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

17. März 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 Anwendungen mit denen von nativen 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] 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.

1.1 Motivation

Im Mai 2020 hat Microsoft mit dem Multi-platform App User Interface (.NET MAUI) einen Nachfolger für das Xamarin.Forms Framework angekündigt, der im Herbst 2021 zusammen mit der sechsten Hauptversion des .NET Frameworks veröffentlicht werden soll. Zum aktuellen Zeitpunkt ist bereits bekannt, dass der Umstieg grundlegende Änderungen mit sich bringt und Anwendungen, 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 federführend von einer Firma 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 mobile Anwendung auf eine vermeintlich zukunftssichere Basis stellen. Denn obwohl Google in der Vergangenheit schon manche Projekte eingestellt hat, wie zum Beispiel Google Nexus oder Google Hangouts, ist damit bei Flutter aufgrund des Erfolges nicht zu rechnen. Nach offizieller Aussage von Tim Sneath, dem Produkt Manager des Frameworks, haben im Jahr 2020 mehr als zwei Millionen Entwickler Flutter verwendet und über 50.000 mobile Anwendungen programmiert.[6] Neben der hohen Verbreitung des Frameworks, konnte das Portal Stackoverflow in seinen jährlichen Umfragen auch eine hohe Beliebtheit unter Softwareentwicklern in den Jahren 2019[7] und 2020[8] ermitteln. Im März 2021 hat Flutter darüber hinaus die zweite Hauptversion von Flutter veröffentlicht, welche zusätzlich Support für die Entwicklung von Webseiten zur Verfügung stellt.[9]

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

[5] Vgl. Hunter 2020, Abgerufen am 28.10.2020.
[6] Vgl. Sneath 2020, Abgerufen am 28.10.2020.
[7] Vgl. Stack Exchange Inc. 2019, Abgerufen am 28.10.2020.
[8] Vgl. Stack Exchange Inc. 2020, Abgerufen am 28.10.2020.
[9] Vgl. Sells 2021, Abgerufen am 28.10.2020.

zentrale Forschungsfrage beantwortet werden soll: "Können Apps komplett automatisiert von Xamarin.Forms zu Flutter übersetzt werden, oder sind manuelle Arbeitsschritte erforderlich?"

1.3 Gliederung

Um diese Forschunsgfrage beantworten zu können, wird in Kapitel 2 auf die theoretischen Grundlagen von Software-Übersetzern eingegangen. Anschließend wird in Kapitel 3 auf den Entwurf des in dieser Arbeit zu implementierenden Compiler eingegangen, bevor in Kapitel 4 die Unterschiede zwischen den Frameworks Xamarin.Forms und Flutter behandelt werden. Darauf aufbauend wird in Kapitel 6 der Source-To-Source Compiler realisiert und in dem darauf folgen Kapitel 7 getestet bevor in Kapitel 8 die Forschungsfrage beantwortet wird und ein Fazit gezogen wird.

2 Compiler

Programmiersprachen dienen als Verständigungsmittel zwischen Programmierern und Rechenanlagen wie z.B Smartphones. 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.[10] Die Beziehung zwischen Softwareentwicklern und Rechenanlagen mit Hilfe von Programmiersprachen werden in Abbildung 2.1 dargestellt.

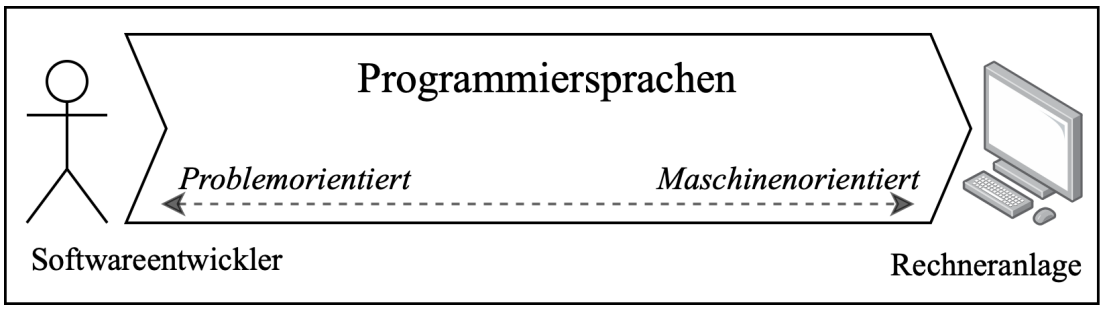

Abbildung 2.1: Programmiersprachen als Schnittstelle

Für die Ausführung einer in einer problemorientierten Programmiersprache geschriebenen Anwendung ist es notwendig, die Sprache in eine maschinenorientierte Form zu überführen.[11] Bereits im Jahre 1952 stelle Rutishauser fest, dass Computer in der Lage sind, diesen Übersetzungsvorgang selbst durchzuführen.[12] Durch die Möglichkeit zur automatischen Übersetzung von problemorientierten Programmiersprachen konnten Hochsprachen entwickelt werden, die menschenfreundliche Sprachelemente anstatt Maschineninstruktionen verwenden.[13]

[10] Vgl. Schneider 1975, S. 15.
[11] Vgl. Schneider 1975, S. 15.
[12] Vgl. Rutishauser 1952, S. 312.
[13] Vgl. Wagenknecht und Hielscher 2014, S. 47.

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:[14]

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 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 passieren, dass von verschiedenen Repräsentationen gesprochen wird. So können alle drei der folgenden Begriffe als Programm bezeichnet werden: Der Quelltext, das ausführbare Programm und der laufende Prozess auf einem Computer. 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 gehören zum Beispiel die sogenannten Binärübersetzer, die den Binärcode eines Programmes für andere Rechner übersetzen, sodass er auf diesen ausgeführt werden kann.[15] Ein Source-to-Source(S2S) Compiler, häufig auch als "Transpiler"bezeichnet, ist ebenfalls eine besondere Ausprägung eines Compilers, die sich wie folgt definieren lässt.[16]

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.[17]

[14] Vgl. Ullman et al. 2008, S. 1.
[15] Vgl. Ullman et al. 2008, S. 27.
[16] Vgl. Rohit, Aditi und Hardikar 2015, S. 1629.
[17] Vgl. Eisenecker 2008, S. 9.

2.2 Compiler Struktur

Zur Übersetzung von Programmen bearbeiten Compiler zwei Teilaufgaben, die Analyse und die Synthese. Während der Analyse wird das Programm in seine Bestandteile zerlegt und mit einer grammatikalischen Struktur versehen. Diese wird anschließend verwendet, um eine Zwischendarstellung zu generieren. Dabei wird überprüft, ob das Programm syntaktisch und semantisch fehlerfrei ist oder ob der Programmierer Änderungen vornehmen muss.[18] Der Begriff Syntax beschreibt den Aufbau eines Programmes, sie legt fest wie Sprachelemente aus anderen Sprachelementen zusammengesetzt sind. Im Gegensatz dazu beschreibt die Semantik die Bedeutung der Programmen und regelt die Bedeutung von Sprachelementen.[19] Außerdem werden bei der Analyse Informationen über das Quellprogramm gesammelt und in der so genannten Symboltabelle abgelegt. Die Synthese konstruiert aus der Zwischendarstellung und den Informationen aus der Symboltabelle das gewünschte Zielprogramm. 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.[20]

Der Vorgang des Kompilierens lässt sich basierend auf diesen zwei Teilaufgaben nach Ullman et al. in mehrere Phasen unterteilen, die in Abbildung 2.2 grafisch dargestellt sind und in diesem Abschnitt detailliert beschrieben werden.[21]

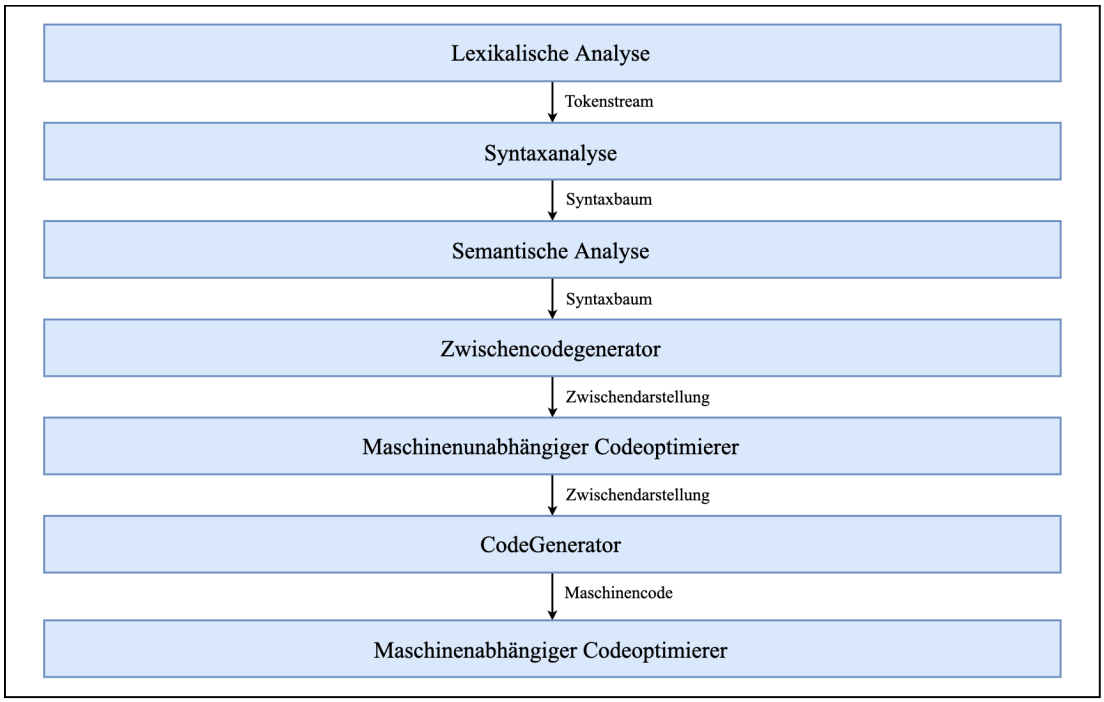

Abbildung 2.2: Phasen eines Compilers[22]

[18] Vgl. Ullman et al. 2008, S. 6f.
[19] Vgl. Schneider 1975, S. 36.
[20] Vgl. Ullman et al. 2008, S. 6f.
[21] Vgl. Ullman et al. 2008, S. 6.
[22] Abbildung in Anlehnung an Ullman et al. 2008, S.6.

2.3 Lexikalische Analyse

Die erste Phase eines Compilers ist die lexikalische Analyse, die den Quelltext in Lexeme untergliedert. 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 sein kann und der Attributwert auf einen Eintrag in der Symboltabelle verweist.[23] In Tabelle 2.1 werden einige beispielhafte Tokens aufgeführt sowie die Information darüber, 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[24]

Der Teil eines Compilers, der die Lexikalische Analyse durchführt, wird als Lexer bezeichnet. Basierend auf der beschriebenen Arbeitsweise ist in 2.3 ein Beispiel dargestellt, das zeigt wie der Lexer aus einer Zeichenfolge mehrere Tokens mit den optionalen Attributwerten extrahiert.

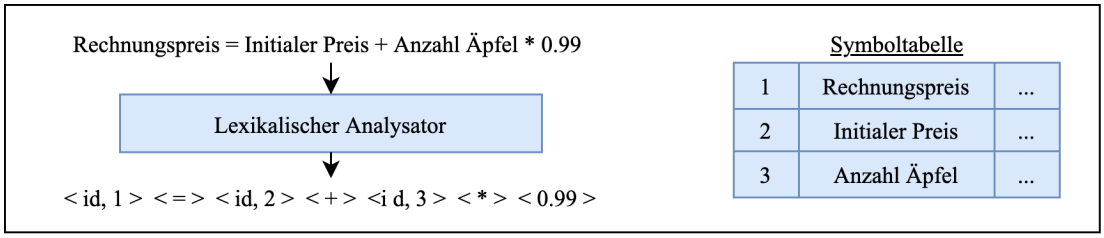

Abbildung 2.3: Lexer Beispiel[25]

Der Lexer interagiert mit anderen Komponenten eines Compilers, dies wird in Abbildung 2.4 visualisiert. Klassischerweise wird der Lexer über den sogenannten Parser, welcher im nächsten Abschnitt eingeführt wird, zur Übermittlung von Tokens aufgefordert, dies wird in der Abbildung mit dem Aufruf "getNextToken"dargestellt.[26]

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

[23] Vgl. Ullman et al. 2008, S. 135 f.
[24] Vgl. Ullman et al. 2008, S. 137.
[25] Abbildung in Anlehnung an Ullman et al. 2008, S.10.
[26] Vgl. Ullman et al. 2008, S. 135.

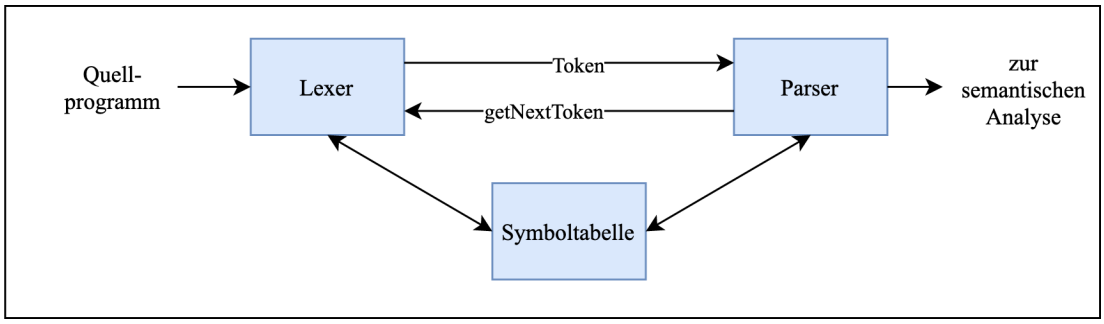

Abbildung 2.4: Interaktion zwischen Lexer und Parser[27]

zum Streichen von Kommentaren im Quelltext und zum Entfernen von Leerstellen, wie Leerzeichen und Tabulatoren. Zudem kann er gefundene Fehler den entsprechenden Zeilennummern zuordnen und dem Entwickler während der Kompilierung so einen genauen Hinweis auf den Ort des Fehlers geben. [28] Häufig werden Lexer daher in zwei kaskadierende Prozesse unterteilt, einen für das Löschen von Kommentaren und Zusammenfassung von Leerraumzeichen und einen für die eigentliche lexikalische Analyse.[29]

2.4 Syntaxanalyse

In der zweiten Phase, der Übersetzung, der Syntaxanalyse, werden durch den bereits erwähnten Parser auch syntaktischer Analysator genannt, die vom Lexer ausgegebenen Tokens in eine baumartige Zwischendarstellung überführt, die die grammatikalische 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 dem Vorrang der Multiplikation vor Addition.[30] Abbildung 2.5 zeigt die Erstellung eines Syntaxbaumes aus den Tokens der Abbildung 2.3. 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.[31]

[27] Abbildung in Anlehnung an Ullman et al. 2008, S.135.
[28] Vgl. Ullman et al. 2008, S. 135.
[29] Vgl. Ullman et al. 2008, S. 136.
[30] Vgl. Ullman et al. 2008, S. 9.
[31] Vgl. Ullman et al. 2008, S. 9.

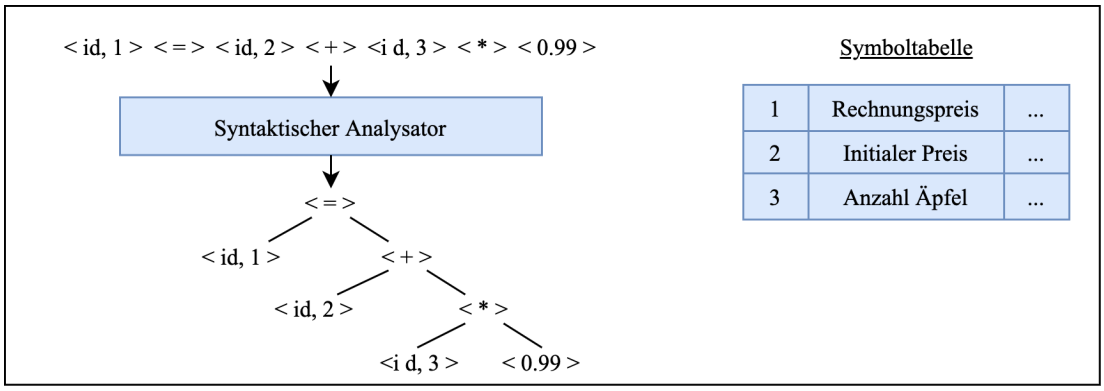

Abbildung 2.5: Syntaxbaum[32]

2.5 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.[33]. Zudem werden hier Typinformationen gesammelt und zur späteren Verwendung im Syntaxbaum oder der Symboltabelle hinterlegt. Auch findet eine Typüberprüfung statt die analysiert, ob jeder Operator die passenden Operanden hat. So wird beispielsweise validiert, ob ein Index eine Ganzzahl ist. Es besteht die Möglichkeit, innerhalb des Baums Typkonvertierungen zu deponieren. So wurde in dem bisherigen Beispiel die Anzahl Äpfel als Ganzzahl behandelt und wird für die Berechnung des Preises in Abbildung 2.6 zu einer Fließkommazahl konvertiert.[34]

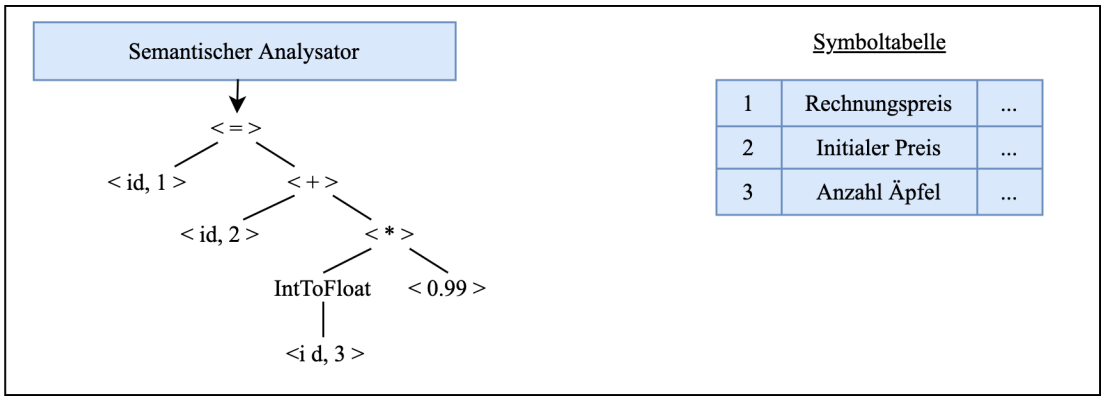

Abbildung 2.6: Typüberprüfung[35]

[32] Abbildung in Anlehnung an Ullman et al. 2008, S.10.
[33] Vgl. Wilhelm, Seidl und Hack 2012, S. 157.
[34] Vgl. Ullman et al. 2008, S. 9ff.
[35] Abbildung in Anlehnung an Ullman et al. 2008, S.10.

2.6 Zwischencodeerzeugung

Während der Übersetzung eines Programms kann der Compiler mehrere Zwischendarstellungen in unterschiedlichsten Formen, zum Beispiel wie die eines Syntaxbaums, erstellen. Nach der semantischen Analyse stellen viele Compiler eine maschinennahe Zwischendarstellung auf niedriger Abstraktionsebene her, die eigentlich für maschinenabhängige Aufgaben wie Befehlsauswahl geeignet ist. Eine Zwischendarstellung, die

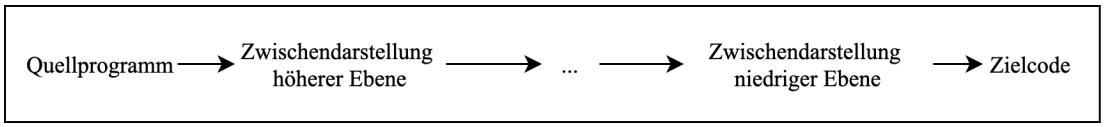

Abbildung 2.7: Zwischendarstellungen[36]

von Compiler zu Compiler in Auswahl oder Entwurf unterschiedlich ist, 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 effizientem Maschinencode kompiliert werden kann und ihre Compiler weitgehend verfügbar sind.[37]

2.7 Codeoptimierung

In dieser Phase wird der Code so optimiert, dass sich daraus ein besserer, das heißt schnellerer oder ressourcenschonender Zielcode ergibt. Der Umfang der Codeoptimierung schwankt dabei von Compiler zu Compiler erheblich.[38] Die Codeoptimierung, die ein Compiler vornimmt, ist im Laufe der Zeit wichtiger und komplexer geworden. Grund für die zunehmende Komplexität sind die immer komplexeren Prozessorarchitekturen, die mehr Gelegenheiten bieten, die Ausführung des Codes zu verbessern. Die gestiegene Bedeutung ergibt sich Beispielsweise aus der steigenden Anzahl an Kernen in modernen Computern und der Möglichkeit, Programme parallel auszuführen.[39]

[36] Abbildung in Anlehnung an Ullman et al. 2008, S.433.
[37] Vgl. Ullman et al. 2008, S. 433.
[38] Vgl. Ullman et al. 2008, S. 11f.
[39] Vgl. Ullman et al. 2008, S. 20.

2.8 Codeerzeugung

Die Überführung aus der Zwischendarstellung in die Zielsprache nennt man Codeerzeugung. Hierbei muss die semantische Bedeutung des Quellprogramms erhalten und hochwertig dargestellt sein. Die größte Herausforderung ergibt sich aus der nicht komplett mathematischen Berechenbarkeit aller Prozesse bei der Überführung. Ein Beispiel wäre die Vergabe von Registern, die nicht effizient berechenbar sind. In der Praxis müssen heuristische Techniken ausreichen- die guten, aber nicht unbedingt optimalen Code liefern. Die Codeoptimierungs- und Codeerzeugungsphasen können mehrfach durchlaufen werden, bevor das Zielprogramm finalisiert ist.[40]

2.9 Der .NET Compiler Roslyn

Für die Arbeit mit der Programmiersprache C# steht mit Roslyn ein Compiler zur Verfügung, der sich aus modularen Bibliotheken zusammensetzt. Durch die Referenzierung dieser Bibliotheken können Programme auf den Funktionsumfang von Roslyn zugreifen. So ist es möglich, den Compiler zu verwenden, ohne das Ziel zu haben, die Programmiersprache C# in plattformnahen Code zu übersetzen. Dabei stehen die Bibliotheken über den Packetmanager Nuget für die Einbindung in eigene Projekte zur Verfügung. Um diese Funktionalität zu gewährleisten, unterteilt Rosyln die Übersetzung in mehrere Phasen, welche wiederum einige der in diesem Kapitel beschriebenen Phasen zusammenfassen. Die erste Phase ist die Erstellung des Syntaxbaums, die zweite Phase ist die semantische Analyse gefolgt von der letzten Phase der Ausgabe der so genannten Intermediate Language als Zielsprache.[41]

[40] Vgl. Ullman et al. 2008, S. 618f.
[41] Vgl. Albahari und Johannsen 2020, S. 1017.

3 Compiler Spezifikation

Zur Entwicklung eines möglichst effektiven Compilers ist es notwendig, die genauen Anforderungen zu ermitteln, um dann passende Softwaretechnische Lösungen zu erarbeiten.[42] Im speziellen Falle besteht die Anforderung darin, vom Quellframework Xamarin.Forms in das Zielframework Flutter zu übersetzen, welche beide für die Entwicklung von plattformunabhängigen Smartphone-Apps verwendet werden können. Für eine genaue Spezifikation des zu realisieren Source-To-Source Compilers ist es notwendig, einen Überblick über das Compiler-Umfeld zu erhalten. Dieses wird in Abbildung 3.1 visualisiert.

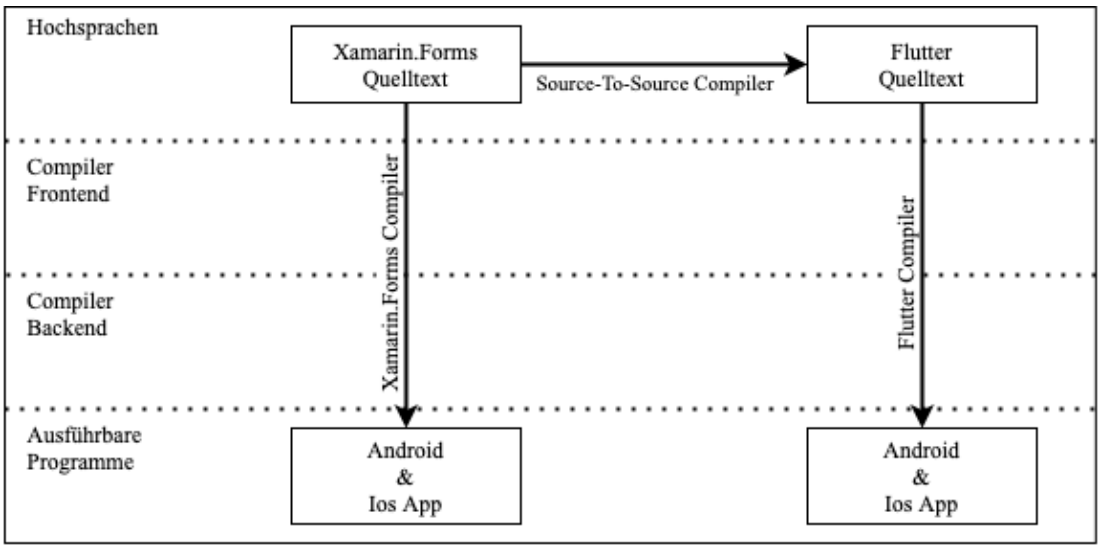

Abbildung 3.1: Umfeld des Source-To-Source Compilerns

Wie auf der Abbildung erkennbar ist, durchlaufen sowohl Xamarin.Forms als auch Flutter bei der Übersetzung zu mobilen Anwendungen die in Kapitel 3.2 erläuterten Phasen, kurz dargestellt als Compiler Front- und Backend. Der Source-To-Source Compiler ist in der Abbildung horizontal dargestellt, was veranschaulichen soll, dass sich sowohl die Quelle, als auch das Ziel der Übersetzung auf einer Abstraktionsebene befinden. Auch bei dieser Übersetzung sind die Compilerphasen aus dem vorherigen Kapitel anzuwenden. So lässt sich die Abbildung 3.1, wie in 3.2 dargestellt, um ein Front- und Backend erweitern.

[42] Vgl. Balzert 2011, S.6.

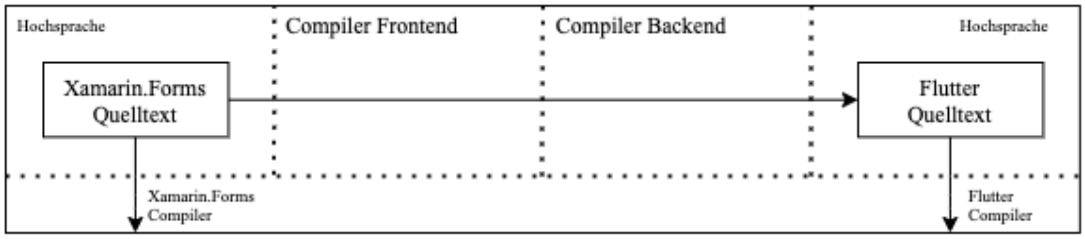

Abbildung 3.2: Source-To-Source Compiler Aufbau

Laut Definition von Compilern, erzeugen diese ein gleichwertiges Programm in einer Zielsprache. Sowohl Xamarin.Forms als auch Flutter stellen mit Hilfe ihrer Compiler gleichwertige Programme in Form von mobilen Apps dar. Da der Source-to-Source Compiler ebenfalls eine gleichwertige Darstellung erzeugt, ist anzunehmen, dass die übersetzte Ursprungs-App gleichwertig zu der übersetzten Flutter App ist.

3.1 Funktionseingrenzung

Zur Beantwortung der Forschungsfrage ist es ausreichend, dass der Prototyp einen begrenzten Funktionsumfang hat. Für eine zielführende Eingrenzung eignen sich die folgenden fünf Aspekte:

- Framework Version: Der in dieser Arbeit zu realisierende Prototyp soll ausschließlich das offiziell von Microsoft veröffentliche Xamarin.Forms in der Version 5.0.0.2012 zu Flutter übersetzten.
- Erweiterungen von Dritten: Viele Firmen und einzelne Entwickler haben Erweiterungen für Xamarin.Forms programmiert. Aufgrund der großen Anzahl und stetigen Veränderung dieser Erweiterungen, werden sie in dieser Arbeit nicht weiter betrachtet.
- Plattformspezifischer Quelltext: Xamarin.Forms erlaubt die Verwendung von plattformspezifischem Quelltext, der in dieser Arbeit keine Beachtung finden wird, da eine gleichwertige Darstellung in Flutter nicht garantiert werden kann.
- Benutzerberflächen (UI): Für die Entwicklung von Benutzeroberflächen kann die Programmiersprache C# verwendet werden, jedoch hat die Alternative XAML (Extensible Application Markup Language) für Entwickler Vorteile,[43] weswegen die Konstruktion von Benutzeroberflächen mit C# in dieser Arbeit nicht berücksichtigt wird.

[43] Vgl. Microsoft Corporation 2017, Abgerufen am 17. März 2021.

- App-Styles: Für die visuelle Darstellung wird in dieser Arbeit ausschließlich das Design-System Material von Google unterstützt. Da die Übersetzung von Darstellungsoptionen sehr aufwendig ist, und für den in dieser Arbeit zu entwickelnden Prototypen nicht notwendig ist.

Diese Eingrenzungen führen in Summe zu einer Vielzahl von nicht in Gänze übersetzbaren Xamarin.Forms Anwendungen. Durch Erweiterungen des Compilers könnte diese Limitierung in Zukunft aufgehoben werden. Im Rahmen dieser Arbeit wird eine mobile Xamarin.Forms Anwendung entworfen, die vollständig übersetzt werden kann, da sie keine der oben definierten Ausschlüsse verwendet.

3.2 Übersetzung von verschiedenen Dateien

Durch seinen modularen Aufbau, kann der im letzten Kapitel eingeführte Roslyn Compiler die Phasen bis zur semantischen Analyse im zu entwickelnden Prototypen übernehmen. Anschließend kann mithilfe des dabei typisierten Syntaxbaumes die Übersetzung in die Zielsprache durchgeführt werden. Die Übersetzung mit Roslyn hat Grenzen, die aus der Zusammensetzung von Xamarin.Forms Projekten resultiert. Wie in Abbildung 3.3 zu erkennen ist, setzten sich Xamarin.Forms Projektmappen aus verschiedenen Dateien zusammen, von denen ausschließlich die Klassen beinhaltenden Dateien mit der Endung ‚cs' analysiert werden können.

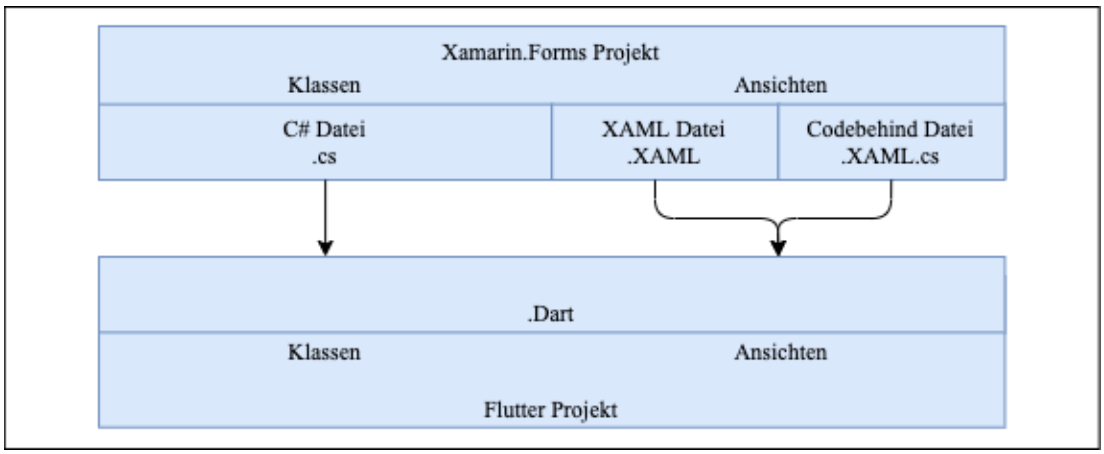

Abbildung 3.3: Compiler Struktur

Neben den Klassen zeigt die Abbildung 3.3, dass auch Ansichten ein Teil von Xamarin.Forms Projekten sind. Diese bestehen aus ‚XAML' sowie ‚XAML.cs' Dateien. Alle Ausgangsdateien müssen zu Dart-Dateien kompiliert werden, um ein Flutter Projekt als Ziel zu ergeben. Die Zusammenführung von ‚XAML' und ‚XAML.cs' Dateien ist dabei notwendig, weil Flutter ohne sogenannte Codebehind Dateien auskommt.

3.3 Grafische Darstellung

Damit Unternehmen und Entwickler ihre bestehenden Xamarin.Forms Anwendungen übersetzen können, muss eine Möglichkeit für die Interaktion mit dem Source-To-Source Compiler existieren. Dieser Compiler ist zur einmaligen und nicht regelmäßigen Verwendung ausgelegt und braucht somit nicht in einer Entwicklungsumgebung integrieren werden. Der Roslyn Compiler ist ausschließlich für das Betriebssystem Windows verfügbar, die zu entwickelnde Oberfläche muss dementsprechend auf Windows Computern lauffähig sein. Abbildung 3.4 zeigt einen Entwurf (engl. Mockup) der geplanten grafischen Benutzeroberfläche (GUI).

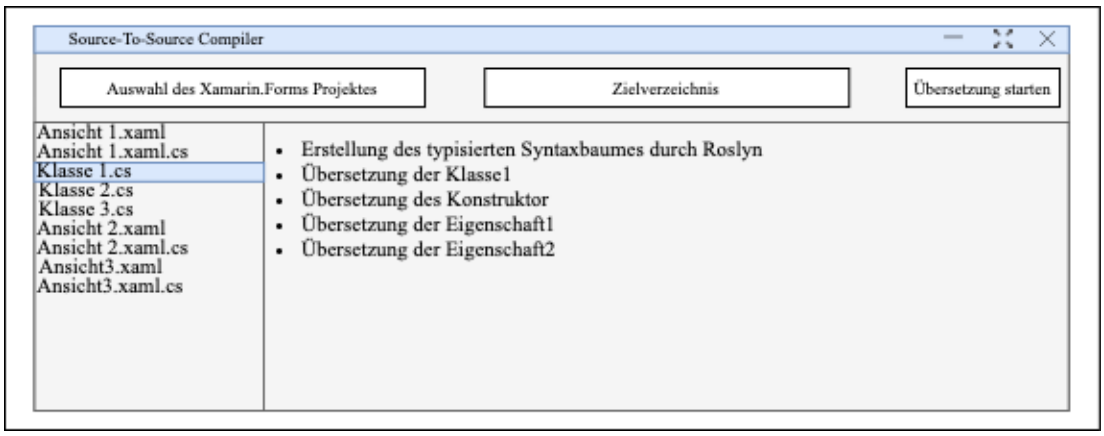

Abbildung 3.4: Mockup der grafischen Oberfläche

Im oberen Teil der GUI befindet sich eine Auswahl für das Quellprojekt und das Zielverzeichnis des Compilers. Der untere Teil der Ansicht zeigt die Ausgabe des Übersetzers, für die im linken Bereich alle bearbeiteten Dateien angezeigt werden. Bei der Auswahl einer Datei werden in dem Bereich daneben alle vorgenommen Übersetzungsschritte aufgeführt.

3.4 Quelltext Optimierung

Die Optimierung des Quelltextes ist, wie in Kapitel 2 beschrieben, eine Phase der Kompilierung. Im Gegensatz zu den dort beschriebenen Aspekten (Geschwindigkeit und Ressourcenschonung) sind für den Source-To-Source Compiler andere Faktoren wie der Austausch von Klassen und Methoden relevant, da diese Ressourcenoptimierung später bei der Übersetzung durch den Flutter Compiler stattfinden. Der Bedarf zum Austausch von Klassen und Methoden resultiert aus unterschiedlichen Arbeitsweisen der Frameworks, sodass eine einfache 1:1 Übersetzung nicht möglich ist. Um dies zu

visualisieren, wird folgend ein Quelltextbeispiel aus beiden Frameworks gezeigt, die die selbe Funktionalität abbilden.

```
var photo = await MediaPicker.PickPhotoAsync();

if(photo == null)
{
 Console.WriteLine("No image selected.");
}
```

Quelltext 3.1: Bilderauswahl in Xamarin.Forms

```
final pickedFile = await picker.getImage(source: ImageSource.camera);

setState(() {
    if (pickedFile != null) {
          _image = File(pickedFile.path);
    } else {
          print('No image selected.');
    }
});
```

Quelltext 3.2: Bilderauswahl in Dart

Beide Crossplatform Frameworks verwenden in diesem Beispiel unterschiedliche Klassen für die Auswahl eines Bildes aus der Smartphonegalerie. Daher ist es notwendig, beide Frameworks zu analysieren und die genauen Unterschiede zwischen den Arbeitsweisen zu verstehen. Zu diesem Zweck werden im nachfolgenden Kapitel sowohl die Frameworks als auch deren Programmiersprachen analysiert. Hieraus resultiert das Verständnis, inwiefern sich Benutzeroberflächen und Sprachen unterscheiden und wie diese übersetzt werden können.

Source-To-Source Compiler bilden eine Brücke zwischen zwei Hochsprachen. Der für die Beantwortung der Forschungsfrage geplante Compiler soll darüber hinaus auch die Arbeitsweisen des Quellprogramms in das Zielprogramm übersetzen. Das heißt, es soll versucht werden, ein frameworkbasierte App in die Form eines Zielframeworks zu überführen.

4 Technische Unterschiede zwischen Xamarin.Forms und Flutter

Die Unterschiede zwischen den Frameworks werden im folgenden genauer betrachtet. Für den technischen Vergleich dient Xamarin.Forms als Grundlage. Die Namen von Abschnitten und Unterabschnitten orientieren sich deshalb an dessen Terminologie. In den jeweiligen Gliederungspunkten wird anschließend genauer betrachtet, wie sich spezielle Arbeitsweisen oder Darstellungsoptionen in Flutter abbilden lassen.

4.1 Projektaufbau

Xamarin.Forms weißt eine andere Projektstruktur auf als Flutter, das nur mit einem Projekt arbeitet. Wärend das Flutter Projekt alle notwendigen Inhalte für iOS und Android inkludiert,[44] setzt sich die Projektmappe bei Xamarin.Forms aus mehreren Projekten zusammen. Es gibt für jede Plattform ein dediziertes Projekt, dass den plattformspezifischen Code, Konfigurationen und Icons beinhaltet, sowie ein Projekt für den plattformunabhängigen Quelltext.[45] Icons und Konfigurationen werden bei Flutter in einem gleichen oder ähnlichen Format und nur in einem anderen Projekt hinterlegt und lassen sich folglich migrieren. In den Ausschlusskriterien in Kapitel 3 dieser Arbeit wurde bereits der plattformspezifische Quelltext von Xamarin.Forms für die Übersetzung exkludiert.

[44] Vgl. Biessek 2019, S. 113.
[45] Vgl. Petzold 2016, S. 25f.

4.2 Ansichten

Ansichten (engl. Views) sind visuelle Elemente, die in zwei Kategorien unterschieden werden können. Steuerelemente (engl. Controls) sind für die Sammlung von Benutzereingaben oder die Ausgabe von Daten verantwortlich und Layouts, beinhalten eine Sammlung von Ansichten und sind für ihre visuelle Anordung auf der Benutzeroberfläche verantwortlich.[46]

4.2.1 Layouts

Ähnlich wie die Ansichten lassen sich auch die Layouts in zwei Kategorien unterteilen. Den Ansichtsseiten (engl. Pages) sowie den generellen Layouts. Die Pages nehmen den gesamten Bildschirm ein und werden in Abbildung 4.1 dargestellt.[47]

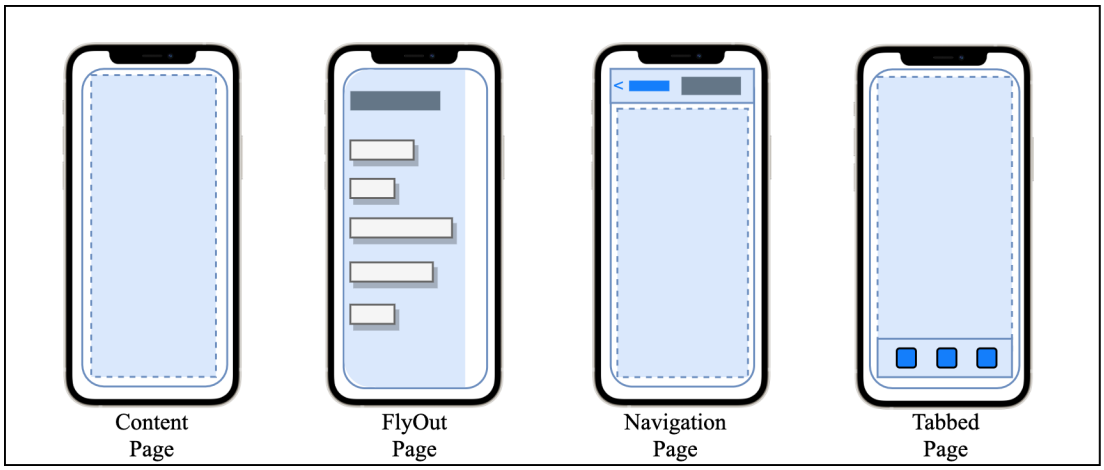

Abbildung 4.1: Xamarin.Forms Pages[48]

‚ContentPage ' ist ausschließlich für die Anzeige einer weiteren Ansicht verantwortlich. Die drei anderen Pages besitzen ein Navigationskonzept. ‚FlyOutPage' teilt den Bildschirm in zwei Bereiche, ein Bereich dient der Navigation. Er enthält ein Menü das wie im Namen enthalten einfliegen kann. Der zweite Bereich zeigt eine Detailansicht, in welcher der Inhalt der angeforderten Seite geladen wird. ‚NavigationPage' bietet eine Navigationsleiste, die einen Titel der aktuellen Seite und eine Navigationsschaltfläche beinhalten kann. ‚TabbedPage' stellt die unterschiedlichen Seiten als Registerkarten dar.[49] Die Ansichtseiten befinden sich in der Regel, innerhalb der XAML-Datei, auf der untersten Ebene, dem so genannten Wurzelknoten. Der Quelltext 4.1 zeigt dies exemplarisch für eine ‚TabbedPage' dargestellt, angegeben.

[46] Vgl. Ritscher 2020, Abgerufen am 17. März 2021.
[47] Vgl. Microsoft Corporation 2016, Abgerufen am 17. März 2021.
[48] Abbildung in Anlehnung an Microsoft Corporation 2016, Abgerufen am 17. März 2021.
[49] Vgl. Microsoft Corporation 2016, Abgerufen am 17. März 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 4.1: Xamarin.Forms ‚TabbedPage' Definition[50]

Es wird eine ‚TabbedPage' mit drei Registerkarten entworfen. Eine Kombination mehrerer Navigationskonzepte ist möglich, das Beispiel zeigt eine Navigationsleiste innerhalb der Registerkarten.

Die verfügbaren Eigenschaften der Ansichtsseiten unterscheiden sich je nach Einsatzszenario. Im folgenden Quelltext 4.2 wird dies exemplarisch an der Realisierung einer ‚FlyoutPage' deutlich. Anders als bei ‚TabbedPage', die aus einer Sammlung von Registerkarten besteht, finden sich im Quelltext die Eigenschaften ‚Flyout' und ‚Detail'.

```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 4.2: Xamarin.Forms ‚FlyoutPage' Definition[51]

Im Unterschied zu Xamarin.Forms kann Flutter auf der Wurzelebene nur den Style der App, nicht aber ein Navigationskonzept definieren. Wie bereits in Kapitel 3 aufgeführt, wird in dieser Arbeit ausschließlich der Material Design Style unterstützt.[52] Quelltext 4.3 zeigt die Realisierung einer MaterialDesign App in Flutter.

[50] Quelltext in Anlehnung an Microsoft Corporation 2020d, Abgerufen am 17. März 2021.
[51] Quelltext in Anlehnung an Microsoft Corporation 2020c, Abgerufen am 17. März 2021.
[52] Vgl. Google LLC 2020b, Abgerufen am 17. März 2021.

```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 4.3: Flutter ‚MaterialApp' Definition[53]

Der Vergleich zwischen den Extensible Markup Language(XML) basierten XAML-Dateien und den bei Flutter verwendeten Dart-Dateien verdeutlicht die Unterschiede in den verwendeten Sprachen zur Benutzeroberflächenentwicklung.

Die zentrale Idee hinter dem Flutter-Framework ist es, eine Benutzeroberfläche aus Widgets aufzubauen. Diese beschreiben das Aussehen der Anwendung basierend auf ihrem aktuellen Zustand. Sobald sich der Status ändert, kann das Framework den neuen mit dem alten vergleichen, um grafische Veränderungen möglichst effektiv vorzunehmen.[54] Damit in Flutter ein Navigationskonzept definiert werden kann, können verschiedene Widgets verwendet und verschachtelt werden, wie in Quelltext 4.4 beispielhaft für eine App mit Registerkarten visualisiert wird.

```dart
        appBar: AppBar(
          bottom: TabBar(
            tabs: [ Tab(icon: Icon(Icons.directions_car)),
                    Tab(icon: Icon(Icons.directions_transit)),
                    Tab(icon: Icon(Icons.directions_bike)),
            ],
          ),
        ),
        body: TabBarView(
          children: [ Icon(Icons.directions_car),
                      Icon(Icons.directions_transit),
                      Icon(Icons.directions_bike),
          ],
        ),
```

Quelltext 4.4: Flutter ‚Tab Layout' Definition[55]

[53] Quelltext in Anlehnung an Google LLC 2020h, Abgerufen am 17. März 2021.
[54] Vgl. Google LLC 2020d, Abgerufen am 17. März 2021.
[55] Quelltext in Anlehnung an Google LLC 2020g, Abgerufen am 17. März 2021.

Die deutlichen Unterschiede bei der Auswahl eines Navigationkonzeptes können dadurch überbrückt werden, dass man zu jeder Xamarin.Forms Page das entsprechende Flutter Widget findet. Der Flutter-Widgetkatalog[56] und die Webseite "Flutter for Xamarin.Forms Developers"[57] wurde für die Recherche des Gegenstückes verwendet. Entsprechende Ergebnisse der Suche können in Tabelle 1 abgelesen werden.

Xamarin.Forms Page	**Flutter Widget**
ContentPage	
FlyOutPage	MasterDetailScaffold
NavigationPage	Scaffold
TabbedPage	TabBar und TabBarView

Tabelle 4.1: Gegenüberstellung Pages

Gegenüberstellungen, in Tabellenform, von Xamarin.Forms Elementen und Flutter Widgets werden auch an anderer Stelle in diesem Kapitel der Übersichtlichkeit halber verwendet. Eine vollständige Referenztabelle die sich aus allen Einzelbetrachtungen zusammensetzt, befindet sich in 8.1.

Neben den Ansichtsseiten bietet Xamarin.Forms weitere Layouts, die Steuerelemente zu visuellen Strukturen zusammenstellen. Abbildung 4.2 stellt die gebräuchlichsten dieser Layouts vor.

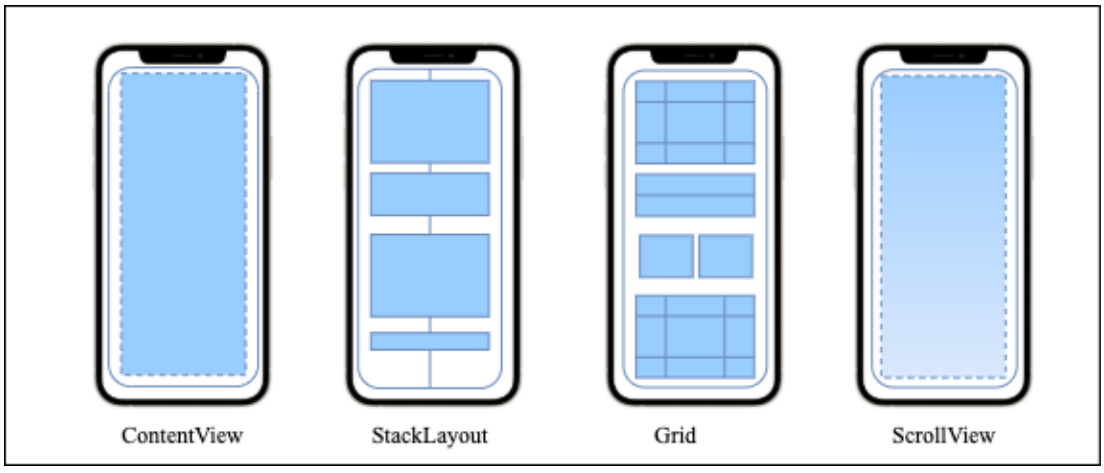

Abbildung 4.2: Xamarin.Forms Layouts[58]

Die vorgestellten Layouts haben unterschiedliche visuellen Eigenschaften und dienen als Sprachelemente von XAML für den Entwurf von Benutzeroberflächen. ‚ContentView' enthält ein einzelnes untergeordnetes Ansichtselement und wird als Basisklassen für benutzerdefinierte Darstellungen verwendet. Das Gestaltungselement ‚StackLayout'

[56] Vgl. Google LLC 2020f, Abgerufen am 17. März 2021.
[57] Vgl. Google LLC 2020b, Abgerufen am 17. März 2021.
[58] Abbildung in Anlehnung an Microsoft Corporation 2018, Abgerufen am 17. März 2021.

legt untergeordnete Elemente in einem entweder horizontal oder vertikal angeordneten Stapel ab. Ein ‚Grid' positioniert seine untergeordneten Elemente in einem Raster aus Zeilen und Spalten, es wird auch dafür verwendet Layouts und Steuerelemente aufeinander zu legen. Das ‚ScrollView' erlaubt das verschieben von Bildschirminhalten und hat wie ein ‚ContentView' nur ein untergeordnetes Element. Neben diesen gängigen Layouts gibt es noch weniger verbreitete. zum Beispiel: Das ‚Frame,' das einen Rahmen um ein visuelles Element zeichnet. Das ‚AbsolutLayout', platziert untergeordnete Elemente an bestimmten Positionen relativ zu ihrem übergeordneten Element und das ‚RelativeLayout' übernimmt die gleiche Aufgabe jedoch nur auf der Ebene des Layout und untergeordneter Elemente.[59]

Basierend auf diesen verfügbaren Layouts werden in Tabelle 4.2 die es entsprechende Flutter Widgets entgegengesetzt.

Xamarin.Forms Layout	**Flutter Widget**
AbsolutLayout	Positioned
ContentView	StatelessWidget
Frame	BoxDecoration
Grid	GridView oder Stack für das Stapeln von Elementen
ScrollView	SingleChildScrollView
StackLayout	Row und Column
ReleativLayout	Positioned

Tabelle 4.2: Gegenüberstellung Layouts

Widgets haben zum Teil erweiterte, oder abweichende Funktionalitäten, sodass Optimierungen durch den Compiler notwendig sind. Damit das Layout Grid in Xamarin.Forms die Möglichkeit für einen Bildlauf bekommen weil der Inhalt zu groß für die Darstellung auf einer Seite ist, wird das Grid in einem ScrollView verschachtelt. Dagegen bietet das GridView vWidget von Flutter die option des Scrollen automatisch an, wenn der Inhalt den sichtbaren Bereich überschreitet.[60] Im Rahmen der Codeoptimierung muss das ScrollView in diesem Anwendungsfall entfernt werden.

4.2.2 Steuerelemente

Steuerelemente sind die sichtbaren Bausteine der Benutzeroberflächen, beispielsweise Schaltflächen, Beschriftungen und Textfelder. Microsoft kategorisiert die Steuerele-

[59] Vgl. Microsoft Corporation 2018, Abgerufen am 17. März 2021.
[60] Vgl. Google LLC 2020c, Abgerufen am 17. März 2021.

mente innerhalb der Frameworkdokumentation anhand ihrer primären Verwendung.[61] Diese Einteilung wird folgend übernommen, obwohl eine klare Abgrenzung der Steuerelemente zu Kategorien nicht uneingeschränkt möglich ist, da einzelne zu mehreren Gruppierungen passen.

Steuerelemente für die Präsentation

Einige Steuerelemente sind ausschließlich für die Darstellung von Inhalten vorgesehen. In Xamarin. Forms gibt es die folgenden Darstellungssteuerelemente, für die eine Flutter Repräsentation notwendig ist. Das Steuerelement ‚BoxView' zeigt in Xamarin.Forms ein einfarbiges Rechteck an. Für die Darstellung von Texten wird auf ‚Label' zurückgegriffen. Bilder können mit Hilfe des ‚Image' Steuerelement angezeigt werden, wobei diese aus verschiedenen Quellen, wie dem Web oder aus den Ressourcen der App geladen werden können. Das Steuerelement ‚Map' kann für die Anzeige von Karten innerhalb der mobilen Anwendung verwendet werden. Um Web und HTML Inhalte innerhalb einer App visualisieren zu können steht das ‚WebView' Steuerelement bereit.[62] Für die Steuerelemente kann nun eine Gegenüberstellung zwischen Xamarin.Forms Elementen und Flutter Widgets vorgenommen werden dies wird in Tabelle 4.3 dargestellt.

Xamarin.Forms Steuerelement	**Flutter Widget**
BoxView	SizedBox
Image	Image
Label	Text
Map	Leamaps oder Google Maps
WebView	webview_flutter
Ellipse	CustomPaint
Linie	CustomPaint
Path	CustomPaint
Polygon	CustomPaint
Polyline und Rectangle	CustomPaint
Rectangle	CustomPaint

Tabelle 4.3: Gegenüberstellung Darstellungssteuerelemente

Zu zeichnende Elemente, wie die ‚Ellipse', ‚Linie', ‚Path', ‚Polygon', ‚Polyline' und ‚Rectangle'wurden nicht besonders aufgeführt, da diese bei Flutter auf die sogenannte Canvas der Benutzeroberfläche gezeichnet werden können.[63]

[61] Vgl. Microsoft Corporation 2020b, Abgerufen am 17. März 2021.
[62] Vgl. Microsoft Corporation 2018, Abgerufen am 17. März 2021.
[63] Vgl. Google LLC 2020a, Abgerufen am 17. März 2021.

Ereignisauslösende Steuerelemente

Xamarin.Forms ist ein ereignisgesteuertes Framework. Die hier behandelten Steuerelemente stellen alle mindestens ein Ereignis zur Verfügung, das mithilfe der in Kapitel 3 erwähnten Codebehind Klassen abonniert werden kann. Sobald ein ein sogenantes Event ausgelöst wird, übermittelt das Framework diese Information an den Empfänger. Die folgenden Steuerelemente werden bei Xamarin.Forms dieser Kategorie zugeordnet. ‚Buttons' sind rechteckige Objekte, die ein ‚clicked' Ereignis auslösen, nachdem sie von einem Anwender gedrückt wurden. Mit ‚ImageButton' steht ebenfalls eine Variante zur Verfügung, die ein Icon statt einem Text anzeigt. Bei einem ‚RadioButton' wird eine Option aus einer Reihe von Möglichkeiten ausgewählt und löst ein Ereignis aus, wenn sich die Benutzerauswahl ändert. Ein weiteres Steuerelement ist ‚RefreshView', dass eine ‚PullToRefresh' Funktionalität für Layouts mit Bildlauf anbietet. Dabei wird durch das herunterziehen des Seiteninhaltes der Wunsch zur Seitenaktualisierung übermittelt. Mithilfe der ‚Searchbar' haben Anwender die Möglichkeit Inhalte innerhalb der App zu suchen. Durch die Eingabe von Textzeichenfolgen kann per Schaltfläche, oder Tastaturtaste, ein Ereignis ausgelöst und der eingegeben Text an die Codebehind-Datei weiterleitet werden. Tabelle 4.9 die ereignisauslösende Steuerelementen von Xamarin.Forms und alternative Flutter Widgets.

Xamarin.Forms Steuerelemente	**Flutter Widget**
Button	Flatbutton
ImageButton	IconButton
RadioButton	RadioButton
RefreshView	pull_to_refresh
SearchBar	flutter_search_bar
SwipeView	flutter_slideable

Tabelle 4.4: Gegenüberstellung ereignisauslösende Steuerelemente

Flutter Widgets verhalten sich nicht exakt gleich wie die Steuerelemente von Xamarin.Forms. Ein Beispiel ist die hier erwähnte SearchBar, die bei Flutter im Gegensatz zu Xamarin.Forms nicht frei platzierbar ist, sondern immer in der Navigationsleiste angezeigt wird.

Die Beziehung zwischen Steuerelementen und Codebehind über Ereignisse wird in den beiden folgenden Quelltextausschnitten demonstriert. Der erste Ausschnitt zeigt XAML Quelltext, durch welchen ein Button dargestellt werden kann. Über die Eigenschaft clicked wird auf eine Methode in der XAML.cs Datei verwiesen, die in dem zweiten Quelltextausschnitt abgebildet ist.

```
<Button Text="Click Me!"
    Grid.Row="0"
    Font="Large"
    BorderWidth="1"
    HorizontalOptions="End"
    VerticalOptions="CenterAndExpand"
    Clicked="Button_Clicked" />
```

Quelltext 4.5: Xamarin.Forms Button Initialisierung

```
private void Button_Clicked(object sender, EventArgs e)
{
 //Button pressed
}
```

Quelltext 4.6: Xamarin.Forms Event Handler

Steuerelemente um Text zu manipulieren

In Xamarin Forms stehen die folgenden beiden Steuerelemente für die Arbeit mit Texten die folgenden zwei Steuerelemente zur Verfügung. Eine Entry für die Eingabe von einzelnen Textzeilen und einem Editor der für die Eingabe von mehrzeiligen Texten verwendet werden kann. Das entsprechenden Flutter Widget wird in Tabelle 4.5 dargestellt.

Xamarin.Forms Page	**Flutter Widget**
Entry	TextField
Editor	TextField

Tabelle 4.5: Gegenüberstellung textmanipulierender Steuerelemente

Die Gegenüberstllung zeigt, dass es ausschließlich das Textfield Widget gibt, 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 4.7: TextField mit mehreren Zeilen in Flutter

Wertsetztende Steuerelemente

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. 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. Bei Xamarin.Forms-Elementen können Element direkt abgefragt werden , um den Status einer seiner Eigenschaften wie den gesetzt Wert zu ermitteln. Das Abrufen von Informationen in Flutter wird von spezialisierten Widgets gehandhabtd. Wenn Sie ein TextField oder ein TextFormField haben, können Sie einen TextEditingController bereitstellen, um Benutzereingaben abzurufen. Die folgenden Steuerelemente bietet Xamarin.Forms an. Das CheckBox Steuerelement ermöglicht dem Benutzer die Auswahl eines booleschen Wert. Ein Switch der die gleiche Aufgabe wie eine Checkbox übernimmt. Ein Slider bieten Benutzern die Option einen Wert aus einem kontinuierlichen Bereich auswählen. Ein Stepper ermöglicht es dem Benutzer, einen doppelten Wert aus einem Bereich von inkrementellen Werten auszuwählen. DatePicker ermöglicht es dem Benutzer, ein Datum mit der Datumsauswahl der Plattform auszuwählen. TimePicker ermöglicht dem Benutzer die Auswahl einer Zeit mit dem spezifischen TimePicker der Plattform.

Xamarin.Forms Page	**Flutter Widget**
CheckBox	Checkbox
Switch	Switch
Slider	Slider
Stepper	number_inc_dec
DatePicker	TextField mit Funktion
TimePicker	TextField mit Funktion

Tabelle 4.6: Gegenüberstellung wertsetzender Steuerelemente

Wird ähnlich wie der Datepicker nicht als Widget sonders als Funktionalität ausgeliefert, ein mögliche Definition ist in Quelltext 4.8 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 4.8: Verwendung von Timepickern 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. ProgressBar verwendet eine Animation, um zu zeigen, dass die Anwendung durch eine langwierige Aktivität fortschreitet

Xamarin.Forms Page	**Flutter Widget**
ActivityIndicator	CircularProgressIndicator
ProgressBar	LinearProgressIndicator

Tabelle 4.7: Gegenüberstellung aktivitätsandeutender Steuerelemente

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. IndicatorView zeigt Indikatoren an, die die Anzahl der Elemente in einer CarouselView darstellen. Picker zeigt ein ausgewähltes Element aus einer Liste von Textzeichenfolgen an. TableView zeigt eine Liste von Zeilen mit optionalen Überschriften und Unterüberschriften an

4.2.3 Listen

Listen sind ebenfalls Steuerelemente für die Anzeige und Interaktion mit Sammlungen, bekommen jedoch wegen ihrer häufigen Verwendung einen eigenen Abschnitt. In

Xamarin.Forms Page	**Flutter Widget**
CarouselView	carousel_slider
IndicatorView	
Picker	TextView mit Funktionalität
TableView	Table

Tabelle 4.8: Gegenüberstellung sammlungsanzeigender Steuerelemente

Xamarin.Form existieren für diese Darstellung zwei Steuerelemente die ListView und die CollectionView, wobei zweiteres eine Optimierung der Listview ist. Das Äquivalent zu einer ListView in Flutter ist ... eine ListView!

In einer Xamarin.Forms wird bei der Arbeit mit einer ListView Anhand einer angegebenen Vorlage für jede Zeile ermittelt, wie diese dargestellt werden muss. Jede Reihe hat dabei ein ein Ereignis, das ausgelöst wird, wenn eine Reihe ausgewählt wird. Um dieses Verhalten in Flutter abzubilden wird die Touch-Behandlung des Widgets in der Liste bereitgestellt. Eine weitere Art der Interaktion mit einer Liste ist mithilfe der ‚SwipeView‘, die es erlaubt einzelne Reihen zur Seite zu schieben und darunter Schaltflächen sichtbar zu machen.

Damit sich die ListView in Xamarin Forms automatisch aktualisiert, wenn wenn Änderungen in der angezeigten Datensammlung ergeben ist es notwendig die Sammlung in einer ObservableCollection vorzuhalten. In Flutter funktionieren die Dinge ein wenig anders. Wenn die Liste der Widgets innerhalb einer setState()-Methode aktualisiert werden würde, würde auffallen, dass sich die Daten visuell nicht zwangsläufig 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 die ListView mithilfe einer Gleichheitsprüfung analyisisert wird wird festgestellt, dass die beiden ListViews gleich sind. Es hat sich nichts geändert, also ist keine Aktualisierung erforderlich. Eine Möglichkeit, die 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 eignet sich für dynamische Listen oder eine Liste mit sehr großen Datenmengen. Dies ist im Wesentlichen das Äquivalent von RecyclerView unter Android, das automatisch Listenelemente für Sie recycelt. Anstatt eine "ListViewßu erstellen, wird ein ListView.builder mit den folgenden zwei Parametern benötigt: Die anfängliche Länge der Liste und eine ItemBuilder-Funktion. Die ItemBuilder-Funktion nimmt eine Position und gibt die Zeile zurück, die an dieser Position gerendert werden soll.

Tabelle 4.3 zeigt die Steuerelemente und Ihre Gegenstücke aus dem Flutter Framework.

Xamarin.Forms Page	**Flutter Widget**
List	List
CollectionView	List
SwipeView	flutter_slideable

Tabelle 4.9: Gegenüberstellung ereignisauslösende Steuerelemente

4.3 Gesten

Viele Elemente haben wir bereits beschrieben Ereignisse, die bei bestimmten Gesten ausgelöst werden, wie der Button bei einem click. Alternativ kann die Klasse "Gesture-Recognizer"(Deutsch Gestenerkenner) verwendet werden um Benutzerinteraktionen auf Ansichten zu erkennen. Dafür werden das Tippen, das Ziehen, Schwenken, Streichen und Drag und Drop unterstützt. In Flutter gibt es zwei sehr ähnliche Möglichkeiten: Wenn das Widget die Ereigniserkennung unterstützt, kann eine Funktion übergeben werden in der die Geste behandelt wird. Zum Beispiel hat der ElevatedButton einen onPressed-Parameter. Alternativ wenn das Widget keine Ereigniserkennung unterstützt, kann es in einem GestureDetector verschachtelt werden.

4.4 Animation

Gut gestaltete Animationen machen eine Benutzeroberfläche intuitiver, tragen zum eleganten Erscheinungsbild einer ausgefeilten App bei und verbessern das Benutzererlebnis. Daher sind sie bei der Übersetzung von bestehenden Xamarin.Forms Anwendungen von großer Wichtigkeit. Xamarin.Forms enthält eine eigene Animationsinfrastruktur, die für die Erstellung einfacher Animationen unkompliziert ist, aber auch vielseitig genug, um komplexe Animationen zu erstellen. Die Animationsklassen zielen auf verschiedene Eigenschaften von visuellen Elementen ab, wobei eine typische Animation eine Eigenschaft schrittweise von einem Wert zu einem anderen über einen bestimmten Zeitraum ändert. Für Flutter stehen vielen Animationen, insbesondere für Material-Widgets, zur Verfügung und werden mit den in ihrer Design-Spezifikation definierten Standard-Bewegungseffekten geliefert, aber es ist auch möglich, diese Effekte anzupassen.

4.5 Navigation

Die Navigation innerhalb der Anwendung hängt bei Xamarin.Forms hauptsächlich von der verwendeten Page und ihrem Navigationkonzept zusammen. So bietet die 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. Der Navigator arbeitet ähnlich wie die Xamarin.Forms NavigationPage, indem er Routen mit den Methoden push() und pop() aufrufen kann , je nachdem, ob man zu einer Ansicht hin oder von ihr zurück navigieren möchte.

Neben der internen Navigation innerhalb einer Anwendung ist es auch möglich zwischen verschiedenen Apps zu navigieren. Dafür greift Xamarin Forms auf ein bestimmes URI-Schema zurrück. Mit dem Befehl Device.OpenUrl("mailto://") kann so z.B. das Standard E-Mail-Programm des Gerätes geöffnet werden verwenden. In Futter kann für diese Integration das Plugin urllauncher verwendet werden.

4.6 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 Benutzeroberflä-

che 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.

4.7 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.

4.8 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 4.9 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 4.9: Flutter Network request

4.9 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.

4.10 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:

4.11 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 4.10 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 4.10: Flutter Font definition

4.12 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.

4.13 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.

4.14 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.[64] In diesem Paket ist das Settingsplugin, welches die Speicherung von Einstellungen in einem Schlüsselwertspeicher erlaubt.[65] 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.[66]

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[67], oder Realm einer Datenbank optimiert für mobile Endgeräte.[68] Beide Datenbanken stehen auch als Plugin für Flutter zur Verfügung, wobei SQLite ausgereift ist,[69] während Realm erst am 5 November 2020 support für Flutter angekündigt hat und noch nicht offiziell zur Verfügung steht.[70]

[64] Vgl. Microsoft Corporation 2020a, Abgerufen am 17. März 2021.
[65] Vgl. Microsoft Corporation 2019, Abgerufen am 17. März 2021.
[66] Vgl. Google LLC 2020e, Abgerufen am 17. März 2021.
[67] Vgl. SQLite Consortium 2020, Abgerufen am 17. März 2021.
[68] Vgl. MongoDB Inc. 2020, Abgerufen am 17. März 2021.
[69] Vgl. Tekartik 2020, Abgerufen am 17. März 2021.
[70] Vgl. Ward 2020, Abgerufen am 17. März 2021.

5 Unterschiede zwischen C# und Dart

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.[71] Anhand dieses Beitrages solllen in diesem Abschnitt die Unterschiede zwischen den Programmiersprachen aufgedeckt und Anhand von Dart- Quelltext Beispielen veranschaulicht werden.

5.1 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.

```
// no need for ? sign to signal nullable.
int myVariable = null;

// Null checking
if (myVariable != null)

// These work the same way as C#.
myVariable = myVariable ?? 0; // Assigns 0 if null

// Short circuits also work.
myVariable?.toString();
```

Quelltext 5.1: Alles kann ich Dart Null sein

[71] Vgl. Pedley 2019, Abgerufen am 17. März 2021.
[71] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.

5.2 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.[72]

Generics werden in Dart sehr ähnlich behandelt wie in C# , mit der Ausnahme, dass Sie eine generische Klasse ohne die Type-Beschränkung übergeben können.[73] Quelltext 5.2 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 5.2: Generics in Dart

5.3 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.[74]

[72] Vgl. Microsoft Corporation 2015b, Abgerufen am 17. März 2021.
[73] Vgl. Cheng 2019, S. 98.
[73] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.
[74] Vgl. Microsoft Corporation 2015a, Abgerufen am 17. März 2021.

In Dart kann der Typ Typedef verwendet werden, um eine Methodensignatur zu definieren und eine Instanz davon in einer Variablen zu halten.[75] Dies wird in Quelltext 5.3 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 5.3: Delegates in Dart

5.4 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 5.4: Optinales "New"Keyword in Dart

[75] Vgl. Pedley 2019, Abgerufen am 17. März 2021.
[75] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.
[75] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.

5.5 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 als Listen (Eng. List) bezeichnet.[76] Dart hat ebenfalls Listen die unter dem gleichen Namen zur Verfügung stehen und die gleichen Funktionalitäten abbilden.[77] Listen sind Listen, und Dictionaries sind Maps.

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

Quelltext 5.5: Listen und Maps in Dart

5.6 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 5.6 als Quelltext dargestellt.

```
class _PrivateClass {
  String _privateField;
}

class PublicClass {
  String publicField;
}
```

Quelltext 5.6: 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

[76] Vgl. Stellman und Greene 2021, S. 413.
[77] Vgl. Meiller 2020, S. 12f.
[77] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.
[77] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.

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.

5.7 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.[78]

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.[79]

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

[78] Vgl. veerbung Google LLC 2020e, Abgerufen am 17. März 2021.
[79] Vgl. interface (C# -Referenz) Google LLC 2020e, Abgerufen am 17. März 2021.

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

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

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

Quelltext 5.7: 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.

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

class MyClass with MyMixin
{
}
```

Quelltext 5.8: Mixin's in Dart

5.8 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

[79] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.
[79] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.

Namespaces und erspart dem Programmierer die Angabe eines vollqualifizierten Namens bei jedem Verwenden einer enthaltenen Methode.[80]

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

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

Quelltext 5.9: Importieren von Paketen in Dart

5.9 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.[81]

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

[80] Vgl. Verwenden von Namespaces (C# -Programmierhandbuch) Google LLC 2020e, Abgerufen am 17. März 2021.
[80] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.
[81] Vgl. .NET-Klassenbibliotheken Google LLC 2020e, Abgerufen am 17. März 2021.

stellen, nichts anderes wird benötigt.

5.10 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.[82]

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

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

Quelltext 5.10: Async und Await in Dart

5.11 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

[82] Vgl. MICROSFT Async await Google LLC 2020e, Abgerufen am 17. März 2021.
[82] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.

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.[83]

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.

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

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

  streamController.add("hello");

  streamController.close();
```

Quelltext 5.11: Events in Dart

[83] Vgl. MICROSFT Events Google LLC 2020e, Abgerufen am 17. März 2021.
[83] In Anlehnung an Pedley 2019, Abgerufen am 17. März 2021.

6 Realisierung

6.1 Umgebung

Für die Realisierung und Verwendung des Compilers ist es notwendig verschiedene Softwarekomponenten erforderlich, die an dieser Stelle mit Ihrer Funktion definiert werde sollen.

- Windows 10:[84] Windows wird als Platform für den Übersetzer zwangsläufig benötigt, da der Roslyn Compiler ausschließlich für dieses Betriebssystem zur Verfügung steht.
- Visual Studio 2019[85]: Die Entwicklungsumgebung für die Entwicklung des Übersetzers. Mit Hilfe von Visual Studio kann ein Projekt angelegt werden, welches den Roslyn Compiler verwenden kann. Darüber hinaus kann mit Visual Studio eine Xamarin.Forms Anwendung entwickelt werden, die für die spätere Qualitätssicherung des Übersetzers essentiell ist.
- Build tools for Visual Studio:[86] Beinhaltet MSBuild als teil der Build-Tool, das hilft, den Prozess der Erstellung eines Softwareprodukts zu automatisieren, einschließlich der Kompilierung des Quellcodes, der Paketierung, des Testens, der Bereitstellung und der Erstellung von Dokumentationen. Mit MSBuild ist es möglich, Visual Studio-Projekte und -Lösungen zu erstellen, ohne dass die Visual Studio IDE installiert ist.
- Android Studio[87] oder Visual Studio Code[88]: Dabei handelt es sich um die Entwicklungsumgebungen für die Entwicklung von Flutter. Für beide Umgebungen

[84] Windows 10 ist über `https://www.microsoft.com/de-de/software-download/windows10ISO` zum download verfügbar.
[85] Visual Studio 2019 ist über `https://visualstudio.microsoft.com/de/downloads/` zum download verfügbar.
[86] Build tools for Visual Studio ist über `https://visualstudio.microsoft.com/de/downloads/` zum download verfügbar.
[87] Android Studio ist über `https://developer.android.com/studio` zum download verfügbar.
[88] Visual Studio Code ist über `https://code.visualstudio.com/` zum download verfügbar.

steht jeweils eine Erweiterung für die Programmiersprache Dart und das Flutter Framework bereit, die für die Arbeit mit Flutter Apps notwendig sind. Im Rahmen dieser Arbeit ist eine dieser Umgebungen notwendig um die übersetzte Anwendung auszuführen.

- Flutter SDK: [89] Das Flutter Software Development Kit (SDK)enthält die Pakete und Kommandozeilen-Tools, die Sie für die plattformübergreifende Entwicklung von Flutter-Apps benötigt werden.

In der Zukunft könnte der Source to Source Compiler auch eine Web-Oberfläche bekommen, diese hätte die zwei folgenden essentielle Vorteile im Gegensatz zu der aktuellen Implementierung. Reduktion des Installationsaufwandes - durch den Betrieb über eine Webseite könnte die Installation von der Anzeige entkoppelt werden. Natürlich wäre eine Client-Server Struktur auch ohne eine Webseite erreichbar, jedoch haben Webseiten darüber hinaus den zusätzlichen Vorteil, dass sie Plattform-unabhängig zur Verfügung stehen, was es zum Beispiel für Xamarin.Forms Entwickelr mit einem Mac OSX Computer erlauben würde ebenfalls von dem Compiler zu profitieren, ohne sich eine Windows Installation vornehmen zu müssen

6.2 Projekterstellung

Mit Hilfe von Roslyn konnte der Name des Projektes extrahiert werden. Dieser Name kann anschließend verwendet werden um ein neues Flutter Projekt zu initialisieren.

Wie der Quelltext zeigt, wird dafür ein neues Projekt mit Hilfe der Kommandozeile angelegt. Damit das Ergebnis nicht von voherigen Übersetzungsvorgängen beeinträchtigt wird, auch im Vorhinein überprüft, ob das Zielverzeichnis existiert und leer ist.

6.3 Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche wurde mit dem Windows Presentation Foundation (WPF) Framework realisiert. Da die Benutzeroberfläche nur auf Windows Computern lauffähig seien muss, auf denen die notwendige Software installiert ist.

[89] Die Flutter SDK ist über `https://flutter.dev/docs/get-started/install/windows` zum download verfügbar.

7 Qualitätssicherung

Nach der erfolgreichen Implementierung des Übersetzers soll mit Hilfe einer zu testenden mobilen Anwendung überprüft werden, ob der Compiler so arbeitet wie zu erwarten ist.

7.1 Testfälle

Um die Anwendung vollständig Testen zu können, ist es notwendig Testfälle zu definieren anhand welcher die Soll-Situation mit der Ist-Situation verglichen werden kann. Zu diesem Zweck werden in diesem Abschnitt Testfälle

Tabelle **??** zeigt die Testfälle für die Metadaten der mobilen Anwendung.

7.2 Testobjekt

Anhand dieser Testfälle kann anschließend ein Xamarin.Forms Projekt erstellt werden, anhand welcher der Source-To-Source Compiler getestet werden kann. Im folgenden werden die Funktionalitäten der mobilen Anwendung dargelegt und mit Hilfe von iOS Screenshots dargestellt, die entsprechenden Android Screenshots werden der Vollständigkeitsannahme in **??** dargestellt.

Bei der Anwendung sind in einem ersten Schritt die Metadaten der Anwendung relevant. Dazu gehören der Name der Anwendung sowie das Anwendungsicon - außerdem sollten die SDK Version der mobilen Anwendung nicht modifiziert werden. Nach dem Start der Anwendung wird eine Menustruktur angezeigt, über den verschiedene Bereiche der Anwendung angesteuert werden können. In Abbildung 7.4 werden Screenshots der mobilen Anwendung gezeigt.

Testfall-ID	Komponente	Beschreibung
1	App-Icon	Prüfen ob das App-Icon übernommen wurde
2	App-Name	Prüfen ob das App-Name übernommen wurde
3	SDK Versionen	Prüfen ob die SDK Versionen übernommen wurden
4	Seitenname	In der Navigationsleiste wird der Name der aktuellen Seite angezeigt
5	Navigation	Mit Hilfe des Menüs kann navigiert werden
6	Zurück Navigation	Über die Navigationsleiste kann zurrück Navigiert werden
7	Gyroscope auswerten	Die Werte werden in der App angezeigt.
8	Accelerometer auswerten	Die Werte werden in der App angezeigt.
9	Compass auswerten	Die Werte werden in der App angezeigt.
10	Magnetometer auswerten	Die Werte werden in der App angezeigt.
11	Sensor nicht verfügbar	Wenn ein Sensor nicht verfügbar ist, wird ein Fehler angezeigt
12	Steuerelemente wurden ausgetauscht	Alle Steuerelemente werden angezeigt
13	Ereignisse funktionieren	Steuerelemente reagieren wie gewohnt
14	Bild aus Ressourcen laden	Ein Bild aus den Ressourcen wird in der App angezeigt
15	Bild aus dem Web laden	Ein Bild aus dem Internet wird in der App

Tabelle 7.1: Testfälle der Test App

Über dieses Menu kann der Anwender zu verschiedenen Seiten navigieren, die folgend kurz erläutert werden sollen. Die in Abbildung 7.2 zeigt die Seite mit den verfügbaren Steuerelementen von Xamarin.Forms, wie sie auch in Kapitel 3 beschrieben wurden. Dazu gehören Ansichten für die Präsentation, für Interaktionen, zum setzen von Werten, für die Manipulation von Texten und für die Anzeige von aktuellen Interaktionen.

Die folgende Option Sensoren öffnet eine Ansicht, auf der die aktuellen Werte von den

Abbildung 7.1: Test Objekt Screenshots I

Abbildung 7.2: Test Objekt Screenshots II

Smartphone-Sensoren ausgegeben werden. Diese Sensoren werden von vielen mobilen Anwendungen verwendet. Dazu gehören der Beschleunigungssensor, der Compass, das Gyroscope und das Magnetometer. Anhand dieser Seite, soll sichergestellt werden, dass

der Compiler auch die Funktionalität dieser Sensoren Abbilden kann.

Abbildung 7.3: Test Objekt Screenshots III

Ebenfalls werden in dem Testprojekt eine Liste und ein Carousel abgebildet, da es sich bei diesen um häufig verwendete Elemente handelt - die in vielen mobilen Anwendungen verwendet wird.

7.3 Testablauf

7.4 Testauswertung

Abbildung 7.4: Test Objekt Screenshots IV

8 Fazit und Ausblick

Flutter immernoch eine Abhängigkeit

8.1 Ausblick

Die Installation der notwendigen Software i

Literaturverzeichnis

Gedruckte Quellen

Albahari, Joseph und Eric Johannsen (2020). *C# in a Nutshell. The Definitive Reference*. Sebastopol: O'Reilly Media Inc.

Balzert, Helmut (2011). *Lehrbuch der Softwaretechnik. Entwurf, Implementierung, Installation und Betrieb*. Heidelberg: Spektrum Akademischer Verlag.

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

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.

Petzold, Charles (2016). *Creating mobile Apps with Xamarin.Forms. Cross.platform C# programming for iOS, Android and Windows*. 1. Aufl. Redmond: Microsoft Press.

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.

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.

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.

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.

Online Quellen

Google LLC (2020a). *Canvas class*. URL: `https://api.flutter.dev/flutter/dart-ui/Canvas-class.html` (besucht am 27. 02. 2021).

– (2020b). *Flutter for Xamarin.Forms developers*. URL: `https://flutter.dev/docs/get-started/flutter-for/xamarin-forms-devs` (besucht am 27. 02. 2021).

– (2020c). *GridView class*. URL: `https://api.flutter.dev/flutter/widgets/GridView-class.html` (besucht am 27. 02. 2021).

– (2020d). *Introduction to widgets*. URL: `https://flutter.dev/docs/development/ui/widgets-intro` (besucht am 27. 02. 2021).

– (2020e). *Shared preferences plugin*. URL: `https://pub.dev/packages/shared_preferences` (besucht am 27. 02. 2021).

– (2020f). *Widget catalog*. URL: `https://flutter.dev/docs/development/ui/widgets` (besucht am 27. 02. 2021).

– (2020g). *Work with tabs*. URL: `https://flutter.dev/docs/cookbook/design/tabs` (besucht am 27. 02. 2021).

Google LLC (2020h). *Write your first Flutter app, part 1*. URL: https://flutter.dev/docs/get-started/codelab (besucht am 27. 02. 2021).

Hunter, Scott (2020). *Introducing .NET Multi-platform App UI*. URL: https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/ (besucht am 27. 02. 2021).

Microsoft Corporation (2015a). *Delegaten (C#-Programmierhandbuch)*. URL: https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/delegates/ (besucht am 27. 02. 2021).

– (2015b). *Generics C# – Programmierhandbuch*. URL: https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/generics/ (besucht am 27. 02. 2021).

– (2016). *Xamarin.Forms Pages*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/pages (besucht am 27. 02. 2021).

– (2017). *Grundlagen zu XAML*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/xaml/xaml-basics/ (besucht am 27. 02. 2021).

– (2018). *Xamarin.Forms Layouts*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/layouts (besucht am 27. 02. 2021).

– (2019). *Xamarin.Essentials Einstellungen*. URL: https://docs.microsoft.com/de-de/xamarin/essentials/preferences (besucht am 27. 02. 2021).

– (2020a). *Xamarin.Essentials*. URL: https://docs.microsoft.com/de-de/xamarin/essentials/ (besucht am 27. 02. 2021).

– (2020b). *Xamarin.Forms Ansichten*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/controls/views (besucht am 27. 02. 2021).

– (2020c). *Xamarin.Forms FlyoutPage*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/app-fundamentals/navigation/flyoutpage (besucht am 27. 02. 2021).

– (2020d). *Xamarin.Forms TabbedPage*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/app-fundamentals/navigation/tabbed-page (besucht am 27. 02. 2021).

MongoDB Inc. (2020). *Realm Mobile Database*. URL: https://www.mongodb.com/realm/mobile/database (besucht am 27. 02. 2021).

Pedley, Adam (2019). *Moving From C# To Dart: Quick Start*. URL: https://buildflutter.com/moving-from-csharp-to-dart-quick-start/ (besucht am 27. 02. 2021).

Ritscher, Walt (2020). *Arranging Views with Xamarin.Forms Layout*. URL: https://bit.ly/34ulgpU (besucht am 27. 02. 2021).

Sells, Chris (2021). *What's New in Flutter 2*. URL: https://medium.com/flutter/whats-new-in-flutter-2-0-fe8e95ecc65 (besucht am 27. 02. 2021).

Sneath, Tim (2020). *Flutter Spring 2020 Update*. URL: https://medium.com/flutter/flutter-spring-2020-update-f723d898d7af (besucht am 27. 02. 2021).

SQLite Consortium (2020). *About SQLite*. URL: https://www.sqlite.org/about.html (besucht am 27. 02. 2021).

Stack Exchange Inc. (2019). *Developer Survey Results 2019*. URL: https://insights.stackoverflow.com/survey/2019 (besucht am 27. 02. 2021).

– (2020). *Developer Survey Results 2020*. URL: https://insights.stackoverflow.com/survey/2020 (besucht am 27. 02. 2021).

Tekartik (2020). *sqflite*. URL: https://pub.dev/packages/sqflite (besucht am 27. 02. 2021).

Ward, Ian (2020). *Realm Mobile Database*. URL: https://github.com/realm/realm-object-server/issues/55 (besucht am 27. 02. 2021).

Anhang I

Xamarin.Forms Page	**Flutter Widget**
ContentPage	
FlyOutPage	MasterDetailScaffold
NavigationPage	Scaffold
TabbedPage	TabBar und TabBarView
AbsolutLayout	Positioned
ContentView	StatelessWidget
Frame	BoxDecoration
Grid	GridView oder Stack für das Stapeln von Elementen
ScrollView	SingleChildScrollView
StackLayout	Row und Column
ReleativLayout	Positioned
BoxView	SizedBox
Image	Image
Label	Text
Map	Leamaps oder Google Maps
WebView	webview_flutter
Ellipse	CustomPaint
Linie	CustomPaint
Path	CustomPaint
Polygon	CustomPaint
Polyline und Rectangle	CustomPaint
Rectangle	CustomPaint
Button	Flatbutton
ImageButton	IconButton
RadioButton	RadioButton
RefreshView	pull_to_refresh
SearchBar	flutter_search_bar
SwipeView	flutter_slideable
Entry	TextField
Editor	TextField
CheckBox	Checkbox
Switch	Switch
Slider	Slider
Stepper	number_inc_dec
DatePicker	TextField mit Funktion
TimePicker	TextField mit Funktion
ActivityIndicator	CircularProgressIndicator
ProgressBar	LinearProgressIndicator
CarouselView	carousel_slider
IndicatorView	
Picker	TextView mit Funktionalität

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)