WILHELM BÜCHNER HOCHSCHULE

MASTERTHESIS

---

# Realisierung eines Source-to-Source Compilers zwischen Xamarin.Forms und Flutter zur automatisierten Transformation bestehender mobiler Anwendungen

---

*Author:*
Julian Pasqué

*Betreuer:*
Dr. Thomas Kalbe

Verteilte und mobile Anwendungen

Fachbereich Informatik

Matrikelnummer: 902953

21. Januar 2021

# Zusammenfassung

# Abstract

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Quellcodeverzeichnis

# 1 Einleitung

Die Entwicklung von verschiedenen mobilen Geräten mit unterschiedlichsten Hardwarekomponenten und Betriebssystemen hat einen stark fragmentierten Markt ergeben.[1] Diese Situation hat einen direkten Einfluss auf die Softwareentwicklung, da die dedizierte Programmierung für die einzelnen Plattformen ressourcenintensiv ist. Durch Realisierung von Web- und hybriden Apps können Softwareprojekte von der darunterliegenden Plattform abstrahieren und plattformübergreifend verwendet werden. Diese Anwendungen haben jedoch, wie schon ausführlich im wissenschaftlichen Diskurs ausgeführt, eine schlechtere Performance und nur begrenzt Zugriff auf die plattformspezifischen Funktionalitäten.[2]

Durch die Kombination der Vorteile von Web- und Hybriden- mit denen von Nativen Anwendungen konnten Frameworks wie Xamarin.Forms und Flutter Programmierern die Möglichkeit bieten, ihre Anwendungen auf mehreren Plattformen bereit zu stellen. Diese Apps haben neben einer guten Performance auch Zugriff auf sämtliche plattformspezifischen Funktionalitäten. Durch die Abstraktion von Hardware und Betriebssystem können Apps mit einer gemeinsamen Quelltextbasis und somit mit geringerem Ressourcenaufwand entwickelt werden.[3]

Der Möglichkeit Ressourcen zu sparen steht das Risiko der Abhängigkeit gegenüber, da sich die oben genannten Frameworks zur Cross-Plattform-Entwicklung in den verwendeten Programmiersprachen sowie ihrer Arbeitsweise grundlegend unterscheiden. Ein Wechsel zwischen den einzelnen Alternativen ist daher mit enormen Arbeitsaufwänden verbunden.[4]

## 1.1 Motivation

Im Mai 2020 hat Microsoft mit .NET MAUI einen Nachfolger für das Xamarin.Forms Framework angekündigt, der im Herbst 2021 zusammen mit der sechsten Version des .NET Frameworks veröffentlicht werden soll. Zum aktuellen Zeitpunkt ist bereits berkannt, dass der Umstieg grundlegende Änderung mit sich bringt und Anwendungen,

[1] Vgl. Joorabchi 2016, S. 3.
[2] Vgl. Keist, Benisch und Müller 2016, S. 110ff.
[3] Vgl. Vollmer 2017, S. 295.
[4] Vgl. Wissel, Liebel und Hans 2017, S. 64.

die mit Hilfe von Xamarin.Forms entwickelt wurden, angepasst werden müssen.[5]

Für Xamarin.Forms Entwickler wird es also unausweichlich sein, grundlegende Änderungen an bereits realisierten Anwendungen vorzunehmen, um in der Zukunft von Aktualisierungen zu profitieren. Unternehmen und einzelne Programmierer stehen vor der Entscheidung, ob ein Umstieg auf das leistungsfähige Flutter sinnvoller ist, als die Anpassungen für das neue noch nicht erprobte .NET MAUI, das von einer Firma federführend entwickelt wird, welche leichtfertig mit der Abhängigkeit von Entwicklern umgeht. Ein automatisierter Umstieg auf das von Google entwickelte Framework Flutter würde also nicht nur die Anpassungen an .NET MAUI vermeiden, sondern die App auf eine vermeintlich zukunftssichere Basis stellen.

## 1.2 Ziel der Arbeit

Im Rahmen dieser Arbeit soll ein Source-To-Source Compiler zwischen den Frameworks Xamarin.Forms und Flutter realisiert werden, mit dessen Hilfe die folgende zentrale Forschungsfrage beantwortet werden soll: "Können Apps komplett automatisiert von Xamarin.Forms zu Flutter übersetzt werden, oder sind manuelle Arbeitsschritte erforderlich?"

## 1.3 Gliederrung

[5] Vgl. Hunter 2020, Abgerufen am 28.10.2020.

# 2 Compiler

Programmiersprachen dienen als Verständigungsmittel zwischen Programmierern und Rechenanlagen. Diese Sprachen haben sich in der Vergangenheit dabei immer mehr an die Terminologie eines bestimmtes Anwendungsgebietes angenähert. Durch diese Entwicklung eigneten sich Programmiersprachen direkt für die Dokumentation von entwickelten Algorithmen und Anwendungen, entfernten sich jedoch weiter von den Gegebenheiten des realen Rechners.[1] Für die Ausführung eines in einer problemorien-

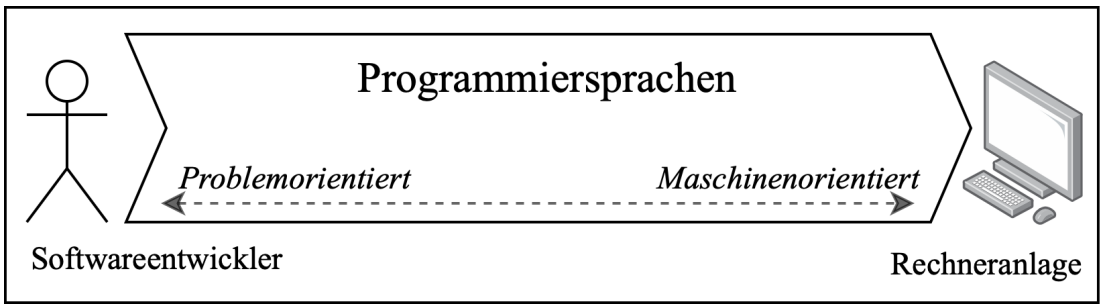


Abbildung 2.1: Programmiersprachen als Schnittstelle

tierten Programmiersprache ist es notwendig, die Sprache in eine maschinenorientierte Form zu überführen.[2] Bereits im Jahre 1951 stelle Rutishauser fest, dass Computer in der Lage sind diesen Übersetzungsvorgang selbst durchzuführen.[3]
Durch die Möglichkeit zur automatischen Übersetzung von problemorientierten Programmiersprachen konnten Hochsprachen entwickelt werden, die menschenfreundliche Sprachelemente anstatt maschineninstruktionen zu verwenden.[4]

## 2.1 Grundbegriffe

Diese historische Einführung zeigt, dass Software zur automatisierten Übersetzung schon seit der Mitte des letzten Jahrhunderts thematisiert wurde, so hat sich in der Wissenschaft eine einheitliche Definition ergeben. Ullman et al. beschreibt die sogenannten Compiler im Jahre 2008 wie folgt:[5]

[1] Vgl. Schneider 1975, S. 15.
[2] Vgl. Schneider 1975, S. 15.
[3] **Quelle fehlt! improve https://link.springer.com/article/10.1007/BF02009622**.
[4] Vgl. Wagenknecht und Hielscher 2014, S. 47.
[5] Vgl. Ullman et al. 2008, S. 1.

**Defintion 1: Compiler**

Ein Compiler ist ein Programm, welches ein anderes Programm aus einer Quellsprache in ein gleichwertiges Programm einer Zielsprache übersetzen kann.

Aus dieser Definition lässt sich ein für diese Arbeit relevanter Fakt ableiten: Compiler sind nicht ausschließlich Übersetzer zwischen zwischen problemorientierten,- und maschinenorientierten Programmiersprachen. Sie sind ausschließlich für die Übersetzung von einer Quellsprache in eine Zielsprache verantwortlich. Die einzige Bedingung ist, dass die Programme gleichwertig sind. Darauf basierend haben sich einige Anwendungsfälle für Compiler entwickelt, die ein anderes Ziel verfolgen. Dazu gehöre zum Beispiel:[6]

- Binärübersetzung: Dabei wird der Binärcode eines Programmes für einen Rechner in den für einen anderen Rechner übersetzt, sodass er das Computerprogramm Ausführung kann.

- Interpreter für Datenbankabfragen: Sprachen werden nicht nur für Software eingesetzt sondern auch für Abfragesprachen wie SQL (Structured Query Language). Diese Datenbankabfragen können zu Befehlen kompiliert werden um in einer Datenbank nach Datensätzen zu suchen.

Ein Source-to-Source(S2S) Compiler , häufig auch als "Transpiler"bezeichnet, ist ebenfalls eine besondere Ausprägung eines Compilers die sich wie folgt definieren lässt.[7]

**Defintion 2: Source-to-Source Compiler**

Ein Source-to-Source-Compiler ist ein Compiler, bei dem sowohl die Quellsprache als auch die Zielsprache eine Hochsprache ist.

### 2.1.1 Programm

Um zu verstehen, wann Programme gleichwertig sind ist es zuerst notwendig ein einheitliches Begriffsverständnis herzustellen. Auch wenn der Begriff Programm für je-

[6] Vgl. Ullman et al. 2008, S. 27.
[7] Vgl. Rohit, Aditi und Hardikar 2015, S. 1629.

dermann geläufig ist, unterscheidet man in der Literatur die Repräsentationen in denen ein Programm vorkommen kann.

- Der Quelltext:
- Ein Objektmodul:
- Ein ausführbares Programm:
- Ein Prozess:

### 2.1.2 Syntax und Semantik

Die Syntax einer Programmiersprache ist das Aussehen bzw. die Struktur des Quelltextes. Der komplette Quelltext eines Programms muss syntaktisch korrekt sein, damit er übersetzt werden kann.
Die Semantik ist die Bedeutung, die den verschiedenen syntaktischen Strukturen zugeordnet werden kann. Es ist also von der Semantik abhängig auf welche Art und Weise die Weiterverabeitung durch den Compiler erfolgt.

## 2.2 Aufgabe

Die Aufgaben des Compilers lassen sich, mit dem Ziel der Übersetzung von Programmiersprachen, in die zwei Unteraufgaben Analyse und Synthese unterteilen.[8]
Bei der Analyse wird das Programm in seine Bestandteile zerlegt und mit einer grammatischen Struktur versehen. Diese wird anschließend verwendet um eine Zwischendarstellung des Quellprogramms zu erstellen. Dabei wird überprüft, ob das Programm syntaktisch oder semantisch nicht wohlgeformt ist, und ob der Programmierer Änderungen vornehmen muss. Außerdem werden bei der Analyse Informationen über das Quellprogramm gesammelt und in einer so genannten Symboltabelle abgelegt.[9]
Bei der Synthese wird aus der Zwischendarstellung un den Informationen aus der Symboltabelle das gewünschte Zielprogramm konstruiert. Der Teil des Compilers, der sich mit der Analyse befasst wird oft als Front-End bezeichnet, derjenige der für die Synthese zustädnig ist als Back-End.[10]

[8] Vgl. Ullman et al. 2008, S. 6.
[9] Vgl. Ullman et al. 2008, S. 6f.
[10] Vgl. Ullman et al. 2008, S. 7.

## 2.3 Phasen

Der Vorgang des Kompilieren lässt sich nach Ullman et al. in mehere Phasen unterteilen, die an dieser Stelle eingeführt werden sollen.[11]

### 2.3.1 Lexikalische Analyse

Die erste Phase eines Compilers ist die sogenannte lexikalische Analyse. Dabei wird der Zeichenstream, der das Quellprogramm bildet in Lexeme gegliedert. Für jedes erzeugte Lexem gibt der lexikalische analysator ein Token in folgender Form aus:[12]

<Name, Attributwert>

Der Name ist dabei ein abstaktes Symbol, das während der nächste Phase, der Syntaxanalyse verwendet wird. Der Attributwert auf einen Eintrag in der symboltabelle für dieses Token zeigt. Diese Informationen werden in den späteren Phasen für die semantische Analyse und die Codegeneriung benötigt.[13]

### 2.3.2 Syntaxanalyse

Die zweite Phase des Compilers ist die Syntaxanalye. Dafür verwendet der sogenannte Parser die vom lexikalischen Analysator ausgegebenen Token um eine baumartige Zwischendarstellung zu erstellen. die die grammatische Struktur der Tokens zeigt. Die Darstellung wird daher auch häufig als Syntaxbaum bezeichnet. Die Knoten stehen dabei für eine Operation und seine Kindknoten für die Argumente dieser Operation. Die Anordnung er Operationen stimmt mit üblichen arithmentischen Konventionen überein, wie zum Beispiel der vorrang der Multiplikation vor Addition.[14]

[11] Vgl. Ullman et al. 2008, S. 6.
[12] Vgl. Ullman et al. 2008, S. 7f.
[13] Vgl. Ullman et al. 2008, S. 7f.
[14] Vgl. Ullman et al. 2008, S. 9.

### 2.3.3 Semantische Analyse

Bei der semantischen Analyse wird der Syntaxbaum und die Informationen aus der Symboltabelle verwendet um das Quellprogramm auf semantische Konsistenz mit der Sprachdefinition zu überprüfen. Außerdem werden in dieser Typinformationen gesammelt und entweder im Syntaxbaum oder in der Symboltabelle hinterlegt um sie in späteren Phasen zu verwenden. Dabei werden außerdem Typen überprüft, daher analysiert ob jeder Operator die passenden Operanden hat.[15]

### 2.3.4 Zwischencodeerzeugung

Bei der Übersetzung eines Quellprogramms in den Zielcode kann der Compiler mehrere Zwischendarstellungen in verschiedenen Formen erstellen. Syntaxbäume sind beispielsweise eine solche Darstellung. Nach der semantischen Analyse stellen viele Compiler eine maschinennahe Zwischendarstellung die für eine Abstrakte Maschine entworfen wurden.[16]

### 2.3.5 Codeoptimierung

In dieser Phase wird der maschinenunabhängige Code so optimiert, dass sich darauf ein besserer Zielcode ergibt. Dabei bedeutet besser, schnellerer code oder code der weniger Ressourcen verbraucht. Der Umfang der Codeoptimierung schwankt dabei von Compiler zu Compiler erheblich.[17]

### 2.3.6 Codeerzeugung

Bei der Codeerzegung werden die Eingaben aus der Zwischendarstellung des Quellprogramms entgegengenommen und auf die Zielsprache abgebildet. Ein entsheidender Aspekt der Codeerzeugung ist die sinnvolle zuweisung von Registern für Variablen, falls es sich bei der Zielsprache um Maschinencode handelt.[18]

[15] Vgl. Ullman et al. 2008, S. 9ff.
[16] Vgl. Ullman et al. 2008, S. 11.
[17] Vgl. Ullman et al. 2008, S. 11f.
[18] Vgl. Ullman et al. 2008, S. 13.

## 2.4 Rekursiver Ansatz

# 3 Cross Plattform Frameworks

Für die Realisierung eine Source-to-Source Compilers gibt es zwei relevante Faktoren, die für die Realisierung ausschlaggebend sind. Zum einen die Programmiersprachen in dem die beiden Frameworks entwickelt werden, da bei der Übersetzung eine Brücke zwischen Quell und Zielsprache geschlagen werden muss. Neben der Programmiersprache ist jedoch auch die Arbeitsweise eines Frameworks von essentieller Bedeutung. Wie die Definition von Compilern bereits einführte, müssen die Programme vor und nach der Übersetzung gleichwertig sein. Dies implementiert, dass das Verhalten der übersetzten Anwendungen nach der Übersetzung identisch seien muss wie das der Ursprungsanwendung. Es ist also notwendig, neben den sprachlichen auch die technischen Unterschiede zwischen den Frameworks zu kennen und diese im Rahmen der compilierung zu optimieren.

## 3.1 Frameworks

Xamarin is eine Open Source-Plattform für das Erstellen mobiler Anwendungen für iOS und Android mit Hilfe des .NET Frameworks, welches von Microsoft weiterentwickelt wird. Dabei ist Xamarin ist eine Abstraktionsebene, die die Kommunikation zwischen Code und dem zugrunde liegenden Plattformcode verwaltet. Xamarin wird in einer verwalteten Umgebung ausgeführt, die Vorteile wie Speicherbelegung und Garbage Collection bietet. Bei Xamarin.Forms handelt es sich um ein Open-Source-Benutzeroberflächenframework., mit dessen Hilfe Entwickler iOS- und Android- Anwendungen aus einer einzigen CodeBase erstellen können. Dabei wird auf die in der .NET Welt bekannten Technologien XAML für die Benutzeroberfläche und C# für die Anwendungslogik zurückgegriffen. Die einzelnen Benutzeroberflächen werden von Xamarin.Forms als native Steuerelemente auf jeder Plattform gerendert.[1]

Flutter ist ebenfalls wie Xamarin.Forms ein Open Source Framework für die Erstellung von 2D mobilen Anwendungen. Dabei werden im Vergleich zu Xamarin.Forms keine nativen Steuerelemente für jede Plattform gerendert sondern beinhaltet eine Sammlung von so genannten Widgets, die von dem Framework vewaltet und gerendert werden. Für die Anzeige dieser Widgets wird auf die 2D engine Skia zugegriffen. Flutter geht diesen Weg, da das Endergebnis der Anwendungen eine höhere Qualität verspricht,

[1] Vgl. Microsoft Corporation 2020, Abgerufen am 28.10.2020.

da die nativen Steuerelemente in Bezug auf Flexibilität und Qualität begrenzt sind. Außerdem ist es durch die Verwendung derselben Renderes einfacher, von derselben Codebasis aus für mehrere Plattformen zu veröffentlichen, ohne eine sorgfältige und kostspielige Planung vornehmen zu müssen, um verschiedene Funktionssätze und API-Merkmale aufeinander abzustimmen.[2]

Dieser essentielle Unterschied zwischen den Frameworks werden in den folgenden Abschnitten dieser Arbeit deutlicher und sind bei der Übersetzung der Anwendungen fokussiert zu Berücksichtigen.

### 3.1.1 Projekte

Xamarin.Forms Projektmappen setzen sich aus mehreren Projekten zusammen. Zwei Projekten für jeweils iOS und Android, welche den plattformspezifischen Code beinhalten sowie ein zusätzliches für den Quelltext, der zwischen den Plattformen geteilt wird. Im Gegensatz dazu gibt es bei Flutter nur ein Projekt, welches alle notwendigen Inhalte für iOS und Android beinhaltet. Xamarin.Forms bietet Entwicklern die Möglichkeit über die plattformbezogenen Projekte die nativen Renderer zu manipulieren. Durch diese sogenannten Custom Renderer (deutsch: benutzerdefinierter Renderer)ist es möglich unterschiedliche Darstellungen und Verhalten je nach Plattform zu erzeugen. Flutter bietet diese Möglichkeit nicht, da wie bereits beschrieben ausschließlich einheitliche Renderer Angeboten werden.

### 3.1.2 Views

Views (zu Deutsch Ansichten) sind visuelle Elemente die in zwei Kategorien unterschieden werden können. Controls, die für die Sammlung von Benutzereingaben oder die Ausgabe von Daten sind. Sowie Layouts die eine Sammlung von Ansichten beinhalten und für die Anordnung der untergeorderten Ansichten in der Benutzeroberfläche verantwortlich sind. Außerdem arbeitet sie mit jeder untergeordneten Ansicht zusammen, um die endgültige Rendering-Größe zu bestimmen.[3]

[2] Vgl. Google LLC 2020a, Abgerufen am 28.10.2020.
[3] Vgl. Ritscher 2020, Abgerufen am 28.10.2020.

### Pages

Pages (zu Deutsch: Ansichtseiten) sind visuelle Elemente einer Anwendung die den gesamten Bildschirm belegen und zu den Layout Views gehören. Xamarin Forms bietet dafür verschiedene Alternativen an, die in 3.1 grafisch dargestellt sind.[4]

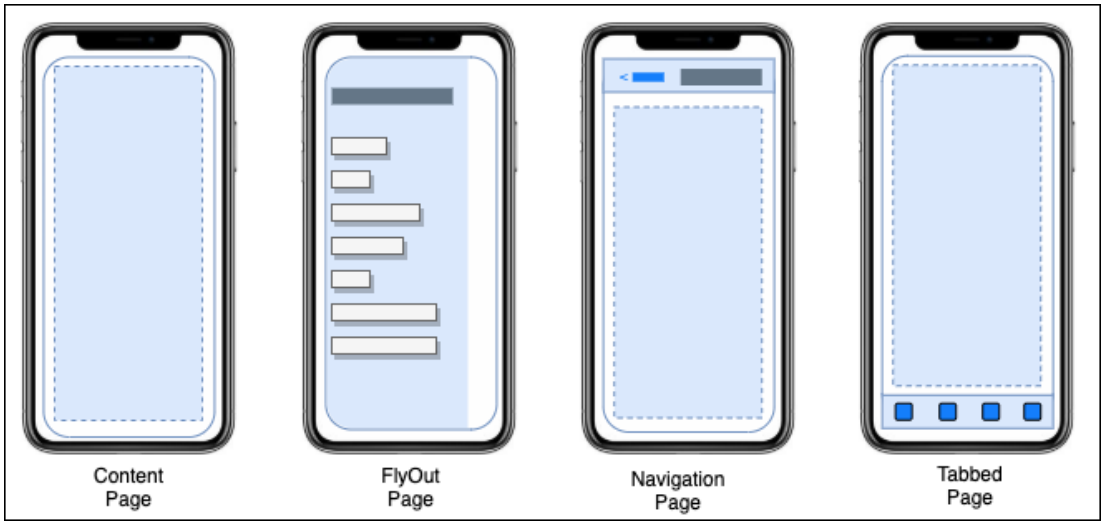


Abbildung 3.1: Xamarin.Forms Pages

Wie die Darstellung präsentiert, hat die Auswahl einer Page einen direkten Einfluss auf das Navigationskonzept innerhalb der Anwendung. Abgesehen von der Content-Page, die ausschließlich eine View anzeigt haben die jeweiligen Seiten das folgende Navigationskonzept:

- FlyOutPage: Eine Seite, die zwei Bereiche für die Seite hat. Typischerweise enthält das Flyout ein Menü über welches zwischen den eigentlichen Inhaltsseiten navigiert werden kann.
- NavigationPage: Eine Seite, die eine Navigationsleiste enthält. Die Seiten werden auf einem Stapel gehalten und es kann zwischen ihnen gesprungen werden. Die Navigationsleiste kann sowohl Navigationsschaltflächen als auch einen Titel enthalten.
- TabbedPage: Eine Container-Seite. Die TabbedPage fungiert als Container, der die mit jeder Registerkarte verbundenen Inhaltsseiten enthält.

Die Auswahl einer Page wird innerhalb des Wurzelknoten der XAML Datei definiert. Dies wird in Quelltext 3.1 dargestellt.

[4] Vgl. Microsoft Corporation 2016, Abgerufen am 28.10.2020.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="MasterThesisSample.SampleTabbedPage">
    <NavigationPage Title="Tab 1"/>
    <NavigationPage Title="Tab 2"/>
    <NavigationPage Title="Tab 3"/>
</TabbedPage>
```

Quelltext 3.1: Xamarin.Forms TabbedPage definition

Das Beispiel zeigt eine TabbedPage mit drei in diesem Falle leeren NavigationsPages die als Children der TabbedPage hinzugefügt werden. Im Gegensatz zu der TabbedPage hat die FlyoutPage keine Sammlung von ChildrenPages sondern ein sogenanntes "Flyout"welches das Menu beinhaltet und die entsprechend ausgewählte Seite in eine Detailansicht läd. Dies wird in Quelltext 3.2 dargestellt.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<FlyoutPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="MasterThesisSample.SampleFlyoutPage">
    <FlyoutPage.Flyout>
        <ContentPage>
            <!-- FlyOut Menu -->
        </ContentPage>
    </FlyoutPage.Flyout>
    <FlyoutPage.Detail>
        <NavigationPage>
            <!-- Place for the DetailPage -->
        </NavigationPage>
    </FlyoutPage.Detail>
</FlyoutPage>
```

Quelltext 3.2: Xamarin.Forms FlyOut definition

Im Gegensatz zu Xamarin.Forms lässt sich bei Flutter auf der Ebene der Wurzel kein Navigationskonzept definieren, sondern ausschließlich das Style der Anwendung. Flutter unterstützt drei alternativen: MaterialApp erzeugt eine App mit dem von Google entwickelten Material Design, CupertinoApp für eine App im iOS-Stil oder die Definition eines eigenen Styles für eine individuelle Anzeige.[5] Quelltext 3.3 zeigt die Definition einer MaterialDesign App in Flutter.

[5] Vgl. Google LLC 2020b, Abgerufen am 28.10.2020.

```dart
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

Quelltext 3.3: Flutter MaterialApp definition

Von diesem Widget aus ist die eigentliche erste Seite ein weiteres zustandsabhängiges Widget. Dieses besteht aus zwei Teilen: Der erste Teil, der selbst unveränderlich ist, erzeugt ein State-Objekt, das den Zustand des Objekts enthält. Das State-Objekt bleibt während der Lebensdauer des Widgets bestehen. Das State-Objekt implementiert die build()-Methode für das zustandsabhängige Widget. Wenn sich der Zustand des Widget-Baums ändert, wird setState() aufgerufen was einen Build des entsprechenden Teils der Benutzeroberfläche auslöst. In Flutter ist die Benutzeroberfläche (auch bekannt als Widget-Baum) unveränderlich, dass bedeutet da der Zustand nicht mehr geändert werden kann, sobald dieser aufgebaut ist. Sie ändern Felder in Ihrer State-Klasse und rufen dann setState() auf, um den gesamten Widget-Baum neu zu erstellen.

Damit die Navigation ähnlich wie in Xamarin.Forms definiert werden kann müssen verschiedene Widgets in einem Widget Baum verschachtelt werden. Quelltext 3.4 zeigt dies für die Arbeit mit Tabbs. In diesem Beispiel wird eine TabBar mit drei Tab-Widgets erstellt und diese innerhalb einer AppBar platziert.[6]

[6] Vgl. Google LLC 2020c, Abgerufen am 28.10.2020.

```
class TabBarDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DefaultTabController(
        length: 3,
        child: Scaffold(
          appBar: AppBar(
            bottom: TabBar(
              tabs: [
                Tab(icon: Icon(Icons.directions_car)),
                Tab(icon: Icon(Icons.directions_transit)),
                Tab(icon: Icon(Icons.directions_bike)),
              ],
            ),
            title: Text('Tabs Demo'),
          ),
          body: TabBarView(
            children: [
              Icon(Icons.directions_car),
              Icon(Icons.directions_transit),
              Icon(Icons.directions_bike),
            ],
          ),
        ),
      ),
    );
  }
}
```

Quelltext 3.4: Flutter Tab Layout definition

### Layouts

Layouts werden in Xamarin.Forms verwendet, um die Steuerelemente der Benutzeroberfläche zu visuellen Strukturen zusammenzustellen. Dabei unterscheidet man zwischen Layouts die ausschließlich einen oder mehrere Inhalte beinhalten können. Xamarin.Forms bietet die folgende Layouts an:

- ContentView: ContentView enthält ein einzelnes untergeordnetes Element, das mit der Eigenschaft "Content"festgelegt wird. Die Eigenschaft Content kann auf jedes View-Derivat gesetzt werden, auch auf andere Layout-Derivate. ContentView wird meist als Strukturelement verwendet und dient als Basisklasse zu Frame.

- Frame: Die Klasse Frame leitet sich von ContentView ab und zeigt einen Rahmen um die Ansicht.
- ScrollView: Ist in der Lage, seinen Inhalt zu scrollen. Die Eigenschaft Content auf eine Ansicht oder ein Layout fest, das zu groß ist, um auf den Bildschirm zu passen. Legen Sie die Eigenschaft Orientierung fest, um anzugeben, ob der Bildlauf vertikal, horizontal oder beides sein soll.
- StackLayout: Positioniert untergeordnete Elemente in einem Stapel entweder horizontal oder vertikal, basierend auf der Eigenschaft Orientation.
- Grid: Grid positioniert seine untergeordneten Elemente in einem Raster aus Zeilen und Spalten. Die Position eines untergeordneten Elements wird über die angehängten Eigenschaften Row, Column, RowSpan und ColumnSpan angegeben.
- AbsolutLayout: Positioniert untergeordnete Elemente an bestimmten Positionen relativ zu ihrem übergeordneten Element. Die Position eines untergeordneten Elements wird über die angehängten Eigenschaften LayoutBounds und LayoutFlags angegeben. Ein AbsoluteLayout ist nützlich, um die Positionen von Ansichten zu animieren.
- RelativeLayout: Positioniert untergeordnete Elemente relativ zum RelativeLayout selbst oder zu ihren Geschwistern. Die Position eines Kindelements wird über die angehängten Eigenschaften angegeben, die auf Objekte vom Typ Constraint und BoundsConstraint gesetzt werden.

Alle verfügbaren Layouts werden in 3.2 grafisch dargestellt.

### Steuerelemente

Xamarin.Forms-Ansichten sind die Bausteine von plattformübergreifenden mobilen Benutzeroberflächen. Ansichten sind Objekte der Benutzeroberfläche wie Beschriftungen, Schaltflächen und Schieberegler, die in anderen grafischen Programmierumgebungen üblicherweise als Steuerelemente oder Widgets bezeichnet werden. Die von Xamarin.Forms unterstützten Ansichten leiten sich alle von der Klasse View ab.

### Steuerelemente für die Darstellung

- BoxView zeigt ein einfarbiges Rechteck an

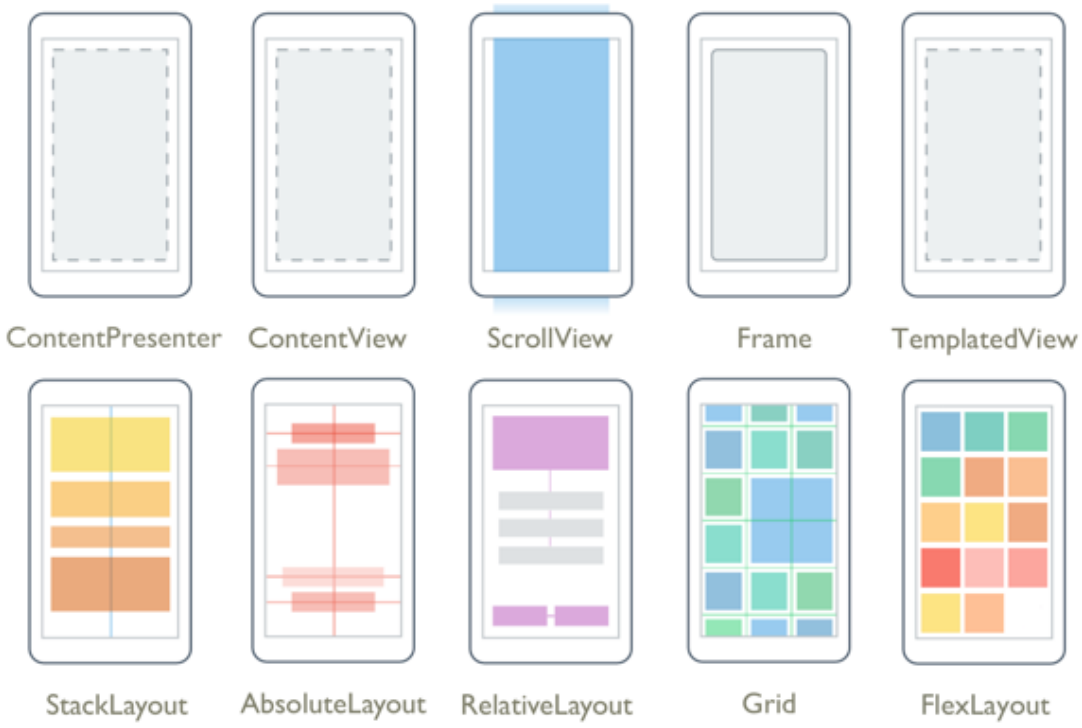


Abbildung 3.2: Xamarin.Forms Layouts

- Ellipse zeigt eine Ellipse oder einen Kreis an
- Label zeigt einzeilige Textstrings oder mehrzeilige Textblöcke an, entweder mit konstanter oder variabler Formatierung.
- Line zeigt eine Linie von einem Startpunkt zu einem Endpunkt an
- Image zeigt ein Bitmap an, diese können über das Web heruntergeladen, als Ressourcen in über das gemeinsame Projekt oder in Plattformprojekte eingebettet werden
- Map zeigt eine Karte an
- OpenGLView zeigt OpenGL-Grafiken an
- Path zeigt Kurven und komplexe Formen an.
- Polygon Polygon zeigt ein Polygon an.
- Polyline zeigt eine Reihe von verbundenen geraden Linien an
- Rectangle zeigt ein Rechteck oder Quadrat an
- WebView zeigt Web-Seiten oder HTML-Inhalte an

### Steuerelemente die Aktionen auslösen

- Button ist ein rechteckiges Objekt, das Text anzeigt und ein Clicked-Ereignis auslöst, wenn es gedrückt wurde.
- ImageButton ist ein rechteckiges Objekt, das ein Bild anzeigt und ein Clicked-Ereignis auslöst, wenn es gedrückt wurde.
- RadioButton erlaubt die Auswahl einer Option aus einer Menge und feuert ein CheckedChanged-Ereignis, wenn die Auswahl erfolgt.
- RefreshView ist ein Container-Steuerelement, das eine Pull-to-Refresh-Funktionalität für scrollbare Inhalte bietet.
- SearchBar zeigt einen Bereich an, in dem der Benutzer eine Textzeichenfolge eingeben kann, sowie eine Schaltfläche (oder eine Tastaturtaste), die der Anwendung signalisiert, eine Suche durchzuführen
- SwipeView ist ein Container-Steuerelement, das sich um ein Inhaltselement legt und Kontextmenüelemente bereitstellt, die durch eine Wischgeste angezeigt werden

### Steuerelemente um Werte zu setzen

- CheckBox ermöglicht dem Benutzer die Auswahl eines booleschen Wertes mit Hilfe einer Art Schaltfläche, die entweder markiert oder leer sein kann
- Schieberegler bieten Benutzern die Option einen Wert aus einem kontinuierlichen Bereich auswählen, der mit den Eigenschaften Minimum und Maximum festgelegt wurde.
- Stepper ermöglicht es dem Benutzer, einen doppelten Wert aus einem Bereich von inkrementellen Werten auszuwählen, die mit den Eigenschaften Minimum, Maximum und Inkrement festgelegt wurden.
- Schalter hat die Form eines Ein/Aus-Schalters, damit der Benutzer einen booleschen Wert auswählen kann
- DatePicker ermöglicht es dem Benutzer, ein Datum mit der Datumsauswahl der Plattform auszuwählen.
- TimePicker ermöglicht dem Benutzer die Auswahl einer Zeit mit dem TimePicker der Plattform

#### Steuerelemente um text zu manipulieren

- Entry ermöglicht dem Benutzer die Eingabe und Bearbeitung einer einzelnen Textzeile
- Editor ermöglicht dem Benutzer die Eingabe und Bearbeitung mehrerer Textzeilen

#### Steuerelemente um eine Aktivität anzudeuten

- ActivityIndicator verwendet eine Animation, um zu zeigen, dass die Anwendung eine langwierige Aktivität ausführt, ohne einen Hinweis auf den Fortschritt zu geben.
- ProgressBar verwendet eine Animation, um zu zeigen, dass die Anwendung durch eine langwierige Aktivität fortschreitet

#### Steuerelemente um Sammlungen anzuzeigen

- CarouselView zeigt eine blätterbare Liste von Datenelementen an
- CollectionView zeigt eine scrollbare Liste mit auswählbaren Datenelementen an, wobei verschiedene Layout-Spezifikationen verwendet werden. Sie soll eine flexiblere und performantere Alternative zu ListView darstellen.
- IndicatorView zeigt Indikatoren an, die die Anzahl der Elemente in einer CarouselView darstellen
- ListView zeigt eine scrollbare Liste mit auswählbaren Datenelementen an
- Picker zeigt ein ausgewähltes Element aus einer Liste von Textzeichenfolgen an und ermöglicht die Auswahl dieses Elements, wenn die Ansicht angetippt wird
- TableView zeigt eine Liste von Zeilen mit optionalen Überschriften und Unterüberschriften an

#### Listen

#### Gesten

#### Animtion

### 3.1.3 Navigation

In Xamarin.Forms bietet die Klasse NavigationPage eine hierarchische Navigation, bei der der Benutzer durch die Seiten, vorwärts und rückwärts, navigieren kann.

Flutter hat eine ähnliche Implementierung, die einen Navigator und Routen verwendet. Eine Route ist eine Abstraktion für eine Seite einer App, und ein Navigator ist ein Widget, das Routen verwaltet. Eine Route bildet grob eine Seite ab. Der Navigator arbeitet ähnlich wie die Xamarin.Forms NavigationPage, indem er Routen push() und pop() kann, je nachdem, ob man zu einer Ansicht hin oder von ihr zurück navigieren möchte.

#### Navigation zu anderen Apps

In Xamarin.Forms verwenden kann, um zu einer anderen Anwendung zu navigieren, ein bestimmtes URI-Schema verwendet werden. So z.B. mit dem Befehl Device.OpenUrl("mailto://") das Standard E-Mail-Programm des Gerätes geöffnet werden verwenden. Um diese Funktionalität in Flutter zu implementieren, muss eine native Plattformintegration oder eine vorhandenes Plugin, wie z. B. urllauncher verwendet werden.

### 3.1.4 Async UI

Dart hat ein Single-Thread-Ausführungsmodell mit Unterstützung für Isolates (eine Möglichkeit, Dart-Code in einem anderen Thread auszuführen), eine Ereignisschleife und asynchrone Programmierung. Sofern Sie kein Isolate erzeugen, wird Ihr Dart-Code im Haupt-Thread der Benutzeroberfläche ausgeführt und von einer Ereignisschleife gesteuert.

Das Single-Thread-Modell von Dart bedeutet nicht, dass Sie alles als blockierende Operation ausführen müssen, die das Einfrieren der Benutzeroberfläche verursacht. Ähnlich wie bei Xamarin.Forms müssen Sie den UI-Thread frei halten. Sie würden async/await verwenden, um Aufgaben auszuführen, bei denen Sie auf die Antwort warten müssen.

In Flutter verwenden Sie die asynchronen Möglichkeiten, die die Sprache Dart bietet, auch async await genannt, um asynchrone Arbeiten auszuführen. Dies ist C# sehr ähnlich und sollte für jeden Xamarin.Forms-Entwickler sehr einfach zu verwenden sein.

Sie können zum Beispiel Netzwerkcode ausführen, ohne dass die Benutzeroberfläche hängen bleibt, indem Sie async await verwenden und Dart die schwere Arbeit erledigen lassen: Sobald der erwartete Netzwerkaufruf erfolgt ist, aktualisieren Sie die Benutzeroberfläche durch den Aufruf von setState(), was einen Neuaufbau des Widget-Unterbaums auslöst und die Daten aktualisiert.

### 3.1.5 Hintergrundarbeiten

Da Flutter Single-Thread-fähig ist und eine Ereignisschleife ausführt, müssen Sie sich nicht um das Thread-Management oder das Erzeugen von Hintergrund-Threads kümmern. Dies ist sehr ähnlich wie bei Xamarin.Forms. Wenn Sie E A-gebundene Arbeiten durchführen, wie z. B. Festplattenzugriffe oder Netzwerkaufrufe, dann können Sie async await verwenden und alles ist bereit.

Wenn Sie andererseits rechenintensive Arbeiten ausführen müssen, die die CPU beschäftigen, sollten Sie sie in ein Isolate verschieben, um ein Blockieren der Ereignisschleife zu vermeiden, so wie Sie jede Art von Arbeit aus dem Hauptthread heraushalten würden. Dies ist ähnlich, wie wenn Sie Dinge über Task.Run() in Xamarin.Forms in einen anderen Thread verschieben.

Für E/A-gebundene Arbeit deklarieren Sie die Funktion als asynchrone Funktion und warten Sie auf lang laufende Aufgaben innerhalb der Funktion.

So würden Sie normalerweise Netzwerk- oder Datenbankaufrufe durchführen, die beide E/A-Operationen sind.

Es kann jedoch vorkommen, dass Sie eine große Datenmenge verarbeiten und Ihre Benutzeroberfläche hängen bleibt. In Flutter verwenden Sie Isolates, um die Vorteile mehrerer CPU-Kerne zu nutzen, um langlaufende oder rechenintensive Aufgaben zu

erledigen.

Isolates sind separate Ausführungsthreads, die sich keinen Speicher mit dem Hauptspeicherheap der Ausführung teilen. Dies ist ein Unterschied zu Task.Run(). Das bedeutet, dass Sie vom Haupt-Thread aus nicht auf Variablen zugreifen oder Ihre Benutzeroberfläche durch den Aufruf von setState() aktualisieren können.

### 3.1.6 Netzwerkaufrufe

In Xamarin.Forms würden Sie HttpClient verwenden. Einen Netzwerkaufruf in Flutter zu machen ist einfach, wenn Sie das beliebte http-Paket verwenden. Dieses abstrahiert einen Großteil des Netzwerks, das Sie normalerweise selbst implementieren würden, und macht es einfach, Netzwerkaufrufe zu tätigen.

Um das http-Paket zu verwenden, fügen Sie es zu Ihren Abhängigkeiten in pubspec.yaml hinzu.

Um eine Netzwerkanfrage zu stellen, rufen Sie await auf die asynchrone Funktion wie in 3.5 dargestellt auf:

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
[...]
  loadData() async {
    String dataURL = "https://jsonplaceholder.typicode.com/posts";
    http.Response response = await http.get(dataURL);
    setState(() {
      widgets = jsonDecode(response.body);
    });
  }
}
```

Quelltext 3.5: Flutter Network request

### 3.1.7 Lebenzyklus

In Xamarin.Forms haben Sie eine Anwendung, die OnStart, OnResume und OnSleep enthält. In Flutter können Sie stattdessen auf ähnliche Lebenszyklusereignisse

hören, indem Sie sich in den WidgetsBinding-Beobachter einklinken und auf das Änderungsereignis didChangeAppLifecycleState() hören.

Die beobachtbaren Lebenszyklus-Ereignisse sind:

‘inactive‘ Die Anwendung befindet sich in einem inaktiven Zustand und empfängt keine Benutzereingaben. Dieses Ereignis ist nur für iOS verfügbar. ‘paused‘ Die Anwendung ist derzeit für den Benutzer nicht sichtbar, reagiert nicht auf Benutzereingaben, wird aber im Hintergrund ausgeführt. Wiederaufgenommen Die Anwendung ist sichtbar und reagiert auf Benutzereingaben. Suspendiert Die Anwendung ist momentan angehalten. Dieses Ereignis ist nur für Android verfügbar.

### 3.1.8 Bilder

Flutter folgt einem einfachen dichtebasierten Format wie iOS. Assets können 1,0x, 2,0x, 3,0x oder ein anderer Multiplikator sein. Flutter hat keine dps, aber es gibt logische Pixel, die im Grunde dasselbe sind wie geräteunabhängige Pixel. Das sogenannte devicePixelRatio drückt das Verhältnis von physikalischen Pixeln zu einem einzelnen logischen Pixel aus.

Assets befinden sich in einem beliebigen Ordner - Flutter hat keine vordefinierte Ordnerstruktur. Sie deklarieren die Assets (mit Speicherort) in der Datei pubspec.yaml, und Flutter holt sie ab.

Beachten Sie, dass vor Flutter 1.0 beta 2 die in Flutter definierten Assets nicht von der nativen Seite aus zugänglich waren, und umgekehrt waren die nativen Assets und Ressourcen für Flutter nicht verfügbar, da sie in separaten Ordnern lagen.

Ab Flutter Beta 2 werden die Assets im nativen Asset-Ordner gespeichert und auf der nativen Seite über den AssetManager von Android aufgerufen:

Ab Flutter beta 2 kann Flutter immer noch nicht auf native Ressourcen zugreifen, noch kann es auf native Assets zugreifen.

Um zum Beispiel ein neues Bild-Asset mit dem Namen myicon.png zu unserem Flutter-Projekt hinzuzufügen und zu entscheiden, dass es in einem Ordner liegen soll, den wir willkürlich images genannt haben, würden Sie das Basisbild (1.0x) in den images-Ordner legen und alle anderen Varianten in Unterordnern, die mit dem entsprechenden Verhältnismultiplikator genannt werden:

### 3.1.9 Schriften

In Xamarin.Forms müssten Sie in jedem nativen Projekt eine eigene Schriftart hinzufügen. Dann würden Sie in Ihrem Element diesen Schriftnamen dem FontFamily-Attribut zuweisen, indem Sie filenamefontname und nur fontname für iOS verwenden.

In Flutter legen Sie die Schriftdatei in einem Ordner ab und referenzieren sie in der Datei pubspec.yaml, ähnlich wie Sie Bilder importieren. Weisen Sie dann die Schriftart Ihrem Text-Widget zu, wie in 3.6 dargestellt.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Sample App"),
    ),
    body: Center(
      child: Text(
        'This is a custom font text',
        style: TextStyle(fontFamily: 'MyCustomFont'),
      ),
    ),
  );
}
```

Quelltext 3.6: Flutter Font definition

### 3.1.10 Plugins

Im .NET-Ökosystem können native Xamarin-Projekte und Xamarin.Forms-Projekte Zugriff auf Nuget und das eingebaute Paketverwaltungssystem zurrückgreifen um. Flutter-Apps enthalten eine native Android-App, eine native iOS-App und eine Flutter-App. In Android fügen Sie Abhängigkeiten hinzu, indem Sie Ihr Gradle-Build-Skript ergänzen. In iOS fügen Sie Abhängigkeiten hinzu, indem Sie Ihr Podfile hinzufügen. Flutter verwendet das eigene Build-System von Dart und den Pub-Paketmanager. Die Werkzeuge delegieren die Erstellung der nativen Android- und iOS-Wrapper-Apps an die jeweiligen Build-Systeme. Generell sollten das pubspec.yaml verwendet werden, um externe Abhängigkeiten zu deklarieren, die in Flutter verwendet werden sollen. Ein guter Ort, um Flutter-Pakete zu finden, ist auf pub.dev.

### 3.1.11 Interaktion mit der Hardware

Flutter führt den Code nicht direkt auf der zugrundeliegenden Plattform aus. Vielmehr wird der Dart-Code, aus dem eine Flutter-App besteht, nativ auf dem Gerät ausgeführt, wobei das von der Plattform bereitgestellte SDK ümgangen"wird. Das bedeutet, wenn Sie zum Beispiel eine Netzwerkanfrage in Dart durchführen, wird diese direkt im Dart-Kontext ausgeführt. Sie verwenden nicht die Android- oder iOS-APIs, die Sie normalerweise beim Schreiben nativer Apps nutzen. Ihre Flutter-App wird immer noch im ViewController oder der Activity einer nativen App als View gehostet, aber Sie haben keinen direkten Zugriff auf diesen oder das native Framework.

Das bedeutet aber nicht, dass Flutter-Apps nicht mit diesen nativen APIs oder mit Ihrem nativen Code interagieren können. Flutter bietet Plattformkanäle, die mit dem ViewController oder der Activity, die Ihre Flutter-Ansicht hostet, kommunizieren und Daten austauschen. Plattformkanäle sind im Wesentlichen ein asynchroner Messaging-Mechanismus, der den Dart-Code mit dem Host-ViewController oder der Activity und dem iOS- oder Android-Framework, auf dem er läuft, verbindet. Sie können Plattformkanäle verwenden, um eine Methode auf der nativen Seite auszuführen oder um z. B. einige Daten von den Sensoren des Geräts abzurufen.

Zusätzlich zur direkten Verwendung von Plattformkanälen können Sie eine Vielzahl von vorgefertigten Plugins verwenden, die den nativen und Dart-Code für ein bestimmtes Ziel kapseln. Zum Beispiel können Sie ein Plugin verwenden, um auf die Kamerarolle und die Gerätekamera direkt von Flutter aus zuzugreifen, ohne eine eigene Integration schreiben zu müssen. Plugins finden Sie auf pub.dev, dem Open-Source-Paket-Repository von Dart und Flutter. Einige Pakete unterstützen möglicherweise native Integrationen auf iOS oder Android oder beides.

Wenn Sie kein Plugin auf pub.dev finden, das Ihren Anforderungen entspricht, können Sie Ihr eigenes schreiben und es auf pub.dev veröffentlichen.

### 3.1.12 Storage

Xamarin.Forms-Entwickler werden wahrscheinlich mit dem Settings-Plugin vertraut sein.

In Flutter greifen Sie auf die gleiche Funktionalität mit dem sharedmpreferences Plugin zu. Dieses Plugin umhüllt die Funktionalität von UserDefaults und dem Android-

Äquivalent SharedPreferences.

In Xamarin.Forms würden die meisten Anwendungen das sqlite-net-pcl-Plugin verwenden, um auf SQLite-Datenbanken zuzugreifen.

In Flutter greifen Sie auf diese Funktionalität mit dem sqflite-Plugin zu.

### 3.1.13 Notifications

## 3.2 Programmiersprachen

C# Dart Vergleich XAML Dart

# Literaturverzeichnis

Google LLC (2020a). *Flutter FAQ*. Website. Online erhältlich unter `https://flutter.dev/docs/resources/faq`; abgerufen am 28. Oktober 2020.

– (2020b). *What is the equivalent of a Page or Element in Flutter?* Website. Online erhältlich unter `https://flutter.dev/docs/get-started/flutter-for/xamarin-forms-devs`; abgerufen am 28. Oktober 2020.

– (2020c). *Work with tabs*. Website. Online erhältlich unter `https://flutter.dev/docs/cookbook/design/tabs`; abgerufen am 28. Oktober 2020.

Hunter, Scott (2020). *Introducing .NET Multi-platform App UI*. Website. Online erhältlich unter `https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/`; abgerufen am 28. Oktober 2020.

Joorabchi, Mona Erfani (Apr. 2016). „Mobile App Development: Challenges and Opportunities for Automated Support“. Diss. Vancouver: University of British Columbia.

Keist, Nikolai-Kevin, Sebastian Benisch und Christian Müller (2016). „Software Engineering für Mobile Anwendungen. Konzepte und betriebliche Einsatzszenarien“. In: *Mobile Anwendungen in Unternehmen*. Hrsg. von Thomas Barton, Christian Müller und Christian Seel, S. 91–120.

Microsoft Corporation (2016). *Xamarin.Forms Pages*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/pages`; abgerufen am 28. Oktober 2020.

– (2020). *What is Xamarin?* Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/get-started/what-is-xamarin`; abgerufen am 28. Oktober 2020.

Ritscher, Walt (2020). *Arranging Views with Xamarin.Forms Layout*. Website. Online erhältlich unter `https://bit.ly/34ulgpU`; abgerufen am 28. Oktober 2020.

Rohit, Kulkarni, Chavan Aditi und Abhinav Hardikar (2015). „Transpiler and it's Advantages“. In: *International Journal of Computer Science and Information Technologies* 6.2.

Schneider, Hans-Jürgen (1975). *Compiler. Aufbau und Arbeitsweise*. Berlin: Walter de Gruyter.

Ullman, Jeffrey D. et al. (2008). *Compiler. Prinzipien, Techniken und Werkzeuge*. 2. Aufl. München: Pearson Studium.

Vollmer, Guy (2017). *Mobile App Engineering. Eine systematische Einführung - von den Requirements zum Go Live*. 1. Aufl. Heidelberg: dpunkt.

Wagenknecht, Christian und Michael Hielscher (2014). *Formale Sprachen, abstrakte Automaten und Compiler. Lehr- und Arbeitsbuch für Grundstudium*. 2. Aufl. Wiesbaden: Springer.

Wissel, Andreas, Chrsitian Liebel und Thorsten Hans (2017). „Frameworks und Tools für Cross-Plattform-Programmierung“. In: *iX – Magazin für professionelle Informationstechnik* 2.

# Eidesstattliche Erklärung

Studierender: Julian Pasqué
Matrikelnummer: 902953

Hiermit erkläre ich, dass ich diese Arbeit selbstständig abgefasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

| | |
|---|---|
| ..... | ..... |
| Ort, Abgabedatum | Unterschrift (Vor- und Zuname) |