

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

30. Januar 2021

# Zusammenfassung

# Abstract

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>III</b>
<b>Abbildungsverzeichnis</b>	<b>V</b>
<b>Tabellenverzeichnis</b>	<b>VI</b>
<b>Quellcodeverzeichnis</b>	<b>VII</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Ziel der Arbeit . . . . .	2
1.3 Gliederrung . . . . .	2
<b>2 Compiler</b>	<b>3</b>
2.1 Grundbegriffe . . . . .	3
2.1.1 Programm . . . . .	4
2.1.2 Syntax und Semantik . . . . .	5
2.2 Aufgabe . . . . .	5
2.3 Phasen . . . . .	6
2.3.1 Lexikalische Analyse . . . . .	6
2.3.2 Syntaxanalyse . . . . .	6
2.3.3 Semantische Analyse . . . . .	7
2.3.4 Zwischencodeerzeugung . . . . .	7
2.3.5 Codeoptimierung . . . . .	7
2.3.6 Codeerzeugung . . . . .	7
2.4 Rekursiver Ansatz . . . . .	8
<b>3 Cross Plattform Frameworks</b>	<b>9</b>
3.1 Frameworks . . . . .	9
3.1.1 Projekte . . . . .	10
3.1.2 Views . . . . .	10
3.1.3 Navigation . . . . .	21
3.1.4 Async UI . . . . .	22
3.1.5 Hintergrundarbeiten . . . . .	22
3.1.6 Netzwerkaufrufe . . . . .	23
3.1.7 Lebenszyklus . . . . .	24
3.1.8 Bilder . . . . .	24
3.1.9 Schriften . . . . .	25

---

3.1.10	Plugins . . . . .	26
3.1.11	Interaktion mit der Hardware . . . . .	26
3.1.12	Storage . . . . .	27
3.2	Programmiersprachen . . . . .	28
3.2.1	In Dart ist alles Null . . . . .	28
3.2.2	Generics . . . . .	28
3.2.3	Delegates . . . . .	29
3.2.4	Das New Keyword . . . . .	29
3.2.5	Listen und Dictionaries . . . . .	30
3.2.6	Zugriffsmodifizierer . . . . .	30
3.2.7	Vererbung . . . . .	31
3.2.8	Namespaces . . . . .	32
3.2.9	Bibliotheken . . . . .	32
3.2.10	Async/Await . . . . .	33
3.2.11	Events . . . . .	33
	<b>Literaturverzeichnis</b>	<b>34</b>

# Abbildungsverzeichnis

2.1	Programmiersprachen als Schnittstelle . . . . .	3
3.1	Xamarin.Forms Pages . . . . .	11
3.2	Xamarin.Forms Layouts . . . . .	15

# Tabellenverzeichnis

# Quellcodeverzeichnis

3.1	Xamarin.Forms TabbedPage definition . . . . .	12
3.2	Xamarin.Forms FlyOut definition . . . . .	12
3.3	Flutter MaterialApp definition . . . . .	13
3.4	Flutter Tab Layout definition . . . . .	14
3.5	Flutter Network request . . . . .	24
3.6	Flutter Font definition . . . . .	26
3.7	Alles kann ich Dart Null sein . . . . .	28
3.8	Generics in Dart . . . . .	29
3.9	Delegates in Dart . . . . .	29
3.10	Optionales "New"Keyword in Dart . . . . .	30
3.11	Listen und Maps in Dart . . . . .	30
3.12	Private und Public Definitionen in Dart . . . . .	30
3.13	Vererbung in Dart . . . . .	31
3.14	Mixin's in Dart . . . . .	32
3.15	Importieren von Paketen in Dart . . . . .	32
3.16	Async und Await in Dart . . . . .	33
3.17	Events in Dart . . . . .	33



# 1 Einleitung

Die Entwicklung von verschiedenen mobilen Geräten mit unterschiedlichsten Hardwarekomponenten und Betriebssystemen hat einen stark fragmentierten Markt ergeben.<sup>1</sup> 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 begrenzten Zugriff auf die plattform-spezifischen Funktionalitäten.<sup>2</sup>

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 plattform-spezifischen Funktionalitäten. Durch die Abstraktion von Hardware und Betriebssystem können Apps mit einer gemeinsamen Quelltextbasis und somit mit geringerem Ressourcenaufwand entwickelt werden.<sup>3</sup>

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.<sup>4</sup>

## 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 bekannt, dass der Umstieg grundlegende Änderungen mit sich bringt und Anwendungen,

---

<sup>1</sup>Vgl. Joorabchi 2016, S. 3.

<sup>2</sup>Vgl. Keist, Benisch und Müller 2016, S. 110ff.

<sup>3</sup>Vgl. Vollmer 2017, S. 295.

<sup>4</sup>Vgl. Wissel, Liebel und Hans 2017, S. 64.

die mit Hilfe von Xamarin.Forms entwickelt wurden, angepasst werden müssen.<sup>5</sup>

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

---

<sup>5</sup>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 bestimmten 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.<sup>1</sup> Für die Ausführung eines in einer problemorien-

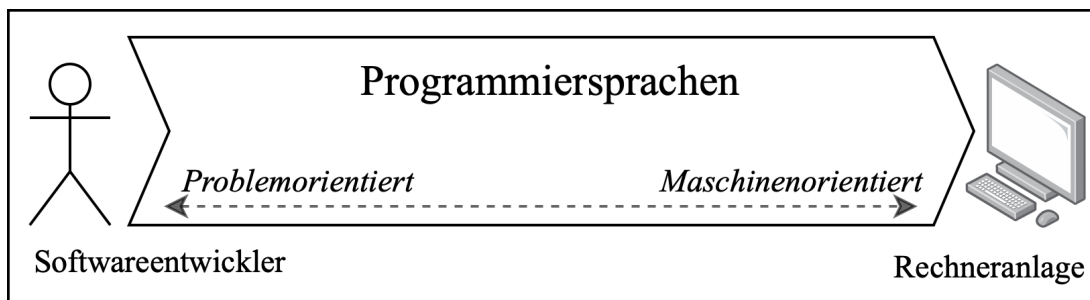


Abbildung 2.1: Programmiersprachen als Schnittstelle

tierten Programmiersprache ist es notwendig, die Sprache in eine maschinenorientierte Form zu überführen.<sup>2</sup> Bereits im Jahre 1951 stellte Rutishauser fest, dass Computer in der Lage sind diesen Übersetzungsvorgang selbst durchzuführen.<sup>3</sup>

Durch die Möglichkeit zur automatischen Übersetzung von problemorientierten Programmiersprachen konnten Hochsprachen entwickelt werden, die menschenfreundliche Sprachelemente anstatt maschineninstruktionen zu verwenden.<sup>4</sup>

### 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:<sup>5</sup>

<sup>1</sup>Vgl. Schneider 1975, S. 15.

<sup>2</sup>Vgl. Schneider 1975, S. 15.

<sup>3</sup>Quelle fehlt! improve <https://link.springer.com/article/10.1007/BF02009622>.

<sup>4</sup>Vgl. Wagenknecht und Hielscher 2014, S. 47.

<sup>5</sup>Vgl. Ullman et al. 2008, S. 1.

**Definition 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 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:<sup>6</sup>

- **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.<sup>7</sup>

**Definition 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-

<sup>6</sup>Vgl. Ullman et al. 2008, S. 27.

<sup>7</sup>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 Weiterverarbeitung 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.<sup>8</sup>

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.<sup>9</sup>

Bei der Synthese wird aus der Zwischendarstellung und 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 zuständig ist als Back-End.<sup>10</sup>

---

<sup>8</sup>Vgl. Ullman et al. 2008, S. 6.

<sup>9</sup>Vgl. Ullman et al. 2008, S. 6f.

<sup>10</sup>Vgl. Ullman et al. 2008, S. 7.

## 2.3 Phasen

Der Vorgang des Kompilieren lässt sich nach Ullman et al. in mehrere Phasen unterteilen, die an dieser Stelle eingeführt werden sollen.<sup>11</sup>

### 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:<sup>12</sup>

<Name, Attributwert>

Der Name ist dabei ein abstraktes 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 Codegenerierung benötigt.<sup>13</sup>

### 2.3.2 Syntaxanalyse

Die zweite Phase des Compilers ist die Syntaxanalyse. 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 der Operationen stimmt mit üblichen arithmentischen Konventionen überein, wie zum Beispiel der vorrang der Multiplikation vor Addition.<sup>14</sup>

---

<sup>11</sup>Vgl. Ullman et al. 2008, S. 6.

<sup>12</sup>Vgl. Ullman et al. 2008, S. 7f.

<sup>13</sup>Vgl. Ullman et al. 2008, S. 7f.

<sup>14</sup>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.<sup>15</sup>

### 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.<sup>16</sup>

### 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.<sup>17</sup>

### 2.3.6 Codeerzeugung

Bei der Codeerzeugung werden die Eingaben aus der Zwischendarstellung des Quellprogramms entgegengenommen und auf die Zielsprache abgebildet. Ein entscheidender Aspekt der Codeerzeugung ist die sinnvolle zuweisung von Registern für Variablen, falls es sich bei der Zielsprache um Maschinencode handelt.<sup>18</sup>

---

<sup>15</sup>Vgl. Ullman et al. 2008, S. 9ff.

<sup>16</sup>Vgl. Ullman et al. 2008, S. 11.

<sup>17</sup>Vgl. Ullman et al. 2008, S. 11f.

<sup>18</sup>Vgl. Ullman et al. 2008, S. 13.

## 2.4 Rekursiver Ansatz



# 3 Cross Plattform Frameworks

Für die Realisierung einer Source-to-Source Compiler gibt es zwei relevante Faktoren, die für die Realisierung ausschlaggebend sind. Zum einen die Programmiersprachen in denen 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 einführt, müssen die Programme vor und nach der Übersetzung gleichwertig sein. Dies implementiert, dass das Verhalten der übersetzten Anwendungen nach der Übersetzung identisch sein 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 compilation zu optimieren.

## 3.1 Frameworks

Xamarin ist 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 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.<sup>1</sup>

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 verwaltet 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,

---

<sup>1</sup>Vgl. Microsoft Corporation 2020a, Abgerufen am 30. Januar 2021.

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.<sup>2</sup>

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., jeweils eins für die Plattform (iOS/ Android), welche den plattformspezifischen Code und assets in Form von Icons und Metadaten beinhalten sowie ein zusätzliches für den Quelltext, der zwischen den Plattformen geteilt wird.<sup>3</sup> Im Gegensatz dazu gibt es bei Flutter nur ein Projekt, welches alle notwendigen Inhalte für iOS und Android beinhaltet. Für plattformspezifischen Code gibt es jeweils einen Ordner für iOS und Android.<sup>4</sup>

### 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 untergeordneten Ansichten in der Benutzeroberfläche verantwortlich sind. Außerdem arbeitet sie mit jeder untergeordneten Ansicht zusammen, um die endgültige Rendering-Größe zu bestimmen.<sup>5</sup>

## Pages

Pages (zu Deutsch: Ansichtseiten) sind visuelle Elemente einer Anwendung die den gesamten Bildschirm belegen und zu den Layout Views gehören. Xamarin Forms bietet

---

<sup>2</sup>Vgl. Google LLC 2020a, Abgerufen am 30. Januar 2021.

<sup>3</sup>Vgl. Petzold 2016, S. 25f.

<sup>4</sup>Vgl. Biessek 2019, S. 113.

<sup>5</sup>Vgl. Ritscher 2020, Abgerufen am 30. Januar 2021.

dafür verschiedene Alternativen an, die in 3.1 grafisch dargestellt sind.<sup>6</sup>

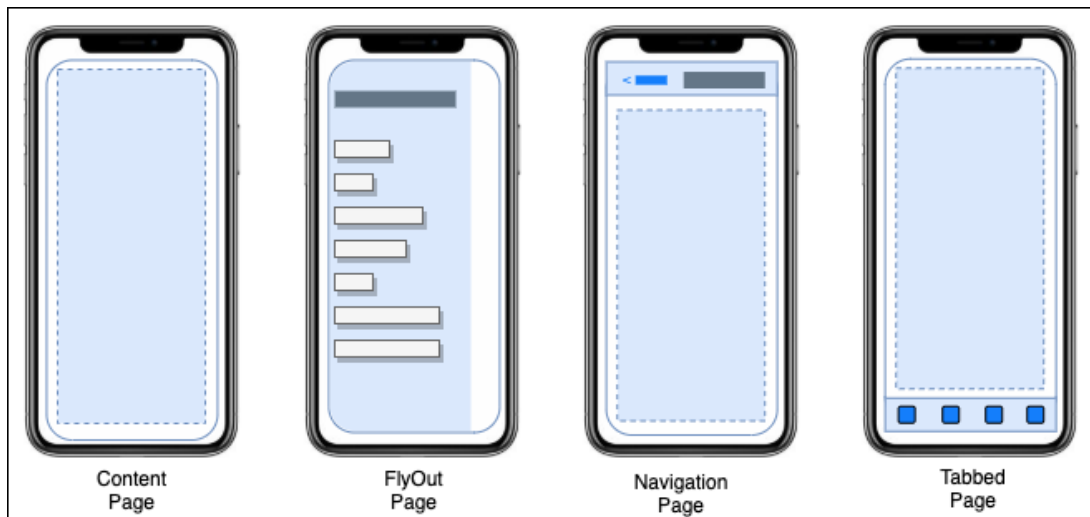


Abbildung 3.1: Xamarin.Forms Pages<sup>7</sup>

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 anzeigen haben die jeweiligen Seiten das folgende Navigationskonzept:<sup>8</sup>

- **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.

<sup>6</sup>Vgl. Microsoft Corporation 2016, Abgerufen am 30. Januar 2021.

<sup>7</sup>Abbildung in Anlehnung an Microsoft Corporation 2016, Abgerufen am 30. Januar 2021.

<sup>8</sup>Vgl. Microsoft Corporation 2016, Abgerufen am 30. Januar 2021.

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

Quelltext 3.1: Xamarin.Forms TabbedPage definition

Das Beispiel zeigt eine TabbedPage mit drei in diesem Falle leeren NavigationPages 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.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <FlyoutPage xmlns="http://xamarin.com/schemas/2014/forms"
3     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4     x:Class="MasterThesisSample.SampleFlyoutPage">
5     <FlyoutPage.Flyout>
6         <ContentPage>
7             <!-- FlyOut Menu -->
8         </ContentPage>
9     </FlyoutPage.Flyout>
10    <FlyoutPage.Detail>
11        <NavigationPage>
12            <!-- Place for the DetailPage -->
13        </NavigationPage>
14    </FlyoutPage.Detail>
15 </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.<sup>9</sup> Quelltext 3.3 zeigt die Definition einer MaterialDesign App in Flutter.

---

<sup>9</sup>Vgl. Google LLC 2020c, Abgerufen am 30. Januar 2021.

```
1 class MyApp extends StatelessWidget {  
2   // This widget is the root of your application.  
3   @override  
4   Widget build(BuildContext context) {  
5     return MaterialApp(  
6       title: 'Flutter Demo',  
7       theme: ThemeData(  
8         primarySwatch: Colors.blue,  
9       ),  
10      home: MyHomePage(title: 'Flutter Demo Home Page'),  
11    );  
12  }  
13 }
```

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, das 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.<sup>10</sup>

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.<sup>11</sup>

---

<sup>10</sup>Vgl. Google LLC 2020b, Abgerufen am 30. Januar 2021.

<sup>11</sup>Vgl. Google LLC 2020d, Abgerufen am 30. Januar 2021.

```
1 class TabBarDemo extends StatelessWidget {  
2   @override  
3   Widget build(BuildContext context) {  
4     return MaterialApp(  
5       home: DefaultTabController(  
6         length: 3,  
7         child: Scaffold(  
8           appBar: AppBar(  
9             bottom: TabBar(  
10              tabs: [  
11                Tab(icon: Icon(Icons.directions_car)),  
12                Tab(icon: Icon(Icons.directions_transit)),  
13                Tab(icon: Icon(Icons.directions_bike)),  
14              ],  
15            ),  
16            title: Text('Tabs Demo'),  
17          ),  
18          body: TabBarView(  
19            children: [  
20              Icon(Icons.directions_car),  
21              Icon(Icons.directions_transit),  
22              Icon(Icons.directions_bike),  
23            ],  
24          ),  
25        ),  
26      ),  
27    );  
28  }  
29 }
```

Quelltext 3.4: Flutter Tab Layout definition

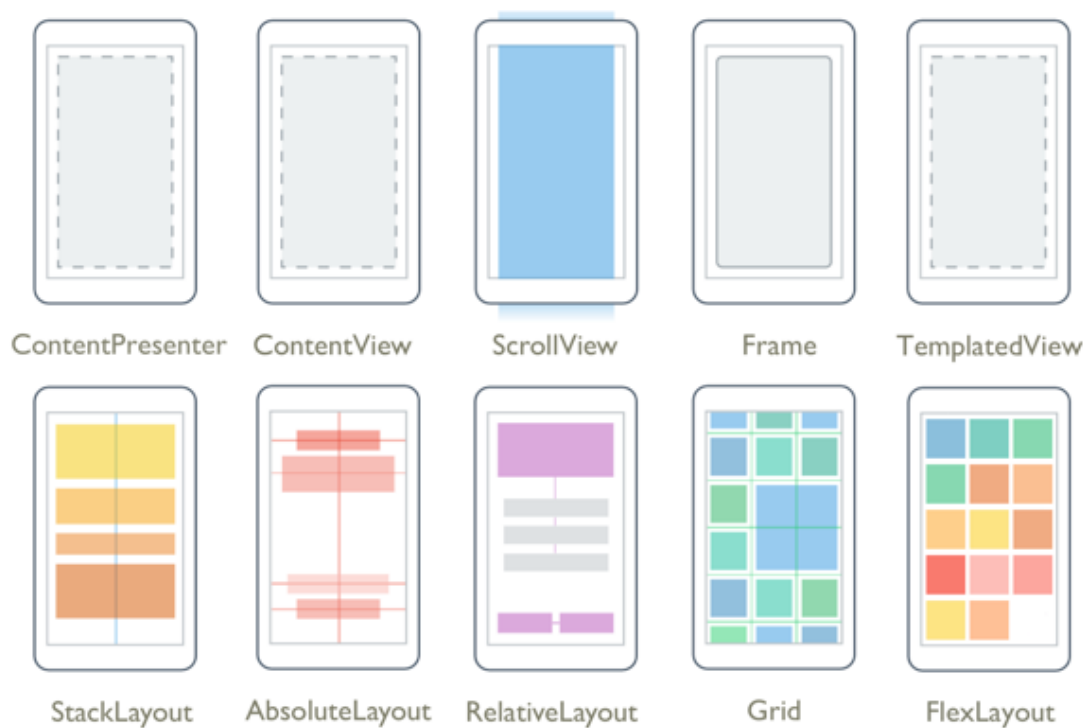
## Layouts

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 eine Vielzahl von Layouts an die in 3.2 grafisch dargestellt werden.

An dieser Stelle ist es nun wichtig zu analysieren, wie man jedes Xamarin.Forms Layout in Flutter nachbilden kann. Dafür wird folgend die Funktion der einzelnen Layouts

---

<sup>12</sup>Abbildung in Anlehnung an Microsoft Corporation 2018, Abgerufen am 30. Januar 2021.

Abbildung 3.2: Xamarin.Forms Layouts<sup>12</sup>

anhand der Xamarin.Forms Dokumentation<sup>13</sup> erklärt und ein Flutter Layout mit den gleichen Eigenschaften herausgestellt.

- **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. In Flutter kann auf das BoxDecoration widget zugegriffen werden es beschreibt, wie ein Kasten auf dem Bildschirm dargestellt werden soll, und kann als Rahmen dargestellt werden.
- **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. In Flutter ist die nächste Entsprechung das SingleChildScrollView-Widget.
- **StackLayout:** Positioniert untergeordnete Elemente in einem Stapel entweder horizontal oder vertikal, basierend auf der Eigenschaft Orientation. Flutter hat ein ähnliches Konzept, anstatt der Orientation werden Kinder durch die Widgets Row

<sup>13</sup>Vgl. Microsoft Corporation 2018, Abgerufen am 30. Januar 2021.

und Column vertical oder horizontal ausgerichtet.

- **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. Außerdem wird das Grid Control häufig verwendet um Steuerelemente zu stapeln. Das nächste Äquivalent zu einem Grid wäre ein GridView. Dieses ist viel leistungsfähiger als das, was Sie in Xamarin.Forms gewohnt sind. Ein GridView bietet einen automatischen Bildlauf, wenn der Inhalt den sichtbaren Bereich überschreitet. Für das Stapeln von Elementen, in Flutter lässt sich dies mit dem Stack-Widget erreichen.
- **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.

## 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 Programmierungsumgebungen 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
- **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 auszuwählen, 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

Das Äquivalent zu einer ListView in Flutter ist ... eine ListView!

In einer Xamarin.Forms ListView erstellen Sie eine ViewCell und möglicherweise einen DataTemplateSelector und übergeben diese an die ListView, die jede Zeile mit dem rendert, was Ihr DataTemplateSelector oder ViewCell zurückgibt. Allerdings müssen Sie oft darauf achten, dass Sie Cell Recycling einschalten, da es sonst zu Speicherproblemen und langsamen Scrollgeschwindigkeiten kommen kann. Aufgrund des unveränderlichen Widget-Musters von Flutter übergeben Sie eine Liste von Widgets an Ihre ListView, und Flutter kümmert sich darum, dass das Scrollen schnell und reibungslos funktioniert

In Xamarin.Forms hat die ListView eine ItemTapped-Methode, um herauszufinden, welches Element angeklickt wurde. Es gibt viele andere Techniken, die Sie vielleicht verwendet haben, wie z. B. die Überprüfung, wenn sich das SelectedItem- oder EventToCommand-Verhalten ändert. In Flutter verwenden Sie die Touch-Behandlung, die von den übergebenen Widgets bereitgestellt wird.

Wenn Sie in Xamarin.Forms die ItemsSource-Eigenschaft an eine ObservableCollection gebunden haben, würden Sie einfach die Liste in Ihrem ViewModel aktualisieren. Alternativ könnten Sie der ItemsSource-Eigenschaft auch eine neue Liste zuweisen. In

Flutter funktionieren die Dinge ein wenig anders. Wenn Sie die Liste der Widgets innerhalb einer `setState()`-Methode aktualisieren, würden Sie schnell sehen, dass sich Ihre Daten visuell nicht geändert haben. Das liegt daran, dass die Flutter-Rendering-Engine beim Aufruf von `setState()` den Widget-Baum betrachtet, um zu sehen, ob sich etwas geändert hat. Wenn sie zu Ihrer `ListView` kommt, führt sie eine `==` Prüfung durch und stellt fest, dass die beiden `ListView`s gleich sind. Es hat sich nichts geändert, also ist keine Aktualisierung erforderlich. Eine einfache Möglichkeit, Ihre `ListView` zu aktualisieren, besteht darin, eine neue Liste innerhalb von `setState()` zu erstellen und die Daten aus der alten Liste in die neue Liste zu kopieren. Dieser Ansatz ist zwar einfach, aber für große Datensätze nicht zu empfehlen. Die empfohlene, effiziente und effektive Art, eine Liste zu erstellen, verwendet einen `ListView.Builder`. Diese Methode ist großartig, wenn Sie eine dynamische Liste oder eine Liste mit sehr großen Datenmengen haben. Dies ist im Wesentlichen das Äquivalent von `RecyclerView` unter Android, das automatisch Listenelemente für Sie recycelt: Anstatt eine `ListView` zu erstellen, erstellen Sie einen `ListView.builder`, der zwei wichtige Parameter benötigt: die anfängliche Länge der Liste und eine `ItemBuilder`-Funktion. Die `ItemBuilder`-Funktion ähnelt der `getView`-Funktion in einem Android-Adapter; sie nimmt eine Position an und gibt die Zeile zurück, die an dieser Position gerendert werden soll. Schließlich, aber am wichtigsten, beachten Sie, dass die `onTap()`-Funktion die Liste nicht mehr neu erstellt, sondern ihr stattdessen etwas hinzufügt.

## Gesten

In `Xamarin.Forms` können Elemente ein Klick-Ereignis enthalten, an das Sie anhängen können. Viele Elemente enthalten auch einen Befehl, der an dieses Ereignis gebunden ist. Alternativ dazu würden Sie den `TapGestureRecognizer` verwenden. In Flutter gibt es zwei sehr ähnliche Möglichkeiten: Wenn das Widget die Ereigniserkennung unterstützt, übergeben Sie ihm eine Funktion und behandeln es in der Funktion. Zum Beispiel hat der `ElevatedButton` einen `onPressed`-Parameter. Alternativ wenn das Widget keine Ereigniserkennung unterstützt, packen Sie das Widget in einen `GestureDetector` und übergeben Sie eine Funktion an den Parameter `onTap`.

In `Xamarin.Forms` würden Sie dem `VisualElement` einen `GestureRecognizer` hinzufügen. Normalerweise wären Sie auf `TapGestureRecognizer`, `PinchGestureRecognizer` und `PanGestureRecognizer` beschränkt, es sei denn, Sie haben einen eigenen erstellt. Flutter kann mit Hilfe des `GestureDetectors` deutlich mehr Gesten behandeln als `Xamarin.Forms`.

## Animtion

In Xamarin.Forms erstellen Sie einfache Animationen mit ViewExtensions, die Methoden wie FadeTo und TranslateTo enthalten. Sie würden diese Methoden in einer Ansicht verwenden, um die gewünschten Animationen auszuführen.

In Flutter animieren Sie Widgets mithilfe der Animationsbibliothek, indem Sie Widgets in ein animiertes Widget einwickeln. Verwenden Sie einen AnimationController, der eine Animation<double> ist, die die Animation pausieren, suchen, stoppen und umkehren kann. Er benötigt einen Ticker, der signalisiert, wenn vsync passiert, und erzeugt eine lineare Interpolation zwischen 0 und 1 auf jedem Frame, während er läuft. Anschließend erstellen Sie eine oder mehrere Animationen und hängen sie an den Controller. Zum Beispiel könnten Sie CurvedAnimation verwenden, um eine Animation entlang einer interpolierten Kurve zu implementieren. In diesem Sinne ist der Controller die "MasterQuelle des Animationsverlaufs und die CurvedAnimation berechnet die Kurve, die die standardmäßige lineare Bewegung des Controllers ersetzt. Wie Widgets arbeiten auch Animationen in Flutter mit Komposition. Beim Aufbau des Widget-Baums weisen Sie die Animation einer animierten Eigenschaft eines Widgets zu, z. B. der Deckkraft einer FadeTransition, und sagen dem Controller, dass er die Animation starten soll.

### 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 Devi-

`ce.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. `url_launcher` 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 Netzwerkaufufe

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, Netzwerkaufufe 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:

```
1 import 'dart:convert';
2
3 import 'package:flutter/material.dart';
4 import 'package:http/http.dart' as http;
5 [...]
6 loadData() async {
7   String dataURL = "https://jsonplaceholder.typicode.com/posts";
8   http.Response response = await http.get(dataURL);
9   setState(() {
10     widgets = jsonDecode(response.body);
11   });
12 }
13 }
```

Quelltext 3.5: Flutter Network request

### 3.1.7 Lebenszyklus

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.

```
1 @override
2 Widget build(BuildContext context) {
3   return Scaffold(
4     appBar: AppBar(
5       title: Text("Sample App"),
6     ),
7     body: Center(
8       child: Text(
9         'This is a custom font text',
10        style: TextStyle(fontFamily: 'MyCustomFont'),
11      ),
12    ),
13  );
14 }
```

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](https://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 Kamera-rolle und die Gerätekamera direkt von Flutter aus zuzugreifen, ohne eine eigene Integration schreiben zu müssen. Plugins finden Sie auf [pub.dev](https://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](https://pub.dev) finden, das Ihren Anforderungen entspricht, können Sie Ihr eigenes schreiben und es auf [pub.dev](https://pub.dev) veröffentlichen.

### 3.1.12 Storage

Ein wesentlicher Bestandteil jeder mobilen Anwendung ist die Fähigkeit, Daten zu persistieren. Manchmal handelt es sich dabei um große Datenmengen, die eine Datenbank erfordern, oft sind es aber auch kleinere Daten wie Einstellungen und Präferenzen, die zwischen den Starts der Anwendung persistiert werden müssen. Xamarin.Essentials stellt Entwicklern plattformübergreifende APIs für ihre mobilen Anwendungen bereit, indem es die einzigartigen Betriebssystem- und Plattform-APIs nativ anspricht.<sup>14</sup> In diesem Paket ist das Settingsplugin, welches die Speicherung von Einstellungen in einem Schlüsselwertspeicher erlaubt.<sup>15</sup> In Flutter wird für die Speicherung von Schlüssel-Wertpaaren auf die gleichen plattformspezifischen APIs zugegriffen wie bei Xamarin Forms diese werden mithilfe eines Plugins angesprochen.<sup>16</sup>

Für die Speicherung in einer Datenbank können Xamarin.Forms Entwickler auf verschiedene Lösungen zurückgreifen zum einen SQLite die am häufigsten verwendete Datenbank-Engine der Welt<sup>17</sup>, oder Realm einer Datenbank optimiert für mobile End-

<sup>14</sup>Vgl. Microsoft Corporation 2020b, Abgerufen am 30. Januar 2021.

<sup>15</sup>Vgl. Microsoft Corporation 2019, Abgerufen am 30. Januar 2021.

<sup>16</sup>Vgl. Google LLC 2020b, Abgerufen am 30. Januar 2021.

<sup>17</sup>Vgl. SQLite Consortium 2020, Abgerufen am 30. Januar 2021.

geräte.<sup>18</sup> Beide Datenbanken stehen auch als Plugin für Flutter zur Verfügung, wobei SQLite ausgereift ist,<sup>19</sup> während Realm erst am 5 November 2020 support für Flutter angekündigt hat und noch nicht offiziell zur Verfügung steht.<sup>20</sup>

## 3.2 Programmiersprachen

Der Wechsel von C# zu Dart ist aufgrund des ähnlichen Stils und der Syntax einfach. Allerdings gibt es bei jeder Sprache Unterschiede, die manche Dinge einfacher und andere schwieriger machen. In diesem Abschnitt werden die Programmiersprachen miteinander verglichen.

### 3.2.1 In Dart ist alles Null

In Dart ist alles null, auch das, was in C# als Wertetyp bezeichnet wird. Dies führt dazu, dass bei der Arbeit mit Wertetypen wie Integer eine Null- Prüfung durchgeführt werden sollte.

```
1 // no need for ? sign to signal nullable.
2 int myVariable = null;
3
4 // Null checking
5 if (myVariable != null)
6
7 // These work the same way as C#.
8 myVariable = myVariable ?? 0; // Assigns 0 if null
9
10 // Short circuits also work.
11 myVariable?.toString();
```

Quelltext 3.7: Alles kann in Dart Null sein

### 3.2.2 Generics

Generics sind C# sehr ähnlich, mit der Ausnahme, dass Sie eine generische Klasse ohne die Type-Beschränkung übergeben können.

<sup>18</sup>Vgl. MongoDB Inc. 2020, Abgerufen am 30. Januar 2021.

<sup>19</sup>Vgl. Tekartik 2020, Abgerufen am 30. Januar 2021.

<sup>20</sup>Vgl. Ward 2020, Abgerufen am 30. Januar 2021.

```
1 class State<Type>
2 {
3   Type getValue() => null;
4 }
5 void processState(State state) {
6   dynamic value = state.getValue();
7 }
8 void processStateWithType(State<String> state) {
9   String value = state.getValue();
10 }
```

Quelltext 3.8: Generics in Dart

### 3.2.3 Delegates

Delegates sind im Wesentlichen Funktionszeiger. In Dart können Sie ein Typedef verwenden, um eine Methodensignatur zu definieren und eine Instanz davon in einer Variablen zu halten.

```
1 class State<Type>
2 {
3   Type getValue() => null;
4 }
5 void processState(State state) {
6   dynamic value = state.getValue();
7 }
8 void processStateWithType(State<String> state) {
9   String value = state.getValue();
10 }
```

Quelltext 3.9: Delegates in Dart

### 3.2.4 Das New Keyword

Dart kann herausfinden, wann Sie ein neues Element definieren, daher müssen Sie nicht new sagen, wenn Sie das nicht wollen.

```
1 class State<Type>
2 {
3     Type getValue() => null;
4 }
5 void processState(State state) {
6     dynamic value = state.getValue();
7 }
8 void processStateWithType(State<String> state) {
9     String value = state.getValue();
10 }
```

Quelltext 3.10: Optionales "New" Keyword in Dart

### 3.2.5 Listen und Dictionaries

Listen sind Listen, und Dictionaries sind Maps.

```
1 var list = new List();
2 var dictionary = new Map<String, String>();
```

Quelltext 3.11: Listen und Maps in Dart

### 3.2.6 Zugriffsmodifizierer

Alle Typen und Typmember verfügen in beiden Sprachen über eine Zugriffsebene. Diese Zugriffsebene steuert, ob sie von anderem Code in Ihrer Assembly oder anderen Assemblys verwendet werden können. Dabei wird in C# mit Mithilfe der Zugriffsmodifizieren (public, private, protected, internal, protected internal, private protected) der Zugriff auf einen Typ oder Member festlegen. In Dart gibt es die Schlüsselwörter public, protected und private nicht. Wenn ein Bezeichner mit einem Unterstrich (\_) beginnt, ist er für seine Bibliothek privat dies wird in 3.12 als Quelltext dargestellt.

```
1 class _PrivateClass {
2     String _privateField;
3 }
4
5 class PublicClass {
6     String publicField;
7 }
```

Quelltext 3.12: Private und Public Definitionen in Dart

Da ein Unterstrich ein valides Zeichen bei der Typ und Typmemberdefinition ist, ist daher bei der Übersetzung darauf zu achten, dass existierende Unterstriche entfernt werden müssen da dies ansonsten potentiell zu fehlerhaften Übersetzungen führen kann. Außerdem kann es durch die Verwendung von Zugriffsmodifizierern wie "protected internal" vorkommen, dass es keine entsprechende Dart Implementierung existiert. Folglich führt dies potentiell zu falschen Zugriffsbeschränkungen bei der Übersetzung von Bibliotheken.

### 3.2.7 Vererbung

Dart hat keine Schnittstellen, Sie haben abstrakte Klassen. Sie implementieren abstrakte Klassen. Wenn Sie erben wollen, erweitern Sie Klassen.

```
1 class BaseClass {  
2     void myFunction() {}  
3 }  
4  
5 class MyClass extends BaseClass {  
6     void myOtherFunction() {  
7         // do something  
8     }  
9  
10    // every function is overridable in Dart.  
11    @override  
12    void myFunction() {  
13        // do something  
14    }  
15 }
```

Quelltext 3.13: Vererbung in Dart

Sie können eine Klasse erweitern und mehrere Klassen implementieren. Dart unterstützt auch Mixin's. Ein Mixin ist wie das Anhängen einer Klasse und das Hinzufügen ihrer Funktionalität zu der Klasse, ohne tatsächlich von ihr zu erben. Dies ist auch ähnlich wie die Schnittstellenimplementierungen von C# 8.0.

```
1 class MyMixin {  
2   void sayHello() => print('Hello!');  
3 }  
4  
5 class MyClass with MyMixin  
6 {  
7 }
```

Quelltext 3.14: Mixin's in Dart

### 3.2.8 Namespaces

Dart hat keine Namespaces. Stattdessen importieren Sie Pakete oder Dateien als solche. Dadurch haben Sie Zugriff auf alle Klassen und Funktionen innerhalb der Datei. Aber wenn es einen Namenskonflikt gibt oder Sie die Dinge etwas lesbarer machen wollen, können Sie sie benennen.

```
1 // Import a core dart library  
2 import 'dart:async';  
3  
4 // Import a package  
5 import 'package:flutter/material.dart';  
6  
7 // Import another file in your application  
8 import 'myfilename.dart' as filename;
```

Quelltext 3.15: Importieren von Paketen in Dart

### 3.2.9 Bibliotheken

Dart verfügt über eine Vielzahl von Kernbibliotheken, die für viele alltägliche Programmieraufgaben wie das Arbeiten mit Objektsammlungen, das Durchführen von Berechnungen und das Kodieren/Dekodieren von Daten unerlässlich sind. Zusätzliche APIs sind in von der Community bereitgestellten Paketen verfügbar. Dieses Konzept ist dem aus .NET bekannten Bibliothek Konzept sehr ähnlich. Neben den Konzept sind auch die Inhalte in einigen Bibliotheken sehr ähnlich. So ähnelt die Bibliothek `dart:async` sehr dem .Net Namespace `System.Threading`. Außerdem ist `dart:Math` sehr ähnlich wie `System.Math` und `dart.io` zu `System.IO`. Darüber hinaus können Sie Funktionen direkt in Dateien haben, ohne eine Klasse oder einen Namespace. Das ist fantastisch, wenn Sie funktionaler programmieren wollen. z. B. können Sie dies in eine Datei ganz allein stellen, nichts anderes wird benötigt.



### 3.2.10 Async/Await

Die asynchrone Programmierung in Dart ist ebenfalls sehr ähnlich zu C#. Sie verwenden die Future-Klasse anstelle von Task. Die `async`- und `await`-Schlüsselwörter sind die gleichen, der einzige kleine Unterschied ist die Platzierung des `async`-Schlüsselworts, das nach dem Methodennamen steht, statt davor.

```
1 void main() async {  
2   var result = await myFunction();  
3 }  
4  
5 Future<String> myFunction() {  
6   // Similar to Task.FromResult  
7   return Future.value('Hello');  
8 }
```

Quelltext 3.16: Async und Await in Dart

### 3.2.11 Events

Anstelle eines Ereignisses in C# mit Delegationen, die dann alle aufgerufen werden, wenn ein Ereignis ausgelöst wird, arbeitet Dart in Streams. Ein Stream ist ähnlich wie ein Ereignis, aber Sie öffnen ihn, hören ihn an und schließen ihn.

Der Vorteil dieses Ansatzes ist, dass Sie viele Dinge tun können, wie z. B. Werte transformieren oder Ereignisse für eine bestimmte Zeitspanne anhalten und vieles mehr.

```
1 StreamController streamController = new StreamController.broadcast();  
2  
3 void main() {  
4   streamController.stream.listen((args) => {  
5     // do something  
6   });  
7  
8   streamController.add("hello");  
9  
10  streamController.close();  
11 }
```

Quelltext 3.17: Events in Dart

# Literaturverzeichnis

Biessek, Allesandro (2019). *Flutter for Beginners. An introductory guide to building cross-platform mobile application with Flutter and Dart 2*. englisch. Birmingham: Packt Publishing Ltd.

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

– (2020b). *Flutter for Xamarin.Forms developers*. Website. Online erhältlich unter [https://pub.dev/packages/shared\\_preferences](https://pub.dev/packages/shared_preferences); abgerufen am 28. Oktober 2020.

– (2020c). *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.

– (2020d). *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.

– (2018). *Xamarin.Forms Layouts*. Website. Online erhältlich unter <https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/layouts>; abgerufen am 28. Oktober 2020.

– (2019). *Xamarin.Essentials Einstellungen*. Website. Online erhältlich unter <https://docs.microsoft.com/de-de/xamarin/essentials/preferences>; abgerufen am 28. Oktober 2020.

- Microsoft Corporation (2020a). *What is Xamarin?* Website. Online erhältlich unter <https://docs.microsoft.com/de-de/xamarin/get-started/what-is-xamarin>; abgerufen am 28. Oktober 2020.
- (2020b). *Xamarin.Essentials*. Website. Online erhältlich unter <https://docs.microsoft.com/de-de/xamarin/essentials/>; abgerufen am 28. Oktober 2020.
- MongoDB Inc. (2020). *Realm Mobile Database*. Website. Online erhältlich unter <https://www.mongodb.com/realm/mobile/database>; abgerufen am 28. Oktober 2020.
- Petzold, Charles (2016). *Creating mobile Apps with Xamarin.Forms. Cross-platform C# programming for iOS, Android and Windows*. englisch. Washington: Microsoft Press.
- 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.
- SQLite Consortium (2020). *About SQLite*. Website. Online erhältlich unter <https://www.sqlite.org/about.html>; abgerufen am 28. Oktober 2020.
- Tekartik (2020). *sqflite*. Website. Online erhältlich unter <https://pub.dev/packages/sqflite>; abgerufen am 28. Oktober 2020.
- 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.
- Ward, Ian (2020). *Realm Mobile Database*. Website. Online erhältlich unter <https://github.com/realm/realm-object-server/issues/55>; abgerufen am 28. Oktober 2020.
- 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)

