Flutter and iOS Application Comparison

An Empirical 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

Contents

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 *web views*[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 examined in 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. Web views 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 2.2). Thereby, Flutter bypasses host platform communication by avoiding the underlying system app SDK for 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 upside 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 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 facets (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 in

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

February of 2021 (**Apple2021e**). 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 3.1.1). 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

The ensuing chapters are structured as follows: In Chapter 2, the reader is given the necessary background knowledge on the Flutter framework in order to contextualize and understand this

thesis. Chapter 3 presents the chosen study methodology in order to evaluate the research goal. Subsequently, in Chapter 5 the results of the study are evaluated and discussed. Finally, key findings are summarized, related back to the research question and future research opportunities are proposed (Chapter 6).

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). Finally, selected architectural features of the Flutter framework will be explained in more detail. All three 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 eloborations 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 end 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**.

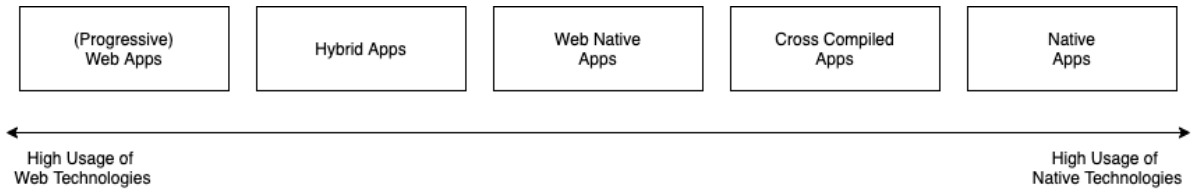

Figure 2.1: Mobile Development Approach Spectrum

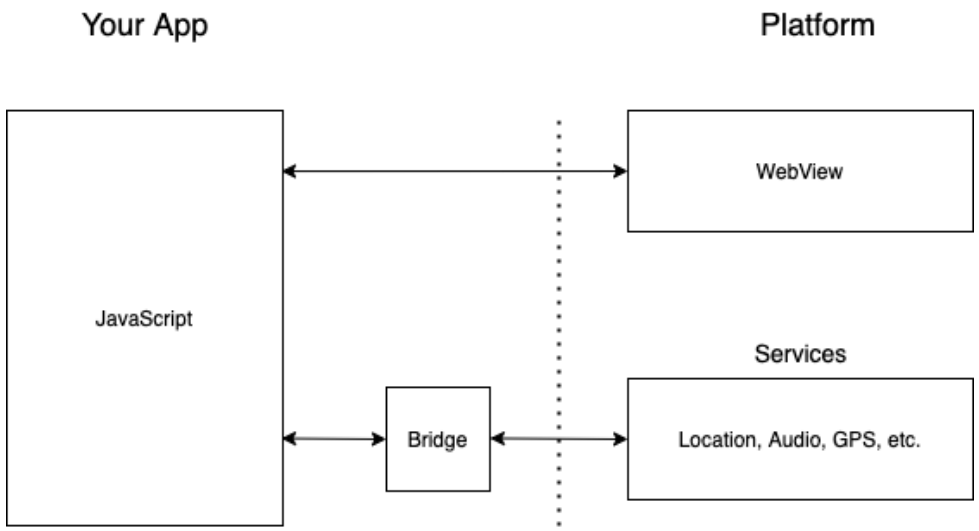

Figure 2.2: Mobile Hybrid Framework Architecture (adapted from **Cunha2018**).

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 Angular (**Angular2021**), React.js (**React2021**) and Vue.js (**Vue2021**). Relying on the Browser and an internet connection, web apps do not have access to hardware capabilites or the file system. Progressive Web Apps (PWAs; **MozillaFoundation2021**) 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, but they are rendered in a platform web view (see Figure 2.2). Additionally they provide the ability to interact with native services such as location, audio and GPS data through a JavaScript 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. 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

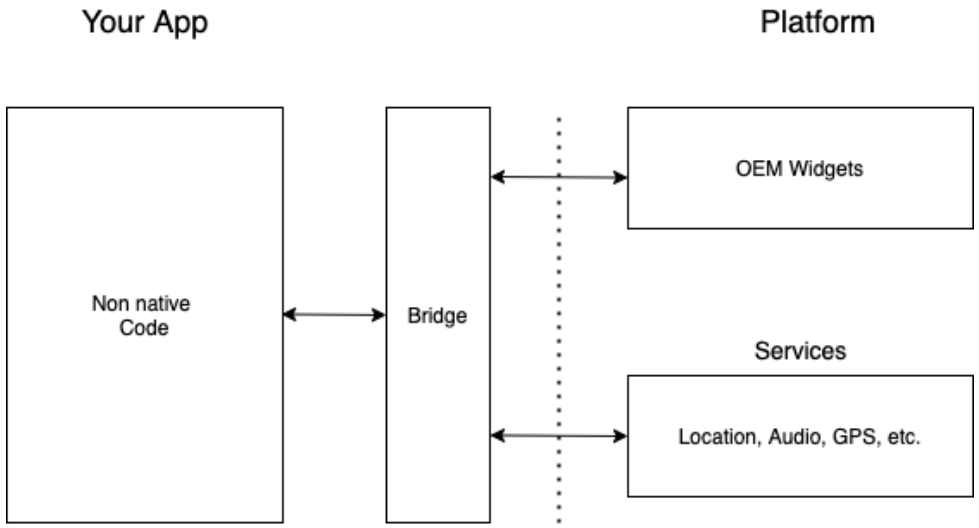

Figure 2.3: Web Native and Cross Compiled Mobile Framework Architecture (adapted from **Cunha2018**).

frameworks for building web native apps include React Native (**ReactNative2021**) and Native Script (**NativeScript2021**).

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 Section 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 (further explained in Section 2.2).

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++.

2.1.6 Summary

The Flutter framework is classfied as a **Cross-Compiled** Mobile Development Approach. Thereby, Flutter is categorically the closest to native mobile development as can be seen in

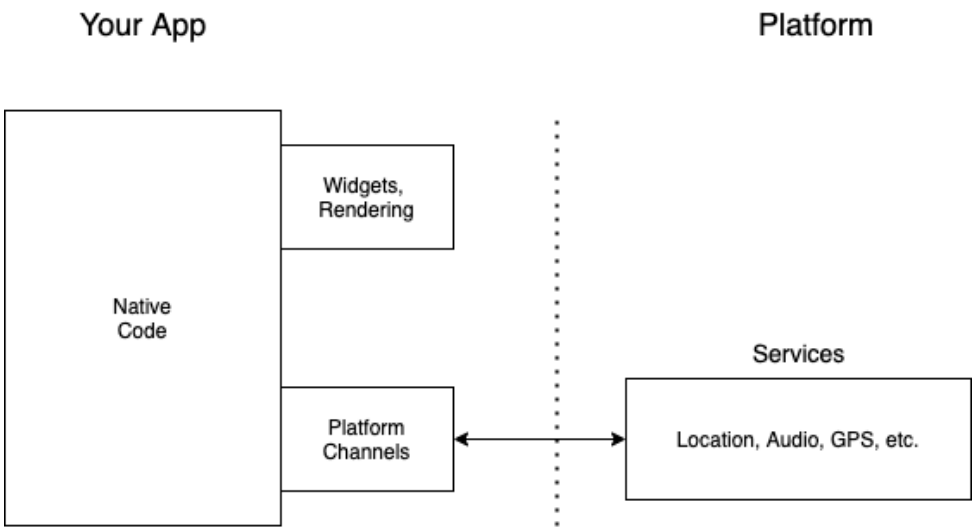

Figure 2.4: Flutter Architecture (adapted from **Cunha2018**).

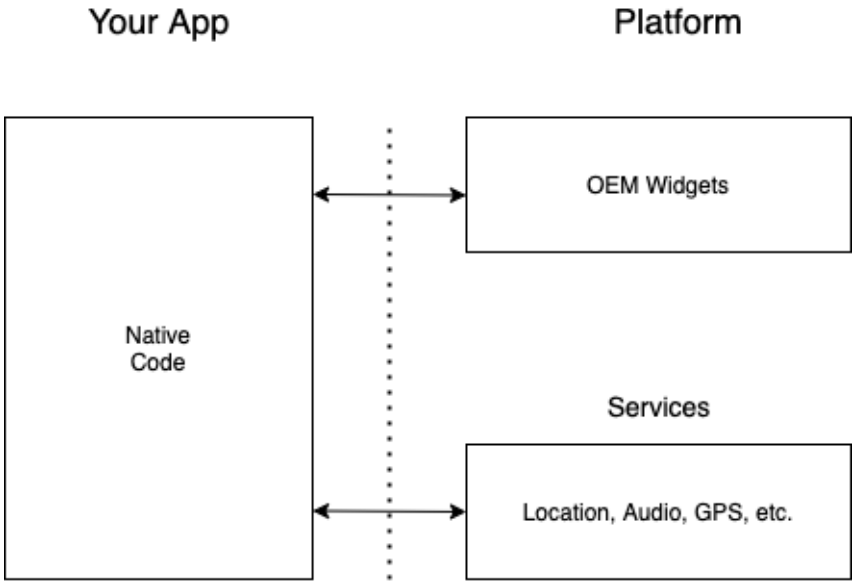

Figure 2.5: Native Mobile Architecture (adapted from **Cunha2018**)

Fig. 2.1. However, Flutter is distinct from other cross-compiled approaches in its technical architecture which is explained in the following section.

2.2 Flutter Framework Architecture

Flutter's architecture is composed of three distinct layers: **framework**, **engine** and **embedder** (see Fig. 2.4).

Framework: 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.

Engine: 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 Section 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 (Dart is Flutter's programming language explained in Section 2.3.1), system event interaction as well as platform channel services (described in 2.3.4) are part of the engine layer.

Embedder: 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 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.

The following sections will go more into depth on particularly interesting parts of Flutter's architecture.

2.3 Presentation of Selected Flutter Features

This Section explores selected specific features of the previously presented Flutter framework architecture (Section 2.2). The aim is to give the reader a more detailed background knowledge of relevant features to contextualize this thesis as well as understand implementation decisions (Chapter 4) and the results presented in Chapter 3.

2.3.1 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 utilize 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.

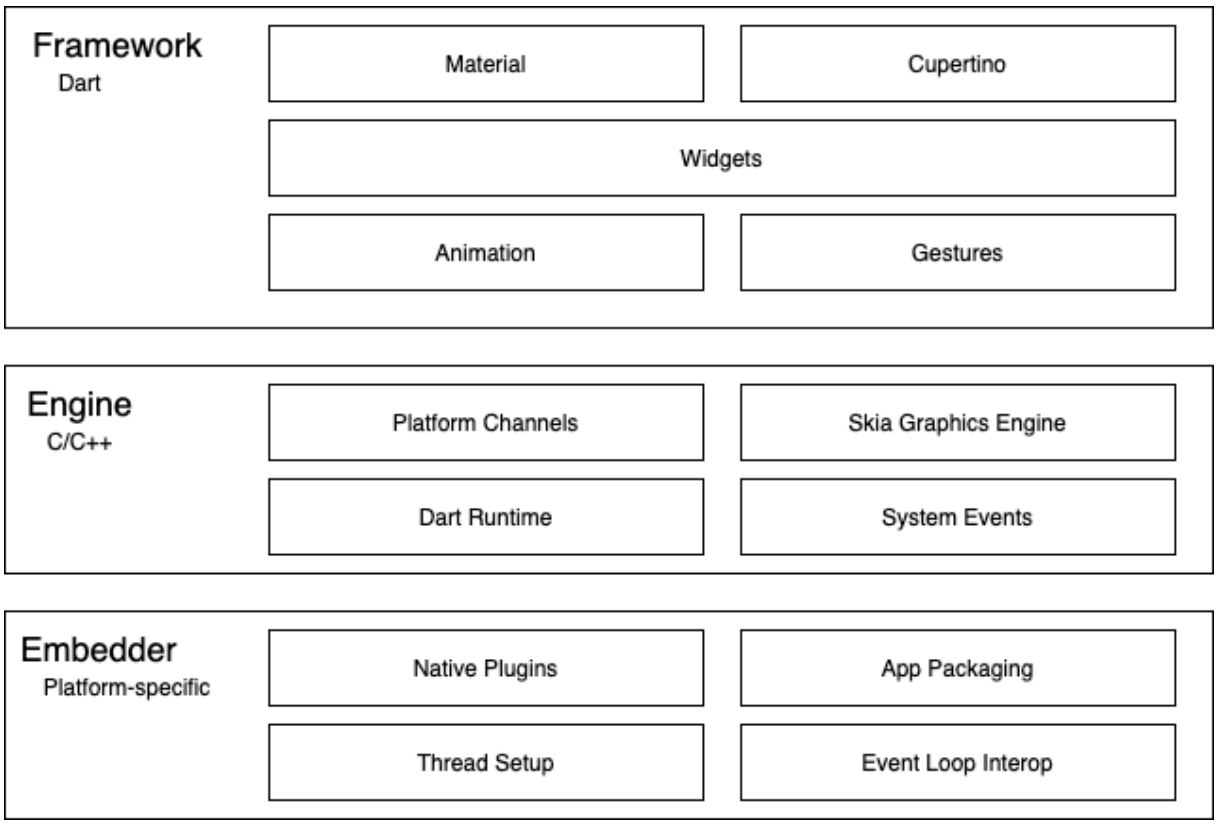

Figure 2.6: Flutter's Layered Architecture (adapted from **FlutterArchitecture2021**)

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**).

2.3.2 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 17):

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

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

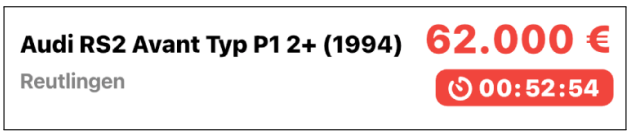

Figure 2.7: Kickdown Posting Overview Car Information UI

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 using the abstraction of **RenderObjects**. When the state changes during runtime Flutter creates a new RenderObject tree, diffs it against the old one, and then redraws only its changes to the screen. RenderObjects are analogous to the **Virtual DOM** diffing of React.js. This reactive 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.

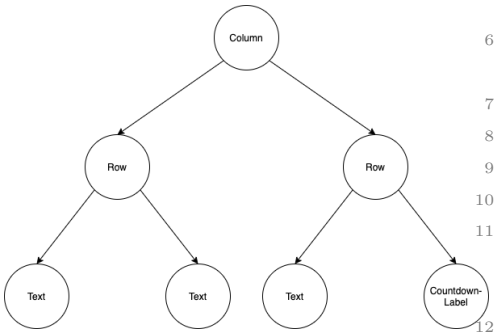

Figure 2.8: Widget Tree Structure Visualization of Code Snippet 17

```
1  Column(
2      children: [
3          Row(
4              children: [
5                  Text("Audi RS2
                       ..."),
6                  Text("62.000 €"
                       ),
7              ],
8          ),
9          Row(
10             children: [
11                 Text("
                       Reutlingen")
                       ,
12                 CountdownLabel
                       (...),
13             ]
14         )
15     ],
16 ),
```

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

2.3.3 Implications of Flutter's Declarative UI Paradigm

Flutter and UIKit are distinct technologies for building UI as they utilize the declarative and imperative programming paradigm respectively. These paradigms fundamentally determine how UI code was written in the Kickdown original and clone application.
The two approaches are detailed by using the example of centering a piece of text which is used in multiple places in the Kickdown app.

UIKit's imperative UI Programming Paradigm

The imperative UI programming paradigm is characterized by explicit instructions to the framework to achieve a desired user interface.
To center text, UIKit takes a considerable amount of instructions as can be seen in Listing 19. Firstly a `UIViewController` is subclassed which manages the entire view hierarchy of the particular screen (**UIViewControllerDocumentation2021**). Furthermore, a `UILabel` - *"A view that displays one or more lines of informational text"* (**UILabelDocumentation2021**) - is created via a closure (see **ClosureDocumentation2021**) for that parent view controller. Inside the closure the `text` property is explicitly set after initialization and the label's `translatesAutoresizingMaskI` property is set to `false` in order for the label to be positioned and sized programmatically. Once the view hierarchy for the view controller is loaded into memory, the `viewDidLoad` function is called by the system (see **viewDidLoadDocumentation2021**). Subsequently, the label is added as a subview and centered horizontally and vertically in the view controller's view based on **Auto Layout** (**AutoLayoutDocumentation2016**) constraints.

```
import UIKit

class InitialViewController: UIViewController {
    private let customTextLabel: UILabel = {
        let label = UILabel()
        label.text = "Custom Text"
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = UIColor.white
        view.addSubview(customTextLabel)
        customTextLabel.centerXAnchor.constraint(equalTo: view.
            centerXAnchor).isActive = true
        customTextLabel.centerYAnchor.constraint(equalTo: view.
            centerYAnchor).isActive = true
    }
}
```

Listing 2.2: iOS UIKit Example of Centering Text on a Screen

```
import "package:flutter/widgets.dart";
class InitialPage extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Center(child: Text("Custom Text"));
    }
}
```

Listing 2.3: Flutter Example of Centering Text on a Screen

Flutter's Declarative UI Programming Paradigm

Contrarily, Flutter's code structure can be more directly mapped to actual user interface elements with its declarative widget composition paradigm.
The same example of centering a piece of text on screen can be written declaratively with Flutter as shown in Listing 8. A widget representing the screen is created by subclassing `StatelessWidget` (see **StatelessWidgetDocumentation2021**). Unlike `StatefulWidget`s (see **StatefulWidgetDocumentation2021**), `StatelessWidget`s do not hold any mutable state and can't change during runtime based on state changes. `StatelessWidget`s have a required `build` method which return any widget child relationships. The method is called once by the system to mount the widget's subtree structure into the global widget tree (explained in Section 2.3.2). Concretely, a `Center` with a `Text` is returned in order to display centered text in the UI.

Advantages and Disadvantages of the Declarative Paradigm

The advantages of using the declarative approach include the entire description of intent in one place and abstraction of details into a clean interface outsourcing implementation complexity to the framework (see **FlutterDeclarative2021**).
However, while Flutter's declarative apprach may be more intuitive for the human mind, it might also introduce a slight performance overhead. This occurs due to the widget subtree diffing calcualation performed for each state change in `StatefulWidget` subtrees (explained in Section 2.3.2).

2.3.4 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 in accordance with the cross-compiled reference architecture introduced in Section 2.1.4. 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.

2.3.5 Flutter's Architectural Limitations

As explained in Section 2.3.2, 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 Selection Process

The procedure for choosing the case study app is based on a 4 step filtering process. Each step may be viewed as a constraint applied upon possible apps progressively reducing the number of potential baseline testing apps:

1. The app is built and maintained by apploft.

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

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 facets commonly appearing in iOS apps are considered for finding the baseline testing app.
For the purposes of finding the baseline app, a **facet** 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 **facets** 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 **facet** if fetched data

is displayed via the UI, but is not a **facet** 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 Guideline

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 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 factes were extracted into Table 3.1 by going through each user interface (constraint 2). Furthermore, a facet had to appear at least twice before being added as a result.
Continuing the filtering process, as per constraint 2 uncommon facets - facets appearing in less than 50% of observed apps - are excluded. This reduces the list of facets 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 an app (**AppleHIGTabBar2021**).
- **Hierarchical navigation** - Screens opened on top of previous screens using a stack structure (**AppleHIGNavigation2021**).
- **Keyboard interaction** - UI for inputing text via a software keyboard.

- **Vertically scrolling collections** - UI collection of items scrolling vertically.
- **Horizontally scrolling collections** - UI collection of items scrolling horizontally.
- **Webview component integration** - Integrated UI component for displaying web content (**AppleHIGWebViews2021**).

Out of the 15 initially tested apps, 5 include all of the above **facets** (conforming to constraint 1 and 2) (see Table 3.2). Kickdown (see Section 3.1.2) 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).

3.1.1 Clone Application Feature Reduction and Implementation

The login and signup mechanism - although a common **facet** - 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 Chapter 4.

3.1.2 Kickdown App Feature Presentation

Kickdown is an online auctioning platform for buying and selling classic and vintag cars (**Kickdown2021**). The original iOS app was built by apploft based on a subset of functionality from the web application and its underlying internal API. The application is divided into two main sections via tab navigation: the overview and the more screen (see Figure 3.1 and 3.2).
On the overview, the user has the ability to scroll through all current offerings. Each offering is shown as a card with a hero image and some basic information including the title, location, current price and remaining time.
Tapping on a posting card brings up a detail view (Figure 3.3) for the respective posting. Besides inspecting detailed information about the particular posting, the user has the option of viewing additional photos in the image gallery (Figure 3.4) and placing a bid via a bottom sheet (Figure 3.5).
The More screen lets the user login, turn on analytics tracking or open informational web views.

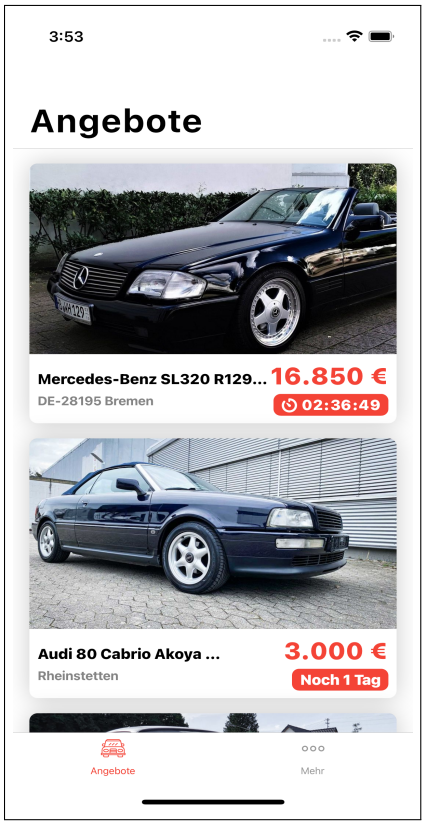

Figure 3.1: Kickdown Postings Overview

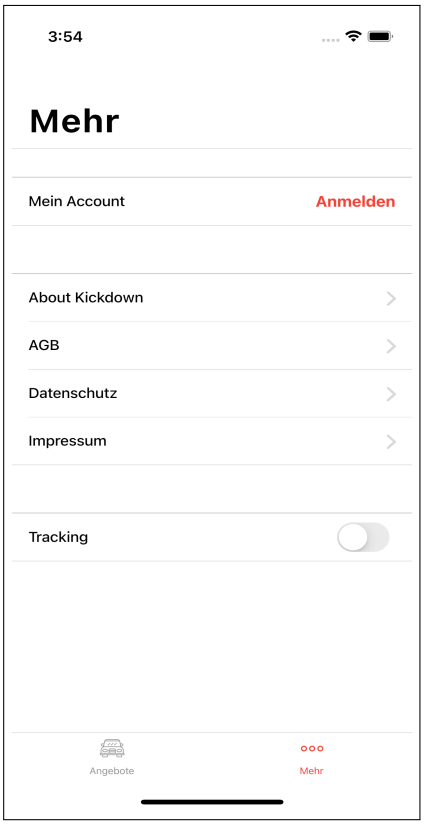

Figure 3.2: Kickdown More Screen

Figure 3.3: Kickdown Detail Screen

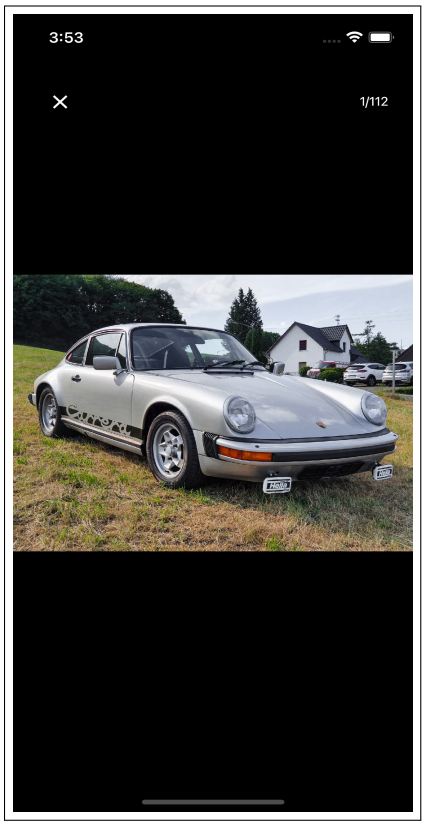

Figure 3.4: Kickdown Gallery View

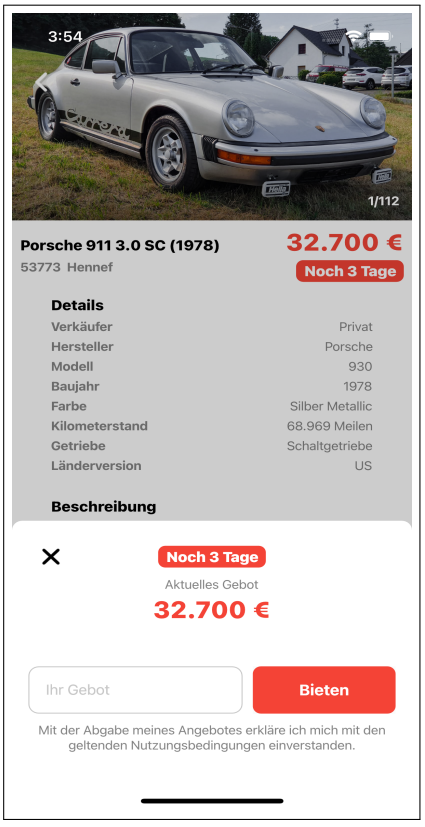

Figure 3.5: Kickdown Bid Preparation Screen

Table 3.1: Initial Facet Extraction for apploft's Apps

App	category	Networking	Login/ Authentication	Maps	Tab Navigation	Stack Navigation	Keyboard Interaction	Vertically Scrolling Collection	Horizontally Scrolling Collection	Webview Components	Camera Interaction
bonprix	shopping	1	1		1	1	1	1	1	1	
couponplatz	shopping	1	1	1	1	1	1	1	1	1	
Fernsehlotterie	lifestyle	1			1	1	1	1	1	1	
Gerolsteiner TrinkCheck	health			1		1		1		1	
Hexal Pollenflug	weather	1			1	1	1	1		1	
HIPP Baby	health	1	1	1	1	1	1	1		1	1
Hipp Bio	food	1				1		1		1	1
Hipp Buddies app	family					1		1	1		
Hipp Windel	shopping	1			1	1		1	1	1	1
Kickdown	shopping	1	1		1	1	1	1	1	1	
Lotto Nds.	entertainment	1	1		1	1	1	1	1	1	
Kulturpunkte	travel	1		1		1		1			
Servus TV	entertainment	1	1		1	1	1	1	1	1	
Starcook	food	1	1			1	1	1	1	1	1
Zeit Online	news	1	1			1	1	1		1	
	9	**13**	**8**	**4**	**9**	**15**	**10**	**15**	**9**	**13**	4

Table 3.2: Table 3.1 After Applying Constraint 2

App	category	Networking	Login/ Authentication	Tab Navigation	Stack Navigation	Vertically Scrolling Collection	Horizontally Scrolling Collection	Webview Components	
bonprix	shopping	1	1	1	1	1	1	1	**7**
couponplatz	shopping	1	1	1	1	1	1	1	**7**
Fernsehlotterie	lifestyle	1		1	1	1	1	1	6
Gerolsteiner TrinkCheck	health				1	1		1	3
Hexal Pollenflug	weather	1		1	1	1		1	5
HIPP Baby	health	1	1	1	1	1		1	6
Hipp Bio	food	1			1	1		1	4
Hipp Buddies app	family				1	1	1		3
Hipp Windel	shopping	1		1	1	1	1	1	6
Kickdown	shopping	1	1	1	1	1	1	1	**7**
Lotto Nds.	entertainment	1	1	1	1	1	1	1	**7**
Kulturpunkte	travel	1			1	1			3
Servus TV	entertainment	1	1	1	1	1	1	1	**7**
Starcook	food	1	1		1	1	1	1	6
Zeit Online	news	1	1		1	1		1	5

3.2 Performance Comparison

The methodology chosen to empirically test the performance hypthesis H_P (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 Section 3.2.1) are easily measured using software tooling (Section 3.2.3) which aids the objectivity, reliability and validity of this particular study methodology. Finally, the data is statistically evaluated in order to make inferences about the hypothesis.

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 is partly responsible for rendering related calculations. For example, Flutter's tree diffing algorithm is handled on the CPU. Therefore, testing CPU utilization assesses whether Flutter's framework processing requirements for rendering related calculations can be fulfilled on the testing hardware (Section 3.2.2).

GPU Utilization

GPU utilization is the GPU equivalent of CPU utilization mentioned in 3.2.1. The GPU is a specialized processing chip optimized graphics computation. The CPU and GPU are tightly integrated on iPhones which make use of the System-on-a-Chip architecture (**Martin2001**, **WikiChip2020**). Therefore a strong correspondence between CPU and GPU usage should be observed in the study. Nonetheless, testing GPU utilization by itself is a valuable metric for determining Flutter's graphic specific processing needs.

Memory Utilization

Memory utilization is the percentage of available memory capacity used for a specific task. A high level of memory usage negatively impacts performance of running tasks as well as interactive responsiveness (**Ljubuncic2015**).

3.2.2 Measurment Process

To reduce measurement confounders, the device is restarted before each individual measurement to ensure that all irrelevant background processes are cancelled.
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 3.1.2) and ensure a sufficient 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.
- **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 Pro 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 6s as the oldest supported device (**Apple2021**). iOS 13 is also the minimum deployment target for the Kickdown app.

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 3.6 and 3.7) is used for CPU (see Paragraph 3.2.1), the **Allocations** tool (see Figure 3.8 and 3.9) for memory usage (see Paragraph 3.2.1) and **Core Animation** tool (see Figure 3.10 and 3.11) for FPS (see Paragraph 3.2.1). For each profiling tool a graph shows the particular metric quantified over time with an associated table showing the exact numeric values with time stamps.

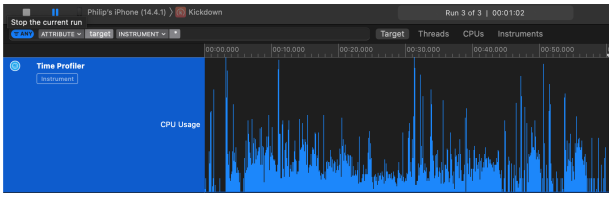

Figure 3.6: Time Profiler Graph

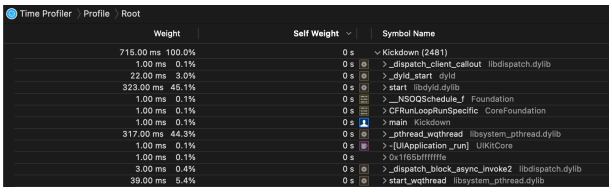

Figure 3.7: Time Profiler Table

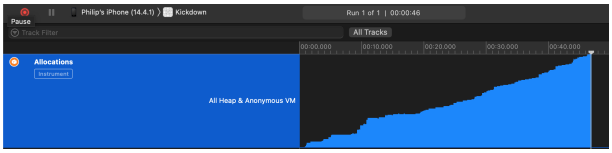

Figure 3.8: Allocations Profiler Graph

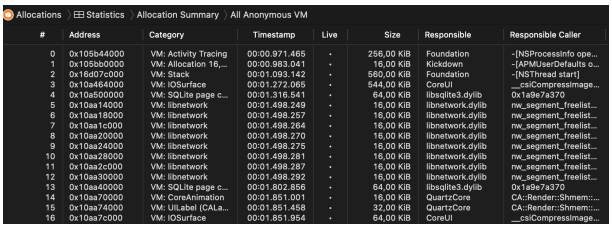

Figure 3.9: Allocations Profiler Table

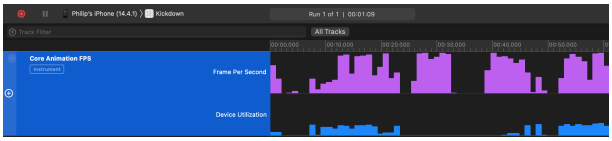

Figure 3.10: Core Animation Profiler Graph

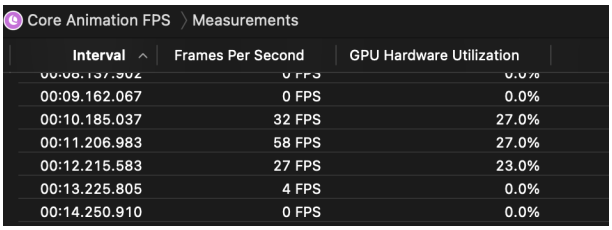

Figure 3.11: Core Animation Profiler Table

3.2.4 Evaluation Process

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

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* (see **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 Section 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 (see **Experts2009**).

As soon as no new perspectives seem to emerge during interviews, no further interviews will be conducted. 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 (see **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 a 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. Due to the ongoing Covid-19 pandemic, the interviews are conducted through video calls, are recoreded with the interviewees consent and the interviewees

are asked to share their iPhone screen via Quicktime player (**Apple2014**). The moderation and recording is facilitated by the author. 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. Both apps are blindly distributed to the participants as "Kickdown A" and "Kickdown B" in order to remove confirmation bias as a confounder (see **Tversky1974**).

3.3.2 Interview Guideline

The interview guideline (see Appendix .1) 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 facets 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 trivalent 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 forms the basis for the evaluation of the usability hypothesis H_U.

Chapter 4

Application Design and Implementation

The methodology utilized to achieve the research objective (see Section 1.3) is to comparatively evaluate a Flutter replica application against an existing native iOS app (detailed in Chapter 3). Thus application design and implementation of the replica will be explored from a perspective of a necessary requirement to conduct both the performance and UX comparison (Sections 3.2 and 3.3) in this chapter.

Naturally, software implementation complexity may impede performance if unscalable algorithms or memory-inefficient datastructures are chosen. Therefore, the goal from a development perspective is to implement the Flutter clone as closely as possible to the baseline application in order to facilitate a fair comparison between both apps.

Consequently, this chapter does not aim to describe the implementation techniques utilized in both apps, but rather contrast selected parts which may affect the outcome of the performance and usability study[1]. Firstly, the specfic framework and programming language versions are mentioned. Secondly, **constant widget precompilation** is introduced as a performance optimization of the Flutter clone (see Section 4.2.2). Finally, the reasoning behind using Cupertino package UI components and a selected deviation from this guideline are explored in Section 4.3.

1. The inclined reader has the opportunity to read through the attached source code of both apps.

4.1 Framework and Programming Language Versions

Preliminarly, it is important to note that the clone app was built with Flutter version 1.22.6 and Dart 2.10.5 whereas the baseline app is built with iOS 13.0 UIKit and Swift 5.0. The underlying implementation of both frameworks and programming languages may potentially affect the system resource usage and therefore also the performance and usability of each app.

4.2 Flutter App Performance Optimizations

This section outlines 2 particularly interesting performance optimizations which were implemented in the Flutter application. Both optimizations do not correspond to the original app as they are Flutter specific.

4.2.1 Partial View Rebuilding

In order to minimize the amount of computation coordinated to the framework, the Flutter clone makes use of **Partial View Rebuilds**. Instead of diffing the entire view tree, the framework only recomputes parts of the UI that are actually affected by the state change. This can be achieved by pushing the state to the leaves of the tree (see **StatefulWidgetPerformance2021**). For example, each Posting has an associated countdown label UI element (as seen on the app overview in Figure 3.1) which needs to calculate its remaining time based on an end date received from the *Kickdown* API every second. By associating a state object - holding the remaining time - with the **CountdownLabel** widget itself rather than the entire or unnecessarily large parts of the view hierarchy, Flutter's internal diffing algorithm runs on a substantially smaller tree data structure.

4.2.2 Constant Widget Precompilation

As part of general performance optimization of the Flutter clone, Dart's `const` constructor is used where possible throughout the codebase. Marking instances with this keyword makes them into immutable compile-time constants. Thereby, `const` widgets are only built once if part of a `StatefulWidget` reducing build cycle times leading to higher frame rates (FPS). Furthermore, instances are **canonicalized** by the compiler making multiple instances the same underlying instance in order to save memory (see **DartConstDocumentation**).
Additionally, this reduces instance tracking and deallocation work for the **Dart garbage col-**

lector (see **DartGarbageCollector2021**).

The process of marking appropriate instances as `const` was facilitated by using the `prefer_const_constructor` linter rule (see **ConstLinterRule2021**) for the Dart Static Analyzer (**DartCodeAnalysis2021**).

4.3 Flutter UI Component Usage

As mentioned in Section 2.2, Flutter uses the Cupertino package (**CupertinoPackageDocumentation2021**) to resemble native iOS UI components from Apple's Human Interface Guidelines. These out-of-the-box components are used where possible as a best practice to map elements of the original app to the Flutter clone. Application elements such as the tab and navigation bar as well as switch controls and page transitions are implemented using Cupertino widgets.

The **More Screen**'s table view (see Fig. 3.2) of the application could not be implemented as easily as in the baseline application as no equivalent Cupertino widgets are available at the time of writing.

4.3.1 More Screen Custom Widget Composition Implementation

The baseline iOS application uses a `UICollectionView` component of the UIKit framework in order to build out the More screen. It is an "[...] object that manages an ordered collection of data items and presents them using customizable layouts" (**UICollectionView2021**). Thereby, the collection view component offers the ability to easily create rows and sections backed by a data source. Furthermore, insets, dividers, detail icons as well as tap interaction UI feedback are easily implemented.

Contrarily, using Flutter there is no equivalent widget for effortlessly creating the desired UI outcome and a custom implementation has been written instead. A simplified version of the code can be seen in Listing 33. A `SliverList` which is a scrollable UI area that places children in a linear array (see **SliverList2021**) is used to layout the individual cells. Each cell is represented by a **CupertinoListTile** which is responsible for adding text and a trailing widget to each cell as well as inserting appropriate dividers and section insets based on the original app specification. Furthermore, it implements a custom `GestureDetector` (see **GestureDetector2021**) which highlights the cell when the user taps down on the cell and triggers an anonymous callback. The callback is used to open webviews using a `SFSafariViewController` (**SFSafariViewController2021**) platform method channel (see Section 2.3.4).

```
SliverList(
    delegate: SliverChildBuilderDelegate(
        (BuildContext context, int index) {
            switch (index) {
            case 0:
                return CupertinoListTile(
                    title: 'Mein Account',
                    trailing: Button03(
                        text: 'Anmelden',
                        onPressed: () {
                            model.onTapLogin();
                        },
                    ),
                    onTap: null,
                    isStartOfSection: true,
                    isEndOfSection: true,
                );
            case 1:
                return CupertinoListTile(
                    title: 'About Kickdown',
                    trailing: DetailIcon(),
                    onTap: () async => model.
                        onTapAboutKickdownTile(),
                    isStartOfSection: true,
                    isEndOfSection: false,
                );
            case 2:
                ...
            }
        },
        childCount: 6,
    ),
);
```

Listing 4.1: Simplified Flutter More View Implementation

Chapter 5

Results

This chapter focuses on the evaluation of the chosen empirical methods and relates back to the originally proposed research question (Section 1.3). Specifically, the executed performance profiling results for both apps along the chosen metrics (see Section 3.2.1) are comparatively evaluated in order to test the validity of the performance hypothesis H_P (Section 1.4). Furthermore, the conducted expert interviews are analysed using the process of interview coding (see Section 3.3.3) to corroborate the usability hypothesis H_U (Section 1.4).

5.1 Performance Comparison

This section presents and contextualizes the performance tracing results based on the benchmarking process for the selected metrics (detailed in Section 3.2). The section concludes with inferences of the results on the performance hypothesis H_P.

5.1.1 General Observations

In most cases, the iPhone 6s consumes less system resources than the iPhone 12 Pro Max (see Section .2). The iPhone 12 Pro Max has a considerably larger screen (6.7 inches vs. 4.7 inches) and a higher pixel density than the 6S (458 pixels/inch vs 326 pixels/inch; **Apple2021d**, **Apple2021c**) naturally requiring more system resources for rendering.

5.1.2 CPU usage

For both the Flutter and original iOS app, the iPhone 12 Pro Max uses more CPU power than the iPhone 6s. Furthermore, for every user action/iPhone combination, the CPU usage is

greater than 100%. This doesn't mean that the processing unit is overloaded as the measured CPU usage is in relation to a single core. The iPhone 12 Pro Max has 2 high performance cores and 4 smaller battery optimized cores. Based on the task, the operating system delegates tasks to either a high performance or a battery optimized core. Whereas the iPhone 6s simply has 2 equally clocked cores for typical parallel work. In terms of CPU usage, it would be errorprone to combine values from the two different phones into a single metric. Therefore, the following metrics are calculated for the two iPhones separately.

Overall, the Flutter clone uses more CPU resources during every single test case when averaged over time (see Fig. 5.1) on both test hardware devices. On average, the Flutter clone uses 43% more CPU power when testing on the iPhone 12 Pro Max while it uses 17% more on the iPhone 6s.

During the *detail view transition*, the Flutter clone has the most similar CPU performance with an additional 19% (12 Pro Max) and 14% (6s) respectively compared to the original iOS app.

The standard deviation of the averaged CPU usage between the use cases lies between 6 and 17% meaning that the CPU requirement for the typical use cases is similar. Contrarily, when looking at the at each individual use case for the averaged run, the CPU usage deviation is very high in every case (app start: 48.17%, detail transition: 53.16%, image gallery: 51.90%, scrolling: 52.12%). This should be mostly due to the fact that the CPU has different load requirements based on the current executed task which may be executed in milliseconds after which it may be idle if no other task is in the queue.

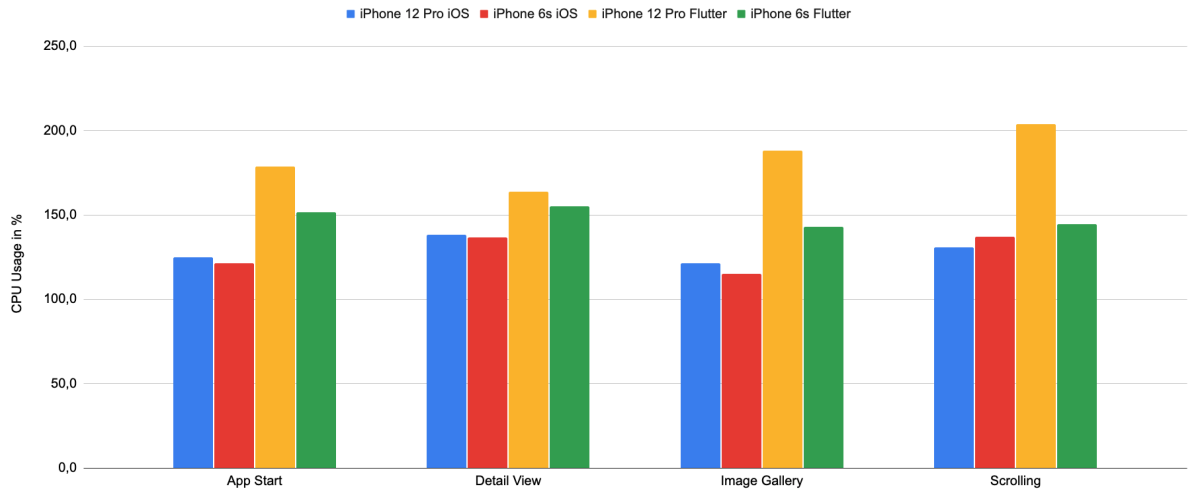

Figure 5.1: Averaged CPU Usage Summary

5.1.3 Memory Usage

The memory usage grows quite steadily throughout the execution of each user action (as can be seen in Figures 2, 5, 8 and 11) as opposed to the CPU usage (Section 5.1.2). When starting the application, a required amount of information is loaded into memory and as the use case is performed more information (i.e. more variables) are added into the memory units.

Flutter is less efficient in terms of its memory usage. Each use case consumes between 53.4 MiB[1] and 67.3 MiB of memory while the iOS baseline app only uses between 32.3 and 41.1 MiB of memory. Thereby, the clone app uses double the amount of memory on average (124.7% additional usage). This may be explained by the fact, that the Flutter app contains its own rendering engine requiring additional space whereas the native rendering engine is built into the operating system itself.

However, the additional memory utilization the Flutter app requires (32.9 MiB) equates to 0.6% and 1.7% of the total RAM capacity for the 12 Pro Max and 6s[2] respectively.

Interestingly, the memory growth throughout each use case is smaller than on iOS (as can be seen in Figures 2, 5, 8 and 11) indicating that Flutter efficiently scales memory during app runtime. In addition, the additional 23.7 MiB the Flutter app consumes on average is negliglible as it is 0.4% of the iPhone 12 Pro Max (6 GB) and 1.2% of the iPhone 6s (2 GB) RAM capacity.

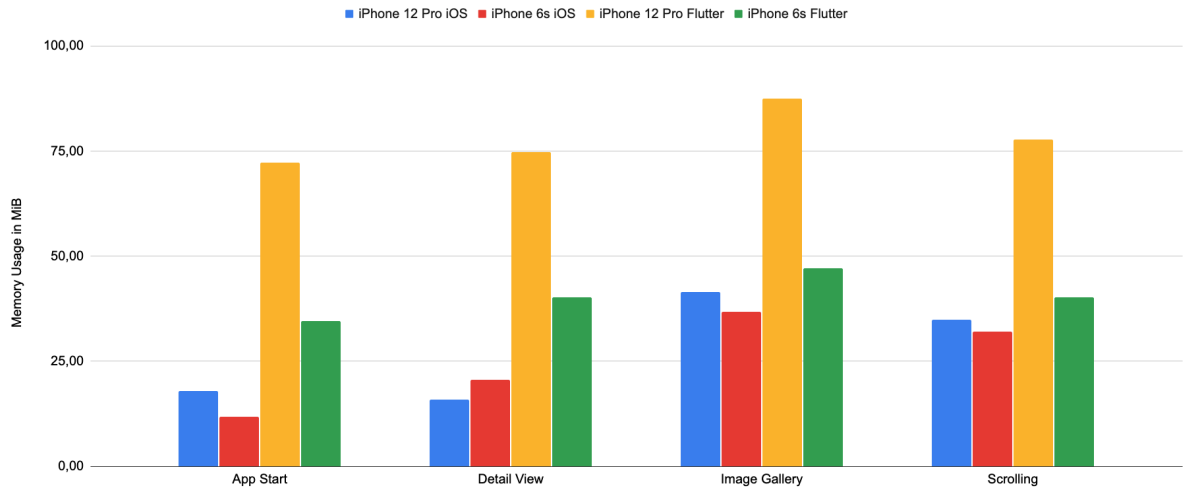

Figure 5.2: Averaged Memory Usage Summary

1. Mebibytes (MiB) are the base 2 equivalent (1,048,576) of Megabytes (MB) (1,000,000)
2. The iPhone 12 Pro Max has 6GB RAM (**GSMArena12ProMax2020**) capacity while the iPhone 6s has 2GB RAM (**GSMArena2015**) capacity

5.1.4 GPU Usage

On average the Flutter app consumes 38% more GPU than the Flutter application. The GPU usage lies between 18.7% 21.9% for Flutter and between 24.2% and 37.4% on iOS (averaged for each use case). The Skia engine of the Flutter app (see Section 2.2) may use the CPU for certain calculations like tree diffing (being a typical CPU task, Section 2.3.2) while iOS is optimized to perform graphics related calculations on the GPU. This requirement inbalance may then also explain the additional CPU usage of the Flutter clone.

Furthermore, throughout each use case the iOS app is subject to larger gradient variation than the Flutter application (see Figures 3, 9, 6 and 12). This may be related to the fact that iOS demands more GPU usage at points in time where intensive graphical computations are required.

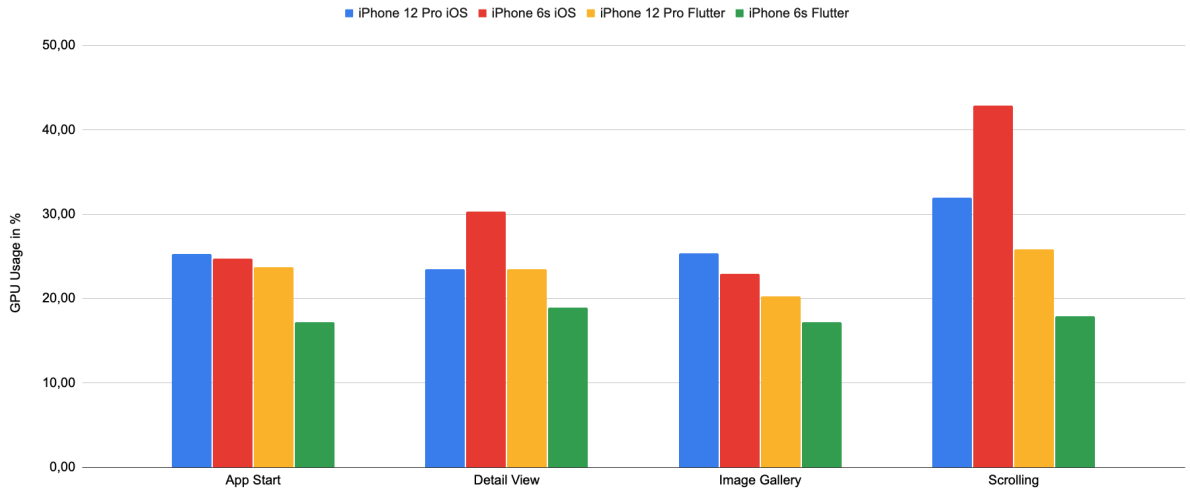

Figure 5.3: Averaged GPU Usage Summary

5.1.5 Hyptothesis Evaluation

Overall, the Flutter application can be said to have a similar performance as iOS. The CPU usage is slightly worse while the GPU usage is slightly better for the Flutter app. In relation, the memory usage is significantly higher in the Flutter clone. However, the relative memory usage for modern phones is sufficiently low and memory growth seems to be slower than on iOS.

5.2 Usability Comparison

This section presents and contextualizes the results from the expert interview coding process (see Section 3.3.3).

In most cases the expert interview participants did not prefer either of the two user experiences as can be seen in Table 5.1. Particularly, cases where no differences could be perceived include:

- opening the application
- switch control interaction
- vertical scrolling on the overview page
- horizontal scrolling in the image gallery

Out of the above, cases 3 and 4 are presumably the most resource intensive UI operations out of all user actions as the collection items are dynamically fetched from a networked data source and are continuously rerendered based on the current users' scroll position. Interestingly, the Flutter clone can provide the same perceived user experience as the iOS original.

When differences were noticed, the participants had to go through a particular use case multiple times in order to detect divergences between the two apps. These could be categorized with the interview coding process (see Table 5.2): *Cross-Cutting differences*, *Detail transition*, *Modal transition*, *Textfield keyboard animation*, *Other specific differences*.
The following sections explain the above categories in detail.

Table 5.1: Interview Use Case Cumulated Preference Choices

Use Case	Preferred iOS App	Preffered Flutter App	Indifferent	Total
App Start & Scroll Behavior	1	0	4	5
Detail Transition, Modal Transition, Textfield interaction	4	1	0	5
Horizontal Scrolling	1	0	4	5
Switch Control & Web View	0	0	5	5
	6	1	13	

Table 5.2: Interview Coding Results

General differences	Flutter App doesn't support dynamic type
Detail Transition differences	Flutter App has more interesting detail transition (mentioned twice) iOS App uses typical navigation stack master-detail transition
Modal Transition	iOS App has more natural bottom sheet animation (mentioned thrice): - Bottom Sheet is not draggable - Animation curve of Flutter app is too linear
Textfield interaction	iOS keyboard animation feels more natural (mentioned twice gap during keyboard animation
Other specific differences	Countdown Timer Spacing is slightly larger in Flutter Status Bar Shadow is not implemented such that lighter images may be seen better Webview close button title is different ("Schließen" vs "Fertig") Alert dialog font letter spacing is larger in Flutter clone app

5.2.1 Cross-Cutting Differences

One participant noticed that the Flutter app did not have dynamic type support. Dynamic type is Apple's accessibility feature for users who need larger text for better readability based on their preferred system text size setting (**Apple2021b**). According to Google, Flutter scales *Text* widgets automatically based on the respective OS setting (**Google2021a**). Upon further inspection, the Flutter app did scale its text, however the scaling factor seemed smaller than on iOS.

5.2.2 Detail Transition

The transition between the overview screen (Figure 3.1) and the detail screen (Figure 3.3) was implemented slightly differently in the Flutter application. The exact navigation stack push/pop transitions of iOS are not yet integrated into the Cupertino package of Flutter. Therefore a custom transition was written for this screen transition. One participant in the usability study noted that he could immediately detect the original app by this transition while two other participants actually preferred the transition in the Flutter clone. Flutter is currently not able to fully mimic the native iOS navigation animation concept. However, transitions that multiple interview participants described as pleasant can be created with Flutter's integrated animation library.

5.2.3 Modal Transition

The default iOS modal presentation style implemented in the bid UI (see Figure 3.5) could not be fully replicated with Flutter as can be seen in Figure 5.4 and 5.5. Three out of the five participants noted this difference. They observed that the modally presented screen did not seem as natural as it was not draggable and its presentation animation curve appeared to be linear.

Flutter's animation system is rich enough to theoretically fully replicate the iOS modal screen transition. However, it would require a substantial amount of code as there are multiple animations running synchronously which may also reverse based on gestures. Furthermore, this process would involve a lot of trial and error since the original transition is closed source. Ideally, the Flutter framework itself will replicate this screen transition and it can simply be set via an argument in the navigation function.

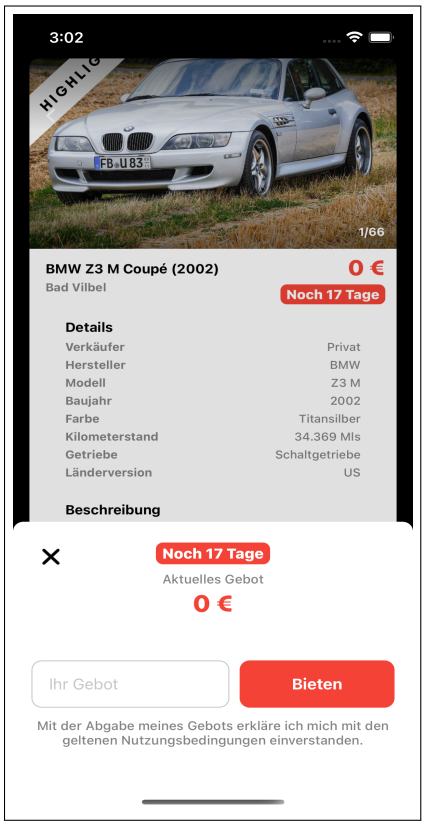

Figure 5.4: Bid UI iOS Original

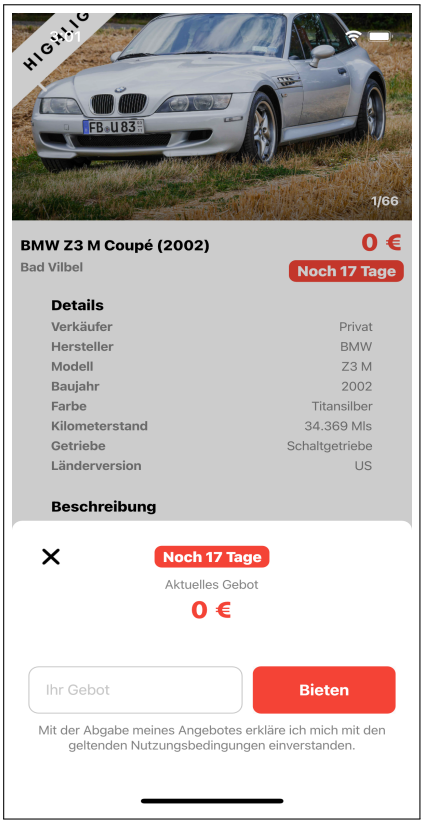

Figure 5.5: Bid UI Flutter Clone

5.2.4 Textfield Keyboard Animation

When interacting with the textfield in the bid UI (see Figure 3.5), the keyboard automatically animates from the bottom for the user. The animation of the original iOS app felt more natural to three participants in the usability study. Further inspection of the Flutter app showed that the keyboard animation is slightly slower than the corresponding animation of the bottom sheet. The user expects that the keyboard pushes up the bottom sheet and they move synchronously, yet in the Flutter app a small gap appears between the two elements for the animation duration due to unmatched speed of the two UI elements. This difference wasn't noticed during the development of the clone application. The animation curve of the sheet may be changed while the keyboard transition is handled by the framework and not configurable. However, simply slowing down the sheet transition should produce a comparable effect as in the original application.

5.2.5 Other Specific Differences

During the conduction of the interviews, individual participants noticed small but interesting differences between the two applications. These fall under the "Other Specific Differences"

category resulting from the interview coding process and will be presented in this section.

One participant noted that the font letter spacing of the alert dialog of the Flutter application is slightly larger as can be seen in Figure 5.6 and 5.7. The alert dialog of the Flutter Cupertino package doesn't implement the font correctly and an open Github issue exists since Feb 17, 2020 (**FlutterCommunity2020**).

The other differences (see Table 5.2) in this category were pointed out by a single participant and the implementation effort of correcting these differences in the Flutter clone are negligible.

5.2.6 Usability Hypothesis Evaluation

Overall, Flutter can fully reconstruct native iOS user interfaces for typical mobile application facets. Furthermore, scrolling and animation fluidity is comparable with Flutter.

However, navigation animations for screen transitions do not look and feel native for iOS users when building with Flutter. Additionally, the iOS system alert dialog styling is slightly different.

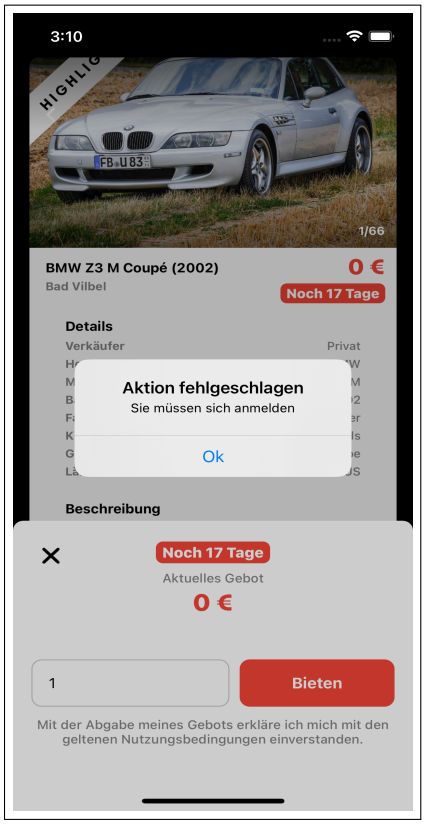

Figure 5.6: Alert Dialog iOS Original

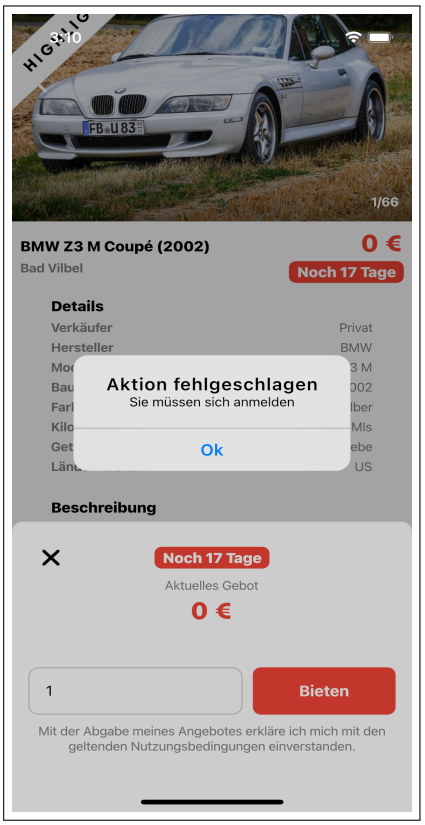

Figure 5.7: Alert Dialog Flutter Clone

Chapter 6

Summary

The aim of this thesis was to evaluate whether apps built with the Flutter framework yield comparable performance and usability. As presented in Section 5.1.5, the Flutter clone application had similar system requirements in terms of CPU, memory and GPU usage. This likely enables the comparable animation fluidity (see Section 5.2.6) tested in the usability study of this thesis. Furthermore, Flutter allows for the creation of iOS interfaces while the animations for screen transitions aren't easily reproducible with Flutter. Therefore, both hypothesis H_P and H_U (laid out in Section 1.3) can be verified in the sense that both performance and usability are comparable with Flutter, but not exactly the same due to small differences summarized above and specifically elaborated on in Chapter 5. Ideally, a statistical significance test would have been conducted in order to find out whether the use of Flutter is statistically significant for determining the performance differences. However, the sample size of n = 3 for each user action, phone, framework combination is too low to have a representative outcome. In future research, testing and measurement could be automated using capture replay technology and transferred into an evaluable data representation. Therefore, it would be useful to conduct a similar study with a larger sample size. Besides validating the findings of this thesis in terms of performance and usability, comparing development time complexity for building common user interfaces would be an interesting future research persuit. Thereby, the business implications mentioned in Section 1.1 can be more easily quantified and the choice when to use Flutter properly weighed against building native apps. The general formalization of a decision process for choosing a UI framework technology for mobile applications is an interesting future research endevour based on this thesis. Specific data patterns described in this thesis, may be empirically verified or falsified by future research. For example, the assumption that Flutter has a fixed

amount of memory allocated for each app due to the rendering engine requirements has a slow memory growth may be empirically tested. The memory growth seen throughout the use cases in this thesis is slower than on iOS suggesting that there is a break-even point. The implications for this specific assumption are therefore interesting regarding large and complex applications which might then be more memory-efficient than native iOS apps.

.1 Interview Guideline

.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

.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

.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)?

.2 Performance Result Graphs

.2.1 App Start

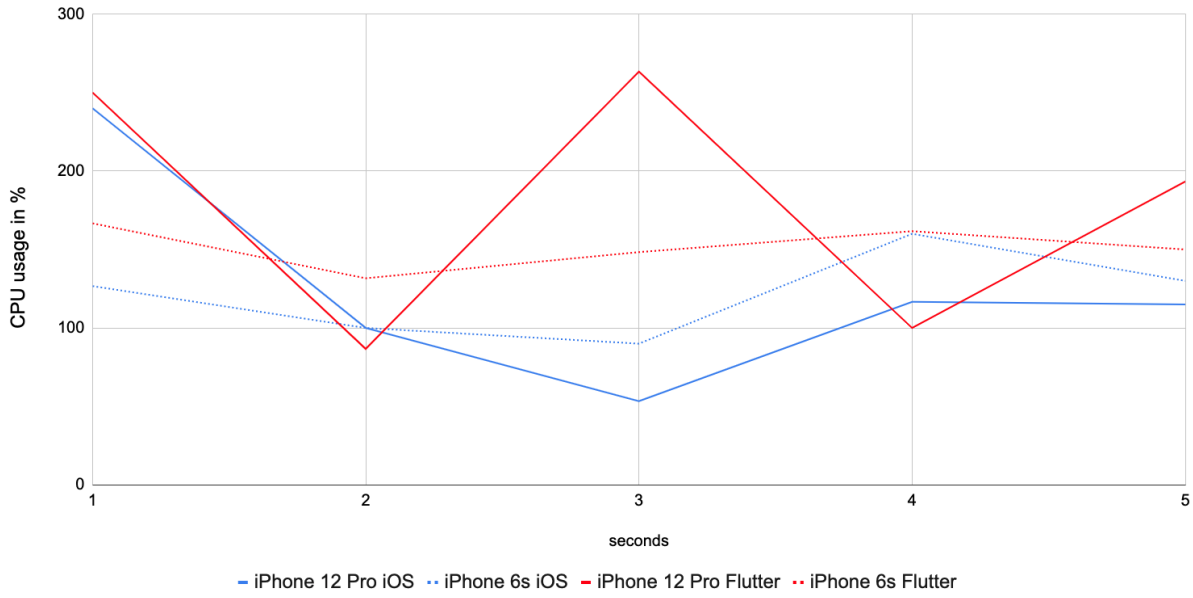

Figure 1: Averaged CPU Usage App Start

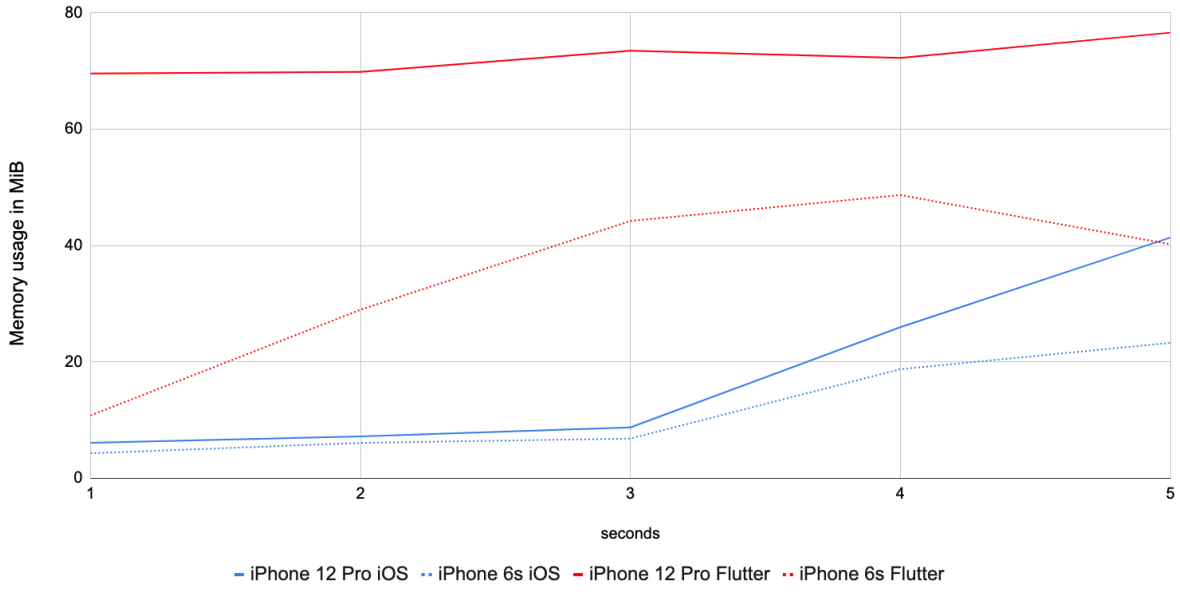

Figure 2: Averaged Memory Usage App Start

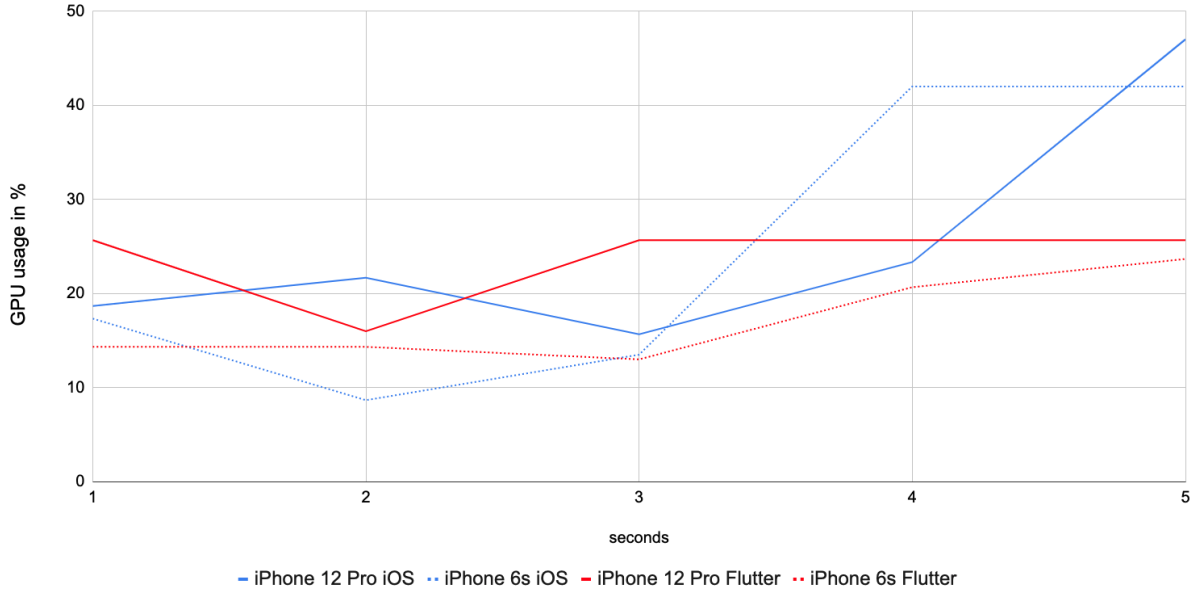

Figure 3: Averaged GPU Usage App Start

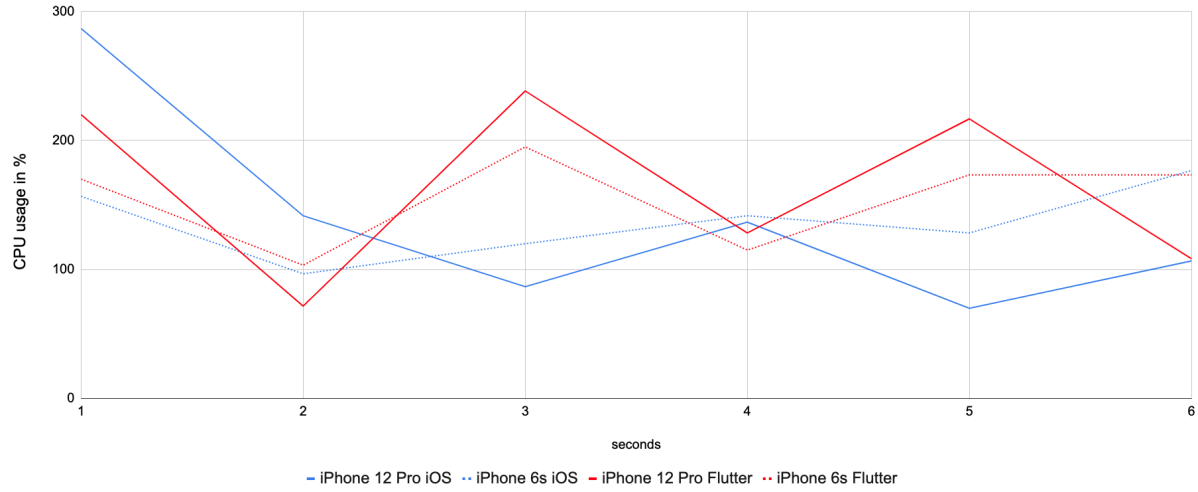

Figure 4: Averaged CPU Usage Detail View

.2.2 Detail View

.2.3 Image Gallery

.2.4 Scrolling

.3 Interview Transcriptions

A: Author

I: Interviewee

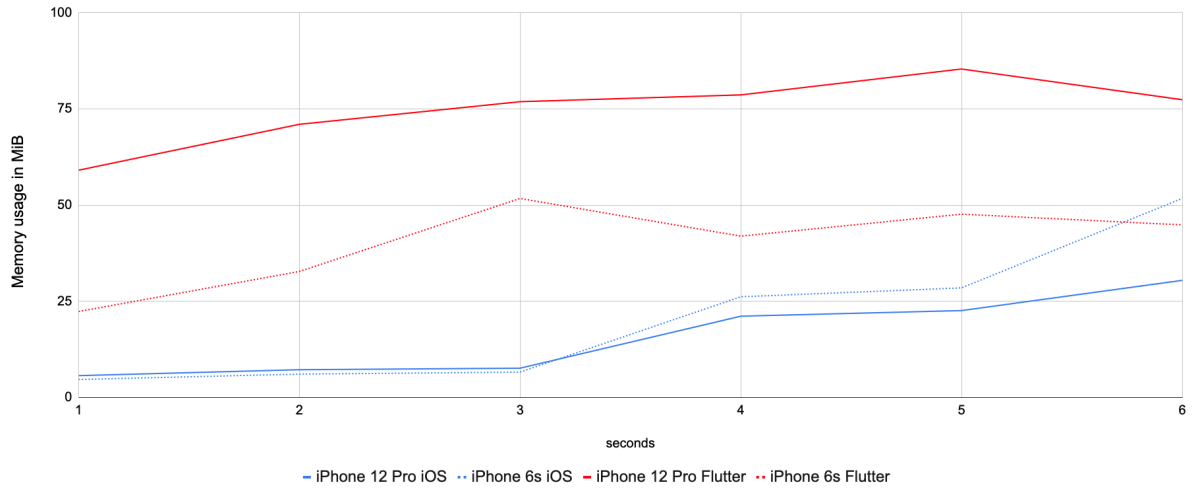

Figure 5: Averaged Memory Usage Detail View

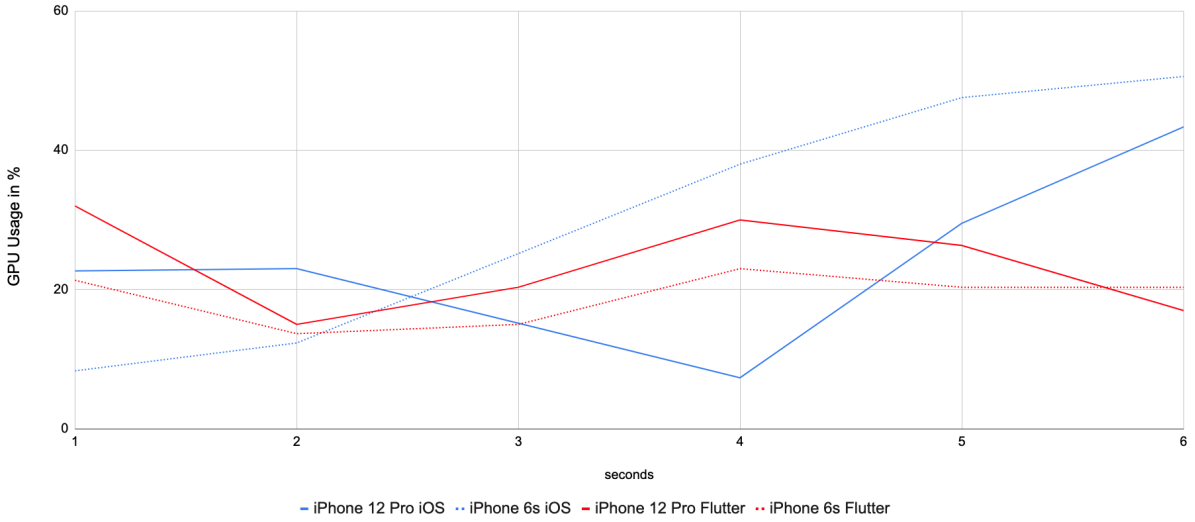

Figure 6: Averaged GPU Usage Image Gallery

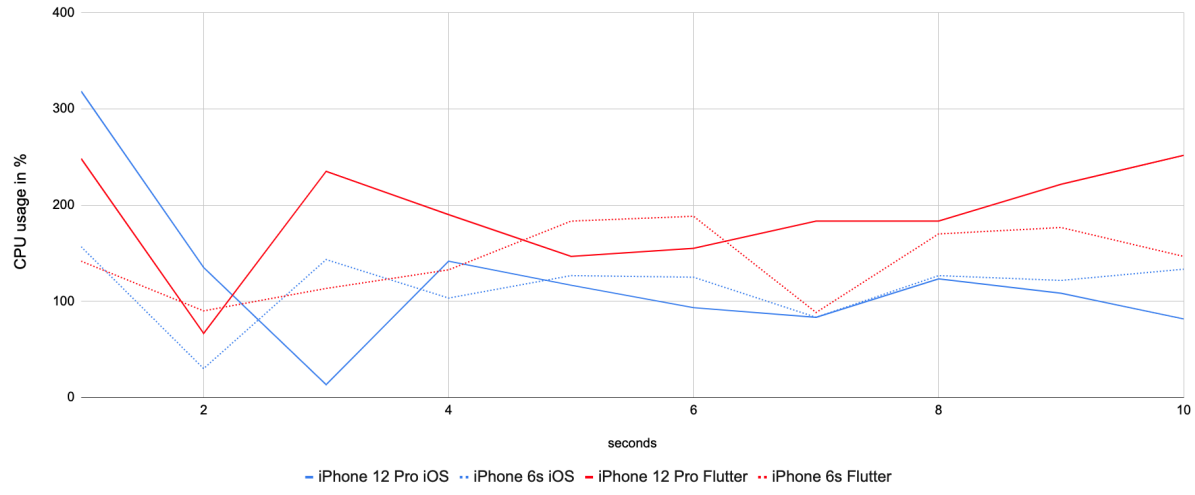

Figure 7: Averaged CPU Usage Image Gallery

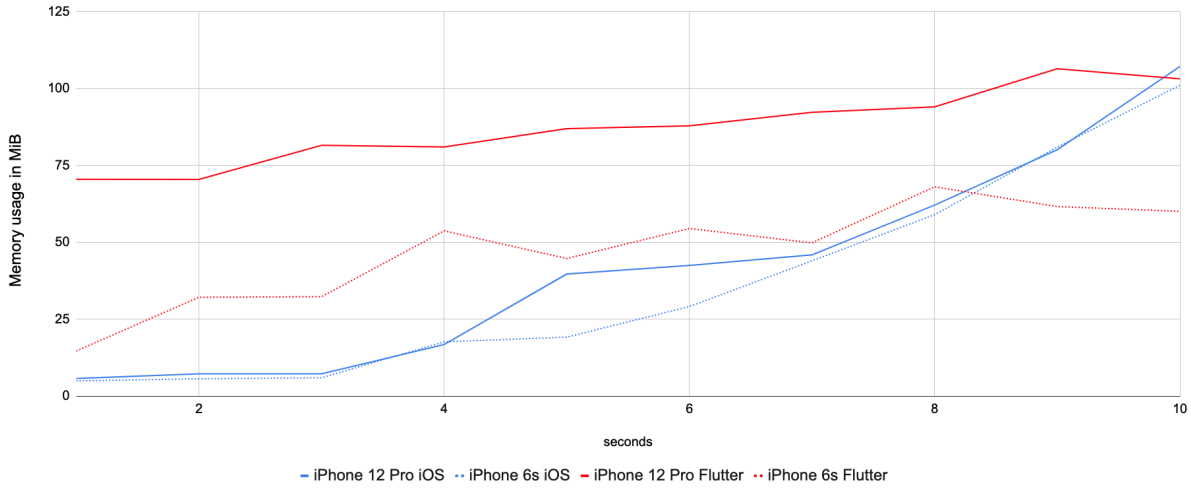

Figure 8: Averaged Memory Usage Image Gallery

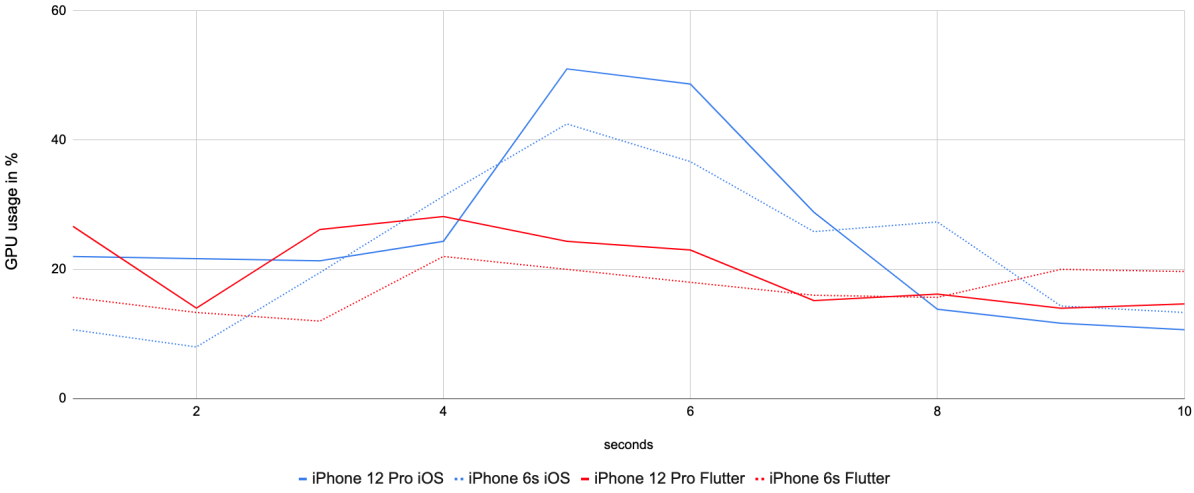

Figure 9: Averaged GPU Usage Image Gallery

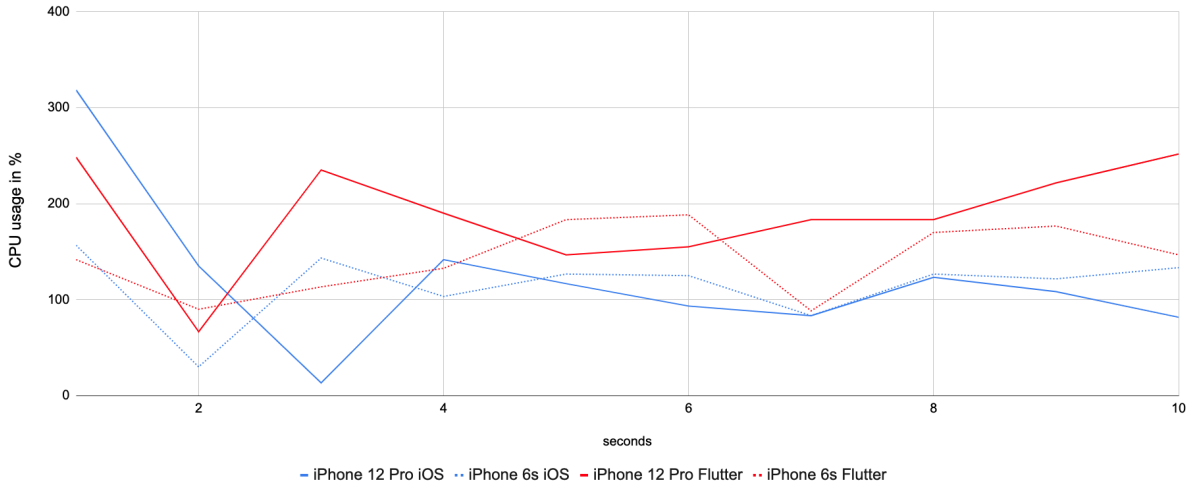

Figure 10: Averaged CPU Usage Scrolling

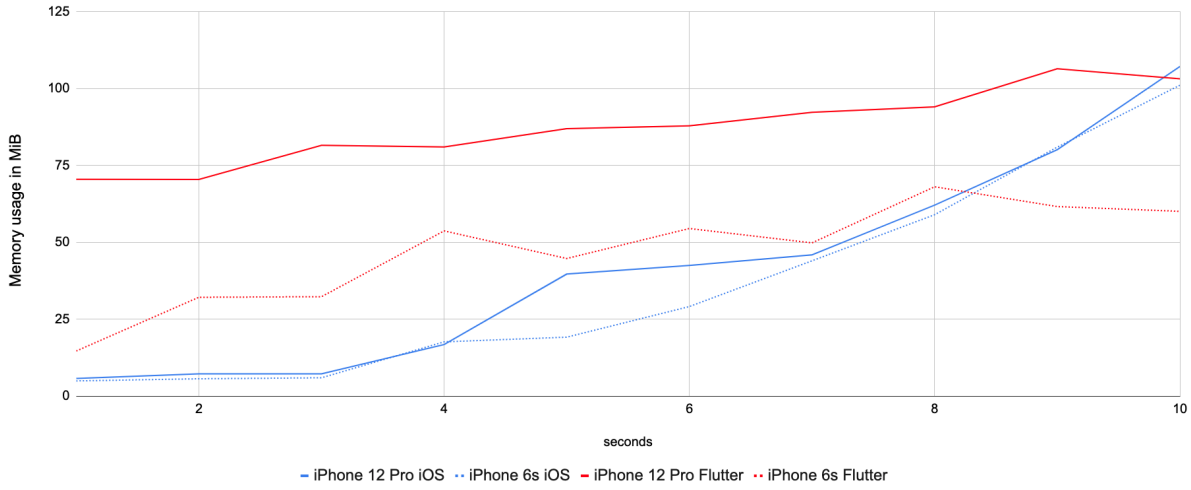

Figure 11: Averaged Memory Usage Scrolling

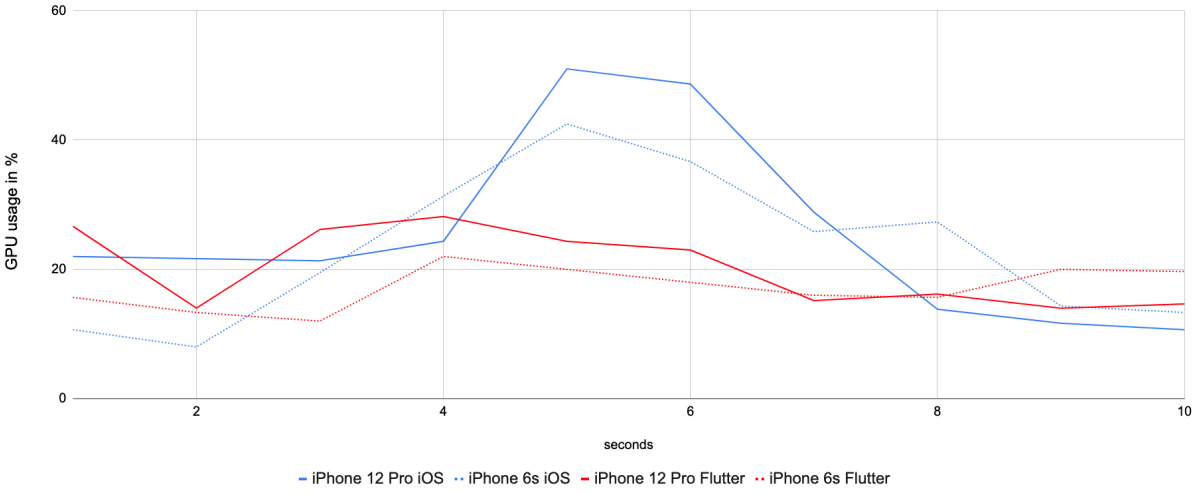

Figure 12: Averaged GPU Usage Scrolling

.3.1 Interview 1

Date: 07.04.21

Job title: Junior UI/UX Designer

Years working in mobile industry: 1,5

Experience with Kickdown App: Has heard of the app, but never used it

A: Öffne Kickdown A und suche den weißen Jaguar aus Böblingen. Dann einmal bitte Kickdown B öffnen und den Golf 1 Cabrio aus Hamburg finden. Konntest du irgendwelche Unterschiede feststellen zwischen den beiden Apps im nachhinein?

I: Also eigentlich sieht es vom UI sehr gleich aus und beim Scrollen konnte ich auch nichts feststellen.

A: Ja, alles gut. Und wenn du dich entscheiden müsstest zwischen eine der beiden Nutzererfahrungen, welche wäre das? "Beide sind" gleich ist auch ok.

I: Also ich kann es nicht sagen, welche ich besser finde. Finde es schwierig.

A: Das ist auch in Ordnung. Ist auch eine gute Antwort. Dann würde ich dich bitten B zu öffnen und dir da dein Lieblingsauto rauszusuchen. Dann kannst du mal rauf tappen. Jetzt kannst du bieten und dort deine Lieblingszahl eingeben.

I: Hier kommt jetzt ein Dialog.

A: Das ist weil du dich nicht angemeldet hast. Das ist in Ordnung. Magst du das gleiche Prozedere bitte einmal für A wiederholen

I: Hier ist jetzt eine andere Transition. Die fand ich bei B irgendwie cooler. Das war so eine Hero Animation. Aber das sheet kommt hier smoother hoch. Das war bei der anderen so abgehackt. Lass mich das nochmal in der anderen App ansehen. Ja, da ist so eine Lücke beim Hochfahren. Die Transition von dem Action Sheet ist schneller als die von der Tastatur und dadurch kommt diese Lücke zustande

A: Wenn du nochmal insgesamt überlegst, also die Transition auf den Screen und dann das Hochfahren des Screens. Wie würdest du diesen ganzen Flow bewerten, wenn du dich zwischen den beiden Apps entscheiden müsstest?

I: Ich fand die Transition bei A besser, aber die Bieten UI bei B besser. Daher würde ich insgesamt sagen beide sind gleich. Aber das ist schwierig.

A: Ok. Dann öffne doch bitte noch einmal Kickdown A. Und von dem ersten Auto einmal das Bild raussuchen, wo die ganzen Zettel zu sehen sind. Du kannst die Gallery öffnen, indem du auf das Bild tippst. Da hast du es auch schon.

I: Oh, das sind wirklich viele Zettel

A: Und in der B kannst du dir nochmal noch das 2. Auto vornehmen und das Bild mit den Zetteln heraussuchen. Es sind nicht ganz so viele, aber hier solltest du einen "Oldheimer Gutachten" sehen können.

I: Hier ist er. Jetzt fragst du mich bestimmt wieder, welches ich besser fand.

A: So langsam erkennst du glaube ich auch den Aufbau dieses Interviews. Konntest du zwischen den beiden Apps irgendwelche Unterschiede feststellen in Bezug auf die Usability?

I: Also das Öffnen der Gallerie funktioniert bei beiden gleich. Das kommt so von unten nach oben rein. Beim Swipen habe ich das Gefühl, dass es bei B erst noch lädt. Wobei das wird animiert oder?

A: Genau, das ist eine Fade Animation.

I: Achso, das finde ich eigentlich schöner. Und sonst finde ich auch keine Unterschiede beim Zoomen oder wenn ich schnell swipe.

A: Dann habe ich nochmal einen letzten Use Case den wir nochmal durchgehen können. Und zwar, wenn du wieder App B öffnest und dort die Mehr Seite öffnest und dort Tracking aktivierst. Und dann kannst du einmal auf Datenschutz tippen. Das gleiche kannst du nochmal bitte für A machen. Du kannst dir eine andere Seite raussuchen, z.B. Impressum.

I: Also bei der App A steht "Fertig" und bei der App B steht "Schließen". Ansonsten sieht das gleich aus.

A: Und beim Switch, erkennst du da irgendwelche Unterschiede?

I: Ne, also die sind auch genau gleich

A: Also dann konntest du hier insgesamt auch keinen Unterschied feststellen?

I: Ne.

A: Ok - vielen Dank!

.3.2 Interview 2

Date: 07.04.21

Job title: UI/UX Designer

Years working in mobile industry: 8

Experience with Kickdown App: supported conceptual development of application

A: Kannst du einmal bitte Kickdown A öffnen und den weißen Jaguar aus Böblingen heraussuchen. Und dann würde ich dich bitten in der App B den Golf 1 Cabrio aus Hamburg herauszusuchen

I: Hab ich.

A: Perfekt. Konntest du beim Öffnen der App und beim Scrollen irgendwelche Unterschiede feststellen?

I: Also in App B sind die Abstände vom Countdown Label etwas anders als in der Original App. Das Spacing zwischen den Zahlen ist etwas größer. Aber sonst sind sie wirklich sehr gleich.

A: Und wenn du dich zwischen den beiden Apps für eine entscheiden müsstest, welche wäre das dann?

I: Das kann ich nicht sagen. Ich finde beides sehr ähnlich.

A: Kannst du bitte einmal in App B ein Auto deiner Wahl aussuchen und kannst auf das Auto bieten. Du bist nicht eingeloggt von daher sollte nichts passieren. Wenn du durch bist, kannst du das gleiche noch einmal in App A machen. Konntest du irgendwelche Unterschiede feststellen? Sowohl bei der Transition auf den Detail View als auch beim Bieten selbst?

I: Da konnte ich im ersten Moment nichts feststellen. Kann ich das ganze auch nochmal durchspielen?

A: Klar. Du kannst dir beide Apps nochmals genauer anschauen.

I: Ok, wenn ich jetzt nochmal darüber schaue, ist die Animation hier [Bottom Sheet zum Bieten] bei A flüssiger. Also würde ich sagen: A.

A: Ok, perfekt. Du bist jetzt noch in A richtig?

I: Ja

A. Dann öffne bitte einmal die Übersichtsseite und von dem ersten Auto das Bild heraussuchen, wo extrem viele Zettel zu sehen sind. Dafür kannst du auch die Gallerie öffnen.

I: Hier.

A: Sehr gut. Und in B kannst du von dem 2. Auto das Bild heraussuchen, wo ebenfalls viele, aber nicht ganz so viele Zettel zu sehen sind. Dort solltest du ein "Oldheimer Gutachten" sehen können.

I: Ah ok. Das müsste es sein - ja.

A: Konntest du bei diesem Use Case irgendwelche Unterschiede feststellen?

I: Garnicht

A: Garnicht?

I: Wenn ich nochmal schaue. Das Öffnen der Gallerie ist gleich. Ich kann hier bei B noch weiter reinzoomen. Bei B kommen die Bilder ein bisschen verzögert reinanimiert. Aber da kann ich sonst keine Unterschiede feststellen, was kaum auffällt. Also, da kann ich mich nicht entscheiden.

A: Alles klar. Dann kommen wir zum nächsten Use Case. Du darfst einmal in App B das Tracking einschalten. Das kannst du über die "Mehr"-Seite machen. Und jetzt einmal "About Kickdown" öffnen. Du kannst die Seite wieder schließen. Das war es auch schon. Nun magst du das gleiche nochmal für A wiederholen. Du kannst auch eine andere Seite öffnen.

I: Du fragst mich bestimmt, ob ich irgendwelche Unterschiede feststellen konnte.

A: Genau.

I: Also, da konnte ich jetzt absolut nichts erkennen.

A: Auch beim Switch betätigen?

I: Alles gleich.

A: Das waren jetzt auch schon alle Use Cases.

.3.3 Interview 3

Date: 07.04.21

Job title: Junior UI/UX Designer

Years working in mobile industry: 8

Experience with Kickdown App: conceptual development of application

A: Du kannst gerne einmal Kickdown B öffnen und dort den weißen Jaguar aus Böblingen heraussuchen

I: Da ist er.

A: Sehr gut. Nun bitte ich dich einmal in Kickdown A den Golf 1 Cabrio aus Hamburg zu finden.

I: Das müsste er sein.

A: Super. Nur vom App Start und Scrollen. Konntest du irgendwelche Unterschiede feststellen?

I: Nein.

A: Das heißt du würdest auch dementsprechend keine der beide Erfahrungen präferieren?

I: Genau.

A: Dann - du bist ja jetzt in Kickdown A - würde ich dich bitten auf den Überblick zu gehen und dort ein Auto deiner Wahl auszusuchen. Dieses kannst du dann einmal Öffnen und dort auf das Auto bieten. Du kannst irgendeinen Betrag eingeben. Du bist auch nicht angemeldet, also wird da auch nichts gesendet. Super - vielen Dank. Du kannst dir gerne einmal in der App B nun ein anderes Auto raussuchen und dort bieten.

I: Ok.

A: Sehr gut. Vielen Dank. Dann jetzt einmal wieder die Frage: Konntest du zwischen den beiden Apps in diesem Use Case irgendwelche Unterschiede feststellen?

I: Bei A hat man bei der Screen Transition die typische Detail Animation. Bei der anderen ist das irgendeine andere Animation. Ansonsten sieht man bei A auch die bottom sheet Animation, wo der Hintergrund so wegfährt. Bei B ist das nicht ganz wie im Original. Und bei B ist die Animation nicht so schön. Ich glaube es liegt einfach an der Animationskurve. Die Animationskurve ist einfach linear.

A: Wenn du dich entscheiden müsstest, welche wäre das hier?

I: Dann wähle ich Variante A.

A: Ich bitte dich nochmal in App B das erste Auto auszuwählen und dort das Bild auszuwählen, wo extrem viele Zettel zu sehen sind.

I: Das ist hier - Bild 7.

A: Cool. Dann kannst du noch einmal in die App A springen und dort beim 2. Auto das Bild heraussuchen, in dem ebenfalls viele Zettel zu sehen sind, allerdings nicht so viele. Da müsstest du ein "Oldheimer Gutachten" sehen.

I: Das meinst du, oder?

A: Konntest du in der User Experience irgendwelche Unterschiede feststellen

I: Also in A ist das erste Bild kurz weg.

A: Genau. Das ist tatsächlich bei beiden Apps so, da die Bilder von der API nochmals neu nachgeladen werden.

I: Ansonsten konnte keine Unterschiede feststellen.

A: Ok. Dann bitte ich dich noch einmal in App A zu gehen und dort einmal Tracking einzuschalten. Und dort einmal "About Kickdown" zu öffnen.

I: Und dann?

A: Das war es schon. Das Gleiche kannst du nochmal bitte für B machen. Du kannst natürlich noch eine andere Seite öffnen - z.B. Impressum. Konntest du da irgendwelche Unterschiede feststellen.

I: Kann ich mir das nochmal genauer ansehen?

A: Ja, klar.

I: Das fühlt sich für mich beides doch sehr ähnlich an. Ich glaube der Font bei der Flutter App ist nicht der Originale. Also er sieht für mich auch nach einem SF Font aus.

A: Ja, ich glaube ich habe nicht alle Fonts importiert.

I: Ah ja.

A: Präferierst du eine der beiden User Experiences?

I: Das kann ich dir nicht sagen

A: Ne, das ist auch in Ordnung. Danke Dir.

.3.4 Interview 4

Date: 07.04.21

Job title: Senior iOS Software Engineer

Years working in mobile industry: 11

Experience with Kickdown App: co-developed the iOS application

A: Du darfst bitte einmal die Kickdown app A öffnen und auf der Übersichtsseite den weißen Jaguar aus Böblingen raussuchen

I: Ah, da ist er.

A: Dann bitte ich dich noch einmal in App B den Golf 1 Cabrio aus Hamburg zu finden.

I: Die App unterstützt auf jeden Fall nicht Dynamic Type.

A: Konntest du zwischen den beiden Apps irgendwelche Unterschiede in der User Experience feststellen?

I: Die App B scheint ein bisschen länger zu laden und Dynamic Type wird wie gesagt nicht unterstützt. Ansonsten ist das sehr gleich für mich.

A: Ok. Und wenn du dich für eine der beiden Apps entscheiden müsstest aus User Experience Sicht?

I: Dann nehme ich natürlich A wegen der Dynamic Type Unterstützung.

A: Sehr gut. Dann kannst du einmal bitte in App B ein Auto deiner Wahl raussuchen.

I: Dann nehmen wir doch den Porsche 911.

A: Und kannst dort einmal bieten. Du kannst da irgendeine Zahl eingeben. Das Angebot wird nicht abgeschickt, da du noch nicht angemeldet bist. Ok - danke! Dann kannst du das Gleiche einmal bitte für App A durchführen. Du kannst dir auch gerne nochmal ein anderes Auto raussuchen.

I: Ok, dann nehme ich mal diesen Porsche.

A: Danke. Das war dann auch der nächste Use Case. Jetzt wieder die Frage an dich: Konntest du irgendwelche Unterschiede feststellen?

I: Ja, also die Transition bei der App ist auf jeden Fall schöner. Das andere ist eher langweilig.

A: Konntest du bei der Animation sonst noch irgendetwas feststellen beim Bieten?

I: Ist mir jetzt erstmal nichts aufgefallen. Lass mich nochmal schauen. Ah doch, also die Animation bei A fühlt sich natürlicher an. Die Transition bei B ist etwas schöner. Aber das Bottom Sheet ist dafür nativer bei App A. Daher würde ich mich in dem Fall auch für A

entscheiden.

A: Dann kannst du bitte einmal App A öffnen, auf das 1. Auto tippen und dort das Bild raussuchen, wo ganz viele Zettel zu sehen sind.

I: Das hier meinst du?

A: Ja, richtig. Dann kannst du bitte in App B das 2. Auto raussuchen und das Bild finden, wo auch Zettel, allerdings einige weniger zu sehen sind.

I: Das hier?

A: Top. Dann wieder die Frage an dich, ob du hier irgendwelche Unterschiede feststellen kontest.

I: Hier bei App B sieht man die Status Bar nicht - das ist natürlich keine gute User Experience. Sonst Beim Swipen und Zoomen fällt mir nichts auf.

A: Wenn du dich hier wieder für eine der beiden entscheiden müsstest, welche wäre das dann?

I: Dann würde ich hier wieder A wählen.

A: Ok. Dann haben wir noch einen letzten Use Case, den wir gemeinsam durchgehen können. Magst du dazu bitte einmal Kickdown B öffnen und dort auf der Mehr Seite das Tracking einschalten.

I: So.

A: Genau - das war's. Das Ganze kannst jetzt noch einmal bitte für App A ausführen. Perfekt - vielen Dank. Waren da für dich irgendwelche Unterschiede festzumachen?

I: Also hier sieht das für mich sehr gleich aus. Da kann ich nichts erkennen.

A: Dementsprechend präferierst du auch keine der beiden Nutzererfahrungen?

I: Nein.

A: Cool - danke Dir.

.3.5 Interview 5

Date: 08.04.21

Job title: iOS Software Engineer

Years working in mobile industry: 8

Experience with Kickdown App: has heard of app, but never used it before

A: Alles klar. Dann lass uns doch direkt mit dem ersten Use Case starten. Öffne bitte einmal Kickdown B und suche dort nach dem weißen Jaguar aus Böblingen.

I: Den hier meinst du?

A: Yes. Dann magst du einmal bitte Kickdown A öffnen und dort den schwarzen Golf 1 Cabrio aus Hamburg finden.

I: Hier ist er.

A: Perfekt. Konntest du da irgendwelche Unterschiede zwischen den beiden Apps feststellen?

I: Lass mich nochmal ein bisschen durchscrollen.

A: Klar.

I: Das fühlt sich für mich sehr gleich an

A: Wenn du dich für eine der beiden Apps entscheiden müsstest, welche wäre das? Es ist auch ok zu sagen, dass du beide gleich gut findest.

I: Ich empfinde beide für gleichwertig.

A: Alles klar. Dann können wir einmal zum 2. Use Case übergehen. Dazu kannst du dir ein Auto deiner Wahl aussuchen in App A und dort Bieten. Du kannst irgendeine Zahl eingeben. Du bist nicht angemeldet und somit wird dein Angebot auch nicht abgeschickt.

I: Alright, dann holen wir uns doch den Porsche.

A: Viel Erfolg! Top - dann kannst du das gleiche einmal bitte in App B durchführen.

I: Der Font ist hier von dem Dialog auf jeden Fall nicht richtig. Das erkenne ich sofort. Das sieht nicht richtig aus für mich.

A: Ah interessant. Gibt es sonst Unterschiede?

I: Das Bottom Sheet ist nicht Draggable in App B. Die Animation fühlt sich auch nicht nativ an. Aber das liegt nicht an Flutter. Das wurde dann einfach nicht genau gleich implementiert

A: Ok danke. Wenn du dich wieder für eine App entscheiden müsstest, welche wäre das?

I: Dann würde ich ganz klar sagen: A.

A: Ok cool. Dann bitte ich dich bitten einmal App A zu öffnen und dort erste Auto auf der Überssichtsseite auszuwählen und dort das Bild mit den Sonnenschirmen zu finden.

I: Das meinst du?

A: Yes. Magst du nun bitte die App B öffnen und dort das 2. Auto auf der Überssichtsseite aussuchen und in der Gallerie das Bild finden, wo man das Nummernschild gut ablesen kann?

I: Hier.

A: Danke. Jetzt wieder die Frage: Konntest du irgendwelche Unterschiede feststellen?

I: Die funktionieren beide gleich. Das Swipen ist bei der Flutter App auch flüssig.

A: Also präferierst du auch keine der beiden Apps in diesem Fall?

I: Die sind beide gleich.

A: Top. Dann habe ich noch einen Use Case vorbereitet. Dafür kannst du in App B bleiben und bitte einmal auf der "Mehr"-Seite das Tracking einschalten und anschließend die "About Kickdown" Seite öffnen.

I:: Ok. Ich muss immer schauen, ob die Seite "Bounce" hat. Das ist ganz wichtig. Aber die scheint einen guten "Bounce" zu haben.

A: Das war es auch schon. Das Ganze kannst du nun nochmals für Kickdown A wiederholen.

I: Ok. Also hier öffnet sich die Webview auf jeden Fall gleich. Lass mich das nochmal prüfen.

A: Klar.

I: Das ist wirklich gleich.

A: Ok. Präferierst du in diesem Beispiel eine der beiden Apps von der User Experience?

I: Ne - die sind beide gleich.

A: Cool - vielen Dank.