Flutter and iOS Application Comparison

An Empirical Metric Analysis of Performance and User Experience

Bachelor Thesis

submitted by:	Philip Krück
Date of Birth:	04.11.1998
Matriculation Number:	3938
Company Supervisor:	Jan Jelschen
First Reviewer:	Dr. Oliver Becker
Word Count:	< 12.000 (text + footnotes)
Degree Program:	B.Sc. Business Informatics (A Track 2018)
University:	Hamburg School of Business Administration
Submission Date:	09.04.2021
Partner Company:	apploft GmbH

HSBA HAMBURG SCHOOL OF BUSINESS ADMINISTRATION

Abstract

To be written... (max: 300 words)

Contents

1 Introduction **1**

1.1 Motivation 2

1.2 Problem Statement 3

1.3 Thesis Objective 3

1.4 Methods 3

1.4.1 Performance comparison 4

1.4.2 Usability comparison 4

1.5 Scope & Limitations 4

1.6 Thesis Structure 5

2 Flutter **6**

2.1 Mobile Development Approaches 6

2.1.1 Web App and Progressive Web App Approach 6

2.1.2 Hybrid Approach 7

2.1.3 Web Native Approach 7

2.1.4 Cross Compiled Approach 8

2.1.5 Native Approach 9

2.2 Flutter Framework Architecture 9

2.2.1 Programming Language and Compilation 10

2.2.2 Rendering and UI State 11

2.2.3 Platform Specific Method Channels 11

2.2.4 Flutter's Architectural Limitations 12

3 Methodology and Study Design **14**

3.1 Baseline App Testing Decision Process 14

3.1.1 Kickdown App Feature Presentation . . . 17
3.2 Performance Comparison . . . 17
3.2.1 Selected Performance Measurement Variables . . . 18
3.2.2 Measurment Process . . . 18
3.2.3 Profiling Tools . . . 19
3.2.4 Evaluation Process . . . 19
3.3 User Experience Comparison . . . 19
3.3.1 Interview Preliminaries and Technicalities . . . 20
3.3.2 Interview Guideline . . . 21
3.3.3 Interview Evaluation . . . 21

4 Application Design and Implementation **23**
4.1 Technicalities . . . 23
4.2 Declarative vs. Imperative UI . . . 24
4.3 Flutter UI Component Usage . . . 24
4.4 Image Gallery Architecture . . . 24

5 Results **25**
5.1 Performance Comparison . . . 25
5.2 Usability Comparison . . . 25

6 Summary **26**
6.0.1 apploft . . . 26

A Appendix **27**
A.1 Interview Guideline . . . 27
A.1.1 Interview Setup . . . 27
A.1.2 Background Questions . . . 27
A.1.3 Main Interview . . . 28

Chapter 1

Introduction

Mobile platforms are dominated by two players - Apple and Google with their respective operating systems iOS and Android. Cumulatively, they form a duopoly in the smartphone operating systems market with a combined usage share of 15.2% for iOS and 84.8% for Android in 2020 according to to the International Data Corporation (**IDC2021**).
To develop a mobile software application (*app*) for both target platforms, the corresponding development environments and technologies are utilized for each platform. This leads to a doubling of cost, development time, and the need for knowledge of two different application development paradigms. As a result, cross-platform frameworks such as Xamarin (**Xamarin2021**), React Native (**Facebook2021**), and Ionic (**Ionic2021**) have been created.

The fundamental principle behind these frameworks is the provision of a unified tech stack operating on a single code base leading to increased development speed while also providing the ability to deploy for both mobile operating systems.

Generally, cross-platform frameworks utilize webviews[1] or a software bridge to communicate with the underlying host platform plugging into native interfaces. In both cases, communication channel delays may occur during app runtime leading to reduced execution speed. In addition, cross-platform frameworks deliver abstract interfaces over multiple native interfaces leading to a decreased subset of functionality and especially impaired UI customizability. These inherent architecture attributes (further see Section 2.1) explain both impeded performance (**Ebone2018** and **Corbalan2019**) and user experience (**Mercado2016** and **Angulo2014**) compared to native technologies.

However, over the past 2 years one particular cross-platform technology with certain dis-

1. Webviews are UI component in both iOS and Android to display web content such as *HTML*, *CSS* and *JavaScript*

tinct attributes has risen strongly in popularity (**Statista2021**): Flutter (**FlutterDev20**) is a newly developed open-source UI toolkit by Google. It has a nonconventional approach to cross-platform development in that every app ships with the framework's rendering engine (Section ...). Thereby, Flutter bypasses host platform communication by avoiding the underlying system app SDK for terms of UI rendering. Flutter apps are compiled in native binary format which can be directly executed by the mobile computing device (see Section 2.2). Furthermore, the framework provides a *"[...] collection of visual, structural, platform, and interactive widgets"* (**GoogleWidgets2021**) for UI customization. It seems that Flutter addresses the exact same issues that are generally criticized about cross-platform solutions.

1.1 Motivation

As a digital agency specialized on native iOS and Android development, **apploft GmbH** (the partnering company of this thesis) (**apploft2021**) is highly interested in Flutter. The implications of using this framework could be wide-ranging, including the extension of the services portfolio to clients with lower budgets. For example, startups which are unsure about product/market fit (**Andreesen2007**) and held by tight budget constraints are especially keen on reaching the maximum number of potential customers with their app. Flutter could be utilized for a fast iteration of a product deployable to both mobile platforms. Worth noting are the benefits of serving smaller customers like startups which include higher growth potential for long-term cooperation and risk diversification in apploft's client portfolio.

The possible updside of implementing Flutter exceeds the acquisition of small clients. Instead of focusing on platform customization with native tooling, more effort could be directed into developing unique custom features. Furthermore, infrastructure setup, package development and app updates would only be necessary for one codebase.

Since the aforementioned economic incentives aren't necessarily company-specific, they are relevant for mobile application developers at large. Scientifically, this thesis may be the basis for future work on Flutter's architecture attributes and their contribution to runtime performance and UI rendering of frontend toolkits, in general.

1.2 Problem Statement

To the best of the author's knowledge, there are no peer reviewed articles comparing the performance or usability to native apps[2] since Flutter was first released in March 2018 (**FlutterReleases2020**), This leaves an especially interesting gap in the literature, since both aspects are the topmost perceived challenges of cross-platform frameworks considered from an industry-perspective (**BioernHansen2019**).

1.3 Thesis Objective

This thesis will focus on comparing Flutter's framework technology to iOS specifically. The assumption made in this paper, is that mimicking Android-specific appearance and behavior should be rather straightforward as both Flutter and Android utilize Google's **Material design** (**Google2021**) for their default components. Additionally, testing the framework against iOS is especially interesting as it seems less likely that Flutter would be able exploit system specific properties for performance optimization in Apple's closed operating system.
The aim of this thesis is derived based on the initially stated problems with cross-platform frameworks and the current lack of research on Flutter's lofty marketing claims to solve those drawbacks. Specifically, Google's assertion that Flutter can match *"native performance"* and the framework can be utilized for building *"expressive and flexible UI"* (**FlutterDev20**) will both serve as inductively derived hypotheses that shall be empirically verified or falsified individually by this thesis:

H_P: The Flutter framework yields comparable **performance** to native iOS app development frameworks for iPhone.

H_U: Comparable **user experiences** can be created with Flutter and native iOS frameworks for iPhone.

1.4 Methods

An archetypal native mobile app has been chosen as a case study for the evaluation of both hypotheses H_P and H_U. Based on typical mobile application features (explained in detail in Section 3.1), *Kickdown* (**Kickdown2021**) - an online car auction app - was chosen for this thesis. The app has already been developed for iOS by apploft and released to the App Store

2. A search for relevant articles has been conducted using Google Scholar, Sci-hub and IEEE Xplore.

in February of 2021 (link to appstore). For the purposes of comparison, a Flutter equivalent has been developed mimicking the relevant subset of UI and functionality of the original app (see Section 3.1). Both the original and Flutter replica app are used for subsequent hypothesis testing.

1.4.1 Performance comparison

The assessment of the performance hypothesis H_P has been conducted by profiling specific performance metrics including CPU, GPU and memory usage for particular use case flows in the original and Flutter application. The profiling results have been analyzed using common statistical techniques. A further explaination of the measurement process is detailled in Section 3.2.

1.4.2 Usability comparison

The analysis of the usability hypothesis H_U has been evaluated by the means of semi-structured expert interviews. Specifically, study participants have been be asked to evaluate both the Kickdown iOS and Flutter application clone along various metrics.

A detailed explanation of the interview process is given in Section 3.3.

1.5 Scope & Limitations

The feature set of the implemented app is representative for most, but not every type of app (see Section 3.1). Therefore the results cannot be generalized to every type of possible app. However, they should be seen as indicators of Flutters value as a cross-platform framework for the archetypal mobile app. Furthermore, the deductively chosen methodology yields the potential of finding adjacent hypothesis which may be further explored by other researchers. Additionally, the usability study doesn't provide statistical significance due to its qualitative nature. Nevertheless, the depth of detail in expert interviews is much greater compared to quantitative methods, and unthought of considerations may be suggested by the interviewees. This research attribute is especially compelling given the current research state on the Flutter framework as mentioned above.

1.6 Thesis Structure

- TODO: Describe structure of thesis and summarize each chapter. Do this once the other chapters are actually written.

Chapter 2

Flutter

Firstly, this chapter contextualizes Flutter's unique approach within mobile development (Section 2.1). Secondly, it presents an architectural overview of the framework (Section 2.2). Both explorations provide the reader with the necessary background knowledge in order to understand implementation choices of the clone application (detailed in Chapter 4) as well as particular elaborations in the study results (Chapter 5).

2.1 Mobile Development Approaches

The following subsections depict the individual mobile development approaches (derived from **Heitkoetter2013** and **Cunha2018**). Each technique can be placed along a spectrum of being built with web technologies on one and native technologies on the other end (see Figure 2.1). Generally, a higher reliance on native over web technologies corresponds to improved performance and usability as shown by **Heitkoetter2013**.

2.1.1 Web App and Progressive Web App Approach

Web apps are run and rendered in the browser. The underlying technologies are HTML, CSS and JavaScript with popular frameworks used for the development process such as Angualar

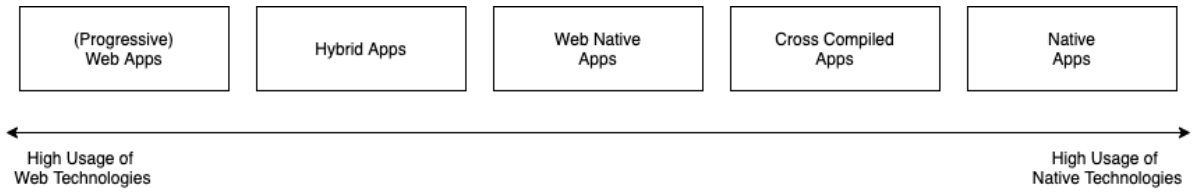

Figure 2.1: Mobile Development Approach Spectrum

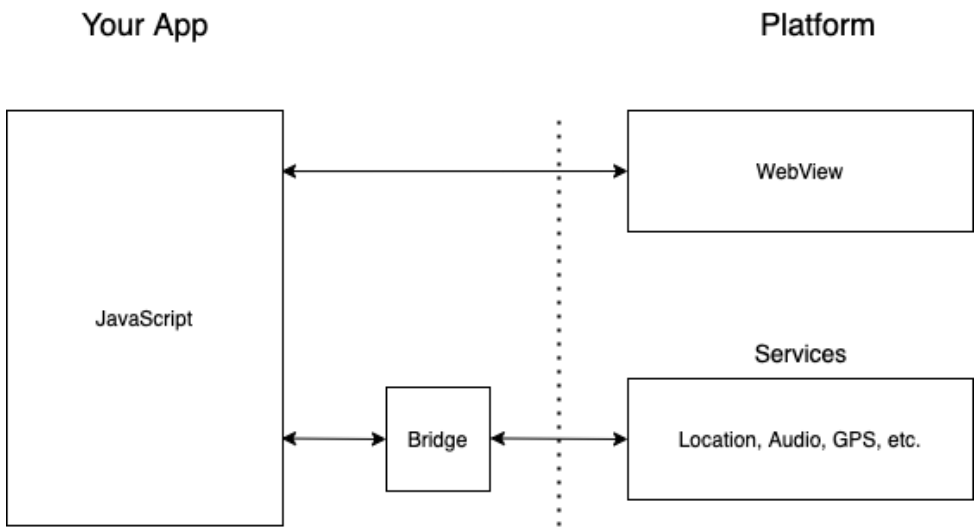

Figure 2.2: Mobile Hybrid Framework Architecture (adapted from **Cunha2018**).

(**Angular2021**), React.js (**React2021**) and Vue.js (**Vue2021**). Relying on the Browser and an internet connection, they do not have access to hardware capabilites or the file system. Progressive Web Apps (PWAs) function similarly to web apps but provide additional capabilities such as offline use, locally cached data, and push notifications. However, the feature set of PWAs is limited to the functionality exposed through the underlying browser.

2.1.2 Hybrid Approach

Hybrid Apps utilize the same technologies as web apps (see Subsection 2.1.1), but they are rendered in a platform web view (see Figure 2.2). Additionally they provide the ability to interact with native APIs such as GPS and sensor data through a platform bridge (see Figure 2.2). Unlike web apps, hybrid apps are shippable through official stores. Popular framework choices for building Hybrid Apps include Ionic (**Ionic2021**) and Cordova (**Cordova2020**).

2.1.3 Web Native Approach

Web Native Apps utilize the OEM components instead of web views for UI rendering while primarily using JavaScript. This is achieved by using a platform bridge for the transpilation of JavaScript into native platform code allowing for OS level user interface component employment and system services calls. Favored frameworks for building web native apps include React Native (**ReactNative2021**) and Native Script (**NativeScript2021**).

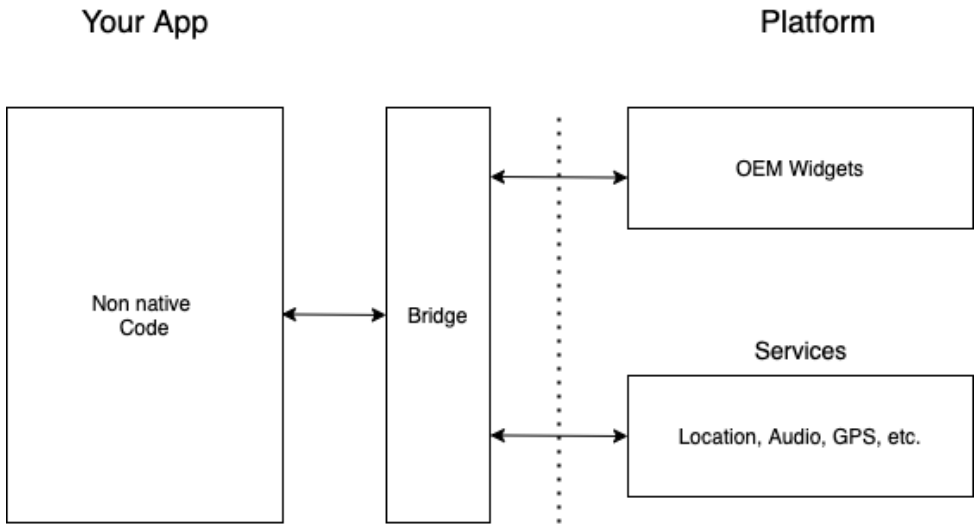

Figure 2.3: Web Native and Cross Compiled Mobile Framework Architecture (adapted from **Cunha2018**).

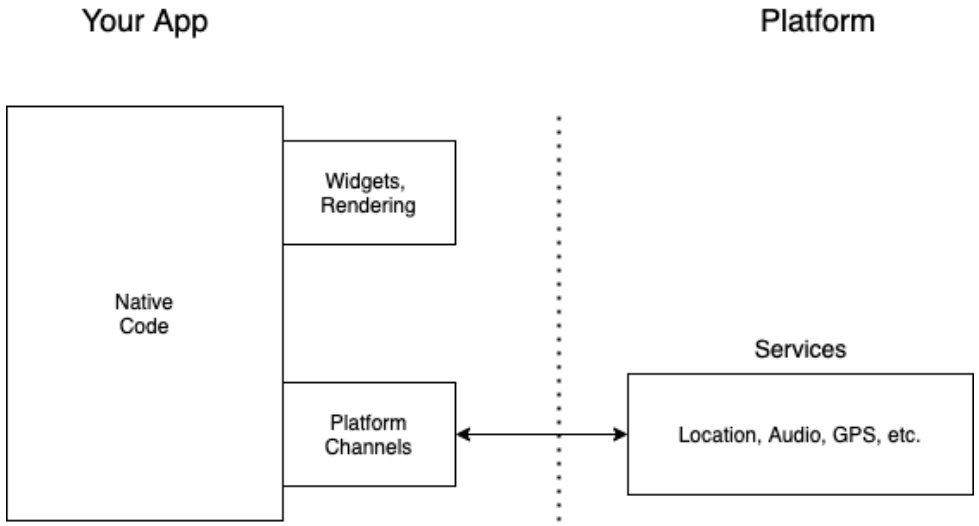

Figure 2.4: Flutter Architecture (adapted from **Cunha2018**).

2.1.4 Cross Compiled Approach

Generally, cross compiled apps take advantage of UI components and services from the underlying host platform similar to web native apps (see Subsection 2.1.3 and Figure 2.3). The OS plugin mechanism works by executing generated byte or machine code on the target device from a compiled language such as C# (used for Xamarin, **Xamarin2021**).

Flutter may be classified as a cross compilation based approach. However, it uniquely leaves the UI rendering process to its **Skia** graphics engine. The framework's architecture and technicalities are further explained in Section 2.2.

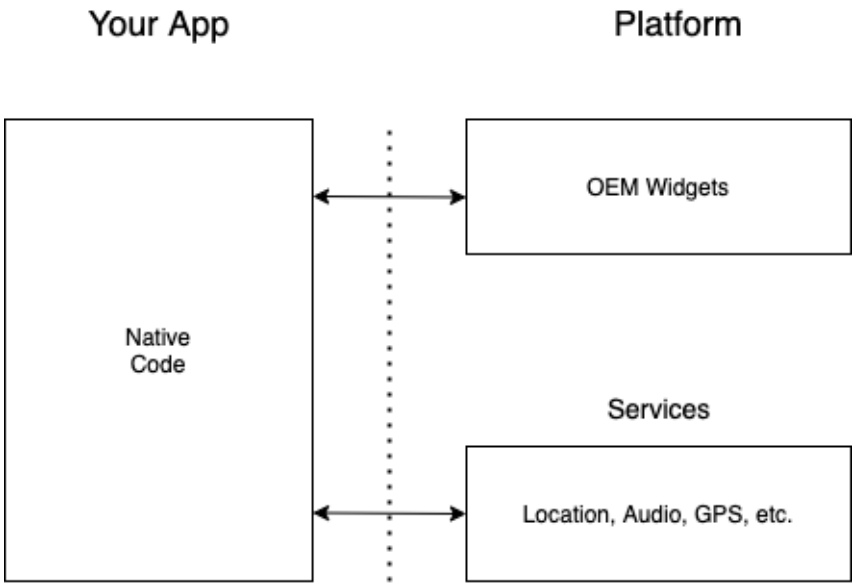

Figure 2.5: Native Mobile Architecture (adapted from **Cunha2018**)

2.1.5 Native Approach

The native approach is facilitated by the platform vendor and characterized by optimal OS integration through high hardware and software cohesion. Separate technology stacks as well as programming languages are used for implementing apps natively. iOS supports Swift and Objective-C whereas Android supports Kotlin, Java and C++.

2.2 Flutter Framework Architecture

Flutter's architecture is composed of three distinct layers: **framework**, **engine** and **embedder**. The framework is the top most layer which app developers interact with. It features animation and gesture APIs as well as structural and preconfigured platform specific UI components delivered through the **Material** (Android) and **Cupertino** (iOS) package.

Below the framework layer lies the engine which is written in C and C++ allowing for the production of native binaries. This act of cross-compilation is Flutter's technical value proposition to increase execution performance as stated in Subsection 2.1.4. The engine layer includes **Skia** - a 2D graphics rendering engine which is also used by the Chrome, Mozilla Firefox and Android (**Skia2021**). Furthermore, Dart runtime management, system event interaction as well as platform channel services (described in 2.2.3) are part of the engine layer.

The Embedder is the lowest layer of Flutter's architecture. Its sole purpose is to integrate the Flutter application into the platform-specific environment by providing native plugins, thread

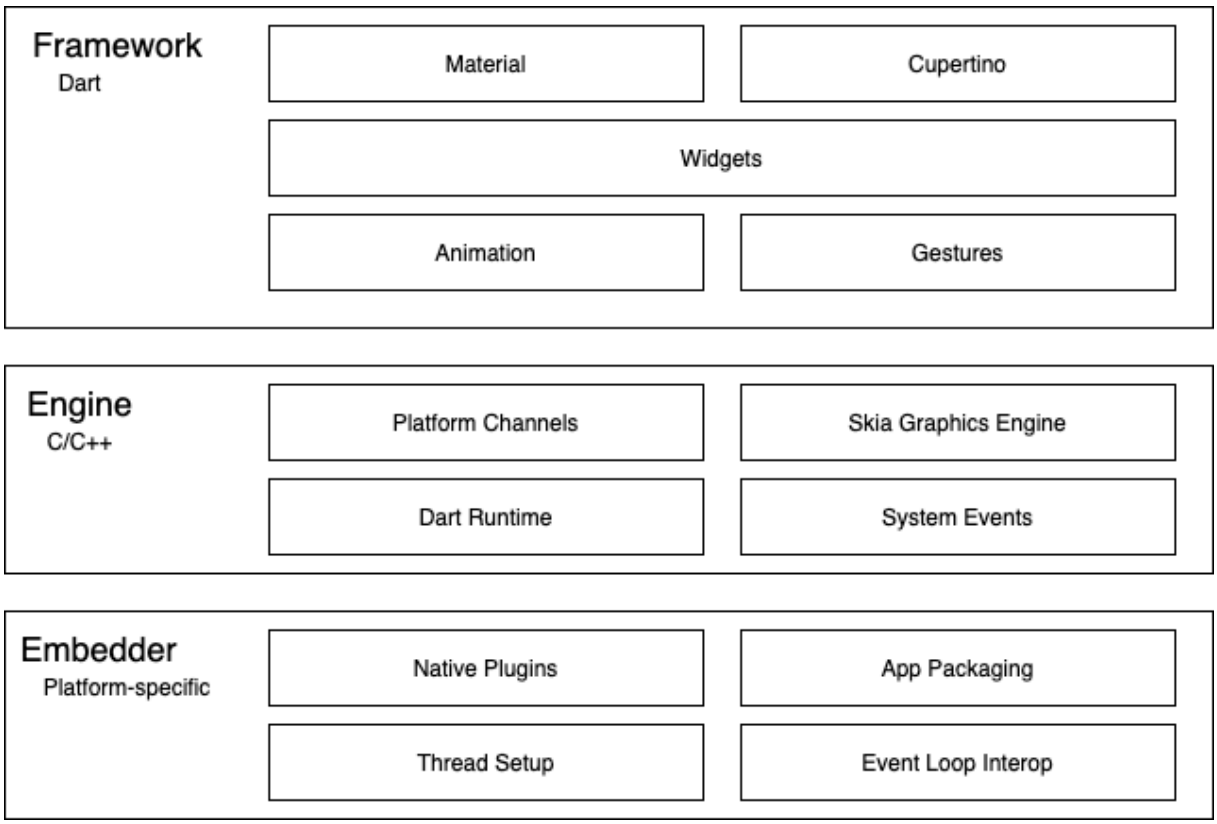

Figure 2.6: Flutter's Layered Architecture (adapted from **FlutterArchitecture2021**)

setup and event loop interoperation. Additionally, app package formatting is provided in the same way as for native apps. The host operating is thus not able to differentiate between a Flutter and a natively written app.

2.2.1 Programming Language and Compilation

Flutter apps are written in Dart - a compiled multiparadigm programming language syntactically similar to Java (see **DartLanguage2021**).

During Flutter development, apps run in the Dart Virtual Machine using Just-In-Time (JIT) compilation (**DartLanguage2021**). This offers stateful **hot reload** which allows reloading the UI after code changes without needing to fully recompile the app leading to faster development cycles.

For release purposes, Flutter applications are Ahead-of-Time (AOT) compiled into native machine code including the Intel x86 and ARM CPU instruction sets in order to optimize production performance (see **FlutterArchitecture2021**).

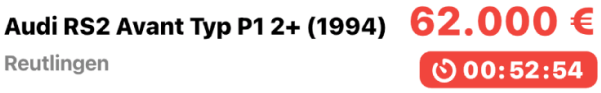

Figure 2.7: Kickdown Posting Overview Car Information UI

2.2.2 Rendering and UI State

Flutter apps are fundamentally composed of widgets which declare structural, stylistic and layout elements for building the user interface. Each widget may have 0, 1 or multiple children which in turn create a tree structure of parent-child relationships.

For example, the information presented for each posting on the overview of the Kickdown app (see Figure 2.7) is built using a simple widget tree structure (see structural Diagram in Figure 2.8 and Code Snippet 2.1):

A **Column** widget has the purpose of laying out 2 **Row** children vertically. Correspondingly, the Row widgets layout out their children horizontally. The first Row contains 2 **Text** widgets which represent the posting's title and current price of the car. Text widgets are terminal nodes as they have no further children[1]. The second Row, contains another Text widget representing the location and a **CountdownLabel** which is a custom built widget abstracting away its internal complexity at its point of use.

The entry point of every Flutter app is either a **MaterialApp** or **CupertinoApp** widget which also marks the root of the tree. Based on this tree structure Flutter can determine where and how elements should be drawn on screen, and instruct its graphics engine accordingly. This concept alone is not yet sufficient to enable modern mobile applications with complex UI changes or animations based on asynchronous events like user interaction. Widgets are mappings from state to a UI representation. When the state changes during runtime Flutter creates a new widget tree, diffs it against the old one and then redraws only its changes to the screen. This declarative approach simplifies UI development in the sense that the developer does not need to keep track of UI state which can grow exponentially with the increase of UI components on screen.

2.2.3 Platform Specific Method Channels

To utilize platform-specific APIs such as camera access, geolocation or other sensor data, Flutter communicates with the platform's native APIs via a method channel (see Figure 2.4).

1. Technically both Text and Image do have an implicit single child widget which is created by the framework.

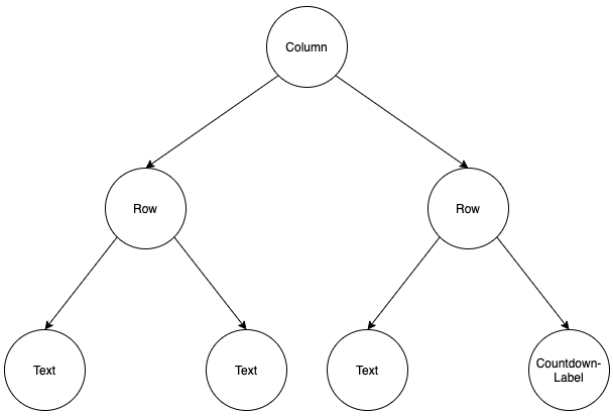

Figure 2.8: Widget Tree Structure Visualization of Code Snippet 2.1

Listing 2.1: Simplified Flutter Code for UI Layout shown in Fig. 2.7

```
Column(
    children: [
        Row(
            children: [
                Text("Audi
                    RS2..."),
                Text("62.000 €"),
            ],
        ),
        Row(
            children: [
                Text("Reutlingen"),
                CountdownLabel(...),
            ]
        )
    ],
),
```

This internal channel is used to execute code written in a host specific language. Thereby, Dart may be used to call Swift or Objective-C on iOS, and Kotlin or Java on Android (see **PlatformChannel2021**). Common functionalities are already provided by Flutter or third party packages (**PubDev2021**). Additionally, custom platform integrations may be implemented as required.

2.2.4 Flutter's Architectural Limitations

As explained in Subsection 2.2.2, Flutter renders its own widgets rather than using OEM components unlike other mobile development frameworks (mentioned in Section 2.1). When an underlying OEM component of the host platform changes with an OS update, Flutters' framework needs to be updated to resemble this new functionality. Thus, Flutter's *framework layer* (see Section 2.2) needs to continuously replicate vendor specific UI changes. However, once the widget has been replicated in the Flutter framework it may even be shipped to older operating systems.

As Google itself is using Flutter in multiple production apps (see **FlutterShowcase2021**), the company has an incentive to swiftly replicate OS features. Additionaly, Google is also the creator of **Material Design** (**Google2021**) which is the default design system on Android.

Furthermore, since every Flutter app ships with the Skia rendering engine, the app size may be larger than natively written apps. However, as Dart code is compiled to machine code

eventually, the marginal size increase for further features should be similar between both native and Flutter apps. The size of the rendering engine is approximately 4.5 mb (according to **FlutterFAQ2021**).

Chapter 3

Methodology and Study Design

This chapter explores the employed methodology for testing both the performance hypothesis H_P and the usability hypothesis H_U (defined in Section 1.3). The goal is to explicitly show the reasoning for the chosen methods as well as provide specific method implementation details to aid transparancy about the obtained results discussed in Chapter 5.
Generally, the employed methodology to achieve the research objective (Section 1.3) is to replicate a relevant subset of functionality of an existing iOS app using Flutter. Thereby, the **original app** acts as a baseline with which the **Flutter clone** can be comparatively evaluated. Based in this comparison, the research question - whether the Flutter framework can match native performance and provide equivalent usability (Section 1.3) - is answered. Instead of creating artificial use cases, taking advantage of an existing app provides realistic instances for performance as well as user interface testing.
The first section in this chapter (Section 3.1) details the decision process for selecting the baseline testing app as well as its feature reduction for further comparison. The subsequent two sections (Sections 3.2 and 3.3) explore the specific methods and reasoning for the performance and usability comparison respectively.

3.1 Baseline App Testing Decision Process

The procedure for choosing the case study app is based on a filtering process of 4 steps (i.e. application of constraints):

1. The app is built and maintained by apploft.

2. The app includes common application **features**.

3. The app uses modern iOS framework technologies.

4. The app conforms to the human interface guidelines (HIG) by Apple (**Apple2021a**).

The reasoning behind selecting the above filtering constraints is detailed in the following paragraphs.

Creation and Maintainance by apploft

This constraint was imposed on the filtering process such that a contact person (apploft employee) is available for code specific questions.
Having a reference to the original source code further provides the ability of implementing the Flutter replica similarly to facilitate comparability with the baseline app. E.g. a particular algorithm could be implemented similarly in the Flutter application. Thereby, equivalent time and space complexities are produced and algorithm implementation can be retracted as a confounding variable.
Furthermore, access to the original source code provides the ability to reduce unnecessary features which are irrevalent regarding the hypotheses evaluation. This reduces the complexity of the Flutter replica.

Inclusion of Common Application Features

The goal of this thesis is to determine whether Flutter is comparable in terms of performance (H_P) and usability (H_U) for the archetypal mobile app (Section 1.4). Therefore, only features commonly appearing in iOS apps are considered for find the app for baseline testing app.
For the purposes of finding the baseline app, a **feature** is defined as either

(a) a generalizable UI component which is non-trivial, or

(b) an underlying technical attribute influencing the user experience.

Trivial UI components (a) such as buttons or text weren't considered **features** as they are omnipresent throughout every app. As for (b), a technical attribute has to influence the user experience to be incorporated as the purpose of this thesis is testing Flutter's value as a UI framework (see Section 1.3). For example, networking can be viewed as a **feature** if fetched data is displayed via the UI, but is not a **feature** if the sole purpose of networking within an app is to extract analytics data.

Use of Modern iOS Frameworks

If this constraint were not applied on the filtering process, old iOS technology could be compared to a modernly built Flutter app. Therefore, constraining the baseline app to be built with modern iterations of iOS framework technology ensures a reasonable comparison against the replica app.

Conformance to Human Interface Guidelines

Conforming to Apple's HIG ensures the original app looks native to the iOS platform. Since Flutter comes from Google, it probably implements the **Material Design** (**Google2021**) rather well. Rebuilding an app that conforms to the Apple's design guidelines is the more interesting case.
In addition, providing a recognizable UX for iOS users would keep participants in the usability study (detailed in Section 3.3) focused on noticing differences instead of being distracted by an ambiguous UI.

Based on the above constraints, a small study was conducted looking at 15 apps developed by apploft (constraint 1) from 9 different iOS App Store categories. The features were extracted into Table [initial table] by going through each user interface (constraint 2). Furthermore, a feature had to appear at least twice before being added as a result.
Continuing the filtering process, as per constraint 2 uncommon features - features appearing in less than 50% of observed apps - are excluded. This reduces the list of features to the following:

- **Networking** - Interaction with a remote API.
- **Login/Authentication** - User log in meachanism through a UI.
- **Tab navigation** - UI component to quickly switch between different sections of app (**AppleHIGTabBar2021**)
- **Hierarchical navigation** - Screens are opened on top of previous screens using a Stack structure (**AppleHIGNavigation2021**).
- **Keyboard interaction** - A UI for inputing text via a software keyboard.

- **Vertically scrolling collections** - A UI collection of items scrolling vertically.
- **Horizontally scrolling collections** - A UI collection of items scrolling horizontally.
- **Webview component integration** - An integrated UI component in the app displaying web content (**AppleHIGWebViews2021**).

Out of the 15 initially tested apps, 5 include all of the above **features** (conforming to constraint 1 and 2) (see Table [table after applying constraint 2]). Kickdown (see Section ...) is chosen among the remaining contestants for the baseline testing app. It was most recently released (Feb 2021) and is therefore built with modern iOS technologies (constraint 3) and complies to the most recent iteration of the **Human Interface Guidelines** (constraint 4).

The login and signup mechanism - although a common **feature** - is removed from the original app for baseline testing. This is due to the fact that textfield and button interaction as well as networking is already present in other parts of the app and would yield no further insight regarding the hypotheses evaluation.
The Flutter app is implemented as closely as possible to the original application to avoid an asymmetrical comparison as detailed in Section **??**.

3.1.1 Kickdown App Feature Presentation

Here I will present what the original features of the Kickdown app are

3.2 Performance Comparison

The methodology chosen to test the performance hypthesis HP (Section 1.3) is a quantitative measurement of computational resources during app runtime. Measurements are performed for specific load conditions (i.e use cases). In the process, the original app acts as an empirical baseline for testing the Flutter replica against.
Directly benchmarking system resources provides insight whether the Flutter framework consumes compute resources efficiently under typically imposed load settings. Furthermore, system benchmarking metrics are the underlying cause of more ephemeral measures for testing the system load itself, e.g. page load time. In addition, the chosen compute resources (explained in Subsection 3.2.1) are easily measured using software tooling (Subsection 3.2.3) which aids the traceability as well as reproducibility of this particular study methodology.

3.2.1 Selected Performance Measurement Variables

The following paragraphs introduce the selected performance measurement variables. Concretely, a brief definition is given as well as the reasoning for including the particular metric in this study with regards to evalutating the performance hypothesis (Section 1.3).

CPU Utilization

CPU utilization is defined as the CPU time (**FSF1988**) of a task divided by its overall capacity expressed as a percentage. The CPU as well as its integrated GPU[1] are responsible for graphics rendering related computations. Generally, high CPU usage is an indicator of insufficient processing power of the executed task. Therefore, testing CPU utilization directly assesses whether Flutter's framework processing requirements for UI rendering can be fulfilled given the testing hardware (see Subsection 3.2.2).

Frames per Second

Frames per Second (FPS) describes the rate at which the system, and in particular, the GPU completes rendering individual frames. The FPS rate directly determines the smoothness of UI animations and transitions (**Google2020**).

Memory Utilization

Memory utilization is the percentage of available memory capacity used for a specific task. A high level of memory usage "[...] affects the performance of actual running tasks, as well as interactive responsiveness" (**Ljubuncic2015**).

3.2.2 Measurment Process

The measurement process for the individual metrics is further split into specific user actions which are executed and tested on both the iOS and Flutter app separately. These were chosen to test all relevant facets of the app (see Section **??**) and ensure a necessary load on the system:

- **app start:** The app is freshly installed on the test device, opened and idle until the visible postings are loaded.
- **scrolling:** On the postings overview screen, the posting cards are vertically scrolled fully to the bottom and subsequently back to the top.

1. The GPU is an integrated chip within the System-on-a-Chip (SoC) architecture (**Martin2001**) for all iPhone processing units (**WikiChip2020**).

- **detail view:** From the postings overview, the first posting is tapped to navigate to the detail view. Afterwards the back button is tapped to navigate back to the overview.
- **image gallery:** The image gallery of a posting is opened from the detail view of a posting and the first 10 images are viewed by swiping.

For each **user action**, the average of all values over time is recorded. This process is then repeated 3 times and averaged. The exact number of experiment repetitions was chosen as a tradeoff between marginal accuracy increase and additional experiment execution time.
Furthermore, 2 testing rounds are devised on separate devices. The iPhone 12 and iPhone 6s are chosen as the upper and lower bounds of hardware performance respectively. The lower bound is defined in this case as per Apples recommendation to set the deployment target to the current operating system version (iOS 14 at time of writing) minus one (iOS 13) which lists the iPhone SE as the oldest supported device (**Apple2021**). iOS 13 is also the minimum deployment target for the Kickdown app.
To reduce measurement confounders, the device is restarted before each measurement use case to ensure that all irrelevant background processes are cancelled.

3.2.3 Profiling Tools

Xcode Instruments (**Apple2019**) - a part of the **Xcode** IDE tool set - are used for profiling the individual metrics. It provides multiple preconfigured profiling trace instruments. For the purposes of this thesis, the **Time Profiler** tool (see Figure ...) is used for CPU (see Paragraph 3.2.1), the **Core Animation** tool (see Figure ...) for FPS (see Paragraph 3.2.1), and **Allocations** tool (see Figure ...) for memory usage (see Paragraph 3.2.1) quantification over time.

3.2.4 Evaluation Process

To better understand the data gathered, it is subsequently examined using exploratory data analysis (EDA) (**Tukey1977**). *To be continued a bit...*

3.3 User Experience Comparison

This Section explores the usability hypothesis evaluation methodology. Specifically, laying out the procedure to answer the question of whether or not the Flutter framework is capable of

reproducing native iOS application user experiences (see Section 1.3).

Generally, as UX is built for other humans, any evaluation is prone to subjectivity and perception biases (**Tversky1974**). Therefore, it is difficult to capture these impressions quantitatively in a reproducable manner.
However, the user experience of an app is directly dependant upon sufficiently underlying performance (e.g. for scrolling fluidity). Therefore the results of the performance comparison (Section 5.1) form the basis of the evaluation of the usability hypothesis H_U. Utilizing a mixed approach as a study methodology combining both the quantitative performance comparison and a qualitative method will draw upon the strengths of both approaches.

Specifically, semi structured interviews with subject matter experts (Cf. **Liebold2009**) are conducted to evaluate the baseline application with the Flutter replica by asking the participants for differences between the two apps (further explained in Subsection 3.3.2). This methodology has the advantage of covering predetermined topics relevant to the research question while also allowing spontaneous discussion possibly leading to novel insights.
Furthermore, expert interviews are an especially useful approach for scientific explorations with no or scant preexisting theory (cf. **Experts2009**).

The number of interviews conducted is limited as soon as no new perspectives seem to emerge from performing further interviews. This is based on the fact that the method of expert interviews aims to provide a breadth of perspectives on a given topic unlike specific quantitative anaylses (Cf. **Liebold2009**).

3.3.1 Interview Preliminaries and Technicalities

The interviews are conducted with employees of apploft. They qualify as subject matter experts in the sense that they have been working in the mobile app industry for multiple years. They come from variety of professional backgrounds including UX and UI design, project management as well as software engineering. Furthermore, some interviewees have actually worked on the original app itself. This diversity among the study participants is especially relevant in order to explore a breadth of perspectives. The interviews are moderated and recorded by the author with a single interviewee per interview. For the comparison, the interviewees receive QR codes with which both apps may be downloaded. Behind each distributed code is a downloadable IPA (iOS App Store Package) binary executable file hosted by an HTTP server. These work exactly the same as any other apps downloaded from the iOS App Store. Due to the ongoing Covid-19

pandemic, the interviews are conducted through video calls and the interviewees are asked to share their iPhone screen via Quicktime player (**Apple2014**).

3.3.2 Interview Guideline

The interview guideline (see Appendix ...) is based on finding out perceptual differences between the iOS baseline and Flutter replica app.

Just like the performance comparison, use cases associated with particular UI features form the basis for the evaluation:

- **App Start and Scroll Behavior** - ...
- **Detail Transition, Modal Transition, Textfield interaction** - ...
- **Horizontal Scrolling** - ...
- **Switch Interaction** - ...

The interviewees are asked to perform a particular use case for app A and app B. Then they are asked to detail differences between the two apps. Asking this open-ended question aims at receiving as much information possible about perceptual differences (Cf. **Helferrich2011**). Subsequently, the participant is asked to determine which of the two apps felt more natural (i.e. had a better UX). A determined tertiary response of: "A", "B" or "same" is expected. The goal of this question is to get overall impression of the usability.
Both questions are asked after each use case execution of the participant. To maintain a high participant engagement during interviews, use cases are described in a more captivating way, e.g. "Please find the blue Mercedes SUV in Kickdown A [Wait until participant has found it.]. Now, please look for the black BMW convertible in the other app.". After each use case, the ordering of app A and B is swapped. E.g. if the first case starts with A, the second starts with B. This choice is made as to avoid recency bias (Cf. **Atkinson1968**). Furthermore, the ordering is also swapped after each interview. In this way, participant X starts with app A while participant Y starts with B. Finally, after the last use case, the participant is asked to answer the two questions with regard to the entire application.

3.3.3 Interview Evaluation

The videos from the interviews are transcribed into a textual format and further processed using **interview coding**. Thereby, each interview is categorized into semantic themes. These themes

among all interviews are then merged into an overall theme structure - also known as code structure. This code structure is forms the basis for the evaluation of the usability hypothesis H_U.

Chapter 4

Application Design and Implementation

This thesis does not use the engineering method (see **Ertas1996**) for evaluating the hypotheses (Section 1.3) and thus application design and implementation will be explained from a perspective of a necessary requirement to execute both the performance and UX comparison (detailed in Chapter3). Naturally, software implementation complexity may impede performance if unscalable algorithms or memory-inefficient datastructures are chosen. Therefore, the goal from a development perspective was to implement the Flutter app as closely as possible to the original app in order to facilitate a fair comparison between both apps.
Firstly, general implementation differences resulting from Flutter's declarative nature are explained and contrasted to UIKit's imperative approach. Secondly, the usage of the Cupertino package and its UI components in the clone application is inspected. Finally, the Image Gallery architecture is examined as a particularly difficult section of the app in both implementations.

4.1 Technicalities

- what is the exact Flutter version that I've been using?
- what is the exact Dart version that I've been using?

4.2 Declarative vs. Imperative UI

4.3 Flutter UI Component Usage

- used as many out of the box cupertino widgets as possible for implementation

- only deviated from above guideline when absolutely necessary

- this was the case for the more view UI which had to be custom built from many smaller Flutter widgets.

- this was also the case for the image gallery which is explained in next section

4.4 Image Gallery Architecture

- image gallery (zooming ability), image caching

Chapter 5

Results

5.1 Performance Comparison

5.2 Usability Comparison

Chapter 6

Summary

- What are my key findings? What is their significance and implications

6.0.1 apploft

- Byproduct of analysis was the development of a decision guide when choosing between the development of a native and Flutter mobile app

- maybe point to future research into Flutter - Reference back to goal in introduction - What are the limitations of my research - Future Research questions - Outlook

Appendix A

Appendix

A.1 Interview Guideline

A.1.1 Interview Setup

- brief interviewee about thesis and their role in the research
- ask if the interview session may be recorded and further processed for scientific inquiry
- ensure technicalities before start of interview with participant:
  - the participant's iPhone is sufficiently charged, has been restarted and is in "Do Not Disturb" mode
  - the participant has installed Quicktime player on their iPhone and are sharing it with their MacBook
  - the participant is sharing their MacBook screen in the video call
  - the participant has received the QR codes for kickdown A and B
- ask the participant if the recording may started

A.1.2 Background Questions

Ask the participant about their...

- job role
- years of experience in the mobile app industry
- previous experience with the Kickdown application

A.1.3 Main Interview

App Start and Scroll Behavior

Instructions

- Please open Kickdown (A/B) and find the [color] [brand] [attribute] car.
- Please open Kickdown (A/B) and find the [color] [brand] [attribute] car.

Questions

1. Are there any differences in terms of user experience in any way between the two apps?
2. If you had to pick one experience over the other, which would you choose (A or B)?

Detail Transition, Modal Transition, Textfield interaction

Instructions

- Please stay in Kickdown (A/B) and tap on a car of your choice. Bid on the car with an amount of your choice.
- Please repeat the process for the Kickdown (A/B). You may choose another car and enter a different amount.

Questions

1. Are there any differences in terms of user experience in any way between the two apps?
2. If you had to pick one experience over the other, which would you choose (A or B)?

Horizontal Scrolling

Instructions

- Please stay in Kickdown (A/B) and open the first posting. Find the picture with the [insert item].
- Please open Kickdown (A/B) and open the second posting. Find the picture with the [insert item].

Questions

1. Are there any differences in terms of user experience in any way between the two apps?
2. If you had to pick one experience over the other, which would you choose (A or B)?

Instructions

- Please stay in Kickdown (A/B) and use the tab navigation to navigate to the "More Screen". Please turn Tracking on.
- Please repeat the process for Kickdown (A/B)

Questions

1. Are there any differences in terms of user experience in any way between the two apps?
2. If you had to pick one experience over the other, which would you choose (A or B)?

Overall

Instructions

- You may test out any functionality of the app that you would like to have.
- *The following questions regard their impression of the entire app*

Questions

1. Are there any differences in terms of user experience in any way between the two apps?
2. If you had to pick one experience over the other, which would you choose (A or B)?