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

10. Januar 2021

Zusammenfassung

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

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

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 geringeren Ressourcenaufwand entwickelt werden.[2]

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.[3]

[1] Vgl. Joorabchi 2016, S. 3.
[2] Vgl. Vollmer 2017, S. 295.
[3] Vgl. Wissel, Liebel und Hans 2017, S. 64.

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 die mit Hilfe von Xamarin.Forms entwickelt wurden angepasst werden müssen.[4]

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

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

2 Compiler

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

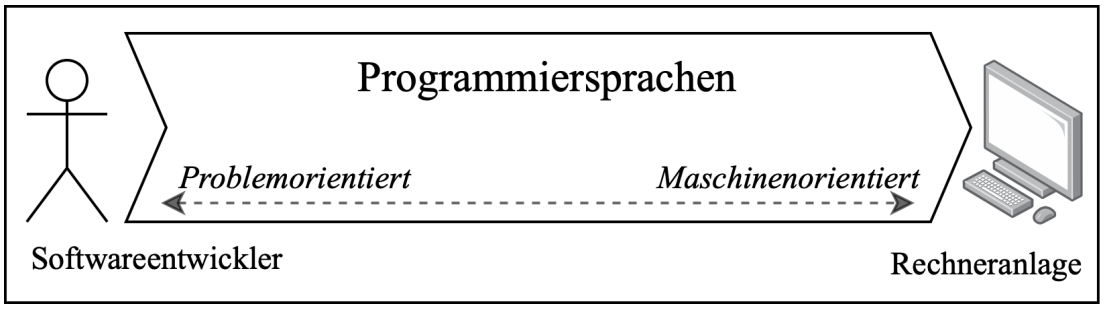

Abbildung 2.1: Programmiersprachen als Schnittstelle

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

2.1 Grundbegriffe

Diese historische Einführung zeigt, dass Software zur automatisierten Übersetzung schon seit der Mitte des letzten Jahrhunderts thematisiert wurde, so hat sich in der

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

Wissenschaft eine einheitliche Definition ergeben. Ullman et al. beschreibt die sogenannten Compiler im Jahre 2008 wie folgt:[5]

Defintion 1: Compiler

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

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

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

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

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

Defintion 2: Source-to-Source Compiler

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

[5] Vgl. Ullman et al. 2008, S. 1.
[6] Vgl. Ullman et al. 2008, S. 27.
[7] Vgl. Rohit, Aditi und Hardikar 2015, S. 1629.

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

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

2.1.2 Syntax und Semantik

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

2.2 Aufgabe

Die Aufgaben des Compilers lassen sich, mit dem Ziel der Übersetzung von Programmiersprachen, in die zwei Unteraufgaben Analyse und Synthese unterteilen.[8]
Bei der Analyse wird das Programm in seine Bestandteile zerlegt und mit einer grammatischen Struktur versehen. Diese wird anschließend verwendet um eine Zwischendarstellung des Quellprogramms zu erstellen. Dabei wird überprüft, ob das Programm syntaktisch oder semantisch nicht wohlgeformt ist, und ob der Programmierer Änderungen vornehmen muss. Außerdem werden bei der Analyse Informationen über das Quellprogramm gesammelt und in einer so genannten Symboltabelle abgelegt.[9]

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

Bei der Synthese wird aus der Zwischendarstellung un den Informationen aus der Symboltabelle das gewünschte Zielprogramm konstruiert. Der Teil des Compilers, der sich mit der Analyse befasst wird oft als Front-End bezeichnet, derjenige der für die Synthese zustädnig ist als Back-End.[10]

2.3 Phasen

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

2.3.1 Lexikalische Analyse

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

<Name, Attributwert>

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

2.3.2 Syntaxanalyse

Die zweite Phase des Compilers ist die Syntaxanalye. Dafür verwendet der sogenannte Parser die vom lexikalischen Analysator ausgegebenen Token um eine baumartige Zwischendarstellung zu erstellen. die die grammatische Struktur der Tokens zeigt. Die Darstellung wird daher auch häufig als Syntaxbaum bezeichnet. Die Knoten stehen dabei für eine Operation und seine Kindknoten für die Argumente dieser Operation. Die

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

Anordnung er Operationen stimmt mit üblichen arithmentischen Konventionen überein, wie zum Beispiel der vorrang der Multiplikation vor Addition.[14]

2.3.3 Semantische Analyse

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

2.3.4 Zwischencodeerzeugung

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

2.3.5 Codeoptimierung

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

2.3.6 Codeerzeugung

Bei der Codeerzegung werden die Eingaben aus der Zwischendarstellung des Quellprogramms entgegengenommen und auf die Zielsprache abgebildet. Ein entsheidender

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

Aspekt der Codeerzeugung ist die sinnvolle zuweisung von Registern für Variablen, falls es sich bei der Zielsprache um Maschinencode handelt.[18]

2.4 Rekursiver Ansatz

[18] Vgl. Ullman et al. 2008, S. 13.

3 Cross Plattform Frameworks

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

3.1 Frameworks

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

Flutter ist ebenfalls wie Xamarin.Forms ein Open Source Framework für die Erstellung von 2D mobilen Anwendungen. Dabei werden im Vergleich zu Xamarin.Forms keine

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

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

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

3.1.1 Projekte

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

3.1.2 Views

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

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

Pages

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

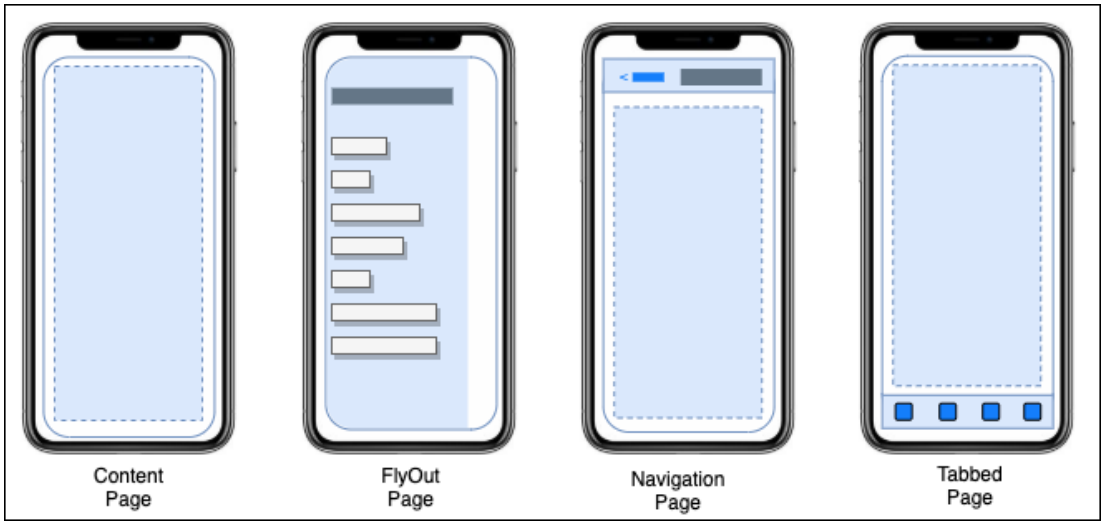

Abbildung 3.1: Xamarin.Forms Pages

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

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

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

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

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

Quelltext 3.1: Xamarin.Forms TabbedPage definition

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

```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 dabei drei alternative Styles: MaterialApp dass eine App mit dem von Google entwickelten Material Design erzeugt, CupertinoApp für eine App im iOS-Stil oder die Definition eines eigenen Styles für eine individuelle Anzeige.

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

Quelltext 3.3: Flutter MaterialApp definition

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

Elemente

Layouts

Listen

Gesten

Animtion

3.1.3 Navigation

Pages. Andere Apps

3.1.4 Async UI

3.1.5 Netzwerk Anfragen

3.1.6 Abhängigkeiten

3.1.7 Lebenzyklus

3.1.8 Schriften

3.1.9 Plugins

Interacting with hardware, third party services, and the platform

3.1.10 Databases and locale storage

3.1.11 Notifications

3.2 Programmiersprachen

C# Dart Vergleich XAML Dart

Literaturverzeichnis

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

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

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

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

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

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

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

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

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

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

Eidesstattliche Erklärung

Studierender: Julian Pasqué
Matrikelnummer: 902953

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

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

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