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

25. Februar 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, tiefgreifende Modifizierungen 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 einer in einer problemori-

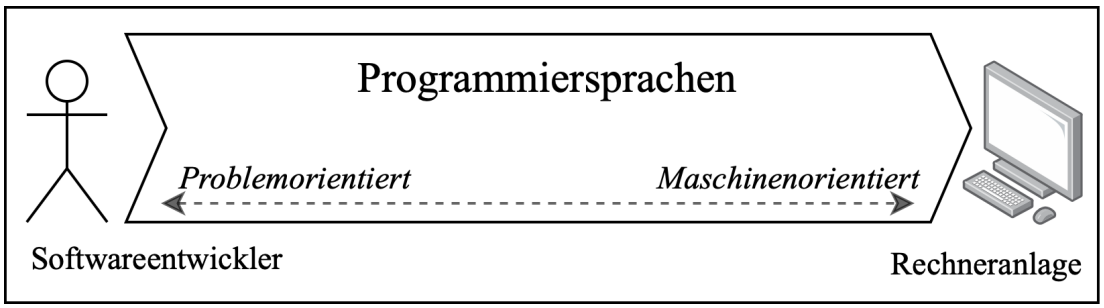

Abbildung 2.1: Programmiersprachen als Schnittstelle

entierten Programmiersprache geschriebenen Anwendung ist es notwendig, die Sprache in eine maschinenorientierte Form zu überführen.[2] Bereits im Jahre 1952 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 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] Vgl. Rutishauser 1952, S. 312.
[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. Auch wenn der Begriff Programm für jedermann geläufig ist, kann es dennoch passieren, das von verschiedenen Repräsentationen eines Programmes gesprochen wird. So können alle drei der folgenden Begriffe als Programm bezeichnet werden: Der Quelltext, dass ausführbare Programm kann er dennoch unterschiedliche Repräsentationen eines Programmes meinen. Für das weitere Verständnis dieser Arbeit ist, mit dem Begriff Programm die ausführbare Anwendung auf den Smartphones des Anwenders gemeint.

Neben der Übersetzung von problem zu maschinenorientierter Sprache gibt es ebenfalls Compiler, die andere Ziele verfolgen. Dazu gehört zum Beispiel die sogenannten Binärübersetzer, die den Binärcode eines Programmes für andere Rechner übersetzen, sodass er auf diesen ausgeführt werden kann.[6] 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.

Der Begriff Hochsprache ist dabei ein Synonym für die bereits eingeführten problemnahen Sprachen wie zum Beispiel C++, Java, C# oder Dart und damit für den Menschen in einer lesbaren und änderbaren Form geschrieben sind.[8]

2.2 Compiler Struktur

Compiler bearbeiten zwei Teilaufgaben für die Übersetzung von Programmen, die Analyse und die Synthese.[9]

[6] Vgl. Ullman et al. 2008, S. 27.
[7] Vgl. Rohit, Aditi und Hardikar 2015, S. 1629.
[8] Vgl. Eisenecker 2008, S. 9.
[9] Vgl. Ullman et al. 2008, S. 6.

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 fehlerfrei 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.[10]

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.[11]

2.3 Phasen

Der Vorgang des Kompilieren lässt sich basierend auf diesen zwei Teilaufgaben nach Ullman et al. in mehrere Phasen unterteilen die inAbbildung 2.2 grafisch dargestellt sind und in diesem Abschnitt detailliert beschrieben werden.[12]

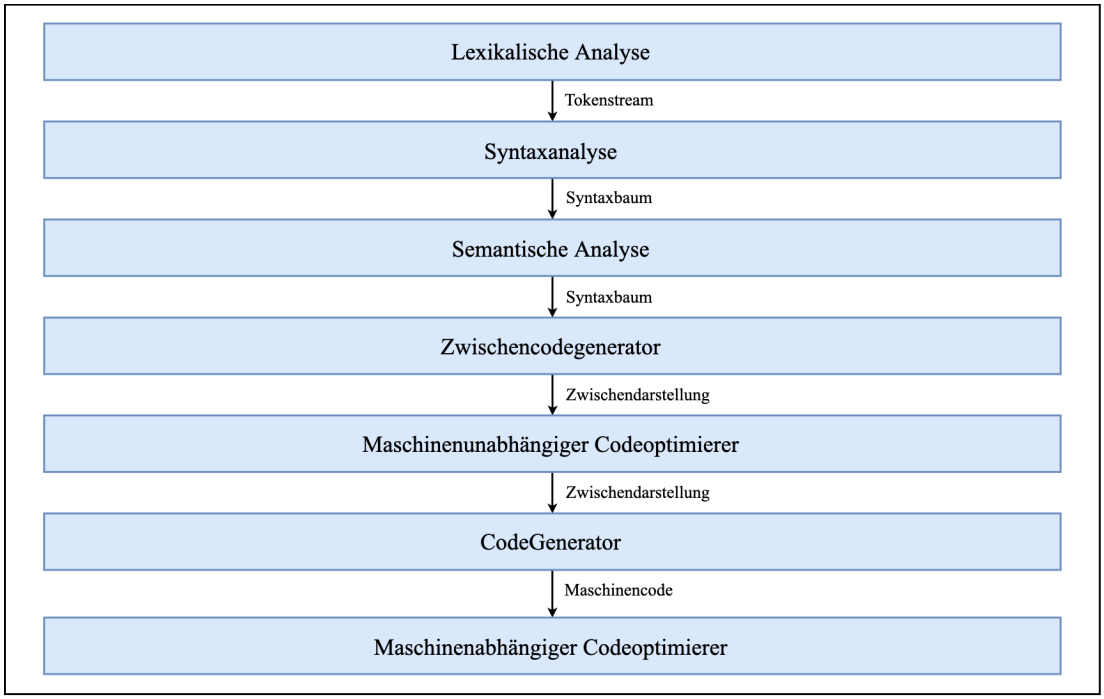

Abbildung 2.2: Phasen eines Compilers[13]

[10] Vgl. Ullman et al. 2008, S. 6f.
[11] Vgl. Ullman et al. 2008, S. 7.
[12] Vgl. Ullman et al. 2008, S. 6.
[13] Abbildung in Anlehnung an Ullman et al. 2008, S.6.

2.3.1 Lexikalische Analyse

Die erste Phase eines Compilers ist die lexikalische Analyse die sich mit der Untergliederung des Quelltextes in Lexeme beschäftigt. Ein Lexem ist die Folge von Zeichen im Quellprogramm, die als Instanz eines Tokens erkannt wurden. Dabei ist ein Token ein Paar aus Namen und einem optionalen Attributwert wobei der Name zum Beispiel ein bestimmtes Schlüsselwort oder eine Folge von Eingabezeichen seien kann und der Attributwert auf einen Eintrag in der Symboltabelle zeigt.[14] In Tabelle 2.1 werden einige Beispielhafte Tokens aufgeführt und aus welchen Lexemen diese extrahiert werden.

Token	**Beschreibung**	**Lexem**
if	Zeichen i,f	if
comparison	Vergleichsoperatoren	<=
id	Buchstaben	pi
number	Numerische Konstanten	3.14159

Tabelle 2.1: Token-Beispiele[15]

Basierend auf dieser Beschreibung ist in 2.3 ein Beispiel dargestellt, welches zeigt wie der Lexer aus einem Zeichenstring mehrere Token mit den optionalen Attributwerten generiert.

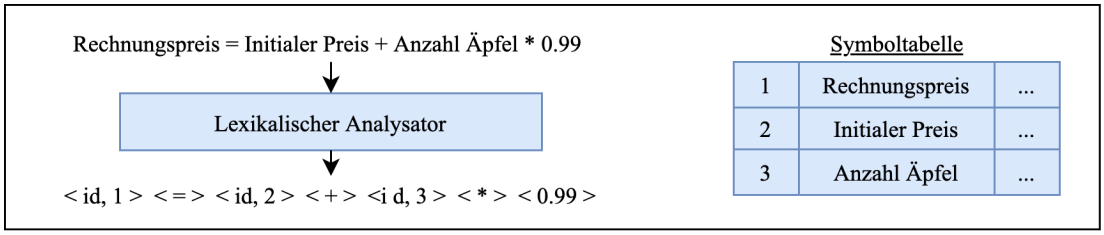

Abbildung 2.3: Lexer Beispiel[16]

Der Lexer interagiert mit den anderen Komponenten eines Compilers, diese Beziehungen werden in Abbildung 2.4 dargestellt. Dabei wird der Lexer häufig durch den Parser aufgerufen, dies wird in der Abbildung mit dem Aufruf "getNextToken"dargestellt.[17]

Da der Lexer derjenige Teil des Compilers ist, der den Quelltext liest, kann er neben der Identifikation von Lexemen auch weitere Aufgaben übernehmen. So eignet er sich Ideal zum Streichen von Kommentaren im Quelltext und zum entfernen von Leerstellen wie Leerzeichen und Tabulatoren. Eine weitere essentielle Aufgabe des Lexers ist es eine Zuordnung von Zeilennummern zu gefunden Fehlern herstellen zu können. Bei

[14] Vgl. Ullman et al. 2008, S. 135 f.
[15] Vgl. Ullman et al. 2008, S. 137.
[16] Abbildung in Anlehnung an Ullman et al. 2008, S.10.
[17] Vgl. Ullman et al. 2008, S. 135.

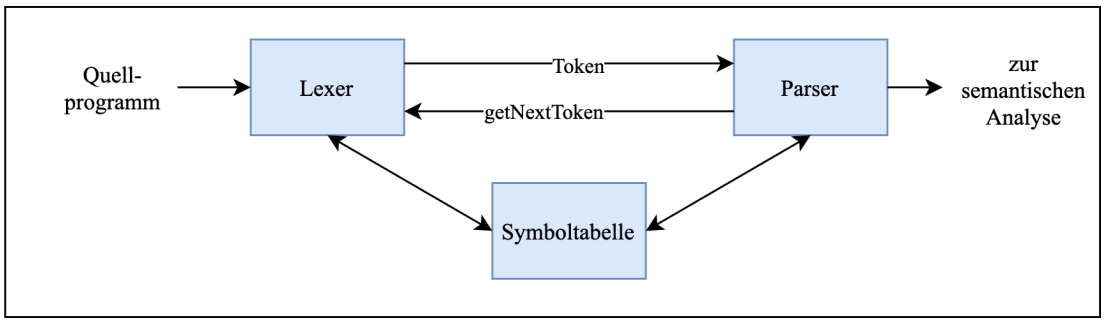

Abbildung 2.4: Interaktion zwischen Lexer und Parser[18]

einem Fehler wärend der Kompilierung kann somit ein Fehler an den Entwickler ausgegeben werden mit einem genauen Hinweis, wo der Fehler aufgetreten ist. [19] Häufig werden Lexer daher in zwei Kaskadierende Prozesse unterteilt, einem Löschen von Kommentaren und Zusammenfassen von Leerraumzeichen und einen für die eigentliche lexikalische Analyse.[20]

2.3.2 Syntaxanalyse

Die zweite Phase des Compilers ist die Syntaxanalye in welcher der sogenannte Parser oder auch syntaktischer Analysator genannt die vom lexikalischen Analysator ausgegebenen Token in eine baumartige Zwischendarstellung überführt, die die grammatische Struktur der Tokens zeigt. Diese Darstellung wird basierend auf ihrem Aussehen häufig als Syntaxbaum bezeichnet. Die Knoten im Syntaxbaum stehen für eine Operation und die 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.[21] Abbildung 2.5 zeigt die Erstellung eines Syntaxbaumes Anhand von einer Reihe von Token, dabei wurden die Tokens aus dem Beispiel der Abbildung 2.3 übernommen. Anhand des Knotens <id, 1> ist jederzeit über die Symboltabelle bekannt, dass das Ergebnis der Rechnung an den Speicherort des Bezeichners Rechnungspreis abgelegt werden muss.[22]

[18] Abbildung in Anlehnung an Ullman et al. 2008, S.135.
[19] Vgl. Ullman et al. 2008, S. 135.
[20] Vgl. Ullman et al. 2008, S. 136.
[21] Vgl. Ullman et al. 2008, S. 9.
[22] Vgl. Ullman et al. 2008, S. 9.

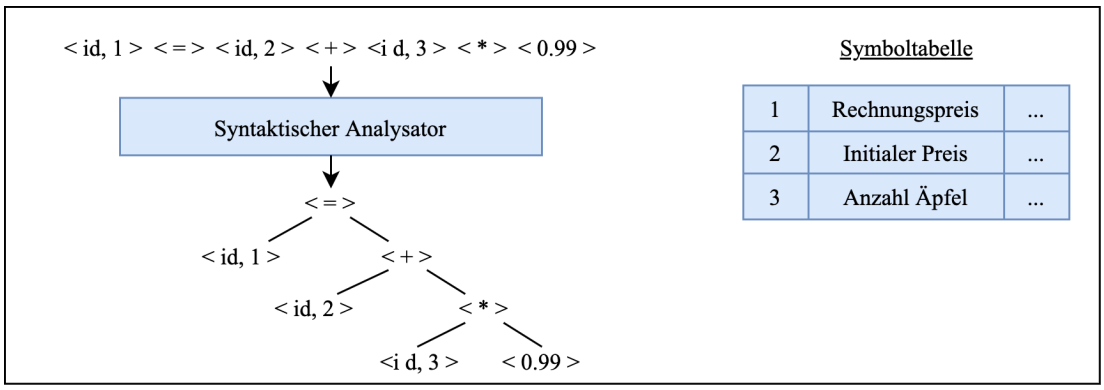

Abbildung 2.5: Syntaxbaum[23]

2.3.3 Semantische Analyse

Bei der semantischen Analyse wird der Syntaxbaum, als Aufgliederung der Programmkonstruktur, zusammen mit den Informationen aus der Symboltabelle verwendet um das Quellprogramm auf semantische Konsistenz mit der Sprachdefinition zu überprüfen.[24]. 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. So wird beispielsweise geprüft wird, ob ein Arrayindex ein Integer ist. Auch Typkonvertierungen können können zu dieser Zeit innerhalb des Baumss hinterlegt werden. So wurde in dem bisherigen Beispiel die Anzahl Äpfel als Ganzzahl behandelt und wird für die Berechnung des Preises in Abbildung 2.7 zu Float konvertiert.[25]

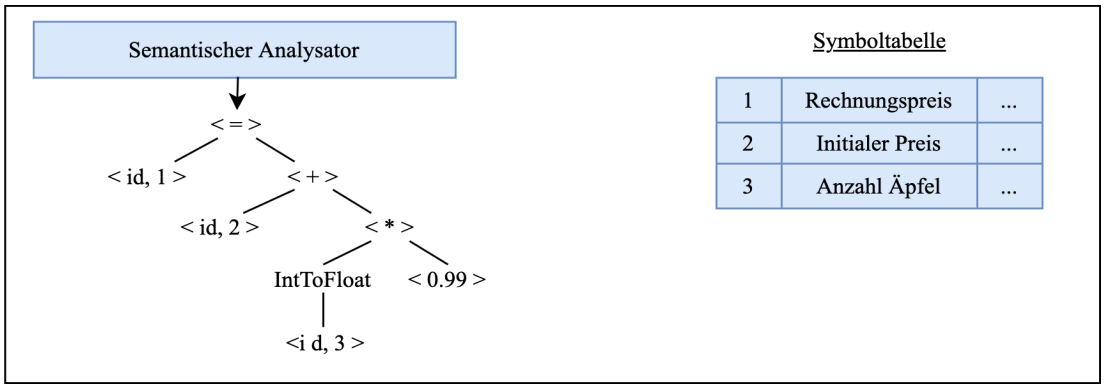

Abbildung 2.6: Typüberprüfung[26]

[23] Abbildung in Anlehnung an Ullman et al. 2008, S.10.
[24] Vgl. Wilhelm, Seidl und Hack 2012, S. 157.
[25] Vgl. Ullman et al. 2008, S. 9ff.
[26] Abbildung in Anlehnung an Ullman et al. 2008, S.10.

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. ç Darstellungen niedriger Ebene sind für maschinenabhängige Aufgaben, z. B. Registerabgabe und Befehlsauswahl, geeignet.

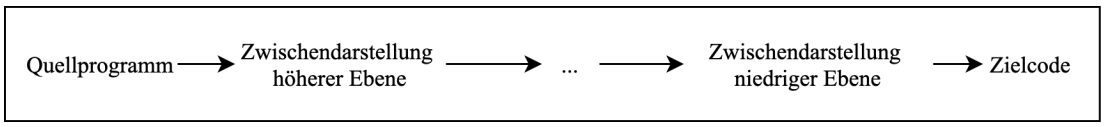

Abbildung 2.7: Zwischendarstellungen[27]

Die Auswahl oder der Entwurf einer Zwischendarstellung ist von Compiler zu Compiler unterschiedlich. Eine Zwischendarstellung kann entweder eine tatsächliche Sprache sein oder aus internen Datenstrukturen bestehen, die von den Phasen des Compilers gemeinsam verwendet werden. Auch wenn C eine Programmiersprache ist, wird sie häufig als eine Zwischenform verwendet, da sie flexibel ist, zu effizienten Maschinencode kompiliert werden kann und ihre Compiler weitgehend verfügbar sind.[28]

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.[29]

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.[30]

[27] Abbildung in Anlehnung an Ullman et al. 2008, S.433.
[28] Vgl. Ullman et al. 2008, S. 433.
[29] Vgl. Ullman et al. 2008, S. 11f.
[30] Vgl. Ullman et al. 2008, S. 13.

Wär

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.

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 Kompilierung 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 2020a, Abgerufen am 25. Februar 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.[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., 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.[3] 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.[4]

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.[5]

Pages

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

[2] Vgl. Google LLC 2020c, Abgerufen am 25. Februar 2021.
[3] **Petzold2016**.
[4] Vgl. Biessek 2019, S. 113.
[5] Vgl. Ritscher 2020, Abgerufen am 25. Februar 2021.

dafür verschiedene Alternativen an, die in 3.1 grafisch dargestellt sind.[6]

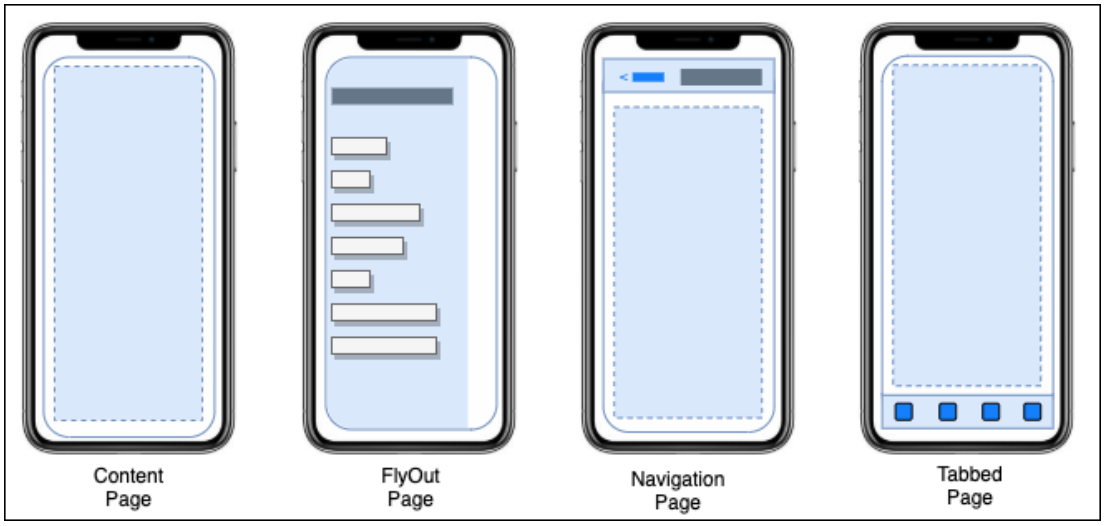

Abbildung 3.1: Xamarin.Forms Pages

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:[7]

- 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.

[6] Vgl. Microsoft Corporation 2016, Abgerufen am 25. Februar 2021.
[6] Abbildung in Anlehnung an Microsoft Corporation 2016, Abgerufen am 25. Februar 2021.
[7] Vgl. Microsoft Corporation 2016, Abgerufen am 25. Februar 2021.

```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.[8] Quelltext 3.3 zeigt die Definition einer MaterialDesign App in Flutter.

[8] Vgl. Google LLC 2020n, Abgerufen am 25. Februar 2021.

```
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.[9]

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.[10]

[9] Vgl. Google LLC 2020d, Abgerufen am 25. Februar 2021.
[10] Vgl. Google LLC 2020o, Abgerufen am 25. Februar 2021.

```
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 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

[10] Abbildung in Anlehnung an Microsoft Corporation 2018, Abgerufen am 25. Februar 2021.

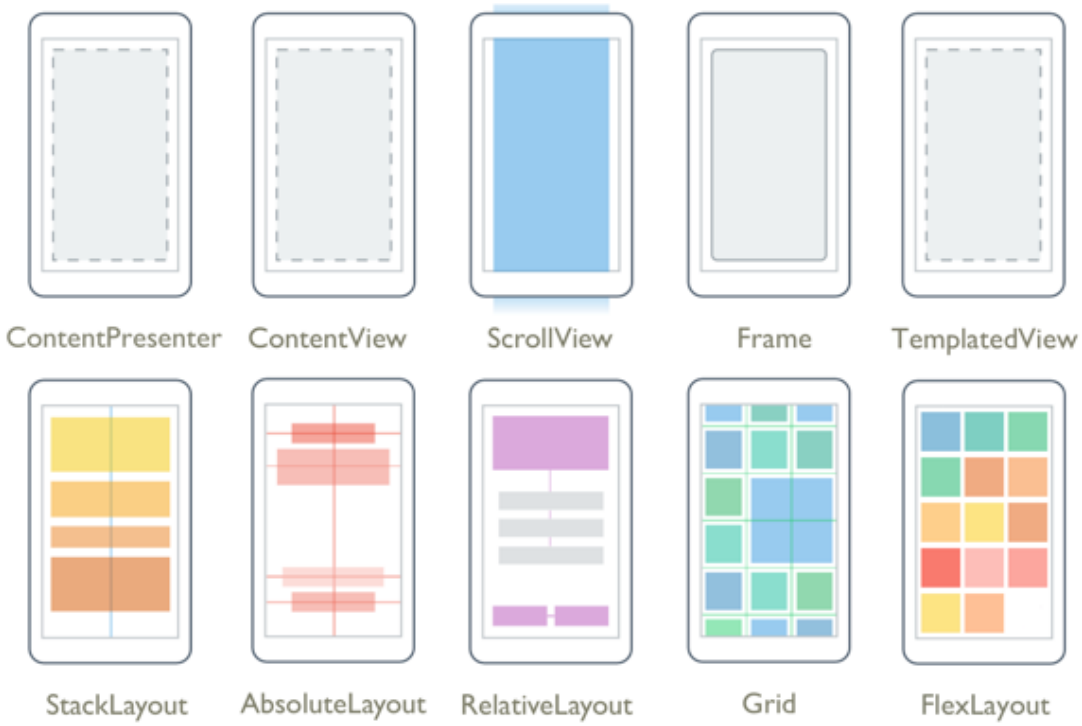

Abbildung 3.2: Xamarin.Forms Layouts

anhand der Xamarin.Forms Dokumentation[11] 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.[12]

- 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.[13]

- StackLayout: Positioniert untergeordnete Elemente in einem Stapel entweder ho-

[11] Vgl. Microsoft Corporation 2018, Abgerufen am 25. Februar 2021.
[12] Vgl. Google LLC 2020j, Abgerufen am 25. Februar 2021.
[13] Vgl. Google LLC 2020i, Abgerufen am 25. Februar 2021.

rizontal oder vertikal, basierend auf der Eigenschaft Orientation. Flutter hat ein ähnliches Konzept, anstatt der Orientation werden Kinder durch die Widgets Row 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.[14] Für das Stapeln von Elementen, in Flutter lässt sich dies mit dem Stack-Widget erreichen.[15]

- 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 Programmierumgebungen üblicherweise als Steuerelemente oder Widgets bezeichnet werden. Die von Xamarin.Forms unterstützten Ansichten leiten sich alle von der Klasse View ab.[16]

[14] Vgl. Google LLC 2020f, Abgerufen am 25. Februar 2021.
[15] Vgl. Google LLC 2020k, Abgerufen am 25. Februar 2021.
[16] Vgl. Microsoft Corporation 2020c, Abgerufen am 25. Februar 2021.

Steuerelemente für die Darstellung

Steuerelemente für die Darstellung sind Elemente die neben der Darstellung und der Benutzerfahrung keine weitere Funktionalität haben. Klassischerweise zeigen diese Informationen die für die Arbeit mit der Anwendung wichtig sind an. Die folgende Aufzählung zeigt die wichtigsten Steuerelemente aus Xamarin.Forms für die Darstellung,[17] wobei zu zeichnende Elemente wie die Ellipse, eine Linie, Path, Polygon, Polyline und Rectangle nicht besonders aufgeführt werden, da diese bei Flutter einfach auf das Canvas gezeichnet werden können.[18]

- BoxView zeigt in Xamarin.Forms ein einfarbiges Rechteck an. In der Praxis wird dieses Control auch häufig für die Darstellung von Seperatoren zwischen zwei Informationen verwendet. Dafür wird in Xamarin.Forms die Höhe oder Breite des Controls auf 1 Pixel limitiert. Für die Verwendung als seperator bietet Flutter das sogenannte Divider Widget an, welches überall verwendet werde kann um Content zu trennen. Mit dem Widget SizedBox bietet Flutter jedoch auch ein Widget für die Zeichnung von Boxen mit einer gewissen Größe an.[19]

- Label zeigt einzeilige Textstrings oder mehrzeilige Textblöcke an, entweder mit konstanter oder variabler Formatierung. Für Label gibt es bei Flutter mehrere alternativen. Das Text dem Xamarin.Forms Label am nahekommenste Widget ist "Text"was die Darstellung mit einem einzigen stil erlaubt.[20] Ebenso bietet Flutter jedoch auch die Möglichkeit RichText in einem entsprechenden Widget anzuzeigen.[21]

- Image zeigt ein Bitmap an, diese können über das Web heruntergeladen, als Ressourcen in über das gemeinsame Projekt oder in Plattformprojekte eingebettet werden. Flutter bietet ebenfalls ein Image Steuerelement an, welches die gleichen Quellen für Bilder unterstützt wie Xamarin.Forms.[22]

- Map zeigt eine Karte an. Für Flutter gibt es mehrere Anbietet für Karten z.B. Leamaps und Google Maps, welches direkt von den Flutter Entwickelern angbeoten wird.[23]

- WebView zeigt Web-Seiten oder HTML-Inhalte an. Das Widget "webview_flutter"bietet eine entsprechende Implementierung für Flutter an.[24]

[17] Vgl. Microsoft Corporation 2018, Abgerufen am 25. Februar 2021.
[18] Vgl. Google LLC 2020b, Abgerufen am 25. Februar 2021.
[19] Vgl. Google LLC 2020a, Abgerufen am 25. Februar 2021.
[20] Vgl. Google LLC 2020l, Abgerufen am 25. Februar 2021.
[21] Vgl. Google LLC 2020h, Abgerufen am 25. Februar 2021.
[22] Vgl. Google LLC 2020g, Abgerufen am 25. Februar 2021.
[23] Vgl. Google LLC 2020e, Abgerufen am 25. Februar 2021.
[24] Vgl. Google LLC 2020m, Abgerufen am 25. Februar 2021.

Steuerelemente die Aktionen auslösen

Die folgenden Steuerelemente sind Elemente, welche etwas in der mobilen Anwendung auslösen. Diese Elemente benötigen eine Interaktion über Gesten des Anwenders mit der Anwendung.

- Button ist ein rechteckiges Objekt, das Text anzeigt und ein Clicked-Ereignis auslöst, wenn es gedrückt wurde. Flutter bietet eine Vielzahl von Buttons an, wobei der "FlatButton"dem Xamarin.Forms button am nähesten kommt.
- ImageButton ist ein rechteckiges Objekt, das ein Bild anzeigt und ein Clicked-Ereignis auslöst, wenn es gedrückt wurde. In Flutter stehen dafür der IconButton zur Verfügung.
- RadioButton erlaubt die Auswahl einer Option aus einer Menge und feuert ein CheckedChanged-Ereignis, wenn die Auswahl erfolgt. Flutter bietet genau wie Xamarin.Forms einen RadioButton an, mit identischer Arbeitsweise
- RefreshView ist ein Container-Steuerelement, das eine PulltoRefresh-Funktionalität für scrollbare Inhalte bietet. Das Widget "pull_to_refresh"bietet ähnlich wie die Xamarin.Forms alternative an durch herunterziehen des Contents eine Aktion auszulösen.
- 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. Das Widget "flutter_search_bar"bietet die Möglichkeit den Inhalt zu Filtern. Jedoch kann es nicht wie bei Xamarin.Forms frei platziert werden, sondern wird im Rahmen der Navigationsleiste abgebildet.
- SwipeView ist ein Container-Steuerelement, das sich um ein Inhaltselement legt und Kontextmenüelemente bereitstellt, die durch eine Wischgeste angezeigt werden. Flutter selber bietet kein Widget mit dieser Funktionalität an. Für Listen steht jedoch mit "flutter_slidableëin OpenSource Widget zur Verfügung.

Steuerelemente um Werte zu setzen

Steuerelemente die Werte setzen sind User Interface Elemente die von einem Anwender durch Interaktion mit Werten versehen werden können um diese in der Anwendung zu verarbeiten. In diesem Unterabschnitt wird nicht auf Elemente eingegangen, die zur Eingabe und Manipulation von Texten zur Verfügung stehen, da diesen Steuerelementen ein eigener Unterabschnitt gewährt wird. Xamarin.Forms bietet eine Vielzahl von

Steuerelementen, die an dieser Stelle aufgeführt werden. Zusätzlich wird wie in den vorherigen Abschnitten erläutert, wie dieses Verhalten bei Flutter generiert werden kann.

- CheckBox ermöglicht dem Benutzer die Auswahl eines booleschen Wertes mit Hilfe einer Art Schaltfläche, die entweder markiert oder leer sein kann. Flutter bietet ebenfalls ein Checkbox Control an, das Material Design Checkbox.
- Switch hat die Form eines Ein/Aus-Schalters, damit der Benutzer einen booleschen Wert auswählen kann, für das es ebenfalls eine entsprechendes Flutter Widget gibt.
- Slider bieten Benutzern die Option einen Wert aus einem kontinuierlichen Bereich auswählen, der mit den Eigenschaften Minimum und Maximum festgelegt wurde. Flutter bietet ebenfalls ein Widget für Slider an
- 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. Von Haus aus gibt es bei Flutter kein Steuerelement, welches ähnliche Eigenschaften anbietet. Die Open Source Community hat mit dem number_inc_dec Plugin aber eine Widget entwickelt welches wie die Xamarin.Forms alternative arbeitet.
- DatePicker ermöglicht es dem Benutzer, ein Datum mit der Datumsauswahl der Plattform auszuwählen. In Flutter stehen dafür kein Widget sondern eine Funktion zur Verfügung, die klassischerweise, wie in Quelltext , in einem Gesture Detector auf einem anderen Widget ausgeführt wird.
- TimePicker ermöglicht dem Benutzer die Auswahl einer Zeit mit dem spezifischen TimePicker der Plattform. Wird ähnlich wie der Datepicker nicht als Widget sonders als Funktionalität ausgeliefert, ein mögliche Definition ist in Quelltext 3.5 dargestellt.

```
GestureDetector(
    onTap: () async {
        TimeOfDay picked = await showTimePicker(
        context: context,
        initialTime: TimeOfDay.now(),
        builder: (BuildContext context, Widget child) {
           return MediaQuery(
           data: MediaQuery.of(context)
           .copyWith(alwaysUse24HourFormat: true),
           child: child,
           );
        },);
     },
     child: Text("SetTime",textAlign: TextAlign.center,)
);
```

Quelltext 3.5: Verwendung von Timepickern in Flutter

Steuerelemente um Text zu manipulieren

Wie bereits im vorherigen Abschnitt beschrieben sind Steuerelemente zur Eingabe und manipulation von Texten Strang genommen ebenfalls Steuerelemente zum setzen von Werten. In Xamarin Forms stehen die folgenden beiden Steuerelemente für die Arbeit mit Texten zur Verfügung:

- Entry ermöglicht dem Benutzer die Eingabe und Bearbeitung einer einzelnen Textzeile., optional kann dieses als Passworteingabe mit Sternen maskiert werden.
- Editor ermöglicht dem Benutzer die Eingabe und Bearbeitung mehrerer Textzeilen

In flutter gibt es ausschließlich das Textfield Widget, welches die Funktionalitäten von beiden Xamarin.Forms controlls bündelt. Standardmäßig bietet das TextField Widget die Eingabemöglichkeit für eine Zeile ähnlich wie das Entry Steuerelement kann aber durch die Eigenschaft "keyboardType: TextInputType.multiline"wie ein entry fungieren wobei optional auch eine Anzahl von maximalen Zeilen übergeben werden kann. Dies wird in Quelltext dargestellt.

```
TextField(
  keyboardType: TextInputType.multiline,
  maxLines: null,
)
```

Quelltext 3.6: TextField mit mehreren Zeilen in Flutter

Steuerelemente um eine Aktivität anzudeuten

In mobilen Anwendungen kann es aufgrund der Limitierten Hardware Ressourcen und der begrenzten Netzwerkanbindung dazu kommen, dass Aktionen etwas länger dauern, als es der Anwender erwarten würden. Dafür stehen in Xamarin.Forms Steuerelemente zur Verfügung, die dem Anwender das Andauern einer Aktivität andeuten. Die folgenden Elemente stehen Xamarin.Forms Entwicklern zur Verfügung:

- ActivityIndicator verwendet eine Animation, um zu zeigen, dass die Anwendung eine langwierige Aktivität ausführt, ohne einen Hinweis auf den Fortschritt zu geben. Für Flutter Anwendungen steht ein ähnliches Widget zur Verfügung das als "CircularProgressIndicator"bezeichnet wird.
- ProgressBar verwendet eine Animation, um zu zeigen, dass die Anwendung durch eine langwierige Aktivität fortschreitet. In Flutter steht mit dem "Linear-ProgressIndicator"Widget ebenfalls eine Progressbar zur Verfügung.

Steuerelemente um Sammlungen anzuzeigen

Bei diesen Steuerelementen handelt es sich um Elemente, die zur Anzeige von Daten-Sammlungen spezialisiert sind. Xamarin.Forms stellt die folgenden Steuelemente zur Verfügung.

- CarouselView zeigt eine blätterbare Liste von Datenelementen an. Für Flutter steht ein ähnliches Widget mit dem "carousel_sliderßur erfügung
- IndicatorView zeigt Indikatoren an, die die Anzahl der Elemente in einer CarouselView darstellen
- Picker zeigt ein ausgewähltes Element aus einer Liste von Textzeichenfolgen an und ermöglicht die Auswahl dieses Elements, wenn die Ansicht angetippt wird. Das Picker Widget ist bereits wie im Date,- und Time-Picker beschrieben

kein Controll sondern eine Funktion, welches durch Gesten auf andere Widges ausgeführt werden kann.

- TableView zeigt eine Liste von Zeilen mit optionalen Überschriften und Unterüberschriften an. Das äquivalente flutter- Widget ist als Table Verfügbar.

Listen

Listen sind ebenfalls Steuerelemente für die Anzeige und Interaktion mit Sammlungen, bekommen jedoch wegen ihrer häufigen Verwendung einen eigenen Abschnitt. In Xamarin.Form existieren ein Listview und ein CollectionView Control, wobei zweiteres eine optimierung der Listview ist. 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 ItemSource-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 ListViews 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 "ListViewßu 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, verpacken 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.

Animation

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 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 3.7 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 3.7: 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 3.8 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 3.8: 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

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.[25] In diesem Paket ist das Settingsplugin, welches die Speicherung von Einstellungen in einem Schlüsselwertspeicher erlaubt.[26] 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.[27]

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[28], oder Realm einer Datenbank optimiert für mobile End-

[25] Vgl. Microsoft Corporation 2020b, Abgerufen am 25. Februar 2021.
[26] Vgl. Microsoft Corporation 2019, Abgerufen am 25. Februar 2021.
[27] Vgl. Google LLC 2020d, Abgerufen am 25. Februar 2021.
[28] Vgl. SQLite Consortium 2020, Abgerufen am 25. Februar 2021.

geräte.[29] Beide Datenbanken stehen auch als Plugin für Flutter zur Verfügung, wobei SQLite ausgereift ist,[30] während Realm erst am 5 November 2020 support für Flutter angekündigt hat und noch nicht offiziell zur Verfügung steht.[31]

3.2 Programmiersprachen

Die Programmiersprachen C# zu Dart haben einen ähnlichen Stils, Syntax und haben sogar gemeinsame Bibliotheksnamen. Pedley, der ursprüngliche Auto des "Flutter für Xamarin.Forms Entwicklers"Dokumentation beschrieb in seinem Blog die Unterschiede zwischen den beiden Programmiersprachen.[32] Anhand dieses Beitrages solllen in diesem Abschnitt die Unterschiede zwischen den Programmiersprachen aufgedeckt und Anhand von Dart- Quelltext Beispielen veranschaulicht werden.

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 3.9: Alles kann ich Dart Null sein

[29] Vgl. MongoDB Inc. 2020, Abgerufen am 25. Februar 2021.
[30] Vgl. Tekartik 2020, Abgerufen am 25. Februar 2021.
[31] Vgl. Ward 2020, Abgerufen am 25. Februar 2021.
[32] Vgl. Pedley 2019, Abgerufen am 25. Februar 2021.
[32] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.

3.2.2 Generics

Generics wurden in .NET als Typparameter eingeführt, wodurch Klassen und Methoden entworfen werden können, bei denen ein Typ erst verzögert werden kann, bis die Klasse oder Methode vom Clientcode deklariert und instanziiert wird. So kann indem z. B. ein generischen Typparameter „T“ verwendet wird, eine einzelne Klasse geschrieben werden, die von unterschiedlichen Methoden verwendet wird, ohne dass Kosten und Risiken durch die Umwandlungen zur Laufzeit anfallen.[33]

Generics werden in Dart sehr ähnlich behandelt wie in C# , mit der Ausnahme, dass Sie eine generische Klasse ohne die Type-Beschränkung übergeben können.[34] Quelltext 3.10 zeigt die Implementation einer generischen State-Klasse in Dart.

```
class State<Type>
{
  Type getValue() => null;
}
void processState(State state) {
  dynamic value = state.getValue();
}
void processStateWithType(State<String> state) {
  String value = state.getValue();
}
```

Quelltext 3.10: Generics in Dart

3.2.3 Delegates

In .Net ist Ein Delegate ist ein Typ, der Verweise auf Methoden mit einer bestimmten Parameterliste und dem Rückgabetyp darstellt. Nach der Instanziierung eines Delegaten können die Instanz mit einer beliebigen Methode verknüpft werden, die eine kompatible Signatur und einen kompatiblen Rückgabetyp aufweisen. Diese können die Methode über die Delegatinstanz aufrufen. Delegates werden dazu verwendet, um Methoden als Argumente an anderen Methoden zu übergeben. Da Ereignishandler ebenfalls Methoden sind können diese durch Delegates aufgerufen werden. Benutzerdefinierte Methoden können also durch Steuerelemente diese Methode aufrufen, wenn ein bestimmes Ereignis wie ein klick auf einen Button eintritt.[35]

[33] Vgl. Microsoft Corporation 2015b, Abgerufen am 25. Februar 2021.
[34] Vgl. Cheng 2019, S. 98.
[34] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.
[35] Vgl. Microsoft Corporation 2015a, Abgerufen am 25. Februar 2021.

In Dart kann der Typ Typedef verwendet werden, um eine Methodensignatur zu definieren und eine Instanz davon in einer Variablen zu halten.[36] Dies wird in Quelltext 3.11 dargestellt.

```
class State<Type>
{
  Type getValue() => null;
}
void processState(State state) {
  dynamic value = state.getValue();
}
void processStateWithType(State<String> state) {
  String value = state.getValue();
}
```

Quelltext 3.11: 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.

```
class State<Type>
{
  Type getValue() => null;
}
void processState(State state) {
  dynamic value = state.getValue();
}
void processStateWithType(State<String> state) {
  String value = state.getValue();
}
```

Quelltext 3.12: Optinales "New"Keyword in Dart

3.2.5 Listen und Dictionaries

C# und .Net haben Sammlungen (Collection) Klassen die alle Auftretenden Probleme des Hinzufügen und entfernen von Elementen aus Arrays behandeln. Diese werden

[36] Vgl. Pedley 2019, Abgerufen am 25. Februar 2021.
[36] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.
[36] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.

als Listen (Eng. List) bezeichnet.[37] Dart hat ebenfalls Listen die unter dem gleichen Namen zur Verfügung stehen und die gleichen Funktionalitäten abbilden.[38] Listen sind Listen, und Dictionaries sind Maps.

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

Quelltext 3.13: 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 3.14 als Quelltext dargestellt.

```
class _PrivateClass {
  String _privateField;
}

class PublicClass {
  String publicField;
}
```

Quelltext 3.14: 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 Zugriffsmodifiziern wie "protected internal"vorkommen, dass es keine entsprechende Dart Implementierung existiert. Folglich führt dies potentiell zu falschen Zugriffsbeschränkungen bei der Übersetzung von Bibliotheken.

[37] Vgl. Stellman und Greene 2021, S. 413.
[38] Vgl. Meiller 2020, S. 12f.
[38] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.
[38] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.

3.2.7 Vererbung

Die Vererbung ist, zusammen mit der Kapselung und der Polymorphie, eines der drei primären Charakteristika des objektorientierten Programmierens. Die Vererbung ermöglicht die Erstellung neuer Klassen, die in anderen Klassen definiertes Verhalten wieder verwenden, erweitern und ändern. Die Klasse, deren Member vererbt werden, wird Basisklasse genannt, und die Klasse, die diese Member erbt, wird abgeleitete Klasse genannt. Eine abgeleitete Klasse kann nur eine direkte Basisklasse haben.[39]

Eine Schnittstelle definiert einen Vertrag. Jede class oder struct, die diesen Vertrag implementiert, muss eine Implementierung der in der Schnittstelle definierten Member bereitstellen. Ab C# 8.0 kann eine Schnittstelle eine Standardimplementierung für Member definieren.[40]

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

```
class BaseClass {
  void myFunction() {}
}

class MyClass extends BaseClass {
  void myOtherFunction() {
    // do something
  }

  // every function is overridable in Dart.
  @override
  void myFunction() {
    // do something
  }
}
```

Quelltext 3.15: 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.

[39] Vgl. veerbung Google LLC 2020d, Abgerufen am 25. Februar 2021.
[40] Vgl. interface (C#-Referenz) Google LLC 2020d, Abgerufen am 25. Februar 2021.
[40] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.

```dart
class MyMixin {
  void sayHello() => print('Hello!');
}

class MyClass with MyMixin
{
}
```

Quelltext 3.16: Mixin's in Dart

3.2.8 Namespaces

Namespaces werden häufig in C# -Programmen auf zwei verschiedene Arten verwendet. Erstens: Die .NET-Klassen verwenden Namespaces, um ihre zahlreichen Klassen zu organisieren. Zweitens: Eigene Namespaces zu deklarieren kann Ihnen dabei helfen, den Umfang der Klassen- und Methodennamen in größeren Programmierprojekten zu steuern. Die meisten C#-Anwendungen beginnen mit einem Abschnitt von using-Anweisungen. Dieser Abschnitt enthält die von der Anwendung häufig verwendeten Namespaces und erspart dem Programmierer die Angabe eines vollqualifizierten Namens bei jedem Verwenden einer enthaltenen Methode.[41]

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.

```dart
// Import a core dart library
import 'dart:async';

// Import a package
import 'package:flutter/material.dart';

// Import another file in your application
import 'myfilename.dart' as filename;
```

Quelltext 3.17: Importieren von Paketen in Dart

[40] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.
[41] Vgl. Verwenden von Namespaces (C#-Programmierhandbuch) Google LLC 2020d, Abgerufen am 25. Februar 2021.
[41] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.

3.2.9 Bibliotheken

Klassenbibliotheken sind das Konzept der freigegebenen Bibliothek für .NET. Sie können damit nützliche Funktionalität auf Module verteilen, die von mehreren Anwendungen verwendet werden können. Sie können auch verwendet werden, um Funktionalität zu laden, die beim Start der Anwendung nicht benötigt wird bzw. nicht bekannt ist. Klassenbibliotheken werden mithilfe des .NET Assembly-Dateiformats beschriebenen.[42]

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 Bilbiothek 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

Das aufgabenbasierte asynchrone Programmiermodell stellt eine Abstraktion über asynchronen Code bereit. Der Quelltext kann dabei in gewohnter Weise als eine Folge von Anweisungen geschrieben werden und auch so gelesen werden, als ob jede Anweisung abgeschlossen wäre, bevor die nächste Anweisung beginnt. Der Compiler führt eine Reihe von Transformationen durch, da möglicherweise einige dieser Anweisungen gestartet werden und eine Task zurückgeben, die die derzeit ausgeführte Arbeit darstellt. Ziel dieser Syntax ist es: Code zu aktivieren, der sich wie eine Folge von Anweisungen liest, aber in einer deutlich komplizierteren Reihenfolge ausgeführt wird, die auf einer externen Ressourcenzuordnung und dem Abschluss von Aufgaben basiert. Vergleichbar ist dies mit der Art und Weise, wie Menschen Anweisungen für Prozesse erteilen, die asynchrone Aufgaben enthalten.[43]

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

[42] Vgl. .NET-Klassenbibliotheken Google LLC 2020d, Abgerufen am 25. Februar 2021.
[43] Vgl. MICROSFT Async await Google LLC 2020d, Abgerufen am 25. Februar 2021.

gleichen, der einzige kleine Unterschied ist die Platzierung des async-Schlüsselworts, das nach dem Methodennamen steht, statt davor.

```dart
void main() async {
  var result = await myFunction();
}

Future<String> myFunction() {
  // Similar to Task.FromResult
  return Future.value('Hello');
}
```

Quelltext 3.18: Async und Await in Dart

3.2.11 Events

Ein Ereignis ist eine Meldung, die von einem Objekt gesendet wird, um das Auftreten einer Aktion zu signalisieren. Die Aktion kann durch Benutzerinteraktionen wie das Klicken auf eine Schaltfläche verursacht werden, oder sie kann durch eine andere Programmlogik, z. B. das Ändern eines Eigenschaftswerts, ausgelöst werden. Das Objekt, von dem das Ereignis ausgelöst wird, wird als Ereignissender bezeichnet. Dem Ereignissender ist nicht bekannt, welches Objekt oder welche Methode die ausgelösten Ereignisse empfangen (behandeln) wird. Das Ereignis ist in der Regel ein Member des Ereignissenders. Beispielsweise ist das Click-Ereignis ein Member der Klasse Button, und das PropertyChanged-Ereignis ist ein Member der Klasse, von der die INotifyPropertyChanged-Schnittstelle implementiert wird.[44]

Anstelle eines Ereignisses in C# mit Delegaten, 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.

[43] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.
[44] Vgl. MICROSFT Events Google LLC 2020d, Abgerufen am 25. Februar 2021.

```dart
StreamController streamController = new StreamController.broadcast();

void main() {
  streamController.stream.listen((args) => {
        // do something
      });

  streamController.add("hello");

  streamController.close();
```

Quelltext 3.19: Events in Dart

[44] In Anlehnung an Pedley 2019, Abgerufen am 25. Februar 2021.

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.

Cheng, Fu (2019). *Flutter Recipes. Mobile Development Solutions for iOS and Android*. Sandringham: Apress.

Eisenecker, Ulrich (2008). *C++: der Einstieg in die Programmierung. Strukturiert und prozedural programmieren*. Witten: W3L.

Google LLC (2020a). *BoxDecoration class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/widgets/SizedBox-class.html`; abgerufen am 25. Februar 2021.

– (2020b). *Canvas class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/dart-ui/Canvas-class.html`; abgerufen am 25. Februar 2021.

– (2020c). *Flutter FAQ*. Website. Online erhältlich unter `https://flutter.dev/docs/resources/faq`; abgerufen am 25. Februar 2021.

– (2020d). *Flutter for Xamarin.Forms developers*. Website. Online erhältlich unter `https://pub.dev/packages/shared_preferences`; abgerufen am 25. Februar 2021.

– (2020e). *flutter_map*. Website. Online erhältlich unter `https://pub.dev/packages/flutter_map`; abgerufen am 25. Februar 2021.

– (2020f). *GridView class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/widgets/GridView-class.html`; abgerufen am 25. Februar 2021.

– (2020g). *Image class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/widgets/Image-class.html`; abgerufen am 25. Februar 2021.

– (2020h). *RichText class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/widgets/RichText-class.html`; abgerufen am 25. Februar 2021.

– (2020i). *SingleChildScrollView class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/widgets/SingleChildScrollView-class.html`; abgerufen am 25. Februar 2021.

– (2020j). *SizedBox class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/painting/BoxDecoration-class.html`; abgerufen am 25. Februar 2021.

Google LLC (2020k). *Stack class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/widgets/Stack-class.html`; abgerufen am 25. Februar 2021.

– (2020l). *Text class*. Website. Online erhältlich unter `https://api.flutter.dev/flutter/widgets/Text-class.html`; abgerufen am 25. Februar 2021.

– (2020m). *webview_flutter*. Website. Online erhältlich unter `https://pub.dev/packages/webview_flutter`; abgerufen am 25. Februar 2021.

– (2020n). *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 25. Februar 2021.

– (2020o). *Work with tabs*. Website. Online erhältlich unter `https://flutter.dev/docs/cookbook/design/tabs`; abgerufen am 25. Februar 2021.

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 25. Februar 2021.

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.

Meiller, Dieter (2020). *Moderne App-Entwicklung mit Dart und Flutter. Eine umfassende Einführung*. Oldenbourg: Walter de Gruyter.

Microsoft Corporation (2015a). *Delegaten (C#-Programmierhandbuch)*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/delegates/`; abgerufen am 25. Februar 2021.

– (2015b). *Generics C# – Programmierhandbuch*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/generics/`; abgerufen am 25. Februar 2021.

– (2016). *Xamarin.Forms Pages*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/pages`; abgerufen am 25. Februar 2021.

– (2018). *Xamarin.Forms Layouts*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/layouts`; abgerufen am 25. Februar 2021.

Microsoft Corporation (2019). *Xamarin.Essentials Einstellungen*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/essentials/preferences`; abgerufen am 25. Februar 2021.

– (2020a). *What is Xamarin?* Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/get-started/what-is-xamarin`; abgerufen am 25. Februar 2021.

– (2020b). *Xamarin.Essentials*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/essentials/`; abgerufen am 25. Februar 2021.

– (2020c). *Xamarin.Forms Ansichten*. Website. Online erhältlich unter `https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/views`; abgerufen am 25. Februar 2021.

MongoDB Inc. (2020). *Realm Mobile Database*. Website. Online erhältlich unter `https://www.mongodb.com/realm/mobile/database`; abgerufen am 25. Februar 2021.

Pedley, Adam (2019). *Moving From C# To Dart: Quick Start*. Website. Online erhältlich unter `https://buildflutter.com/moving-from-csharp-to-dart-quick-start/`; abgerufen am 25. Februar 2021.

Ritscher, Walt (2020). *Arranging Views with Xamarin.Forms Layout*. Website. Online erhältlich unter `https://bit.ly/34ulgpU`; abgerufen am 25. Februar 2021.

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

Rutishauser, Heinz (1952). „Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen“. In: *Journal of Applied Mathematics and Physics (ZAMP)* 3, S. 312–313.

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 25. Februar 2021.

Stellman, Andrew und Jennifer Greene (2021). *Head First C#. A Learner's Guide to Real-World Programming with C# and .NET Core*. englisch. Sebastopol: O'Reilly.

Tekartik (2020). *sqflite*. Website. Online erhältlich unter `https://pub.dev/packages/sqflite`; abgerufen am 25. Februar 2021.

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 25. Februar 2021.

Wilhelm, Reinhard, Helmut Seidl und Sebastian Hack (2012). *Übersetzerbau. Syntaktische und semantische Analyse*. Bd. 2.

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)