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

28. März 2021

Zusammenfassung

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API Application programming interface

GUI Graphical user interface

OOP Objektorientierte Programmierung

S2S Source-to-Source

UI User Interface

URI Uniform Resource Identifier

WPF Windows Presentation Foundation

XAML Extensible Application Markup Language

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. Die Unterschiede zwischen den Programmiersprachen werden in Kapitel 5 behandelt. 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.

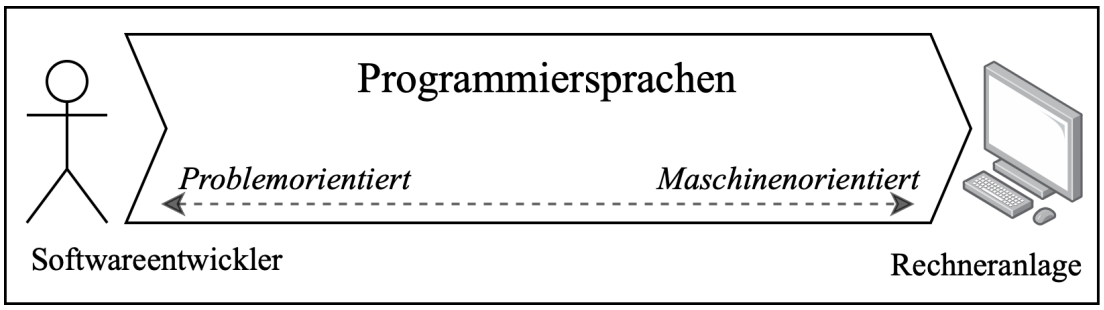

Abbildung 2.1: Programmiersprachen als Schnittstelle

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]

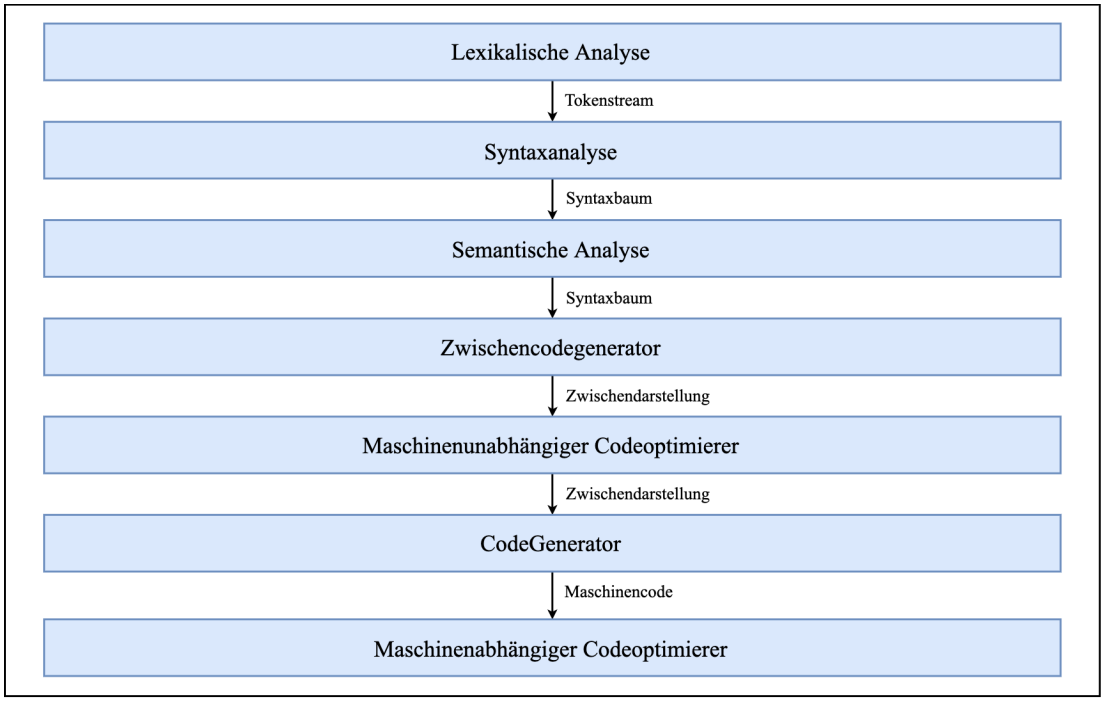

Abbildung 2.2: Phasen eines Compilers[22]

[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.

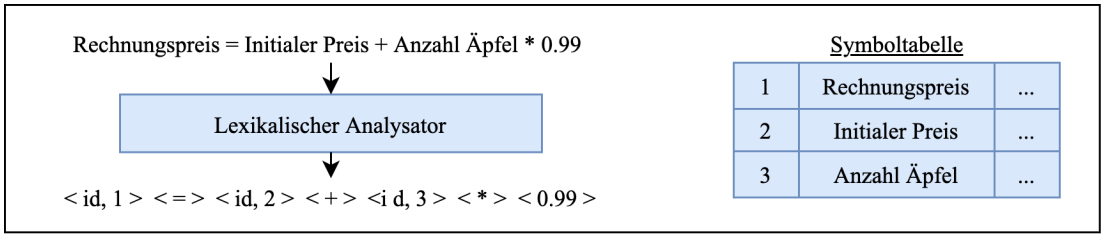

Abbildung 2.3: Lexer Beispiel[25]

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.

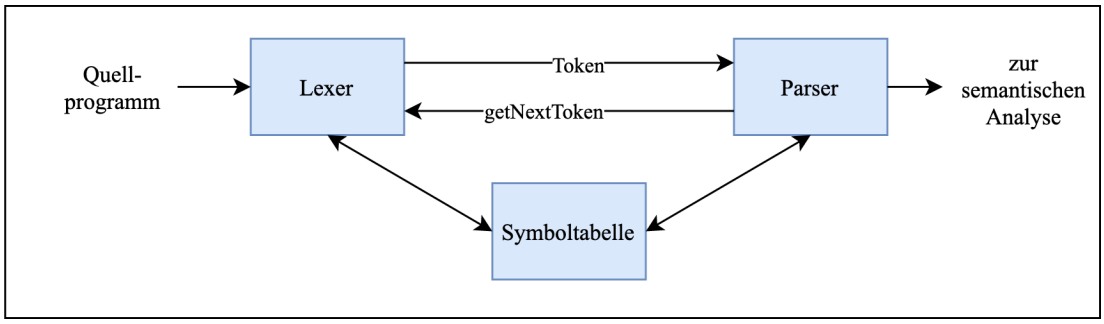

Abbildung 2.4: Interaktion zwischen Lexer und Parser[27]

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.

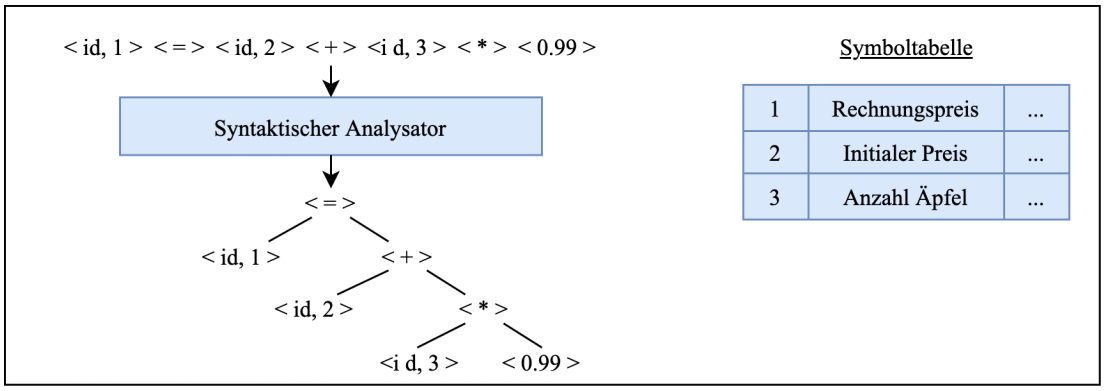

Abbildung 2.5: Syntaxbaum[32]

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]

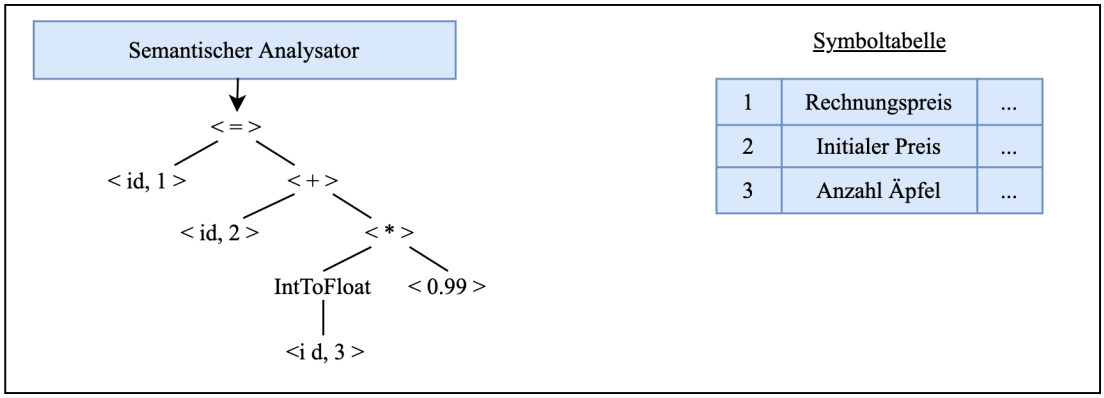

Abbildung 2.6: Typüberprüfung[35]

[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

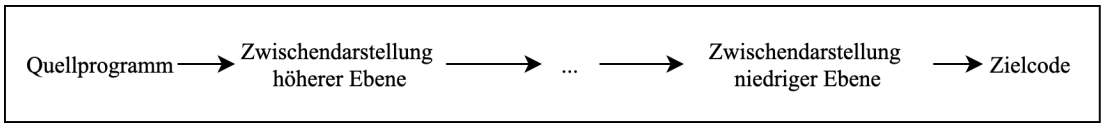

Abbildung 2.7: Zwischendarstellungen[36]

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.

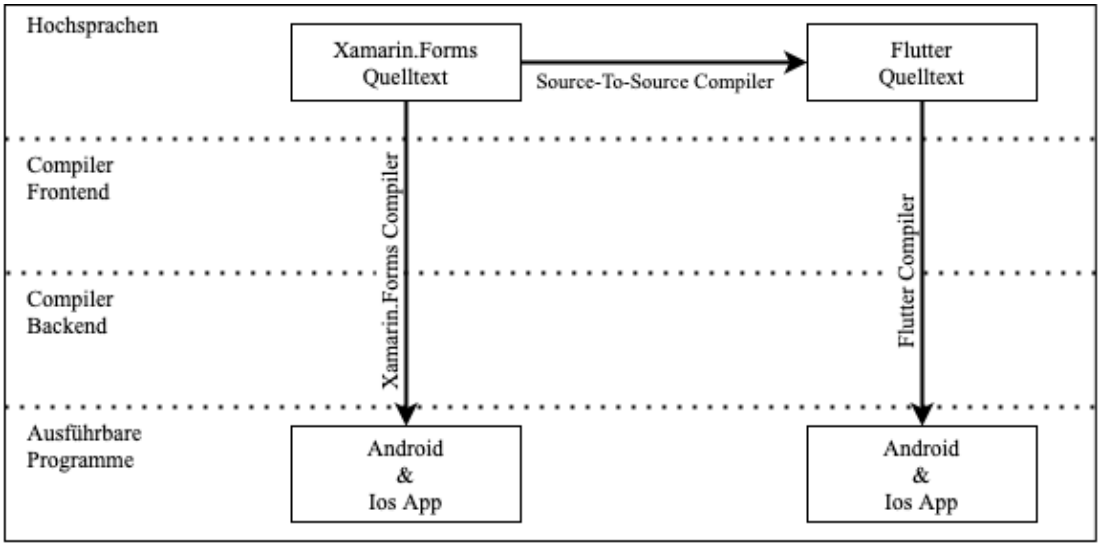

Abbildung 3.1: Umfeld des Source-To-Source Compilerns

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.

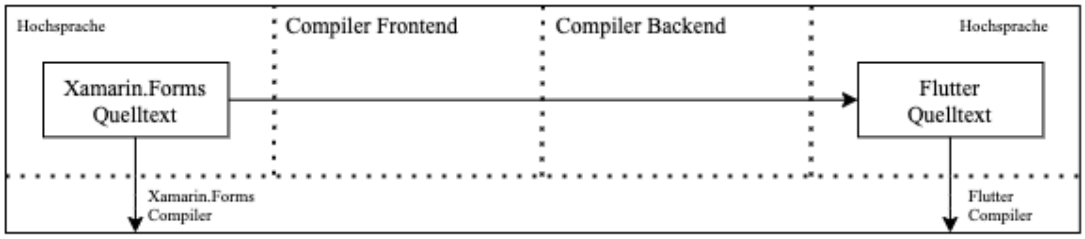

Abbildung 3.2: Source-To-Source Compiler Aufbau

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.
- User Interface (UI): Für die Entwicklung von Benutzeroberflächen kann die Programmiersprache C# verwendet werden, jedoch hat die Alternative Extensible Application Markup Language (XAML) für Entwickler Vorteile,[43] weswegen die Konstruktion von Benutzeroberflächen mit C# in dieser Arbeit nicht berücksichtigt wird.

[43] Vgl. Microsoft Corporation 2017b, Abgerufen am 28. 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.

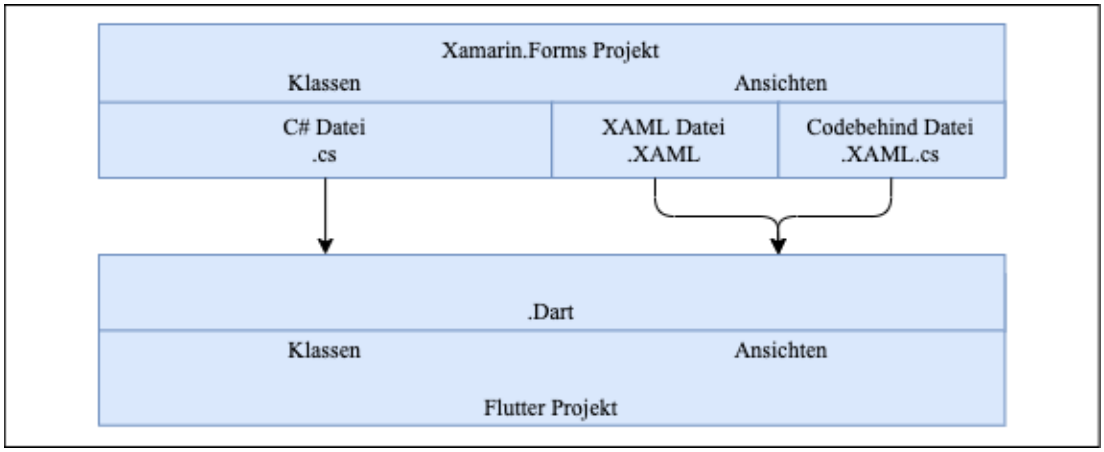

Abbildung 3.3: Compiler-Struktur

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 Graphical user interface (GUI).

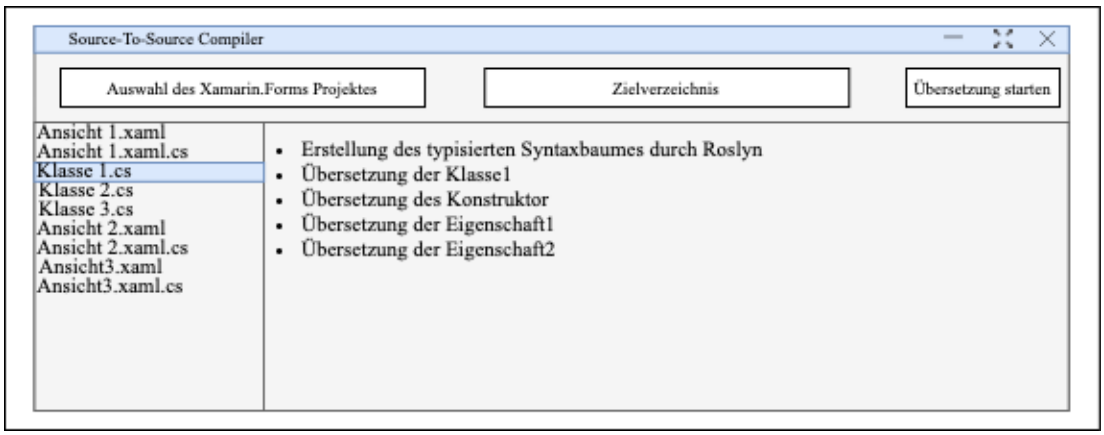

Abbildung 3.4: Mockup der grafischen Oberfläche

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 weist 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 sogenannte Lösung bei Xamarin.Forms aus mehreren Projekten zusammen. Es gibt für jede Plattform ein dediziertes Projekt, das 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 weiteren Projekt hinterlegt und lassen sich folglich migrieren.

4.1.1 Metadaten

Zu den Metadaten einer Anwendungen gehören unter anderem der Name der Anwendung, Informationen wie die benötigte Betriebssystemversion, das ‚App Launcher Icon', das auf dem Smartphonebildschirm angezeigt wird und die Berechtigungen, die von

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

der mobile Anwendung während der Ausführung beantragt werden können. Diese Eigenschaften werden bei Xamarin.Forms innerhalb der nativen Projekte verwaltet. In iOS können Änderungen mittels der Datei ‚Info.plist' definiert werden.[46] Android speichert Metadaten innerhalb der ‚AndroidManifest.xml'.[47] Flutter verwendet für die Verwaltung der Metadaten die identischen Dateien, daher ist es möglich, diese aus dem Xamarin.Forms Projekt zu kopieren und innerhalb des Flutter Projektes zu sichern. Dafür muss die ‚Info.plist'im Verzeichnis ‚ios/Runner/' und die ‚AndroidManifest.xml' in ‚android/app/src/main/' gespeichert werden. In diesen Dateien wird außerdem der Identifizierer der Anwendung definiert. Durch eine Kopie der Konfigurationsdatei und Erhöhung der Versionsnummer wird eine spätere Kompilierung der Flutter-App demnach als eine Aktualisierung der Xamarin.Forms App erkannt.[48]

4.1.2 Bilder und Startbildschirm

Innerhalb von Apps können Bilder das Benutzererlebnis verbessern und helfen, eine Aktion zu veranschaulichen oder komplexe Botschaften zu verdeutlichen.[49] In iOS und Android werden sie in verschiedenen Auflösungen bereit gestellt. Das Betriebssystem wählt während der Laufzeit die beste Ressource basierend auf den Eigenschaften des Smartphonedisplays aus. Xamarin.Forms verwendet die Application programming interfaces (APIs) der nativen Plattformen zum Laden lokaler Bilder und unterstützt daher die plattformspezifischen Funktionalitäten.[50] Zur Verwendung nativer Bilddateien müssen die Bilder in Xamarin.Forms zu jedem Anwendungsprojekt hinzugefügt werden und vom gemeinsamen Xamarin.Forms-Code referenziert werden. Flutter verwendet im Gegensatz zu Xamarin.Forms keine nativen APIs, sondern ein einfaches dichte-basiertes Format, ähnlich dem von iOS. Für die Anzeige von Bildern arbeitet Flutter mit sogenannten logischen Pixeln. Alle Bilderressourcen können sich in einem beliebigen Ordner innerhalb des Projektes befinden, da Flutter keine vordefinierte Ordnerstrukturen hat.[51] Der Startbildschirm (engl. Splashscreen) ist der Einstiegspunkt der mobilen App. Er dient als Ladebildschrim und wird bei Xamarin.Forms in den plattformspezifischen Projekten gespeichert und kann ähnlich wie die ‚Info.plist'- und die ‚AndroidManifest.xml'-Dateien in das Flutter Projekt kopiert werden, da die technische Implementierung identisch ist.[52]

[46] Vgl. Microsoft Corporation 2017d, Abgerufen am 28. März 2021.
[47] Vgl. Microsoft Corporation 2018a, Abgerufen am 28. März 2021.
[48] Vgl. Vaibhavi 2020, Abgerufen am 28. März 2021.
[49] Vgl. Google LLC 2020h, Abgerufen am 28. März 2021.
[50] Vgl. Microsoft Corporation 2020c, Abgerufen am 28. März 2021.
[51] Vgl. Google LLC 2020c, Abgerufen am 28. März 2021.
[52] Vgl. Google LLC 2020b, Abgerufen am 28. März 2021.

4.1.3 Benutzerdefinierte Schriftarten

Eine Schriftart (engl. Font) wird verwendet, um das Design und den Inhalt so klar und effizient wie möglich darzustellen, dafür haben Android und iOS eine Vielzahl vorinstalliert. Wenn eine mobile App jedoch auf eine benutzerdefinierte Schriftart zurückgreifen möchte muss diese mit der Anwendung mitgeliefert werden. In Xamarin.Forms mussten Fonts bis zum Jahre 2020 in jedem nativen Projekt referenziert werden. Seit der Version 4.5 können diese Dateien auch plattformübergreifend verwendet werden und befinden sich daher innerhalb des geteilten Projekts.[53] In Flutter werden die Fonts wie in der neueren Version von Xamarin.Forms in einem Ordner abgelegt.[54] Eine Verwendung des Schriftsatzes ‚MyCustomFont' in Flutter wird in Quelltext 4.1 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 4.1: Flutter Verwendung von Schriftsätzen[55]

4.1.4 Plattformspezifischer Quelltext

In den Ausschlusskriterien des dritten Kapitels wurde der plattformspezifische Quelltext von Xamarin.Forms für die Übersetzung exkludiert, der Vollständigkeit halber sollen die Unterschiede zwischen den Plattformen jedoch trotzdem erwähnt werden. Xamarin.Forms unterteilt nativen Quelltext in zwei Kategorien, den sogenannten ‚DependencyServices' die es erlauben, Funktionenalitäten in den Plattformprojekten aus dem geteilten Code aufzurufen,[56] sowie ‚Custom Renderers', die es ermöglichen das Aussehen und Verhalten von Steuerlementen anzupassen.[57] In Flutter wird für die

[53] Vgl. Versluis 2020, Abgerufen am 28. März 2021.
[54] Vgl. Google LLC 2020o, Abgerufen am 28. März 2021.
[55] Quelltext in Anlehnung an Google LLC 2020o, Abgerufen am 28. März 2021.
[56] Vgl. Microsoft Corporation 2019c, Abgerufen am 28. März 2021.
[57] Vgl. Microsoft Corporation 2019b, Abgerufen am 28. März 2021.

Ausführung von plattformspezifischen Funktionalitäten auf ‚platform-channels' zurückgegriffen.[58] Für ‚Custom Renderer' gibt es bei keine Alternative, da das Framework keine nativen Steuerelemente verwendet sondern Widgets anzeigt.

4.2 Erweiterungen

Erweiterungen sind Programmergänzungen, die auch von externen Entwicklern beigetragen werden können. Sie dienen der Reduzierung des Entwicklungsaufwandes, da nicht jede Funktionalität eigens implementiert werden muss. Im .NET-Ökosystem können Xamarin.Forms-Projekte auf das Paketverwaltungssystem Nuget zugreifen, um Erweiterungen zu einer App hinzuzufügen.[59] Bei Flutter wird für diesen Fall mit der pubspec.yaml Datei eine Referenz auf das ausgewählte Plugin gesetzt. Dabei werden in Dart geschriebene Pakete von plattformunabhängigen Plugins unterschieden. Diese Plugins können für Android (mit Kotlin oder Java) oder für iOS (mit Swift oder Objective-C) geschrieben sein.[60] In Quelltext 4.2 wird ein Auschnitt der pubspec.yaml Datei gezeigt, in welcher das in Unterabschnitt 4.2.3 erwähnte Plugin ‚url_launcher' geladen wird.

```
1 dependencies:
2   flutter:
3     sdk: flutter
4   url_launcher: '>=5.4.0 <6.0.0'
```

Quelltext 4.2: Erweiterungen in Flutter[61]

In dem Beispiel muss das Plugin mindestens die Version 5.4 haben, darf jedoch die Version 6 nicht überschreiten. Aufgrund der Vielzahl an existierenden Erweiterungen für beide Frameworks ist es nicht möglich eine vollständige Gegenüberstellung zu erstellen. Im Folgenden wird jedoch anhand von drei gängigen Anwendungsszenarien der Einsatz von Erweiterungen erläutert.

4.2.1 Interaktion mit der Hardware

Android und iOS nutzen einzigartige Betriebssystem- und Plattform-APIs, auf die Xamarin.Forms Entwickler zugreifen können. Mit der Erweiterung Xamarin.Essentials

[58] Vgl. Google LLC 2020t, Abgerufen am 28. März 2021.
[59] Vgl. Microsoft Corporation 2020a, Abgerufen am 28. März 2021.
[60] Vgl. Google LLC 2020p, Abgerufen am 28. März 2021.
[61] Quelltext in Anlehnung an Google LLC 2020p, Abgerufen am 28. März 2021.

bietet Microsoft eine plattformübergreifende API, auf die von gemeinsamem Code aus zugegriffen und die direkt auf der Plattform ausgeführt werden kann. Der Dart Quelltext, aus dem eine Flutter-App besteht, wird nicht auf der zugrundeliegenden Plattform, sondern nativ auf dem Gerät ausgeführt. Es werden also nicht die iOS oder Android APIs, benutzt. Flutter Apps können über Plattformkanäle jedoch auch mit den nativen APIs interagieren, um beispielsweise Daten von Sensoren des Gerätes abzurufen.[62]

4.2.2 Speicherung von Daten

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 gespeichert werden müssen.

Für die Speicherung in einer Datenbank können Xamarin.Forms Entwickler auf verschiedene Lösungsansätze zurückgreifen. Zum einen ‚SQLite', die am häufigsten verwendete Datenbank-Engine der Welt[63], oder ‚Realm', eine Datenbank optimiert für mobile Endgeräte.[64] Beide Datenbanken stehen auch als Plugin für ‚Flutter' zur Verfügung, wobei SQLite ausgereift ist,[65] während ‚Realm' erst am 5 November 2020 Support für Flutter angekündigt hat und noch nicht offiziell zur Verfügung steht.[66]

Die Xamarin.Essentials Erweiterung stellt Entwicklern das ‚Settingsplugin' bereit, welches die Sicherung von Präferenzen und App-Einstellungen erlaubt.[67] In Flutter wird für diese Speicherung mit Hilfe des Plugins ‚shared_preferences' auf die gleichen plattformspezifischen API zugegriffen.[68] Die Verwendung der gleichen APIs ist ein wichtiger Faktor, da die von Anwendern gespeicherten Daten, auch nach einem Frameworkwechsel zu Flutter, noch verfügbar sind.

[62] Vgl. Google LLC 2020t, Abgerufen am 28. März 2021.
[63] Vgl. SQLite Consortium 2020, Abgerufen am 28. März 2021.
[64] Vgl. MongoDB Inc. 2020, Abgerufen am 28. März 2021.
[65] Vgl. Tekartik 2020, Abgerufen am 28. März 2021.
[66] Vgl. Ward 2020, Abgerufen am 28. März 2021.
[67] Vgl. Microsoft Corporation 2019a, Abgerufen am 28. März 2021.
[68] Vgl. Google LLC 2020l, Abgerufen am 28. März 2021.

4.2.3 Navigation zu anderen Anwendungen

Mobile Anwendungen können die Möglichkeit zur Navigation zu anderen Apps realisieren. Dafür greift Xamarin Forms auf ein bestimmtes Uniform Resource Identifier (URI)-Schema zurück, mit dem Ressourcen eindeutig bezeichnet werden. Durch den Befehl ‚Launcher.OpenAsync("mailto://")', der Bestandteil der Xamarin.Essentials Erweiterung ist, wird z.B. das Standardprogramm für E-Mails des Smartphones gestartet.[69] Mithilfe des Plugins ‚url_launcher' kann die Funktionalität zum Öffnen von anderen Anwendungen zu Flutter Apps hinzugefügt werden.[70] Die unterstützten URI-Schemata und ihre Aktionen werden in Tabelle 4.1 dargestellt.

URI-Schemata	**Aktion**
http:<URL>	URL wird im Standardbrowser geöffnet
mailto:<E-Mail Adresse>	Erstellt eine Email an die angegebene E-Mail Adresse
tel:<Telefonnummer>	Ruft die Rufnummer an
sms:<Telefonnummer>	Schreibt eine SMS an die Rufnummer

Tabelle 4.1: Unterstützte Schemata des ‚url_launcher' Plugins

4.3 Lebenzyklus

Durch das Navigieren zu anderen Anwendungen ändert sich der Status von Ausführung im Vordergrund zu Ausführung im Hintergrund. Der aktuelle Status der App bestimmt die Handlungsoptionen. Hat eine Vordergrund App die Aufmerksamkeit des Benutzers, ist sie priorisiert beim Zugriff auf die Systemressourcen, einschließlich der CPU. Gleichzeitig muss eine Hintergrund-App möglichst inaktiv sein, da sie sich außerhalb des Bildschirms befindet. Mobile Anwendungen müssen auf Änderungen des Lebenszyklus-Status reagieren können, um ihr Verhalten entsprechend anzupassen.[71] Xamarin.Forms bietet dafür die drei Methoden ‚OnStart' , ‚OnResume' und ‚OnSleep' an, die aufgerufen werden wenn sich der Status verändert.[72] Bei Flutter kann auf die ‚didChangeAppLifecycleState' Methode zurückgegriffen werden, die bei Änderungsereignissen ausgelöst wird und die ebenfalls die drei Lebenszyklen beinhaltet.[73]

[69] Vgl. Microsoft Corporation 2020d, Abgerufen am 28. März 2021.
[70] Vgl. Google LLC 2020n, Abgerufen am 28. März 2021.
[71] Vgl. Apple Inc. 2020, Abgerufen am 28. März 2021.
[72] Vgl. Microsoft Corporation 2020i, Abgerufen am 28. März 2021.
[73] Vgl. Google LLC 2020e, Abgerufen am 28. März 2021.

4.4 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 zuständig, Layouts beinhalten eine Sammlung von Ansichten und sind für ihre visuelle Anordung auf der Benutzeroberfläche verantwortlich.[74] Im folgenden Abschnitt werden Xamarin.Forms und Flutter-Ansichten gegenübergestellt. Damit soll ein Konzept für die Übersetzung von grafischen Benutzeroberflächen gelegt und die Möglichkeiten des Austausches von Ansichten für den Compilers validiert werden.

4.4.1 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.[75]

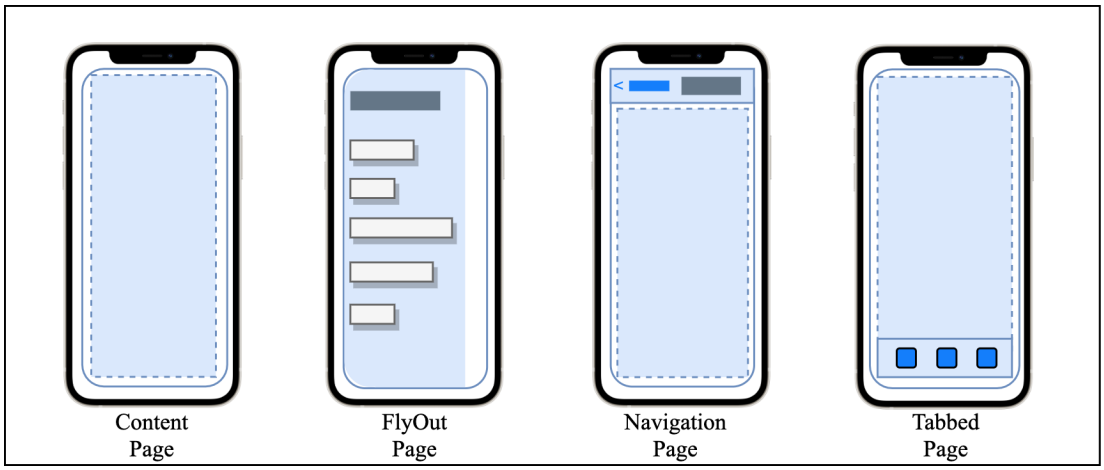

Abbildung 4.1: Xamarin.Forms Pages[76]

Die ‚ContentPage' ist ausschließlich für die Anzeige einer weiteren Ansicht verantwortlich. Die drei anderen Pages besitzen ein Navigationskonzept. Die ‚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

[74] Vgl. Ritscher 2020, Abgerufen am 28. März 2021.
[75] Vgl. Microsoft Corporation 2016b, Abgerufen am 28. März 2021.
[76] Abbildung in Anlehnung an Microsoft Corporation 2016b, Abgerufen am 28. März 2021.

dar.[77] Die Ansichtsseiten befinden sich in der Regel innerhalb der XAML-Datei auf der untersten Ebene, dem so genannten Wurzelknoten. Der Quelltext 4.3 zeigt dies exemplarisch für eine ‚TabbedPage'.

```
<?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"/>
</TabbedPage>
```

Quelltext 4.3: Xamarin.Forms ‚TabbedPage' Definition[78]

Es wird eine ‚TabbedPage' mit zwei 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 4.4 wird dies exemplarisch an der Realisierung einer ‚FlyoutPage' deutlich. Anders als bei der ‚TabbedPage', die aus einer Sammlung von Registerkarten besteht, finden sich im Quelltext die Eigenschaften ‚Flyout' und ‚Detail'.

```
<?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/>
    </FlyoutPage.Flyout>
    <FlyoutPage.Detail>
        <NavigationPage/>
    </FlyoutPage.Detail>
</FlyoutPage>
```

Quelltext 4.4: Xamarin.Forms ‚FlyoutPage' Definition[79]

Im Gegensatz 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.[80] Quelltext 4.5 zeigt die Realisierung einer MaterialDesign App in Flutter.

[77] Vgl. Microsoft Corporation 2016b, Abgerufen am 28. März 2021.
[78] Quelltext in Anlehnung an Microsoft Corporation 2020m, Abgerufen am 28. März 2021.
[79] Quelltext in Anlehnung an Microsoft Corporation 2020j, Abgerufen am 28. März 2021.
[80] Vgl. Google LLC 2020f, Abgerufen am 28. März 2021.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(primarySwatch: Colors.blue,),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

Quelltext 4.5: Flutter ‚MaterialApp' Definition[81]

Der Vergleich zwischen den 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 Status vergleichen, um grafische Veränderungen möglichst effektiv vorzunehmen.[82] Um in Flutter ein Navigationskonzept zu definiert , können verschiedene Widgets verwendet und verschachtelt werden, wie in Quelltext 4.6 beispielhaft für eine App mit Registerkarten visualisiert wird.

```
appBar: AppBar(
    bottom: TabBar(
        tabs: [ Tab(icon: Icon(Icons.directions_transit)),
                Tab(icon: Icon(Icons.directions_bike)),],
    ),
    title: Text('Tabs Demo'),
    ),
    body: TabBarView(
    children: [ Icon(Icons.directions_transit),
                Icon(Icons.directions_bike), ],
),
```

Quelltext 4.6: Flutter ‚Tab Layout' Definition[83]

Die deutlichen Unterschiede bei der Auswahl eines Navigationkonzeptes können überbrückt werden, indem zu jeder Xamarin.Forms Page das entsprechende Flutter Widget gefunden wird. Der Flutter-Widgetkatalog[84] und die Webseite "Flutter for Xamarin.Forms Developers"[85] wurde für die Recherche des Gegenstückes verwendet. Entsprechende Ergebnisse der Suche können in Tabelle 4.2 abgelesen werden.

[81] Quelltext in Anlehnung an Google LLC 2020s, Abgerufen am 28. März 2021.
[82] Vgl. Google LLC 2020j, Abgerufen am 28. März 2021.
[83] Quelltext in Anlehnung an Google LLC 2020r, Abgerufen am 28. März 2021.
[84] Vgl. Google LLC 2020q, Abgerufen am 28. März 2021.
[85] Vgl. Google LLC 2020f, Abgerufen am 28. März 2021.

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

Tabelle 4.2: Gegenüberstellung Pages

Gegenüberstellungen in Tabellenform, von Xamarin.Forms Elementen und Flutter Widgets werden auch an anderer Stelle in diesem Kapitel zugunsten der Übersichtlichkeit verwendet. Flutter Widgets, die im Text nicht expliziert aufgeführt werden, sind den Xamarin.Forms Elementen in Funktionalität und Aussehen nahezu identisch. Eine vollständige Referenztabelle, die sich aus allen Einzelbetrachtungen zusammensetzt, befindet sich in Anhang I .

Die Navigation durch die Anwendung ist ein wichtiger Bestandteil und hängt wie beschrieben von der verwendeten Ansichtsseite ab. Die ‚NavigationPage' bietet, eine hierarchische Navigation, bei der der Benutzer durch die Seiten vorwärts und rückwärts navigieren kann.[86] Flutter hat eine ähnliche Implementierung, welche die Widgets ‚Navigator' und ‚Routen' verwendet.[87]

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

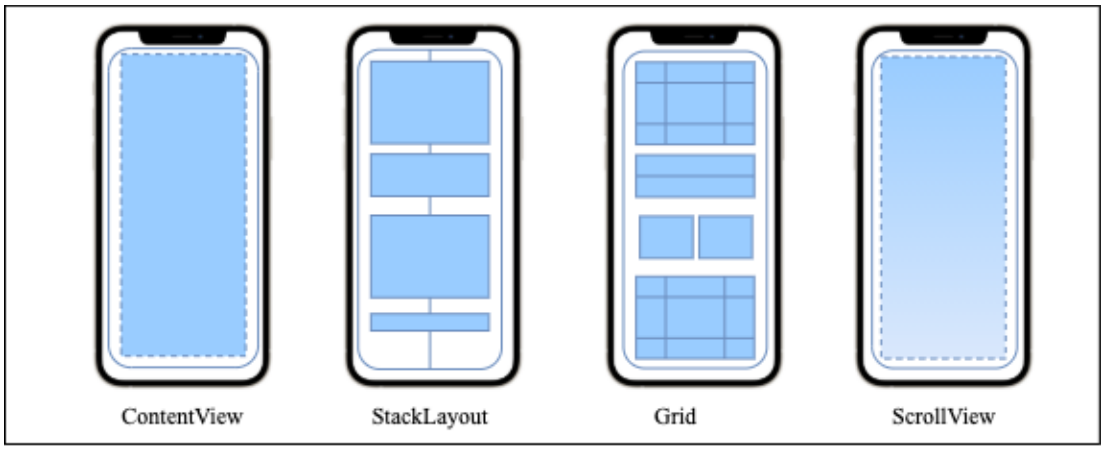

Abbildung 4.2: Xamarin.Forms Layouts[88]

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 Basisklasse

[86] Vgl. Microsoft Corporation 2020l, Abgerufen am 28. März 2021.
[87] Vgl. Google LLC 2020k, Abgerufen am 28. März 2021.
[88] Abbildung in Anlehnung an Microsoft Corporation 2018b, Abgerufen am 28. März 2021.

für benutzerdefinierte Darstellungen verwendet. Das Gestaltungselement ‚StackLayout' 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 übereinander 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. Das ‚RelativeLayout' übernimmt die gleiche Aufgabe, jedoch nur auf der Ebene des Layouts und untergeordneter Elemente.[89] Basierend auf diesen verfügbaren Layouts werden in Tabelle 4.3 die entsprechenden Flutter Widgets entgegengesetzt.

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

Tabelle 4.3: 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 bekommt, weil der Inhalt zu groß für die Darstellung auf einer Seite ist, wird das ‚Grid' in einem ‚ScrollView' verschachtelt. Dagegen bietet das ‚GridView' Widget von Flutter die Option des Scrollens automatisch an, wenn der Inhalt den sichtbaren Bereich überschreitet.[90] Im Rahmen der Codeoptimierung muss das ‚ScrollView' in diesem Anwendungsfall entfernt werden.

4.4.2 Steuerelemente

Steuerelemente sind die sichtbaren Bausteine der Benutzeroberflächen, beispielsweise Schaltflächen, Beschriftungen und Textfelder. Microsoft kategorisiert die Steuerelemente innerhalb der Frameworkdokumentation anhand ihrer primären Verwendung.[91]

[89] Vgl. Microsoft Corporation 2018b, Abgerufen am 28. März 2021.
[90] Vgl. Google LLC 2020g, Abgerufen am 28. März 2021.
[91] Vgl. Microsoft Corporation 2020h, Abgerufen am 28. März 2021.

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' Steuerelements 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.[92] Für die Steuerelemente kann nun eine Gegenüberstellung zwischen Xamarin.Forms Elementen und Flutter Widgets vorgenommen werden, wie in Tabelle 4.4 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 4.4: Gegenüberstellung Darstellungssteuerelemente

Zu zeichnende Elemente, wie die ‚Ellipse', ‚Linie', ‚Path', ‚Polygon', ‚Polyline' und ‚Rectangle' wurden nicht gesonders aufgeführt, da diese bei Flutter auf die sogenannte Canvas der Benutzeroberfläche gezeichnet werden können.[93]

[92] Vgl. Microsoft Corporation 2018b, Abgerufen am 28. März 2021.
[93] Vgl. Google LLC 2020d, Abgerufen am 28. März 2021.

Ereignisauslösende Steuerelemente

Xamarin.Forms ist ein ereignisgesteuertes Framework. Die hier behandelten Steuerelemente stellen alle mindestens ein Ereignis zur Verfügung, das durch die in Kapitel 3 erwähnten Codebehind Klassen abonniert werden kann. Sobald ein sogenanntes 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 einen Text anzeigen und 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', das 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. Nach der Eingabe von Textzeichenfolgen kann per Schaltfläche, oder Tastaturtaste, ein Ereignis ausgelöst und der eingegeben Text an die Codebehind-Datei weitergeleitet werden. Tabelle 4.5 zeigt die ereignisauslösenden Steuerelementen von Xamarin.Forms und alternative Flutter Widgets.

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

Tabelle 4.5: 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 mittels Ereignissen 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 4.7: Xamarin.Forms Button Initialisierung

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

Quelltext 4.8: Xamarin.Forms Event Handler

Steuerelemente zur Textmanipulation

In Xamarin.Forms stehen für die Arbeit mit Texten die Steuerelemente‚Entry‘zur Eingabe von einzelnen und ‚Editor‘ von mehreren Textzeilen bereit. Tabelle 4.6 zeigt die Gegenüberstellung zu Flutter Widgets.

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

Tabelle 4.6: Gegenüberstellung textmanipulierender Steuerelemente

Wie in der Übersicht erkenntlich besitzt Flutter ausschließlich das Widget ‚TextField‘, das beide Funktionalitäten der Xamarin.Forms Steuerelemente bündelt. Standardmäßig bietet das ‚TextField‘ Widget die Eingabemöglichkeit für eine Zeile ähnlich dem ‚Entry‘ Steuerelement, kann aber durch das Setzen einer Eigenschaft erweitert werden. Dies wird in Quelltext 4.9 dargestellt.

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

Quelltext 4.9: Eingabefeld mit mehreren Zeilen in Flutter

Steuerelemente zur Wertsetzung

Wersetzung bedeutet das Ergänzen von Steuerelmenten mit Eingaben durch den Anwender der App. Die folgenden Steuerelemente bietet Xamarin.Forms in dieser Kategorie an. Das ‚CheckBox' Steuerelement ermöglicht dem Benutzer die Auswahl eines booleschen Wertes (wahr, falsch). Die gleiche Funktionalität, bei einem anderen visuellen Erscheinungsbild (siehe Abbildung 4.3) bietet der ‚Switch'.

Abbildung 4.3: Darstellung von den Steuerelementen ‚Checkbox' und ‚Switch'

Die beiden linken Darstellungen der jeweiligen Steuerelemente zeigen den Zustand mit dem boolschen Wert falsch, die rechten wahr.

Ein ‚Slider' gewährt den Anwendern die Option einen Wert aus einem kontinuierlichen Bereich, ein ‚Stepper' aus einem Bereich von inkrementellen Werten auszuwählen. Eine Datumsauswahl wird durch das ‚DatePicker' Steuerelement ermöglicht, die Zeitauswahl mit ‚TimePicker'. Die Tabelle 4.7 präsentiert die gewohnte Gegenüberstellung von Xamarin.Forms Steuerlementen zu Flutter Widgets.

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

Tabelle 4.7: Gegenüberstellung wertsetzender Steuerelemente

Für die Steuerelemente ‚DatePicker' und ‚TimePicker' steht kein entsprechendes Widget zur Verfügung. Durch ‚TextField' und mithilfe einer Fingergeste wird eine Funktion aufgerufen, die den Auswahldialog für Datum und Uhrzeit öffnet und anschließend die Auswahl in das Textfeld einträgt. Quelltext 4.10 repräsentiert diese Funktion in Dart am Beispiel einer Zeitauswahl.

```
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 4.10: Verwendung von Timepickern in Flutter[94]

Die gesetzten Werte können bei Xamarin.Forms aus den Codebehind Klassen abgefragt werden, um den Status eines Steuerelementes zu ermitteln. Das Abrufen von Informationen in Flutter wird von speziellen Widgets, beispielsweise dem ‚TextEditingController' durchgeführt.

Aktivitätsandeutende Steuerelemente

In mobilen Anwendungen kann es aufgrund der limitierten Hardware Ressourcen und begrenzten Netzwerkanbindung zu zeitaufwendigen Aktionen kommen. Zur Visualisierung dieser Ladezeit stehen in Xamarin.Forms die folgenden Steuerelemente zur Verfügung. Der ‚ActivityIndicator' zeigt durch eine Animation, dass eine langwierige Aktivität ausführt wird, die ‚ProgressBar' kann mittels Ladebalken auch den Fortschritt darstellen. Die Tabelle 4.8 veranschaulicht die adäquaten Flutter Widgets.

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

Tabelle 4.8: Gegenüberstellung aktivitätsandeutender Steuerelemente

[94] Quelltext in Anlehnung an Google LLC 2021d, Abgerufen am 28. März 2021.

Sammlunganzeigende Steuerelemente

Die überwiegende Anzahl von mobilen Anwendungen visualisieren Sammlungen von Daten.[95] Xamarin.Forms stellt hierfür ebenfalls Steuelemente zur Verfügung. ‚CarouselView' zeigt eine blätterbare Liste von Datenelementen an. ‚IndicatorView' stellt mithilfe von Indikatoren die Anzahl der Elemente in einer ‚CarouselView' dar. ‚Picker' bietet die Möglichkeit eine Auswahl aus einer Sammlung zu entnehmen und anschließend in einem Textfeld auszugeben. Die Tabelle 4.9 zeigt die entsprechenden Flutter Widgets an.

Xamarin.Forms Steuerelemente	**Flutter Widget**
CarouselView	carousel_slider
IndicatorView	carousel_slider
Picker	flutter_material_pickers
TableView	Table

Tabelle 4.9: Gegenüberstellung sammlungsanzeigender Steuerelemente

Listen

Listen sind Steuerelemente und dienen ebenfalls der Anzeige und Interaktion von Sammlungen. Aufgrund der langsamen Ladezeiten von ‚ListView' hat Microsoft im Jahre 2019 mit ‚CollectionView' ein zweites optimiertes Steuerelement für die Anzeige von Listen zur Verfügung gestellt. ‚SwipeView' erlaubt einzelne Reihen zur Seite zu schieben und darunter liegende Schaltflächen sichtbar zu machen. Tabelle 4.10 stellt die Listen aus Xamarin.Forms mit denen aus Fluttter gegenüber.

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

Tabelle 4.10: Gegenüberstellung Listen

Die Xamarin.Forms ‚ListView 'ermittelt anhand einer Vorlage, wie eine Zeile dargestellt werden muss. Jede Reihe, die durch den Benutzer ausgewählt wird, löst ein Ereignis aus. Um dieses Verhalten in Flutter abzubilden, wird die Geste des Widgets in der Liste bereitgestellt.

[95] Vgl. Hindrikes und Karlsson 2020, S. 180.

Damit sich die ListView im Falle von Änderung in der angezeigten Sammlung automatisch aktualisiert, ist es notwendig, die Daten in einer ‚ObservableCollection' vorzuhalten, da somit die Benutzeroberfläche über Änderungen informiert wird. Eine Möglichkeit, die ‚ListView' in Flutter zu aktualisieren, besteht darin, eine neue Instanz des Widgets zu erstellen und die Daten aus der alten in die neue Liste zu kopieren. Dieser Ansatz ist zwar einfach umsetzbar, aber für große Datensätze nicht zu empfehlen. Eine effektive Änderung für dynamische oder umfangreiche Listen ist mit dem ListView.Builder möglich.

4.4.3 Ausrichtung von Steuerelementen

Innerhalb von ‚Stacklayouts' können Xamarin.Forms Steuerelemente mit Hilfe von ‚HorzintalOptions' und ‚VerticalOptions' ausgerichtet werden. Für diese Eigenschaften können die folgenden Werte gesetzt werden:[96]

- ‚Start' positioniert die Ansicht bei einer horizontalen Ausrichtung an der linken Seite des übergeordneten Layouts, und bei vertikaler Ausrichtung am oberen Rand des übergeordneten Layouts.
- ‚Center' zentriert die Ansicht in der Mitte des übergeordneten Layouts.
- ‚End' platziert bei horizontaler Ausrichtung auf der rechten Seite und bei vertikaler Ausrichtung am unteren Rand des übergeordneten Layouts.
- ‚Fill' sorgt im Falle einer horizontalen Ausrichtung dafür, dass die Ansicht die Breite oder Höhe des übergeordneten Layouts ausfüllt.

Neben diesen Werten stehen auch noch Ausprägungen mit der Erweiterung ‚AndExpand' zur Verfügung. Die Werte ‚StartAndExpand', ‚CenterAndExpand', ‚EndAndExpand' und ‚FillAndExpand' werden verwendet, um die Ausrichtungspräferenz festzulegen und zu bestimmen, dass die Ansicht mehr Platz einnimmt, wenn dieser im übergeordneten ‚StackLayout' verfügbar ist.[97]

In Flutter kann mit den Eigenschaften ‚crossAxisAlignment' und ‚mainAxisAlignment' definiert werden, wie eine Zeile oder Spalte ihre untergeordneten Widgets ausrichtet. Bei einem ‚Row' Widget verläuft die Hauptachse horizontal und die Querachse vertikal bei dem ‚Column' Widget andersherum. Ähnlich wie bei Xamarin.Forms stehen auch

[96] Vgl. Microsoft Corporation 2017c, Abgerufen am 28. März 2021.
[97] Vgl. Microsoft Corporation 2017c, Abgerufen am 28. März 2021.

hier ‚Start', ‚End' und ‚Center' zur Verfügung.[98] Eine Zentrierung von Layouts in Flutter kann durch eine Kombination von zentrierten ‚Row' und ‚Column' Widget erreicht werden. In der Praxis wird jedoch auf das ‚Center' Widget zurrückgegriffen, welches die gleiche Aufgabe mit weniger Quelltext realisiert.[99]

4.4.4 Gesten

Für die Interaktion mit der Benutzeroberfläche werden Gesten verwendet. Die Steuerelemente von Xamarin.Forms stellen Ereignisse für die häufig verwendeten Interaktionen bereit, wie im Unterpunkt ereignisauslösende Steuerelemente aufgeführt. Alternativ kann die Klasse ‚GestureRecognizer' verwendet werden, um seltenere Benutzerinteraktionen auf Ansichten zu erkennen, beispielsweise der Klick auf eine Steuerelement für die Darstellung.[100] In Flutter gibt es zwei ähnliche Möglichkeiten: Wird die Ereignisserkennung durch das Flutter-Widget z.B. den ‚ElevatedButton' unterstützt, kann eine Funktion übergeben werden, in der eine Geste behandelt wird. Ist keine Ereigniserkennung mittels Widget möglich, kann es in einem ‚GestureDetector' verschachtelt werden, wie im Rahmen des ‚Timepickers' in Quelltext 4.10 dargestellt.[101]

4.4.5 Animationen

Gut gestaltete Animationen machen eine Benutzeroberfläche intuitiver, tragen zum eleganten Erscheinungsbild einer ausgefeilten App bei und verbessern das Benutzererlebnis.[102] Xamarin.Forms enthält eine eigene Animationsinfrastruktur, die für die Erstellung einfacher Bildsequenzen unkompliziert, aber auch vielseitig genug ist, um komplexe Varianten zu erstellen. Die Klassen 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.[103] In Flutter stehen viele Animationen, insbesondere für Material-Widgets, zur Verfügung. Sie sind mit den in ihrer Design-Spezifikation definierten Standard-Bewegungseffekten geliefert, wobei diese Effekte anpassbar sind.[104]

[98] Vgl. Google LLC 2021b, Abgerufen am 28. März 2021.
[99] Vgl. Google LLC 2021c, Abgerufen am 28. März 2021.
[100] Vgl. Microsoft Corporation 2020k, Abgerufen am 28. März 2021.
[101] Vgl. Google LLC 2020m, Abgerufen am 28. März 2021.
[102] Vgl. Google LLC 2020i, Abgerufen am 28. März 2021.
[103] Vgl. Microsoft Corporation 2020e, Abgerufen am 28. März 2021.
[104] Vgl. Google LLC 2020i, Abgerufen am 28. März 2021.

4.4.6 Übersetzungsbeispiel

Nach der Einführung aller verfügbaren Seiten, Layouts und Steuerelemente soll exemplarisch ein UI von Xamarin.Forms zu Flutter überführt werden. Als Beispielseite wurde die im linken Bereich der Abbildung 4.4 dargestellte Login-Seite mithilfe von Xamarin.Forms entwickelt. Der entsprechende XAML-Quelltext wird im rechten Bereich der Abbildung gezeigt. Die Login-Seite zeigt eine zentrierte Form mit zwei Eingabefeldern für Benutzernamen und Passwort und eine Schaltfläche für die Ausführung des Loginvorganges.

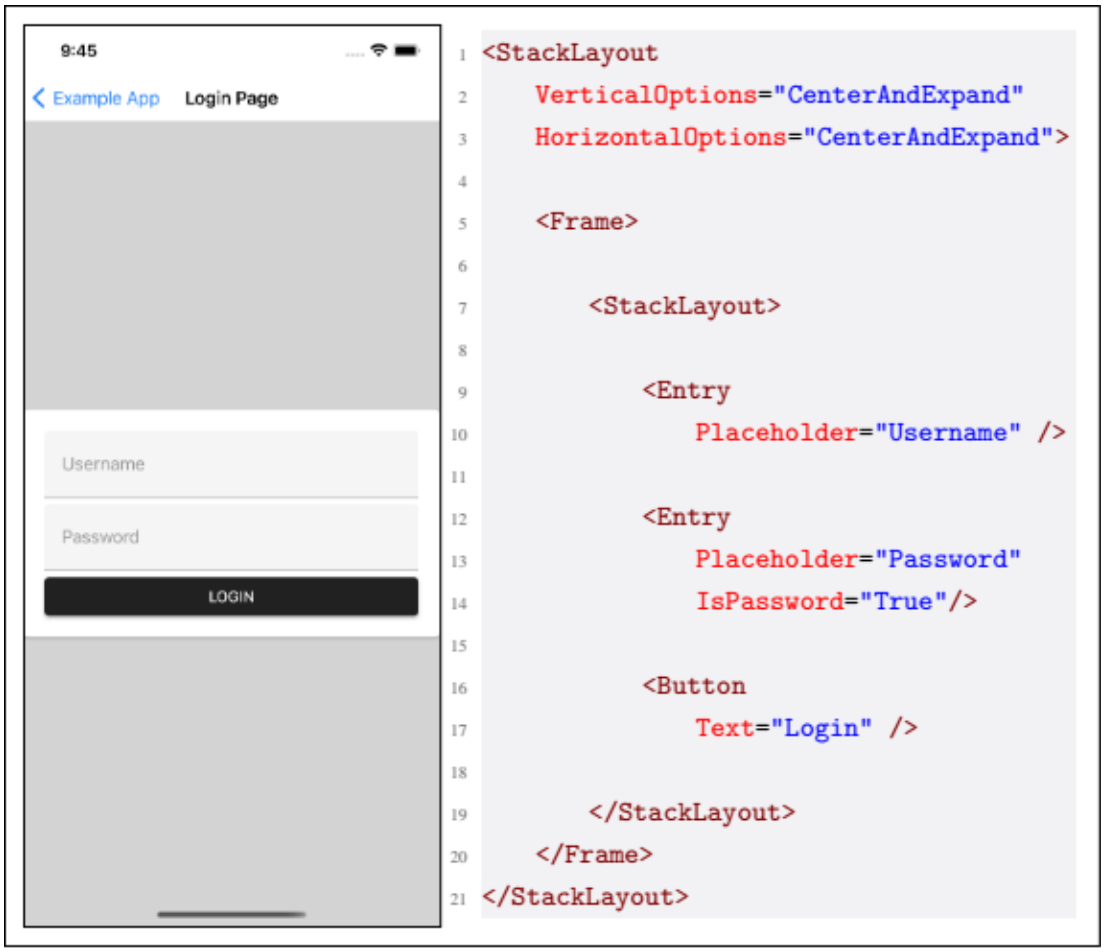

Abbildung 4.4: Darstellung einer exemplarische Login-Page

Mithilfe der Gegenüberstellungen von UI-Elementen und Widgets, kann anschließend eine Transformation des Xamarin.Forms Layout-Baums zu dem Flutter Widget Baum durchgeführt werden. Zur Verdeutlichung wird in Abbildung 4.5 eine Baumstruktur verwendet, welche die Verschachtlung innerhalb der Benutzeroberfläche visualisiert. Das übergeordnete ‚StackLayout' von Xamarin.Forms sorgt durch ‚CenterAndExpand' als Eigenschaftswert für eine zentrierten Darstellung. In Flutter übernimmt dies das ‚Center' Widget. Für alle Steuerelemente des obigen Quelltextes wird auf in diesem Kapitel verwiesene Widgets zurückgegriffen.

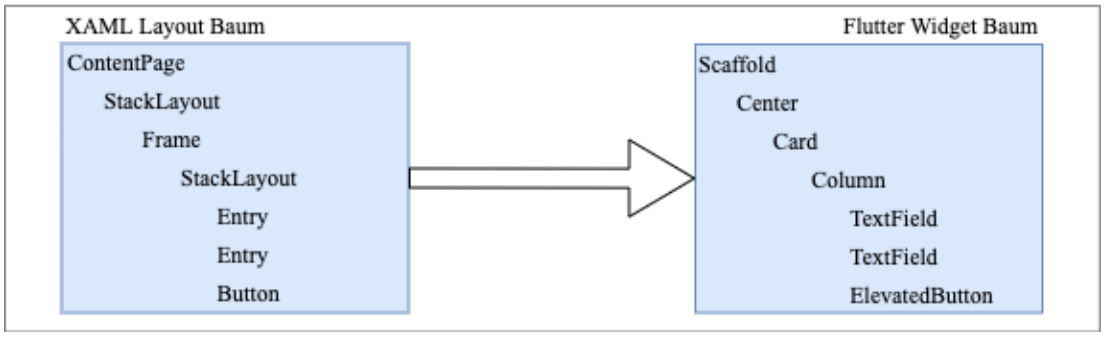

Abbildung 4.5: Überführung von Xamarin.Forms zu Flutter

Um die Funktionsfähigkeit des Widget-Baums sicherzustellen, wird er innerhalb einer Flutter-App realisiert, siehe Quelltext 4.11, und das visuelle Ergebnis mit der in Abbildung 4.4 dargestellten Xamarin-Forms App verglichen.

```
@override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey,
      appBar: AppBar( title: Text("LoginPage"),),
      body: Center(
        child: Card(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: <Widget>[
              TextField(
                obscureText: false,
                decoration: InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Username',
                ),
              ),
              TextField(
                obscureText: true,
                decoration: InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Password',
                ),
              ),
              ElevatedButton(
                child: Text('Login'),
              ),
            ],
          ),
        ),
      ),
    );
  }
```

Quelltext 4.11: Exemplarische LoginPage in Dart

Bei der Ausführung dieser App fällt auf, dass visuelle Unterschiede im Vergleich zu der vorher entwickelten Xamarin.Forms Variante existieren. Im oberen linken Teil der Abbildung 4.6 wird das zentrale Element der Login-Form dargestellt, wobei im Gegensatz zur Xamarin.Forms Variante die Abstände zwischen den Eingabefeldern fehlen. Außerdem ist der Login-Button nicht über die verfügbare Breite gestreckt. Abstände die das ‚StackLayout' automatisch zwischen Elemente legt, müssen in Flutter durch das ‚Padding' Widget gesondert nachgestellt werden. Eine weitere Angleichung ist durch das Widget ‚SizedBox' durchzuführen, um die Breite des ‚ElevatedButtons' anzupassen. Somit kann der zentrale Bereich des Widget-Baums neu generiert werden, wie er im unteren Linken Bereich der Abbildung 4.6 veranschaulicht. Der angepasste Quelltext, siehe hierzu auch Anhang II, erzeugt durch oben genannte Flutter Anpassungen eine Login-Ansicht mit dem im rechten Teil angezeigten UI die der Xamarin.Forms Variante sehr nahekommt.

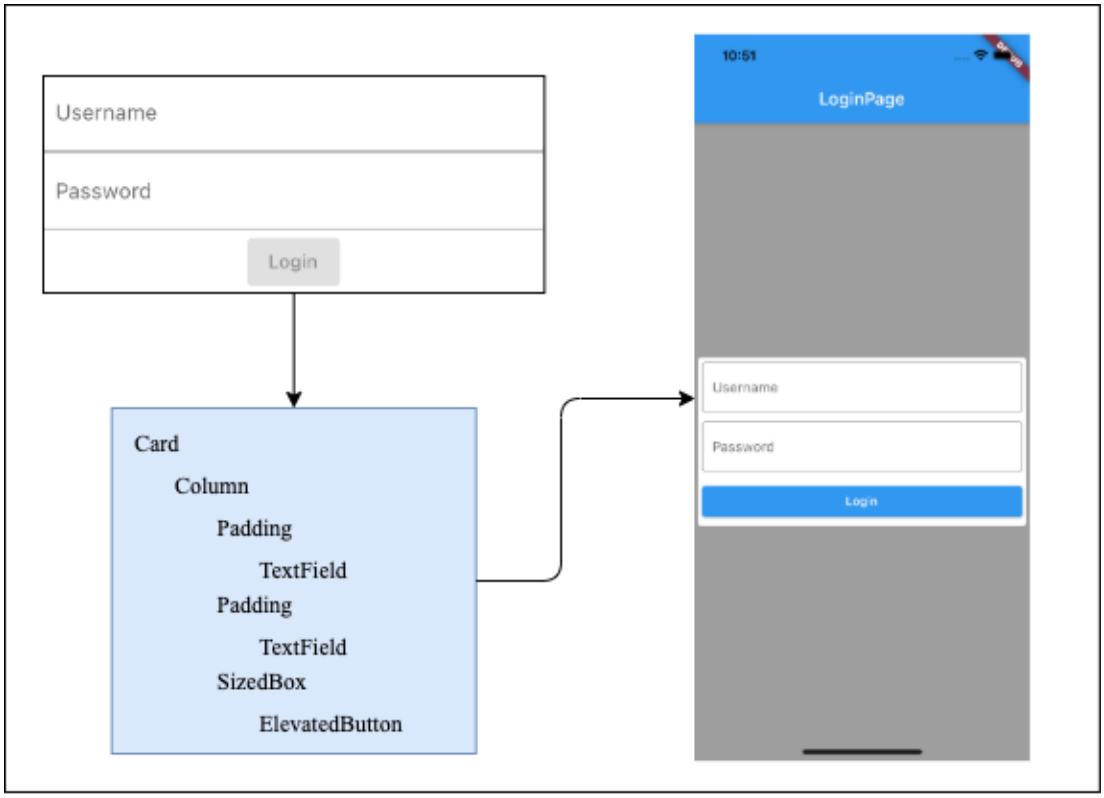

Abbildung 4.6: Flutter LoginPage Screenshot und Widget-Baum

Durch diese Validierung ist ein Beleg für die Annahme, dass der Austausch von UI-Elementen eine App mit vergleichbarer Benutzeroberfläche generieren kann, erbracht. Jedoch ist für die erfolgreiche Übersetzung von Xamarin.Forms nicht nur die Analyse der Ansichten von elementarer Bedeutung, sondern auch die Berücksichtigung der Eigenschaften und ihrer Kombinationen sowie der nicht im Quelltext ersichtlichen Standartwerte.

5 Unterschiede zwischen C# und Dart

Nach den in Kapitel 4 behandelten Unterschieden zwischen den Frameworks wird in diesem Kapitel die Heterogenität zwischen den Hochsprachen C# und Dart behandelt. Die beiden objektorientierten Programmiersprachen haben einen ähnlichen Stils und eine vergleichbare Syntax was den Umstieg von Xamarin.Forms zu Flutter vereinfachen kann.[105] Alle Unterschiede zwischen den Programmiersprache können innerhalb des Quelltextes der mobilen Anwendung vorkommen und haben daher einen direkten Einfluss auf die mit den Frameworks entwickelten Apps.

5.1 Klassendesign

Die zentrale Betrachtung dieses Abschnittes sind die sogenannten Klassen und Objekte der Objektorientierte Programmierung (OOP). Dabei handelt es sich um ein Ansatz der von problemorientierten Programmiersprachen verwendet wird und menschlichem konzeptionellen Denken nahe kommt. Dabei werden die sogenannten Objekte durch bestimmte, charakteristische Merkmale beschrieben die in der Klassendefinition festgelegt werden müssen.[106] Das OOP Paradigma wird sowohl von C# und Dart verwendet, jedoch gibt es vereinzelt Unterschiede bei der Realisierung.

5.1.1 Referenz- und Wertetypen

Das von C# verwendete .NET- Typsystem unterscheidet in Werte- und Referenztypen. Der Unterschied zwischen beiden ist in der Allozierung des Systemspeichers zu finden. Eine Variable eines Wertetyps enthält eine Instanz des Typs. Dies unterscheidet sich von einer Variablen eines Referenztyps, die eine Adresse auf die Speicherzellen des Typs

[105] Vgl. Pedley 2019, Abgerufen am 28. März 2021.
[106] Vgl. Witte 2013, S. 11f.

enthält.[107] C# bietet die folgenden eingebauten Wertetypen: Ganzzahlige numerische Typen (Integer) Fließkomma numerische Typen (Float und Double), boolean und Char welcher ein Zeichen darstellt. Jeder Variablen dieser Typen muss immer einen zum Typ passender Wert zugewiesen sein. So muss zu jeder Zeit in einer Instanz des Types Integer eine Zahl verwaltet werden.[108] Um die Notwendigkeit eines Wertes zu umgehen gibt es in der Hochsprache die sogenannte nullable Typen die neben den zulässigen Werten zusätzlich den Wert null zulassen.[109]

Die Programmiersprache Dart war klassischerweise ‚nicht Null sicher', dass bedeutet das sowohl Wert als auch Referenztypen nicht zu jeder Zeit einen Wert repräsentieren mussten. Dies führte dazu, dass bei der Arbeit mit Variablen Null- Prüfung ‚dargestellt in Quelltext 5.1, durchgeführt werden musste um Laufzeitfehler zu vermeiden.

```
1 int myVariable = null;
2 if (myVariable != null)
3     myVariable = 0;
```

Quelltext 5.1: Null-Sicherheit in Dart 1.x[110]

Im März 2021 hat die Programmiersprache Dart den Support für Null-Sicherheit hinzugefügt. So können Typen, wie die Wertetypen in C# nicht den Wert Null erhalten, es sei den der Entwickler entscheidet sich bewusst für das Gegenteil. Mit dieser ‚Null Sicherheit' werden die Laufzeit-Nullreferenzfehler zu Analysefehlern zur Bearbeitungszeit die nicht mehr zwangsläufig zum Absturz der Anwendung führen.[111] Quelltext 5.2 visualisiert die Arbeit mit den neuen Datentypen in Dart.

```
1 int i = 42; // Inferred to be an int.
2 int? aNullableInt = null;
3 if (aNullableInt != null)
4     aNullableInt = 0;
```

Quelltext 5.2: Null-Sicherheit in Dart 2.x[112]

Nach dieser Veränderung verhält sich die Programmiersprache Dart identisch wie C# sogar das Fragezeichen für die Definition einer Variable die Potentiell null seien kann ist identisch. Durch diese Veränderung muss der Compiler keine speziellen Änderungen mehr vornehmen, der Vollständigkeit halber sollte diese Änderung trotzdem nicht unerwähnt bleiben.

[107] Vgl. Kühnel 2019, S. 155f.
[108] Vgl. Microsoft Corporation 2020f, Abgerufen am 28. März 2021.
[109] Vgl. Bayer 2008, S. 167.
[110] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.
[111] Vgl. Google LLC 2021a, Abgerufen am 28. März 2021.
[112] Quelltext in Anlehnung an Google LLC 2021a, Abgerufen am 28. März 2021.

5.1.2 Datentypen

Das Verständnis des Unterschieds zwischen Referenz und Wertetypen ist elementar für die Programmierung mit .NET. Ein Unterschied besteht in der Initialisierung von Referenztypen. Während bei Werttypen einfach der entsprechende Wert zugewiesen werden kann, muss bei Referenztypen expliziert ein Objekt erzeugt oder ein bestehendes Objekt zugewiesen werden. Die Erzeugung eines neues Objekts geschieht dabei mit dem Operator ‚new'.[113] In Dart muss dieser Operator nicht verwendet werden, da analysiert wird, wann ein neues Objekt initiiert werden muss.[114] Dies wird in Quelltext 5.3 dargestellt.

```
1 var p1 = Point(2, 2);
2 var p2 = Point.fromJson({'x': 1, 'y': 2});
```

Quelltext 5.3: Optionales ‚new' Keyword in Dart[115]

Nach diesem wesentlichen Unterschied kann nun ein Vergleich der Verfügbaren Datentypen in 5.1) durchgeführt werden.

Datentyp	**C#**	**Dart**
Ganzzahl	sbyte byte short ushort int uint long ulong	Integer BigInt
Fließkommazahlen	double float decimal	Double
Strings	String	String
Textzeichen	char	
Array	Array	
Booleans	bool	bool
Lists	List	List
Maps	Dictionairy	Map

Tabelle 5.1: Gegenüberstellung Datentypen

[113] Vgl. Eller und Kofler 2005, S. 93.
[114] Vgl. Google LLC 2020a, Abgerufen am 28. März 2021.
[115] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.

Die Tabelle zeigt einige Unterschiede zwischen den Verfügbaren Datentypen, auf die im folgenden genauer eingegangen werden soll. Auf den Datentyp Boolean wird nicht weiter eingegangen, da sich diese nicht unterscheiden.

Zahlen

Wie die Tabelle zeigt, stehen bei C# eine Vielzahl von Typen für die Arbeit mit Ganzzahlen zur Verfügung. Diese haben einen Bereich von 8 bis 64 Bit und kommen jeweils mit und ohne Vorzeichen. Bei Dart steht keine Vergleichbare Auswahl zur Verfügung. In der Praxis gibt es aber auch Dart unterschiedliche interne Darstellungen, je nachdem, welcher Integer-Wert zur Laufzeit tatsächlich verwendet wird. Für besonders große Zahlenwerte steht jedoch auch der DatenTyp Bigint zur Verfügung.[116]Für Fließkommastellen steht in Flutter nur der Datentyp double zur Verfügung, während in C# double, float und decimal zur Verfügung stehen.

Textzeichen und Strings

Im Gegensatz zu C# steht bei Flutter kein Datentyp für einzelne Charaktere zur Verfügung. Als alternative wird auf den Datentyp String zurrückgegriffen, und diesem nur ein Zeichen zugewiesen. Für die Zuweisung eines String können entweder einfache oder doppelte Anführungszeichen verwenden, um eine Zeichenkette zu erstellen. Im Gegensatz zu C# wo doppelte Anführungszeichen für String und einzelne für einen einzelnen Charakter verwendet werden.

Arrays und Listen

Die vielleicht gebräuchlichste Sammlung in fast jeder Programmiersprache ist das Array, also eine geordnete Gruppe von Objekten. In Dart sind Arrays List-Objekte, daher werden sie häufig auch kurz als Listen bezeichnet..[117] Diese Liste enthält einen internen Puffer, der erweitert wird, wenn die Liste erweitert wird. In C# gibt es sowohl Arrays als auch Listen für die Behandlung von Sammlungen. Listen ermöglichen ein einfaches Einfügen von Elementen. Dafür und auch für andere Aktionen unterstützten die Liste mehrere Operationen. Array-Speicher ist statisch und kontinuierlich während Listenspeicher dynamisch ist.

[116]Vgl. Star 2019, Abgerufen am 28. März 2021.
[117]Vgl. Google LLC 2020a, Abgerufen am 28. März 2021.

Für die Übersetzung von C# zu Dart ist es folglich notwendig, alle Arrays zu Listen umzuwandeln.

Hashtabellen

Im Allgemeinen ist eine Hashtabellen ein Objekt, das Schlüssel und Werte miteinander verknüpft. Sowohl Schlüssel als auch Werte können beliebige Objekttypen sein. Jeder Schlüssel kommt nur einmal vor, aber sie können denselben Wert mehrfach verwenden. Dart unterstützt Hashtabellen durch den Typen Map[118] und Csharp durch den Typen Dictionary. Im Rahmen der Übersetzung muss der Typ Map demnach mit Dictionairy ersetzt werden.

5.1.3 Modifizierer

Alle Klassen und Eigenschaften verfügen in beiden Sprachen über eine Zugriffsebene. Diese 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 5.4 als Quelltext dargestellt.

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

Quelltext 5.4: Private und Public Definitionen in Dart[119]

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

[118] Vgl. Google LLC 2020a, Abgerufen am 28. März 2021.
[119] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.

Folglich führt dies potentiell zu falschen Zugriffsbeschränkungen bei der Übersetzung von Bibliotheken.

5.1.4 Vererbung

Die Vererbung ist, eines der primären Charakteristika der Objektorientierung. Sie 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 in C# nur eine direkte Basisklasse haben.[120] Veerebung ist in Dart ebenso möglich wie in C# dies wird in Quelltext 5.5.

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

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

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

Quelltext 5.5: Vererbung in Dart[121]

Um Mehrfachverebung zu umgehen wird in C# auf Schnittstellen (engl. Interfaces) zurückgegriffen, die eine Art Vertrag definieren. Jede Klasse oder Struktur, die diesen Vertrag implementiert, muss eine Implementierung der in der Schnittstelle definierten Member bereitstellen.[122] Dart kennt im Gegensatz zu C# keine Schnittstellen sondern verwendet stattdessen das Konzert der Mixins. Mixin-basierte Vererbung bedeutet, dass zwar jede Klasse genau eine Oberklasse hat, ein Klassenkörper aber in mehreren Klassenhierarchien wiederverwendet werden kann. Dies wird in Quelltext 5.6 visualisiert.

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

[120] Vgl. Microsoft Corporation 2020g, Abgerufen am 28. März 2021.
[121] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.
[122] Vgl. Kühnel 2019, S. 258f.

```
4
5 class MyClass with MyMixin
6 {
7 }
```

Quelltext 5.6: Mixin's in Dart[123]

5.1.5 Überführung einer beispielhaften Klassen

Basierend auf dem in diesem Abschnitt eingeführten Unterschieden kann nun eine Klasse von C# zu Dart übersetzt werden.

5.2 Namespaces

Namespaces werden häufig in C# -Programmen auf zwei verschiedene Arten verwendet. Erstens: Die .NET-Klassen verwenden Namespaces, um ihre zahlreichen Klassen zu organisieren. Zweitens: Eigene Namespaces zu deklarieren kann Ihnen dabei helfen, den Umfang der Klassen- und Methodennamen in größeren Programmierprojekten zu steuern. Die meisten C# -Anwendungen beginnen mit einem Abschnitt von using-Anweisungen. Dieser Abschnitt enthält die von der Anwendung häufig verwendeten Namespaces und erspart dem Programmierer die Angabe eines vollqualifizierten Namens bei jedem Verwenden einer enthaltenen Methode.[124]

Dart hat keine Namespaces, stattdessen werden Pakete oder Dateien als solche importiert. Dadurch kann ein direkter Zugriff auf alle Klassen und Funktionen innerhalb der Datei gewährt werden. Wenn es zu Namenskonflikten kommt können können Dateien benannt werden.5.7 zeigt die Arbeit mit Paketen und Dateien in Dart.

```
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 5.7: Importieren von Paketen in Dart[125]

[123] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.
[124] Vgl. Microsoft Corporation 2015c, Abgerufen am 28. März 2021.

5.3 Generische Typen

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 deklarierst 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.[126]

Generics werden in Dart sehr ähnlich behandelt wie in C# , mit der Ausnahme, dass eine generische Klasse ohne die Type-Beschränkung übergeben werden kann.[127] Quelltext 5.8 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 5.8: Generics in Dart[128]

5.4 Delegaten

In .Net ist ein Delegate 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.[129]

[125] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.
[126] Vgl. Microsoft Corporation 2015b, Abgerufen am 28. März 2021.
[127] Vgl. Cheng 2019, S. 98.
[128] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.
[129] Vgl. Microsoft Corporation 2015a, Abgerufen am 28. März 2021.

In Dart kann der Typ Typedef verwendet werden, um eine Methodensignatur zu definieren und eine Instanz davon in einer Variablen zu halten.[130] Dies wird in Quelltext 5.9 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 5.9: Delegates in Dart[131]

5.5 Bibliotheken

Klassenbibliotheken sind das Konzept der freigegebenen Bibliothek für .NET. Somit können nützliche Funktionalität auf Module verteilt werden, 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.[132]

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 die bereits im letzten Kapitel erwähnt wurden. 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 Funktionen direkt Inhalt von Dateien sein, ohne eine Klasse oder einen Namespace.

[130] Vgl. Pedley 2019, Abgerufen am 28. März 2021.
[131] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.
[132] Vgl. Microsoft Corporation 2016a, Abgerufen am 28. März 2021.

5.6 Asynchrone Benutzeroberfläche und Parallelität

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.[133]

Die Verwendung von asychronen Methoden für lang laufende Aufgaben, wie das herunterladen von Daten, trägt dazu bei, dass die Benutzeroberfläche reaktionsfähig bleibt , während die Nichtverwendung dieser Methoden oder die unsachgemäße Verwendung dazu führen kann, dass die Benutzeroberfläche Ihrer App nicht mehr auf Benutzereingaben reagiert, bis die lang laufende Aufgabe abgeschlossen ist. Für beide in dieser Arbeit behandelten Frameworks gilt, dass arbeitsaufwendige Aufgaben nicht auf in dem Thread durchgeführt werden sollten, die für die Benutzeroberfläche zuständig ist, um ein einfrieren dieser zu verhindern. In Flutter werden die asynchronen Möglichkeiten, die die Sprache Dart bietet, auch ‚async' und ‚await' genannt, um asynchrone Arbeiten auszuführen. Dies ist dem Vorgehen in C# sehr ähnlich und ist der Lösung in Xamarin.Forms sehr ähnlich die ebenfalls die Schlüsselwörter ‚async' und ‚await' verwendet.

```
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[134]

Um die Vorteile von mehreren Prozessorkernen nutzen zu können, reicht dieses Konzept

[133] Vgl. Microsoft Corporation 2020b, Abgerufen am 28. März 2021.
[134] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.

jedoch nicht aus, in Flutter müssen potentielle Hintergrund-Thread manuell verwaltet werden, ähnlich wie bei ‚async‘ und ‚await‘ ist das vorgehen vergleichbar mit dem Vorgehen in C# . So steht für den Start von rechenintensive Arbeiten sogenannte ‚Isolate‘ zur Verfügung, dies ähnlich arbeiten wie Tasks in C# . Dabei sind ‚Isolates ‘ sind separate Ausführungsthreads, die sich keinen Speicher mit dem Hauptspeicherheap der Ausführung teilen. Dies ist ein Unterschied zu ‚Tasks‘ . Das bedeutet, dass vom Haupt-Thread aus nicht auf Variablen zugegriffen werden kann oder Ihre Benutzeroberfläche durch den Aufruf von ‚IsetState()‘ aktualisieren können.

5.7 Netzwerkaufrufe

Um Netzwerkaufrufe durchzuführen, um Daten von einem Server abzurufen oder Benutzereingaben des Anwenders zu übermittelt wird in Xamarin.Forms die Klasse HttpClient verwenden. Für die Arbeit in Flutter wird dafür das http-Paket verwendet, welches von einem großteil der Netzwerkfunktionalitäten abstrahiert und es einfach macht Netzwerkaufrufe zu tätigten. Um eine Netzwerkanfrage zu stellen ist es wichtig, die vorher eingeführten Schlüsselwörter ‚async‘ und ‚await‘ zu verwenden, damit die Benutzeroberfläche auch während der Anfrage Reaktionsfähig bleibt. Ein Netzwerkanfrage in Flutter wird in 5.11 dargestellt.

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

Quelltext 5.11: Flutter Network request

[134] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.

5.8 Ereignisse

Ein Ereignis ist eine Meldung, die von einem Objekt gesendet wird, um das Auftreten einer Aktion zu signalisieren. In dem vorherigen Kapitel wurde schon eingeführt, dass Xamarin.Forms ein ereignisgesteuertes Framework ist, welches Ereignisse wie den Klick auf eine Schaltfläche nutzen um Aktionen auszulösen. Events können in C# auch ohne XAML-Dateien verwendet werden beispielsweise wenn eine Programmlogik, z. B. das Ändern eines Eigenschaftswerts, ausgelöst. 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.[135]

Anstelle von Ereignisse werden in Dart sogenannte Streams verwendet, die ähnlich wie Ereignisse arbeiten. 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, für den start werden sie geöffnet, danach abgheört und zum Ende hin geschlossen.. Der Vorteil dieses Ansatzes ist, dass dabei Werte transformiert oder Ereignisse für eine bestimmte Zeitspanne anhalten und vieles mehr.

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

Quelltext 5.12: Events in Dart[136]

[134] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. März 2021.
[135] Vgl. Microsoft Corporation 2017a, Abgerufen am 28. März 2021.
[136] Quelltext in Anlehnung an Pedley 2019, Abgerufen am 28. 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:[137] Windows wird als Platform für den Übersetzer zwangsläufig benötigt, da der Roslyn Compiler ausschließlich für dieses Betriebssystem zur Verfügung steht.
- Visual Studio 2019[138]: 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:[139] 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[140] oder Visual Studio Code[141]: Dabei handelt es sich um die Entwicklungsumgebungen für die Entwicklung von Flutter. Für beide Umgebungen steht jeweils eine Erweiterung für die Programmiersprache Dart und das

[137] Windows 10 ist über `https://www.microsoft.com/de-de/software-download/windows10ISO` zum download verfügbar.
[138] Visual Studio 2019 ist über `https://visualstudio.microsoft.com/de/downloads/` zum download verfügbar.
[139] Build tools for Visual Studio ist über `https://visualstudio.microsoft.com/de/downloads/` zum download verfügbar.
[140] Android Studio ist über `https://developer.android.com/studio` zum download verfügbar.
[141] Visual Studio Code ist über `https://code.visualstudio.com/` zum download verfügbar.

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: [142] 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 Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche kann mithilfe des Windows Presentation Foundation (WPF) realisiert. Da die Benutzeroberfläche nur auf Windows Computern lauffähig seien muss, auf denen die notwendige Software installiert ist.

6.3 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 vorherigen Übersetzungsvorgängen beeinträchtigt wird, auch im Vorhinein überprüft, ob das Zielverzeichnis existiert und leer ist.

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

6.4 XAML zu Dart

6.5 XAML zu Dart

,

6.6 C# zu Dart

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 den Compiler zu Testen ist es notwendig anhand vorher definierter Testfälle zu überprüfen, ob der Compiler die Xamarin.Forms App so übersetzt, wie es zu erwarten ist. Zu diesem Zweck werden in diesem Abschnitt Testfälle definiert die sich aus den ermittelten Unterschieden zwischen beiden Frameworks, sowie der Programmiersprachen ergeben.

Tabelle 7.1 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 Anhang IV: Android Screenshots 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

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

der Anwendung angesteuert werden können. In Abbildung 7.1 werden Screenshots der mobilen Anwendung gezeigt.

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 Smartphone-Sensoren ausgegeben werden. Diese Sensoren werden von vielen mobilen

Abbildung 7.1: Test Objekt Screenshots I

Abbildung 7.2: Test Objekt Screenshots II

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.

Abbildung 7.4: Test Objekt Screenshots IV

7.3 Testablauf

7.4 Testauswertung

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.

Bayer, Jürgen (2008). *Visual C# 2008. Windows-Programmierung mit dem .NET Framework 3.5*. München: Pearson Deutschland GmbH.

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.

Eller, Frank und Michael Kofler (2005). *Visual C#. Grundlagen, Programmiertechniken, Windows-Programmierung*. München: Pearson Deutschland GmbH.

Hindrikes, Daniel und Johan Karlsson (2020). *Xamarin.Forms Projects. Build Multi-platform Mobile Apps and a Game from Scratch Using C# and Visual Studio 2019*. Birmingham: Packt Publishing Ltd.

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.

Kühnel, Andreas (2019). *C# 8 mit Visual Studio 2019. Das umfassende Handbuch*. Bonn: Rheinwerk.

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.

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.

Witte, Joerg (2013). *Programmieren in C#. Von den ersten Gehversuchen bis zu den Sieben-Meilen-Stiefeln*. Wiesbaden: Springer Vieweg.

Online Quellen

Apple Inc. (2020). *Managing Your App's Life Cycle*. URL: `https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle` (besucht am 27. 02. 2021).

Google LLC (2020a). *A tour of the Dart language*. URL: `https://dart.dev/guides/language/language-tour#using-constructors` (besucht am 27. 02. 2021).

– (2020b). *Adding a splash screen to your mobile app*. URL: `https://flutter.dev/docs/development/ui/advanced/splash-screen` (besucht am 27. 02. 2021).

– (2020c). *Adding assets and images*. URL: `https://flutter.dev/docs/development/ui/assets-and-images` (besucht am 27. 02. 2021).

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

– (2020e). *didChangeAppLifecycleState method*. URL: https://api.flutter.dev/flutter/widgets/WidgetsBindingObserver/didChangeAppLifecycleState.html (besucht am 27. 02. 2021).

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

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

– (2020h). *Imagery*. URL: https://material.io/design/communication/imagery.html (besucht am 27. 02. 2021).

– (2020i). *Introduction to animations*. URL: https://flutter.dev/docs/development/ui/animations (besucht am 27. 02. 2021).

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

– (2020k). *Navigate with named routes*. URL: https://flutter.dev/docs/cookbook/navigation/named-routes (besucht am 27. 02. 2021).

– (2020l). *Shared preferences plugin*. URL: https://pub.dev/packages/shared%5C_preferences (besucht am 27. 02. 2021).

– (2020m). *Taps, drags, and other gestures*. URL: https://flutter.dev/docs/development/ui/advanced/gestures (besucht am 27. 02. 2021).

– (2020n). *url_launcher*. URL: https://pub.dev/packages/url_launcher (besucht am 27. 02. 2021).

– (2020o). *Use a custom font*. URL: https://flutter.dev/docs/cookbook/design/fonts (besucht am 27. 02. 2021).

– (2020p). *Using packages*. URL: https://flutter.dev/docs/development/packages-and-plugins/using-packages (besucht am 27. 02. 2021).

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

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

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

Google LLC (2020t). *Writing custom platform-specific code*. URL: https://flutter.dev/docs/development/platform-integration/platform-channels (besucht am 27. 02. 2021).

– (2021a). *Sound null safety*. URL: https://dart.dev/null-safety (besucht am 27. 02. 2021).

– (2021b). *Center*. URL: https://api.flutter.dev/flutter/widgets/Center-class.html (besucht am 27. 02. 2021).

– (2021c). *Lay out multiple widgets vertically and horizontally*. URL: https://flutter.dev/docs/development/ui/layout%5C#lay-out-multiple-widgets-vertically-and-horizontally (besucht am 27. 02. 2021).

– (2021d). *showTimePicker function*. URL: https://api.flutter.dev/flutter/material/showTimePicker.html (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).

– (2015c). *Verwenden von Namespaces (C#-Programmierhandbuch)*. URL: https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/namespaces/using-namespaces (besucht am 27. 02. 2021).

– (2016a). *.NET-Klassenbibliotheken*. URL: https://docs.microsoft.com/de-de/dotnet/standard/class-libraries (besucht am 27. 02. 2021).

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

– (2017a). *Behandeln und Auslösen von Ereignissen*. URL: https://docs.microsoft.com/de-de/dotnet/standard/events/ (besucht am 27. 02. 2021).

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

Microsoft Corporation (2017c). *Layoutoptionen in (Xamarin.Forms)*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/layouts/layout-options (besucht am 27. 02. 2021).

– (2017d). *Working With Property Lists in Xamarin.iOS*. URL: https://docs.microsoft.com/de-de/xamarin/ios/app-fundamentals/property-lists (besucht am 27. 02. 2021).

– (2018a). *Arbeiten mit dem Android-Manifest*. URL: https://docs.microsoft.com/de-de/xamarin/android/platform/android-manifest (besucht am 27. 02. 2021).

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

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

– (2019b). *Xamarin.Forms Custom Renderers*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/app-fundamentals/custom-renderer/ (besucht am 27. 02. 2021).

– (2019c). *Xamarin.Forms DependencyService*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/app-fundamentals/dependency-service/ (besucht am 27. 02. 2021).

– (2020a). *An introduction to NuGet*. URL: https://docs.microsoft.com/de-de/nuget/what-is-nuget (besucht am 27. 02. 2021).

– (2020b). *Asynchrone Programmierung mit async und await*. URL: https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/concepts/async/ (besucht am 27. 02. 2021).

– (2020c). *Images in Xamarin.Forms*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/images?tabs=macos (besucht am 27. 02. 2021).

– (2020d). *Launcher Class*. URL: https://docs.microsoft.com/en-us/dotnet/api/xamarin.essentials.launcher?view=xamarin-essentials (besucht am 27. 02. 2021).

– (2020e). *Simple Animations in Xamarin.Forms*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/user-interface/animation/simple (besucht am 27. 02. 2021).

Microsoft Corporation (2020f). *Value types (C # reference)*. URL: https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/builtin-types/value-types (besucht am 27. 02. 2021).

– (2020g). *Vererbung (C#-Programmierhandbuch)*. URL: https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/classes-and-structs/inheritance (besucht am 27. 02. 2021).

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

– (2020i). *Xamarin.Forms App Lifecycle*. URL: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/app-fundamentals/app-lifecycle (besucht am 27. 02. 2021).

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

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

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

– (2020m). *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).

Star, Ford (2019). *The Dart Language: When Java and C# Aren't Sharp Enough*. URL: https://www.toptal.com/dart/dartlang-guide-for-csharp-java-devs (besucht am 27. 02. 2021).

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

Vaibhavi, Rana (2020). *Change Application Name and Icon in Flutter project(Android and iOS)*. URL: https://medium.com/@vaibhavi.rana99/change-application-name-and-icon-in-flutter-bebbec297c57 (besucht am 27. 02. 2021).

Versluis, Gerald (2020). *Embedded Fonts: Custom Fonts in Xamarin.Forms*. URL: https://devblogs.microsoft.com/xamarin/embedded-fonts-xamarin-forms/ (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: Gegenüberstellung von visuellen Elementen

Xamarin.Forms Element	**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	carousel_slider
Picker	TextView mit Funktionalität
TableView	Table
List	List
CollectionView	List
SwipeView	flutter_slideable

Tabelle: Gegenüberstellung von visuellen Elementen

Anhang II: Optimierte Flutter-LoginPage

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.grey,
    appBar: AppBar( title: Text("LoginPage"),),
    body: Center(
      child: Card(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: <Widget>[
            Padding(
              padding: const EdgeInsets.all(5.0),
              child: TextField(
                obscureText: false,
                decoration: InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Username', ),
              ),
            ),
            Padding(padding: const EdgeInsets.all(5.0),
              child: TextField(
                obscureText: true,
                decoration: InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Password',),
              ),
            ),
            SizedBox(
              width: double.infinity,
              child: Padding(
                padding: const EdgeInsets.all(5.0),
                child: ElevatedButton(
                    child: Text('Login'),),
              ),
            ),
          ],
        ),
      ),
    ),
  );
}
```

Quelltext: Optimierte Flutter-LoginPage

Anhang III: Testfälle für den Compiler

ID	Beschreibung	Status
1	Das Icon der Anwendung wird sowohl für Android als auch für iOS übernommen	✓
2	Der Name der Anwendung wird sowohl für Android als auch für iOS übernommen	✓
3	FlyOutPage wird zu MasterDetailScaffold übersetzt	✓
4	NavigationPage wird zu Scaffold übersetzt	✓
5	TabbedPage wird zu TabBar und TabBarView übersetzt	✓
6	AbsolutLayout wird zu Positioned übersetzt	✓
7	ContentView wird zu StatelessWidget übersetzt	✓
8	Frame wird zu BoxDecoration übersetzt	✓
9	Grid wird zu GridView übersetzt	✓
10	ScrollView wird zu SingleChildScrollView übersetzt	✓
11	StackLayout wird zu Row und Column übersetzt	✓
12	RelativLayout wird zu Positioned übersetzt	✓
13	BoxView wird zu SizedBox übersetzt	✓
14	Image wird zu Image übersetzt	✓
15	Label wird zu Text übersetzt	✓
16	Map wird zu Leamaps oder Google Maps übersetzt	✓
17	WebView wird zu webview_flutter übersetzt	✓
18	Ellipse wird zu CustomPaint übersetzt	✓
19	Linie wird zu CustomPaint übersetzt	✓
20	Path wird zu CustomPaint übersetzt	✓
21	Polygon wird zu CustomPaint übersetzt	✓
22	Polyline wird zu CustomPaint übersetzt	✓
22	Rectangle wird zu CustomPaint übersetzt	✓
23	Rectangle wird zu CustomPaint übersetzt	✓
24	Button wird zu Flatbutton übersetzt	✓

25	ImageButton wird zu IconButton übersetzt	✓
26	RadioButton wird zu RadioButton übersetzt	✓
27	RefreshView wird zu pull_to_refresh übersetzt	✓
28	SearchBar wird zu flutter_search_bar übersetzt	✓
29	SwipeView wird zu flutter_slideable übersetzt	✓
30	Entry wird zu TextField übersetzt	✓
31	Editor wird zu TextField übersetzt	✓
32	CheckBox wird zu Checkbox übersetzt	✓
33	Switch wird zu Switch übersetzt	✓
34	Slider wird zu Slider übersetzt	✓
35	Stepper wird zu number_inc_dec übersetzt	✓
36	DatePicker wird zu TextField mit Funktion übersetzt	✓
37	TimePicker wird zu TextField mit Funktion übersetzt	✓
38	ActivityIndicator wird zu CircularProgressIndicator übersetzt	✓
39	ProgressBar wird zu LinearProgressIndicator übersetzt	✓
40	CarouselView wird zu carousel_slider übersetzt	✓
41	IndicatorView wird zu carousel_slider übersetzt	✓
42	Picker wird zu TextView mit Funktionalität übersetzt	✓
43	TableView wird zu Table übersetzt	✓
44	List wird zu List übersetzt	✓
45	CollectionView wird zu List übersetzt	✓
46	SwipeView wird zu flutter_slideable übersetzt	✓
47	Die Flutter App lässt sich mit Hilfe des Flutter Compilers zu mobilen Anwendungen übersetzten	✓

Tabelle: Testfälle für den Compiler

ID	**Beschreibung**	**Status**
1	Das Xamarin.forms Projekt wird über seine Projektmappe ausgewählt	✓
2	Für das Flutter-Projekt muss ein Zielordner angegeben werden	✓
2	Wenn beide Auswahlen getroffen wurden kann die Übersetzung gestartet werden	✓
3	Nach der Übersetzung werden alle übersetzten Dateien angezeigt	✓

4	Bei einem click auf die Dateien werden die durchgeführten Schritte angezeigt	✓

Tabelle: Testfälle für die grafische Benutzeroberfläche

Anhang IV: Android Screenshots

Abbildung 1: Test Objekt Screenshots I

Abbildung 2: Test Objekt Screenshots II

Abbildung 3: Test Objekt Screenshots III

Abbildung 4: Test Objekt Screenshots IV

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)