstalliert und verwendet werden. Weiterhin gibt es ebenfalls keine Funktion zur Ausgabe einer lokalen Benachteiligung. Hierfür kann zum Beispiel das Plugin.LocalNotfication installiert werden.

Mit Hilfe dieses Paketes soll in der Klasse *NotificationPage.xaml* eine Benachrichtigung erstellt und ausgegeben werden. Bevor aber das Paket installiert und eingebunden werden kann, muss die Android‑Mindestversion auf Android 10.0 oder höher, gestellt werden. Dazu wird in die Eigenschaften des *HalloWorldProject.Android* Projekts navigiert und unter Android-Manifest die Einstellungen verändert. Um eine Benachrichtigung zu erhalten, wird ein Button angelegt, welcher dieses Ereignis ausführen soll.



Die Methode zum Senden der Benachrichtigung enthält eine Variable *notification*. Diese Variable ist vom Typ *NotificationRequest* und enthält Parameter wie den Titel, eine Beschreibung und eine Identifikation und wird mit *LocalNotificationCenter.Current.Show aufgerufen*. Zusätzlich können Events angelegt werden. Ein Event ist das Anzeigen eines Alerts, wenn eine Benachrichtigung eintrifft. Diese wird auf dem Main Thread der Applikation ausgeführt. Um diese Seite unter Android nutzen zu können, muss die Hauptdatei *MainActivity.cs* modifiziert werden. Es muss auch hier das Paket *LocalNotification* eingebunden und ein neuer Channel angelegt werden.



Die vollständige Beschreibung von Xamarin.Essentials ist zu finden unter <https://learn.microsoft.com/de-de/xamarin/essentials/>. Sollten andere Funktionen benötigt werden, die noch nicht im Projekt eingebunden sind, empfiehlt es sich die Suchfunktion von NuGet zu nutzen. An dieser Stelle findet man als Entwickler auch zahlreiche Informationen zu den Versionen, den Voraussetzungen, dem Paket- Entwicklern und eine Beschreibung, die in den meisten Fällen einen Link zu einer ausführlichen Dokumentation enthält.

## **Veröffentlichung von nativen Apps**

Um eine Native App zu veröffentlichen, muss sie auf einer entsprechenden Plattform hochgeladen werden. Die gängigsten Plattformen sind hierbei der AppStore von Apple und der Play Store von Google. Jedoch benötigt es einige Schritte, um die App final hochzuladen, sodass andere Nutzer sie benutzen können. Bevor die App veröffentlicht werden kann, gibt es einige Kosten, die berücksichtigt werden müssen. Falls die App Daten speichert oder auf sie zugreifen soll, wird eine Datenbank benötigt, die Kosten orientieren sich hierbei daran, wie viele User auf die Datenbank zugreifen und wie groß diese sein muss. Neben einer Datenbank tauchen auch Kosten für eine API auf, welche die Datenbank verwaltet, dies sind aber nur fällig, falls man eine API von Drittanbietern nutzt. Ein weiterer Punkt für die Kosten wäre der Server, welcher benötigt wird, um die Anwender der App zu hosten, hier richtet sich der Preis nach der Anzahl der aktiven Nutzer. Die letzten Kostenpunkte sind Wartung und Service, bei allen angebrachten Kostenpunkten handelt es sich um laufende Kosten. Das bedeutet, dass diese Kosten jährlich bzw. monatlich anfallen und bei der Preiskalkulierung beaufsichtigt, werden müssen. Neben den laufenden Kosten verlangen die Inhaber der Plattform, auf welcher die App hochgeladen wird, auch Geld. Bevor die App im App-Store veröffentlicht werden kann, wird als allererstes eine Mitgliedschaft im Apple Developer Programm benötigt, welche 99 € pro Jahr kostet. Weiterhin bekommt Apple anfangs eine Provision von 30 % und nach einem Jahr nur noch 15 % von den Einnahmen der App. Um die App zu veröffentlichen, benötigt man außerdem einen ITunes Connect Account und eine DUNS -Nummer. Des Weiteren muss die App den Gemeinschaftsrichtlinien entsprechen und Security und DSGVO Vorgaben erfüllen. Wenn alles abgeschlossen ist, kann die App über App-Store Connect hochgeladen werden. Danach kann man Sceenshots, Titel und Beschreibung hinzufügen und nachdem sie von Apple überprüft wurde, wird die App freigeschaltet. Wenn die App im Play-Store hochgeladen werden soll, benötigt man ein Entwicklerkonto, welches 25 € pro Jahr kostet. Zudem wird auch eine Provision von anfangs 30 % und nach einem Jahr 15 % verlangt. Um die App final zu veröffentlichen, müssen auch hier die Gemeinschaftsrichtlinien, Security- und DSGVO Vorgaben beachtet werden. Ist das alles ordnungsgemäß, kann die APP per APK-Datei hochgeladen werden und zum Schluss noch Screenshots, Titel, Kurzbeschreibung, Vorschaugrafik und Kategorie festgelegt werden. Freigeschaltet wird die App aber erst, nachdem sie von Google geprüft wurde.

# **Aufgaben**

## **geführte Aufgabe**

## **selbstständige Aufgabe**

# **Quellenverzeichnis**