JÖNKÖPING UNIVERSITY
School of Engineering

Evaluation of The Software Development Process for A Multi-Platform Solution in Flutter

Main Subject area: *Computer Engineering*
Author: *Dennis Andersson & Axel Axelsson*
Supervisor: *Garrit Schaap*
JÖNKÖPING *2021 02*

This final thesis has been carried out at the School of Engineering at Jönköping University within Computer Engineering. We are responsible for the presented opinions, conclusions, and results.

Before we proceed, we would like to thank our supervisor Garrit Schaap who has guided and motivated us throughout this bachelor thesis.

Examiner:	Johannes Schmidt
Supervisor:	Garrit Schaap
Scope:	15 hp (first-cycle education)
Date:	2021-06-22

Abstract

Throughout the years of software development, the number of frameworks and software technologies have rapidly increased. This not only increases the difficulties of choosing the right software, but also makes it harder to find developers in a specific area. To create a multi-platform solution, a company would usually need competence in different areas such as frontend, backend, and mobile development. Such a solution requires an investment of a lot of time and resources. An alternative to developing a multi-platform solution opposed to the conventional way is with the software development kit created by Google called Flutter.

The purpose of this bachelor thesis is to investigate the software development process when creating multi-platform solutions in Flutter. The study will point out eventual differences with the software development process between Flutter and conventional development.

To achieve this, the study collected secondary research and conducted interviews with two developers who both worked with Flutter. In addition to this a Portfolio application was built during an experiment phase with a daily diary as the data collection process to validate the experiences given from the interview and secondary research. The results point towards some general benefits of using Flutter when creating multi-platform solutions. These benefits are only applicable when creating a product that shares the same user interface and have no OS specific features that requires need for deep integration into the targets device system.

1 Introduction

This bachelor thesis is done in collaboration with Sweco. Sweco does not have a separate objective with the work being undertaken but are primarily involved to aid us with knowledge and experience. They are taking part of the new knowledge we generate with this work and the work process we carry out.

1.1 Definitions

1.1.1 Multi-platform solution

The definition of multi-platform in this paper adds mobile development to the web solution stack. This means having three entry points, namely Android, iOS, and a web application that share the same backend which includes servers and databases. All components included can be visualized in figure 1.

1.2 Background

Having a thought-out software development process is essential when creating any type of software (for an in-depth explanation of the software development process see section 3.1). A lot of steps need to be completed for a product to reach a launch ready phase. Therefore, a lot of different definitions of the software development process exists. With one of the goals being to potentially cut down on development time and cost

The issue the software development process tries to solve remains because companies and customers usually want their product to reach as broad of a market as possible. This leads to a process that involves a high number of technologies for each phase. According to a survey done by IDC, companies continue to be very interested in multiple platforms (Dhillion, 2015). This often means that they want a multi-platform solution. The process of developing this requires a lot of time and money, as well as developers with a lot of different skillsets (Mercado, 2016).

To combat these challenges, several tools to build cross-platform applications have emerged to aid developers (Dhillion, 2015). The goal with these tools is to achieve a single codebase that can be compiled and executed on different platforms.

Previous studies have been conducted on cross-platform development tools over the years. The majority of these have had focus on benchmarking and performance between different frameworks and not on evaluating development in them in the context of the software development process. A study conducted by Sauma & Ziai (2020) at Jonkoping University did a comparative study between the development process on Flutter vs React Native with respect to the developer's productivity.

This thesis is different in that we do not look deeply into the software development process in terms of how the developer's productivity can alter, by looking at quality and simplicity. Instead, our paper will focus on a more complete view of the software development process and how its parts can look like when utilizing Flutter. On top of that, the thesis focuses on evaluating Flutter as a suitable option for creating a multi-platform solution.

Flutter as a cross-platform tool for mobile development have existed since May 2017. In 2021 Flutters web support went from beta to a stable version (Flutter, 2021). Since this web support is so new, there lacks knowledge and research on how to work with Flutter for a multi-platform solution. This thesis will fill a gap in the research field that is not yet explored and that can potentially give companies some valuable knowledge on considering Flutter as a tool to create a multi-platform solution in.

The aim of this study is to look further into the software development process when creating multi-platform solutions in Flutter and investigate if there are any differences to the software development when using Flutter vs conventional development.

1.3 Problem statement

Throughout the years of software development, the number of frameworks and software technologies have rapidly increased. This increases the difficulties of choosing the right software (Mercado, 2016).

To create a multi-platform solution, a company would usually need competence in different areas such as frontend, backend, and mobile development. Such a solution requires an investment of a lot of time and resources. The variation from several different development technologies makes it hard to give a unified user-experience across the entire solution (Dhillion, 2015). Several codebases must be maintained resulting in longer times when bug-fixing and implementing new features (Biørn-Hansen, 2019).

One example of a conventional way of building a multi-platform solution is with a website using client-side software HTML, JavaScript, and CSS. While with mobile development, having two separate codebases natively for Android and iOS written in Java or Kotlin and Objective-C or Swift. These parts all need to be connected to a server-side. This consists of both a run-time environment such as Node.js and a database created with a suitable technology such as firebase. Even for a simple application this development becomes extensive and multifaceted. Figure 1 illustrates this simplified version of a multi-platform solution, and in addition lists more common techniques required to work on developing multi-platform solutions.

An alternative to developing a multi-platform solution opposed to this conventional way is with the software development kit created by Google called Flutter. (Flutter, n.d.).

The approach Flutter takes is different in the sense that only one language, namely Dart, is required to write a single codebase for the frontend. This codebase can then be compiled to deliver a native executable to work on any operating system. As illustrated in figure 2, this can be used to create a multi-platform solution in iOS, Android, desktop and a progressive web application that can run on the web (Flutter , 2020).

The reason why our thesis focuses on Flutter and no other cross-platform development tool, is because Flutter is the only cross platform development tool that offers a stable version of web support. This makes Flutter a legitimate tool to create multi-platform solutions in.

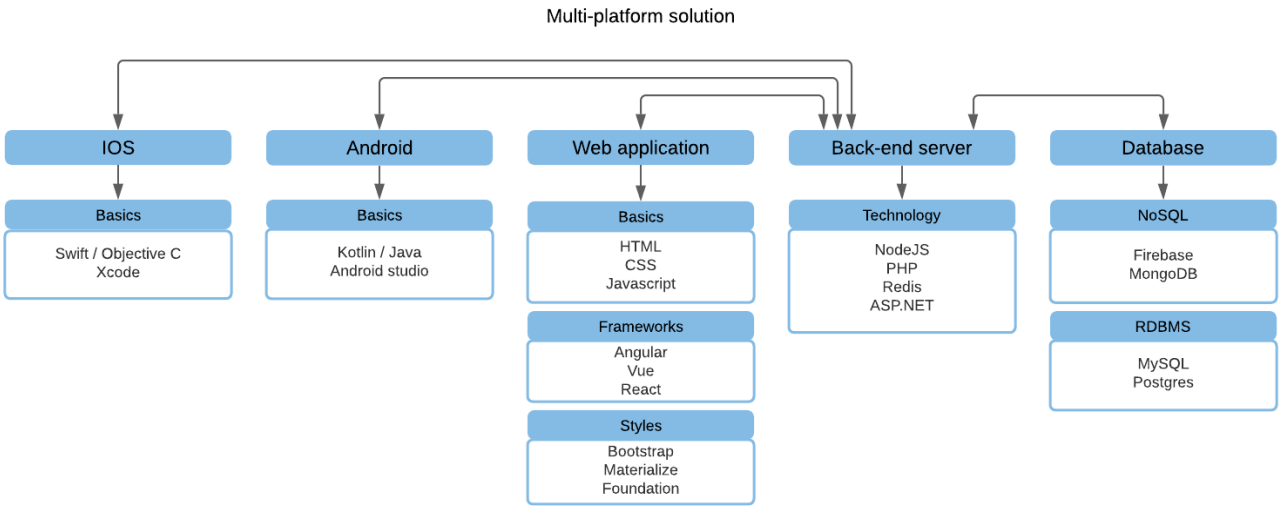

Figure 1: A conventional multi-platform solution and optional technologies

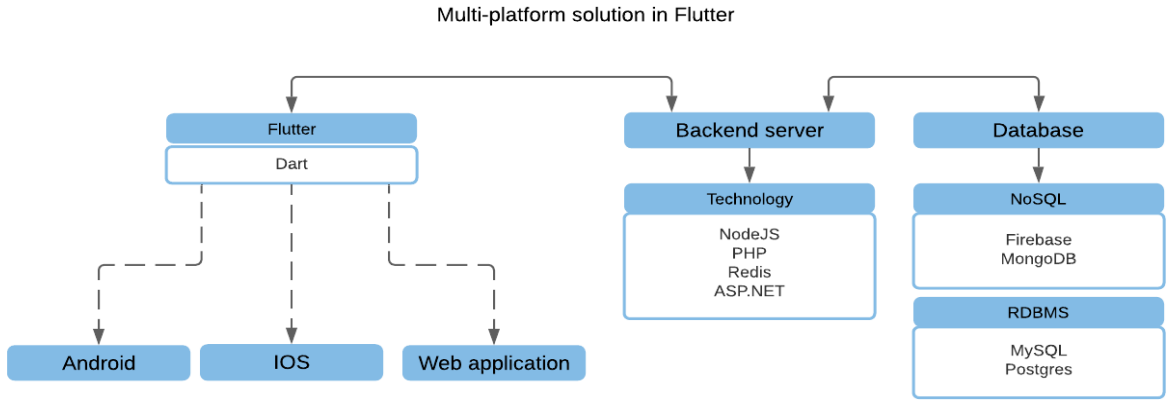

Figure 2: A multi-platform solution in Flutter

1.4 Purpose

The purpose of this bachelor thesis is to investigate the software development process when creating multi-platform solutions in Flutter. The study will focus on highlighting differences with the software development process between Flutter and conventional development.

1.5 Research questions

There needs to be an investigation of how the software development process in Flutter looks like. The findings will be evaluated and used to inform about how the process of developing a multi-platform solution in Flutter can look like.

RQ1: How can a development process in Flutter look like?

The findings from RQ1 together with desk-research will be further discussed to give insight into Flutter and its potential differences.

RQ2: What are the differences between using Flutter vs conventional development in the context of a developer's perspective.

1.6 Scope and Limitations

No extensive look will be given on every detailed part of the software development process. The only concern is within the software development process that might be affected by using Flutter. This paper will not examine any benchmark differences with Flutter or compare performance regarding other software technologies.

1.7 Disposition

The study structure is divided into the following chapters:

Chapter 2 – Method and implementation: Describes and motivates the approach, data collection and data analysis of the paper. The chapter ends with a small description of how validity and reliability has been secured.

Chapter 3 – Theoretical Framework: Goes over the software development process used in this thesis and further discussion around Flutter and its interesting components related to the framework.

Chapter 4 – Experiment: Method of choice describes and motivates the method choice for the experiment. Application concept goes over the application meant to be built for this thesis. The chapter ends with a work process going over the structure of working with the experiment.

Chapter 5 – Results: Lays out the results gathered in this thesis in a systematic way. The structure is laid out according to the software development process defined in the Theoretical framework.

Chapter 6 – Analysis: The results gathered in chapter 5 is systematically analyzed with the same structure and format.

Chapter 7 – Result discussion describes how the results have been connected to the purpose and research questions for this thesis. The chapter ends with a limitation section that goes over the parts left out in this paper.

Chapter 8 – A few conclusions and implications are given here. The chapter ends with ideas for future work and how the study can be carried on for further research.

2 Method and implementation

2.1 Approach

As a basis for the thesis regarding the software development process, desk research will be performed. This will give a more thorough understanding of the different parts involved within the process, to help evaluate the development of multi-platform solutions. To evaluate the software development process in Flutter a case study with qualitative research will be conducted. The research entails interviews with a complementary experiment.

A case study is most suitable when you want to focus on a single unit for analysis (Saldaña, 2011). Unlike studies where you focus on many participants to get a broader view, this allows us to go more in-depth.

For the first part of this thesis, a case study will be suitable, to get in-depth knowledge of software development in Flutter. It allows us to interview developers who are experienced with working with Flutter and inquire about their approach and thoughts about developing in Flutter. Their knowledge and experience with developing will then be combined with desk research and an experiment as a foundation for RQ1.

Using a case study together with an experiment will not be enough to do an exact description of every specific little detail regarding the software development process and how it can affect the creation of a multi-platform solution using Flutter. But the evaluation from RQ1 will still be enough to draw at least some valuable insights into the differences with working in Flutter. These differences will be beneficial when considering Flutter regardless of how the software development process looks like.

When enough data through our desk research has been collected. An interview process can be laid out in preparation for our interviews. The interviews will be performed with developers that are currently working with software development. These developers will provide knowledge with developing multi-platform solutions in Flutter. Further details on how this process went ahead can be read in section 2.2.2.

The data from the interviews will yield a software development processes that are accurate representations of how a real-world instance of a process can look like. The data and desk research will be the foundation behind an experiment. The experiment will be used to evaluate the software development process for Flutter by developing a multi-platform solution. The experiment is because Flutter's web part is relatively new having just left beta and the findings from RQ1 regarding Flutter can be heavily biased. Therefore, developing our multi-platform solution in Flutter can be used to strengthen the potential differences when utilizing Flutter.

As a final evaluation, a qualitative comparison method will be done to form a conclusion of the potential differences with Flutter.

2.2 Data collection

2.2.1 Desk Research

Desk research or secondary research is effective in the starting phase of a research process. Secondary research is the act of reviewing previous research to gain a broader understanding of a subject. This is almost always the first point of departure before answering the papers primary research questions. This is because most of the basic information can be easily fetched which then can be used as a foundation in the research process (Stewart, 1993).

For this paper, external desk research with focus on online material will be performed. The desk research will lead to complementary information that aids us in the process of forming our interview structure in a scientific way. It will also strengthen our evaluation of Flutter with online lectures and research material that covers adequate information about the software development process both in Flutter and conventionally.

2.2.2 Interview process

Interviews will be conducted for RQ1. An interview is best suited to gain insight into a subjects' opinions and experience of a phenomena. This results in an established knowledge base for how certain software development parts in Flutter can look like. These parts are applicable to the real world and used by companies. The interview types will be semi-structured and consist of open-ended pre-determined questions (appendix 2).

The participants will be sent an introduction text in advance with important details of the interview to prepare them about the topic at hand and what the study sought to entail. The document is also a step for upholding full confidentiality and transparency between interviewers and the participants regarding how the video and text material will be handled in this paper. This document is also sent out early enough to give the participants at least 24hours to withdraw or raise any concern if anything in the introduction was unclear (appendix 1).

A pilot interview will be conducted with a person of similar expertise to ensure that the questions of the interview were stated in a way that is easily understood and provides informative answers that align with our work. The answers from the pilot interview will be analyzed by discussing what kind of data we received. Based on this, both the interview questions as well as the introduction text can be improved and clarified.

The participants of the interviews are chosen for their expertise and experience with the subject of matter. The interview targets will be developers that have been working with Flutter and have extensive knowledge about it.

The interviews will be conducted over Microsoft teams and be recorded. One of us are responsible for taking the lead on asking the pre-determined questions, while the other

one can focus on listening more attentively, taking notes, and noticing any interesting answers and asking follow-up questions that the lead interviewer might miss. For the interviews, a follow-up mail can be sent to the participant with a few additional follow-up questions or clarifications that were identified during the analysis.

2.2.3 Comparative research method

A comparative research method will be applied to the software development process identified in RQ1 and additional secondary research in the form of online material. This comparative analysis is qualitative and allows a systematic approach to identify what is shared and what differs between the instances. The value of this method is that it provides a wider lens to view the context of the different software development processes (Hammond, 2020). Which will help this paper with answering RQ2 and lay out beneficial discussions in chapter four and chapter five.

2.2.4 Experiment

To have enough data to evaluate the software development in Flutter, as well as discuss the potential differences of making an entire multi-platform solution in Flutter vs conventional development. An experiment will be conducted to fill in the gaps from the findings in RQ1. Experiments are also an appropriate method for evaluating new ideas and theories. This was used to evaluate how the software development could look like in Flutter.

During the experiment process of developing a multi-platform solution in Flutter, a diary will be used as the method of choice. A diary is a good complement for other methods and is particularly important when observations is not possible (Hammond, 2020). Therefore, since the experiment will be based on our own experiences with developing in Flutter a diary document is the obvious choice to use. This will provide data and records over the course of the entire experiment.

2.3 Data Analysis

2.3.1 Interview analysis

The gathered material from the interviews will be analyzed using a qualitative content analysis. Content analyses involve systematic reading of content and assigning labels to identify interesting or meaningful content. The qualitative method to the analysis allows for identifications of patterns and the meaning of the content. Based on the findings of the content the course of the research could be changed. This approach is based on our open-ended research questions that allows the findings to guide the research (Bengtsson, 2016).

The recordings from the interviews will be transcribed word for word. The transcribed interviews are then individually read through multiple times and relevant content are

outlined by being marked and commented upon. After this, we will discuss our different findings and relevant content. The result is then categorized, summarized, and presented in the thesis.

2.4 Validity and reliability

A case study can become biased with our interpretation of the gathered data and subsequent argumentation as basis for the new findings. Therefore, we will only try in an objective way to highlight the differences between the software development processes without judging which one might be better or worse.

The nature of qualitative studies makes generalization of a phenomena very hard due to the in-depth focus of smaller sample sizes or even single cases. This also leads to the study being almost impossible to replicate since the data is gathered from a very specific context and source (Bengtsson, 2016).

3 Theoretical framework

3.1 Software Development Process

The software development process, also known as the software development lifecycle (SDLC), is the process of developing software and is often broken up into distinct phases. “A lifecycle covers all the phases of software from its inception with requirements definition through to launch and maintenance” (Ruparelia, 2010).

The software development process and its phases and steps are illustrated in figure 3. This is the final definition of the software development process as this thesis uses.

The SDLC allows software to be developed in a systematic way and increase the chance of delivering completed software projects within the timeframe and budget specified while still maintaining high quality (Apoorva Mishra, 2013). By applying a SDLC, a software company can organize their work efficiently when developing and working on software.

There exist many different models of the software development life cycle. These models are generally used to describe the SDLCs internal phases and how it can be approached (Ruparelia, 2010). These phases illustrate the work needed to be done to complete any software development. Although the SDLCs differ from each other and contain different number of steps with different names, the phases are similar in their essence, but just organized differently.

In general, every software developed will have to go through some form of requirements capture to understand a problem, a solution needs to be designed, coding the actual solution, testing of the created solution, the solution needs to be deployed and then comes maintenance of the product (Apoorva Mishra, 2013).

The content of each phase described below is based on the phases from Pham (2019) but with some changes to better suit how we will explore them.

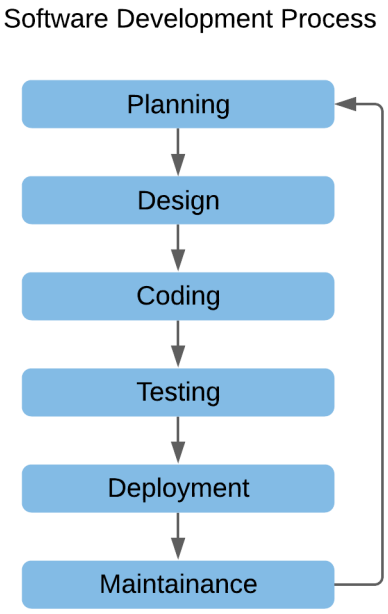

Figure 3, illustration of a software development process

3.1.1 Planning

Normally the planning part during development is mainly planning the current iterative cycle and the work being required for it. This is based on the requirements for the product and the priority of the products specification.

According to Dawson (2014), two main processes are required to produce any software, the project management process, as well as the software development process.

The project management consists of all management activities that is required outside of the actual coding of a product. For example, forming a team, planning the software development, defining, and controlling the product, as well as estimating the time – cost – scope of the product, happens in the planning phase before the actual development can begin. These areas might be relevant of how to approach working with Flutter and are included in the planning phase.

3.1.2 Design

In this phase the pre-designed solution architecture and user-interfaces in the form of mock-ups are implemented. How this is approached and implemented in the solution is of relevance to the software development.

3.1.3 Coding

This phase is concerned with the actual building of the product. Technical details of how the software is assembled will be discussed here.

3.1.4 Testing

Testing of the completed parts is a necessary step to ensure that the solution works as intended. By finding bugs and problems before the solution is deployed to the public potential risks can be mitigated.

3.1.5 Deployment

Deployment is the action of launching a new product or a big update to the public, this only happens once for each new version of the solution, and often when the development on that version is completed satisfactory.

3.1.6 Maintenance

This is a phase in which a product is often put into when active development on the product is finished. Here the product is maintained to ensure that it is working as intended. Updates and features could still be implemented this phase. The characteristics of a software product that is maintained is different from a new product in active development. Most concern here is laid on steps like bug fixes and minor changes while ensuring that the product is up and running. The internal process of the work here is usually longer.

3.2 Flutter

A problem with most native applications as stated earlier in the thesis is the time, and cost consumption required to achieve a native multi-platform solution. For the end user this results in nothing more than two devices, having the same capabilities, which is to be expected in today's day and age. If you use a navigation system on Android & iOS or on the web it should work the same in terms of core functionalities and user experience (Chadha, 2017). But "from a software engineering perspective, implementing the same feature on different platforms requires the use of vastly dissimilar languages, APIs, and software architectures" (Chadha, 2017).

The goal with Flutter is to alleviate this unnecessary process by creating a portable framework, capable of natively compiling an application to any target device with a single codebase (Flutter, 2021).

With Flutter 2.0 released, web became another stable target device for Flutter. The vision is not to be a full replacement for the traditional document-based experience using HTML, CSS, and JavaScript but to instead provide everything that is great about Flutter for the mobile experience, with highly interactive, graphically rich content and compiling it into the web landscape (Flutter Team, 2019).

The following subparagraphs will briefly describe some of the primary parts that makes Flutter different in its approach.

3.2.1 Widgets

This is the core part of Flutter. The whole user interface is constructed using widgets, everything from buttons, text, and images are built using widgets. This includes more than what the user can interact with. Even styling, padding, margins, and your entire application itself is a widget (Flutter, 2021).

Essentially there are two types of widgets in Flutter, Stateless widgets and Stateful widgets. Just as the name implies, if a widget does not do anything that can alter the state of the application like a static text on your screen, then it is referred to as a stateless widget. On the contrary if it does alter the state, like for example a button that sets off some logic execution, then it is stateful.

Widgets have a parent and child relationship, and since the entire application is built using widgets, this leads to a parent and child hierarchy of widgets. An illustration of this can be seen in figure 4.

This makes styling and building user interfaces very intuitive, because you write all your dart code in a declarative way with a single codebase. Instead of having to wrap your traditional HTML code into containers, to separately style it by writing CSS. A code snippet illustrating a basic flutter program can be seen in figure 5.

However, as with everything regarding technology, one potential limitation with this approach is a problem referred to as the state management problem. In short terms this means that since Flutters UI is a widget tree, setting or altering the state of a parent would lead to all its associated children to re render. This could cause some performance drawbacks, especially on large applications (Boukhary, 2019).

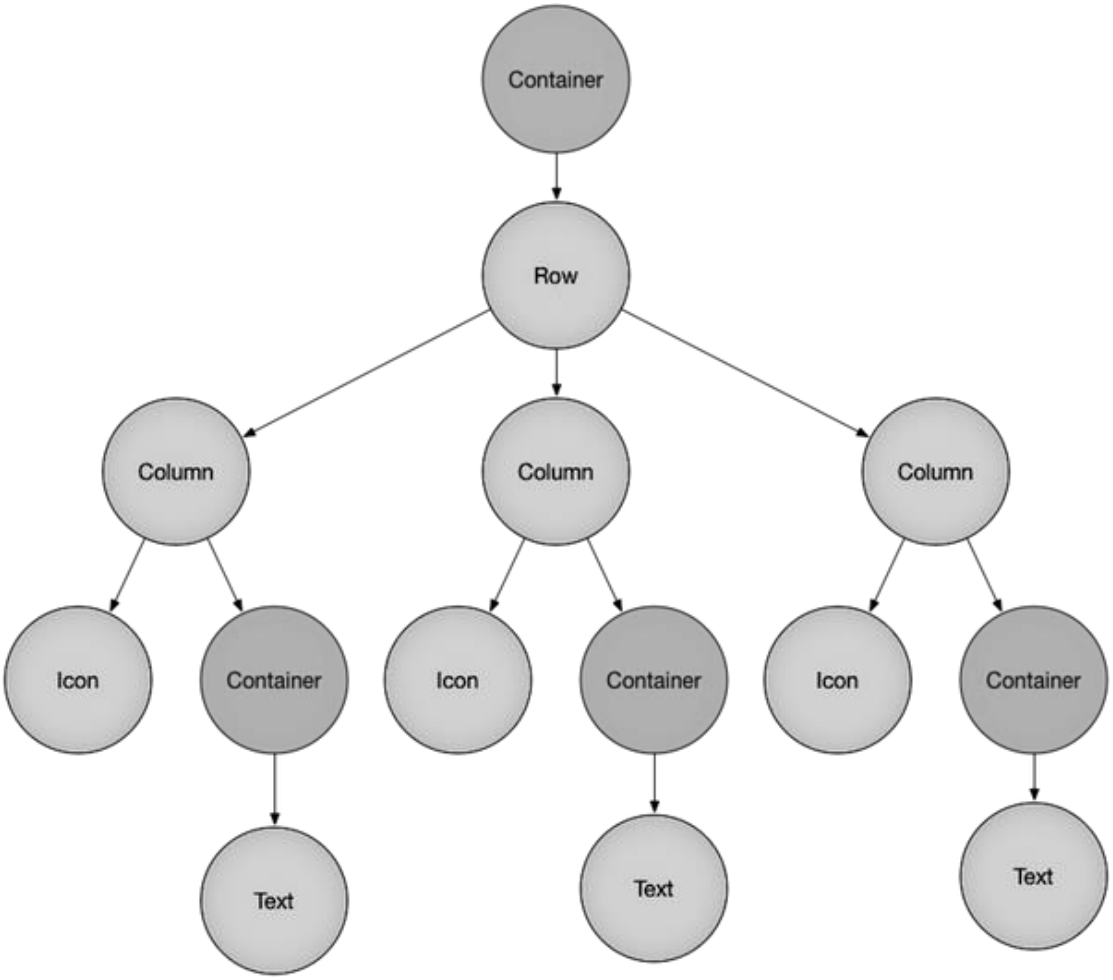

Figure 4: A simple representation of a widget tree

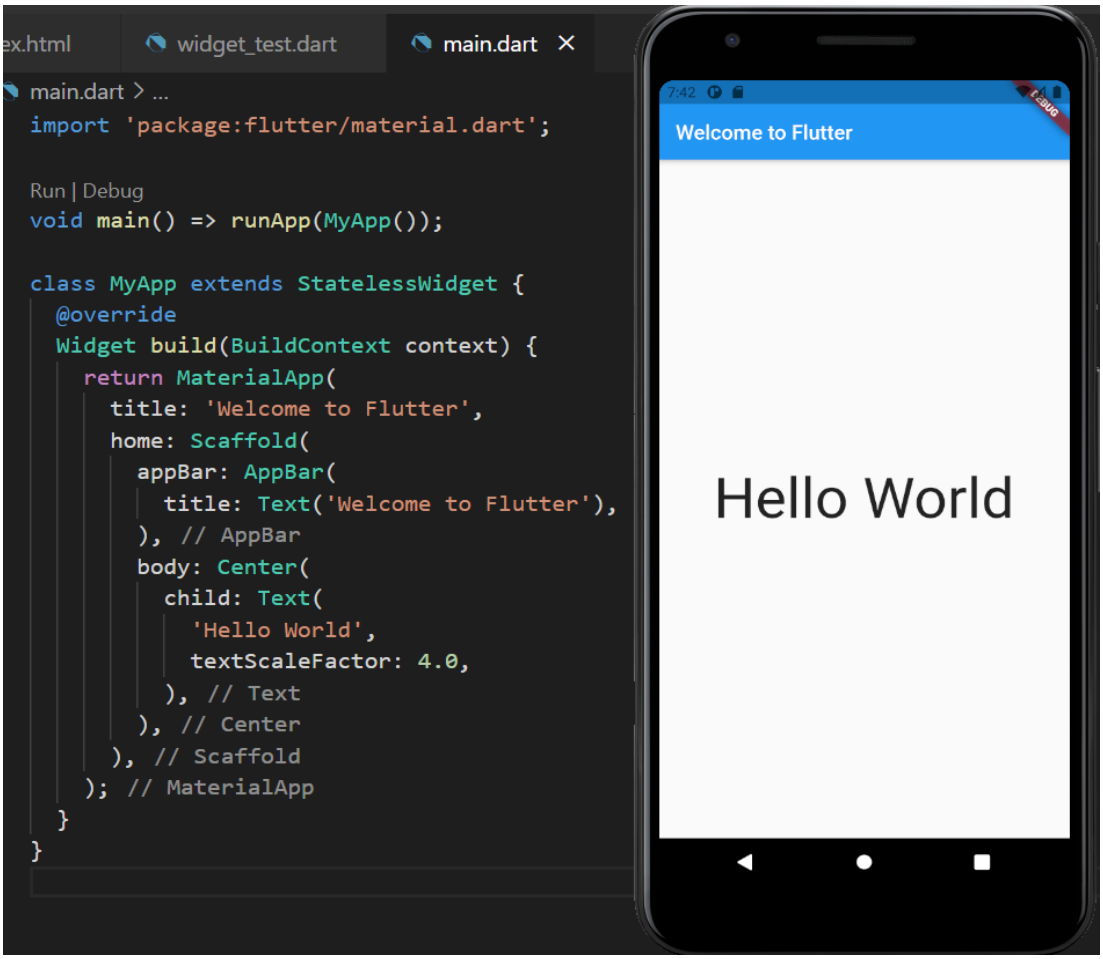

Figure 5: A sample project

3.2.2 Testing

Testing in Flutter is split up into three parts, unit testing, widget testing and integration testing. Unit testing is written to check the correctness of a single function, method, or class under different conditions (Flutter, n.d.). When dealing with external dependencies in the codebase Flutter needs to utilize non prebuilt packages to imitate that dependency. An example of such a package is the popular mock library, Mockito (Mockito | Dart package, n.d.).

Widget testing or component testing tests a single widget to verify that the behavior, interaction, and UI of that widget matches the expected outcome (Flutter, n.d.)

Integration testing is more extensive and meant to test the complete application or large parts of it, with the goal being that all the widgets and services works together (Flutter, n.d.). In its essence this test should verify that the application works as expected by a consumer.

3.2.3 Web-support

As previously mentioned, Flutter does not envision their web support to be a complete replacement to traditional websites, therefore it is clearly indicated on their website that Flutter is most valuable when you want to create any of the following scenarios:

3.2.3.1 SPA (Single Page Application)

A web application that consists of a single page. You load a single HTML page once and nothing more during the entire session of visiting the site. The browser only renders the information necessary when an action is performed by the end user with the help of client-side JavaScript.

Compared to a regular multi-page application that was historically more common, with links navigating the client user across different parts of the website, forcing the website to render multiple times (Solovei, 2018).

3.2.3.2 PWA (Progressive Web Application)

A progressive web application is a web-solution built to give a native like experience across all devices. The goal is to give a unified user experience across mobile and desktop (Arroyo, 2020), so that browsing websites on a mobile device does not feel like two separate technologies for the end user.

A website in Flutter is essentially always a progressive web application. By just a few clicks, anyone can turn their single page application into a progressive web app.

3.2.3.3 Turning existing native application for Android and iOS into a website

As the title says, because the framework is the same and web is just another target device, any existing Flutter application can easily be turned into a website if the company desires to expand their product at any given time.

It is therefore not recommended or suited to use Flutter for text-rich, static content such as websites with a lot of articles and static text content like newspaper sites and blogs (Flutter, 2021).

These recommendations and guidelines will be taken into consideration during the experiment phase, and therefore a website suited for the technology of choice will be built. The experiment will also build and compile two additional target devices, one for Android and one for iOS. To highlight the portability aspect of Flutter when turning a PWA into a mobile application.

3.2.4 Flutter features

This section will go through some key features that is beneficial when developing applications in Flutter with regards to web development as focus point. These features will be documented and explored while conducting the experimental part of this paper.

3.2.4.1 UI–inspector

Because Flutters entire view is built up using widgets, having a visual representation of the hierarchical structure of the parent and child relationship is beneficial when dealing with large applications in Flutter. The UI – inspector is therefore a tool to combat this problem and can be used when working with any type of layout (Flutter, 2021).

3.2.4.2 Hot reload & Hot restart

A benefit that always gets brought up in every paper looking at Flutter for mobile development is the feature hot reload. This feature allows code changes to be loaded into the virtual machine so the widget tree can be rebuilt while preserving the application state (Flutter, 2021).

The benefit with this is that developers can save time during development because small changes like altering some color scheme or adding a new button to the screen does not have to cause a full recompilation of the project. For Flutter web hot reload is currently not supported as an option. Instead, something called hot restart is used. This is essentially the same functionality except that the flutter app gets restarted and does not preserve its state which is comparable to refreshing a website.

3.2.5 Architectural differences

This section will briefly highlight some of the internal differences between the architectural structure of Flutter web and how the approach is slightly different to Flutter for mobile development.

3.2.5.1 Compilation

Flutter has four compilers that is being used for different purposes. Two for mobile and two for web:

1. Mobile
    - Just in time + Virtual machine
    - Ahead of time + Runtime
2. Web
    - Dev_compiler (dartdevc)
    - Dart2js

Two common and traditional approaches when trying to execute computer instructions is by using either something called Ahead of time (AOT) or interpretation (Aycock, 2003).

Interpretation is a high-level compilation method that interprets the code one step at a time during runtime (Poeplau, 2020). This makes it a very suitable compilation method for debugging purposes because the program will crash once it approaches a code block that it cannot interpret. The downside with pure interpretation is the performance drawbacks it gives. Compilation never takes place and no conversion to machine code occurs, it simply interprets source code and gives the desired results back. This makes interpretation slow compared to compiling source code directly into machine code (Poeplau, 2020). AOT compilation as the name implies, does all the compilation prior to runtime. It compiles a single native executable already at build time by converting all source code straight into machine code (Poeplau, 2020). This gives opposite benefits and drawbacks compared to the interpretation process, because this lets the application run fast in terms of performance but is much harder to debug when something goes wrong.

Just in time (JIT) compilation is a mix between these two styles. Just like the interpreter it runs during runtime but instead of just interpreting every code block it compiles the code block straight into machine code one step at a time. This makes JIT faster than traditional interpretation but still gives you all the debugging benefits of it (Aycock, 2003).

The reason to why Flutter separates mobile development into an JIT and AOT compiler becomes more understandable now. It is because of the nature of these compilers. An JIT compiler is simply more suited for debugging purposes because it compiles during runtime, which allows for benefits such as hot reload to accelerate the development process. Therefore, during development mode, when using a virtual machine through the dart virtual machine JIT is utilized (Dart overview, u.d.).

When the development phase is finished and a launch ready product is ready to be produced, Flutter uses an AOT compiler to translate the Dart code into assembly files which later gets compiles into binary code suitable for the architecture in mind in this case native ARM for mobile (Dart overview, u.d.).

As mentioned previously, with Flutter you have the same framework across all platforms but for targeting web applications, some small adjustments had to be made. This is because compared to mobile development, the web runs on JavaScript. So instead of compiling your native code into ARM, it gets compiled into JavaScript when targeting the web (Flutter , 2020). Therefore, Flutter uses two different compilers when dealing with the process of building a single page application.

The Dev_compiler is used during development time and lets you run and debug your web application in a web browser. It is like that of an JIT compiler in the sense that it supports incremental compilation (Dart, n.d.). It is therefore more suitable and recommended to use during development because it supports benefits such as faster refreshing and better integration with Google Chromes development tools (Dart, n.d.).

For compiling a production ready application Flutter recommends using the Dart2js compiler instead. The compiler is more optimized for this purpose, and it simply packages the entire application into a source file that can be natively executed on a browser (Dart, n.d.).

Overall, the platform differences between compiling to different platforms are so small that no further look in the experiment phase or theoretical discussing will be taken regarding this.

3.2.5.2 Rendering process

For mobile development, Flutter only has a single rendering mode called Skia. Skia is an open-source graphic library that Google acquired back in 2005. It runs across a variety of different platforms because it provides a common set of APIs. It serves as the graphics engine for products such as Google Chrome and Chrome OS, and Android (Skia, n.d.). This makes it a suitable graphic rendering tool for Flutter to utilize. Flutter web comes with two different rendering modes. One that utilizes a combination of Skia and web elements to minimize download size, and another one that combines Skia with web assembly to deploy graphics on the web (Skia, n.d.).

1. **HTML renderer**

   Provides a smaller download size by "combining HTML elements, CSS, Canvas elements, and SVG elements" (Flutter, n.d.).

2. **CanvasKit renderer**

   Unlike HTML renderer, this rendering mode provides a larger download size but is more consistent for both mobile and desktop. Both in terms of performance and graphical look (Flutter, n.d.).

According to Google the reason behind this is because generally you do not want a large download size for websites when browsing on a mobile device because it can impact performance and lead to additional bandwidth fees. While for desktop this is generally not an issue. (Flutter, 2021).

By default, Flutter automatically chooses which renderer to use if no additional setup specifications are made during the creation of an initial project (Flutter, n.d.).

Google does not clearly state how much more download size that can be expected by using CanvasKit, or how greatly the performance and visual representation is for the application when using HTML renderer instead.

The reason this can become a real issue is because on average a website is reasonably small. According to httparchive, a website that regularly keeps track of size and quantity of many websites. An average website between 2018 – 2021 is ~2Mb for desktop (HTTP Archive, 2021). This means that if Flutter greatly exceeds this number, it can

potentially scare customers and developers away from adopting or considering Flutter as a multi-platform solution. This is because large increase in size leads to increasing loading time for clients visiting your website (Manhas, 2014). It also reduces your overall search ranking on Google because speed is a landing page factor for Google search and Google advertisement for mobile browsers as of 2018 (Addy Osmani, 2019).

4 Experiment

4.1 Method of choice

It is important to point out the validity aspect of this experiment. Since this will be an interpretation of our own observations when developing a multiplatform solution in Flutter. The ability to replicate the same experience and result might not be possible. We believe that the findings collected from this experiment can be of value when compared to secondary research for pointing out some differences between using Flutter vs conventional development. The findings can also be of use for further studies but will most likely not be enough data to give a clear and concise answer on its own.

The diary will be written separately and will not be discussed between the participants until the experiment phase is completed to avoid inflicting biased opinions into the process. the diary itself will be written in a structured way according to the papers own established version of the software development process described in figure 3.

Four main questions will be in mind during the documentation process when writing. An example of how the diary template looks like for a given day is visualized in figure 6. The reason for structuring the diary in this way is that it facilitates the analysis process. Because data and relevant parts can then be more clearly compared to the software development process once the experiment is done.

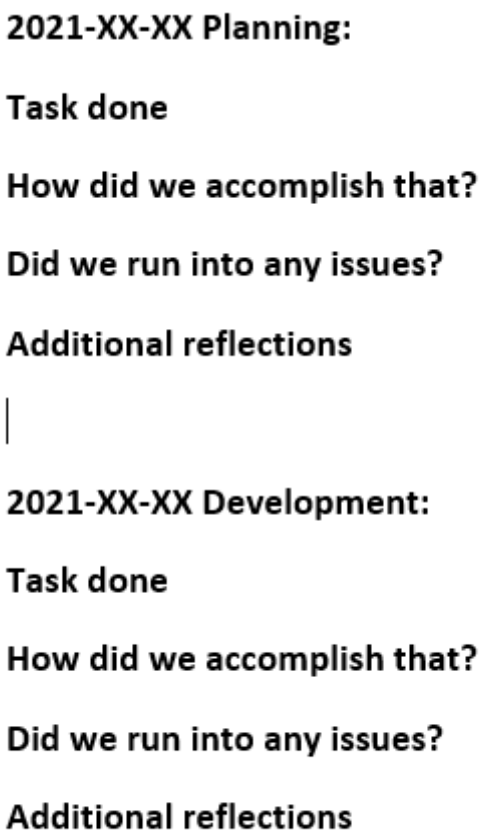

Figure 6: A model of the diary template

4.2 Application concept

The application concept built for this thesis was a portfolio platform. On the platform a user can navigate to various social medias through a footer, visit an about page and personalized profile page. Lastly a user can visit a project page that displays all the projects currently uploaded by the user. Here the user can apply all four CRUD operations on the different projects to dynamically create/delete projects. This concept allows for both user-interfaces and logic to be created for the application.

The application idea is based on an application that could solve an actual need, which makes the experiences working on it more valid. The size and design of the application will also be enough to follow structured work of the software development process where each phase and step can be documented.

The codebase for the frontend part of the portfolio application will be built using Flutter. The backend will utilize Firebase as the service for hosting the database. Firebase was chosen because we have previous experience working with it, and both technologies are owned by Google, which makes the integration process as straightforward as possible. Since the goal is to look further into Flutter, the choice of technologies for the backend does not matter.

4.3 Work process

The process of creating the application follows the software development process as it is laid out in the theoretical framework. The work process follows an agile scrum methodology to achieve a better structure on the work because it is more in line with how developers work in the industry. This causes the work to be more feature-driven and combines the steps described in the development phase in iterative sprint cycles. The different actions required for the sprint is categorized into its relevant step when it is documented in the diary.

5 Results

Here the results from the interviews and experiment will be presented. The chapter is divided into phases of the software development process as discussed in chapter 3. For each of the areas discussed, the findings will be presented in a structured way that clearly states where the evidence comes from.

The transcription of the interview data can be seen in appendix 3 and appendix 4. The interview data have been corrected grammatically for clarity.

5.1 Software development process

5.1.1 Interview data

The interviews with both the lead developer and the google developer expert shared similar opinions regarding the development process. The overarching theme was that the development process was not bound by the type of software that is being utilized by a company. The team structure and process of working remained the same. Both companies still needed to go through the same traditional agile workflow of having daily sprint meetings, retrospectives, and code reviews.

The only potential benefit covered regarding this, was that you get a unified user experience across multiple devices with a single codebase, and this can save time during development. This is pointed out by the google developer expert as a key reason during development because for the most part, developers are not experts in multiple platforms at the same time. Namely native Android, iOS, and web applications in this case. This is only an advantage if you are a small startup firm with limited resources or one or two people trying to create their own application across multiple devices. As soon as you want to separate the design into two separate entities, Flutter is not the right tool to use.

The Google developer expert also points out the benefit of being able to share the user interface between all the platforms. Because now a days most applications written will look and behave very similar regardless of the operating system that is running the application. For this reason, a single codebase makes sense and being able to share the UI saves a ton of time.

5.2 Planning

5.2.1 Interview data

The google developer expert and the lead developer had similar opinions when talking about gathering requirements for a project. The shared opinion was that this step is not affected by the choice of using Flutter as a client-side application. It rather depends more on the type of product that the client wants, or the overall operational structure of the company. The google developer expert worked at an agency which meant they had a more rushed schedule of delivering the products as fast as possible to the customer.

The lead developer worked at a company that focused purely on their own application instead of selling multiple products to different clients. But the process of gathering requirements was just about the same, it was not affected by using Flutter. It was more about the fact that they had their internal application that they worked on, and stakeholders and product mangers decided the scope and requirements of the application.

Forming teams when using Flutter is no different compared to native development in terms of size. Both the lead developer and the Google developer expert had the same team size while using Flutter compared to native development. Both developers worked in some type of agile method to determine the scope of the product, and divided the instructions given by the project manager between team members.

Both the lead developer and the Google developer expert touched upon the fact that time-scope-cost can potentially be improved by using Flutter. Instead of having five people divided into smaller separated groups as in native development, they can with Flutter work within the same codebase but with different views and functionalities. This saves time according to the google developer expert, but only if the application built is mostly UI-centric. If the product requires special sensors or packages that is not supported, the development time compared to native development can be much slower instead.

5.2.2 Experiment data

The experiment did not look deeply into the definition of planning and what it entails for the software development process. This was mainly because we were limited to only two people, making the emulation of having a real planning with communication to clients, forming a team structure, as well as time, scope, and cost discussions nearly impossible to perform in a meaningful way.

5.3 Coding

5.3.1 Interview data

The interview process covered a few points regarding coding benefits and drawbacks with Flutter. Features such as Hot reload was pointed out as a convenient tool for developers, which worked well even on large enterprise systems.

The biggest drawback with Flutter pointed out by both the lead developer and the Google developer expert, was the support for handling OS specific tasks. If a package or plugin made for Flutter does not already exist it can be difficult if not impossible to call some of these OS specific API’s. Some of the drawbacks mentioned could arise if you want any type of background process in your application. For example, scanning for Bluetooth devices in the background while the application is not active, or having a

payment process system where you must type in your credit card number and then show a website for them to 3D verify themselves with a face scanning method.

When discussing flutter web, none of the participants in the interview utilized Flutter web if all they wanted was a website. The main reason behind this was that web specific APIs are more suitable and easier to connect directly with JavaScript. The google developer also mentions that the speed in development with web frameworks like Vue.js is faster compared to Flutter web. The size was another contributing factor, the lead developer had considered the idea of turning their website built with react into another target device for Flutter. But the size scared the company away from considering it, because of reasons discussed in section 3.2.4.2.

5.3.2 Experiment data

The experience gathered from us during the experiment shares similar experiences with some Flutter features. Hot reload was a convenient tool to have, the only drawback was that this functionality was not added to flutter web. Flutter web as mentioned in the theoretical framework only comes with hot restart as of writing this. Hot restart proved to be much slower compared to hot reload. The feature also had a habit of crashing or freezing during development of the experiment, which created long loading times even for small changes. The only time hot reload could be utilized were during the time testing happened on a mobile emulator or physical mobile device, which worked flawlessly for us. Another good experience was the documentation written for Flutter. Everything is extensively written and easy to access on the website so getting a project up and running for the first time without any prior experience only took a few instructions detailed inside a written guide by the Flutter team (Flutter, n.d.).

The issue with Flutter having a hard time calling dependencies closely related to hardware was not an issue during the experiment because the application built did not handle any sensors or payment systems that needed specific packages to work.

Overall Flutter web gave the same impression as if we were doing Flutter for mobile, except that it was for another target device. The same packages and plugins were available to us, and the web part never limited us during development.

5.4 Design

5.4.1 Interview data

According to the Google developer expert, Flutter is faster to get into since Flutter is using declarative UI, which abstracts away a lot of the development. When building user interfaces in native Android development you need to understand the whole application lifecycle, how different activity instances transition between different states, like on-resume or on-start. When coding in Flutter those states are handled automatically, and the developer only needs to think about how the layout of the

components should be placed on the screen and what should happen when a user integrates with it.

But Flutter removes all these hurdles for newcomers to build applications. However, it is sometimes unusual for people to build applications in a declarative way. Especially if the person has a background in C or Java. But once it clicks, it becomes much faster to build applications.

Both respondents lifted that creating UIs in Flutter allows the user to implement any design without limitations because Flutter gives the user control to manipulate every pixel on a canvas board.

5.4.2 Experiment data

The observation from the experiment is that declarative UI allows the developer to conveniently create a user interface, giving great control over all elements of the screen. Getting a grasp on how to think about declarative design was not intuitive at first, but after a few days of coding in Flutter the declarative approach seems cleaner and easier to use. Creating basic interfaces that looks good by default and serves the intended purpose became easy and fast.

A few of the widgets in the application were created as custom classes and were able to be reused. The code for these components could be reused in several different views by sending in different data for them. With a good plan of the modularity of the views and which components could better serve as being implemented as their own classes this sped up the development and made the project uniform.

5.5 Testing

5.5.1 Interview data

Regression testing was something the lead developer spoke of as the most necessary and important step during their development process in terms of testing. They had separate employees working on quality assurance for verifying all the necessary components. This involved manual testing like purchase flow and verification steps that could be unique for a specific country etc. They checked this frequently and extensively during every single code review.

5.5.2 Experiment data

During the experiment two of the three methods of testing, mentioned in the theoretical framework were implemented in the Portfolio application developed by us. For the unit testing to work, an external package called Mockito had to be used. The reason behind this was because all the data in the Portfolio application was controlled by a fire-store backend service. Therefore, to test the functionalities of the classes without depending on a database connection, one can use mock objects to do the mocking of a real service.

The experience gathered was that this way of doing unit testing ended up being complicated. Despite extensive research and troubleshooting work, the unit testing did not end up working correctly. This was due to errors received when using Mockito that permitted us from mocking a fully runnable test case.

The second implementation of a test scenario was a simple widget test. This part was very intuitive and easy to understand, it was already prebuilt within Flutter and did not rely on any external packages. The experience is that this part was well thought out from the creators.

5.6 Deployment

5.6.1 Interview data

The Google developer expert and the lead developer both brought up the same reasoning when talking about deploying a Flutter application. Shipping a Flutter project for the first time or deploying a new version of it, results in all target devices getting the same updates and features within the same time frame. The benefit of having a unified deployment schedule between iOS, Android, and Web. leads to more consistency between platforms.

5.7 Maintenance

5.7.1 Interview data

According to the Google developer expert one crucial part of developing with Flutter is that a developer can build identical applications for multiple platforms with a single codebase.

This allows a development team to build and maintain applications with just a single developer. This is crucial when working with maintenance because bug fixes and new features is created once and shipped to all platforms simultaneously

5.7.2 Experiment data

The experience when implementing small features or bug fixes, was that it is very similar to the work being done in the coding and design phase. A feature or bug only needs to be solved within the same codebase and all target platforms will receive the update when the new version is launched.

6 Analysis

6.1 General

During the experiment phase a similar experience was shared by us regarding the development process. Previous projects of similar size and scope as the one built during the experiment have been created along the university period, but with different languages and technologies. The impression here was that nothing during the actual development process stood out while using Flutter to build applications. The same backend project had to be used and built for Android, iOS, and web, respectively. The same version control system had to be setup, and the entire agile process remained the same as with any other native technology.

The only potential benefit covered regarding this, was that you get a unified user experience across multiple devices with a single codebase that is described in the results chapter for the software development process. As pointed out in the interview, most of the code is the user interface, not the logic. Therefore, reusing code across several platforms that are supposed to behave in the same way is a good approach.

All cross-platform SDKs and frameworks changes the software development process by their nature of just having to create one codebase that can function on several target devices running different operating systems.

Many of the benefits of using cross platform tools is due to having a single codebase. Code only needs to be written once for both the user interface and the business logic. Creating and planning the solution architecture can be achieved with just a single instance, and only one codebase needs to be maintained.

6.2 Planning

One of the key points brought up from the result chapter regarding planning was about gathering requirements. Both the Google developer expert and the lead developer spoke of this process as something that is not affected by the type of software used by the company. Requirement gathering highly depends on other reasons such as, company structure and listening to stakeholders or clients.

A reason for this conclusion might come from the way both these developers worked on a day-to-day basis. Both had an agile approach to their workflow and according to literature, several agile methods exists that all have a phase/step for gathering requirements. But the common factor among them is “to deliver software faster and ensure that the software meets customer’s changing needs and expectations” (C.R & Thomas, 2011).

This can, therefore, point towards the fact that the company formation and the structure of how the workflow goes, plays a key role in how requirements are gathered and not because they necessarily use Flutter or any other type of software product.

Another point brought up was in terms of forming teams. The results indicate that forming teams is no different in Flutter compared to native development when speaking purely of team size. The same product can still be developed, but the difference becomes apparent during the process of dividing the work. In Flutter a team of the same size can according to the Google developer expert, achieve the same results as with native development but much faster. This could indicate that Flutter might be able to achieve the same work with less developers compared to native. This can therefore save time, money, or both by speeding up the workflow, and cutting the number of developers needed.

The importance of being able to achieve more work with less developers is an important factor for productivity. Previous research is all on board with less communication overhead being a good thing. Data from International Software Benchmarking Standards Group shows that three main factors impact software development productivity, with team size being one of them (Rodríguez, Sicilia, & E. García, 2012). In a paper written by Rodríguez et al (2012) the average team size of 9 or more people leads to less productivity during development.

6.3 Coding

In a bachelor thesis written by Christine Björemo about app development in cross-platform tools. An interview process with several senior and junior developers was conducted. The overall experience gathered from this thesis points towards similar benefits with Flutter. Hot reload and good documentation was pointed out as being two of the biggest benefits for Flutter. The senior developer goes as far as saying that hot reload was the single biggest contributor to a faster development time in Flutter (Björemo, 2020).

This together with the lead developers' positive feedback on hot reload and our own experience with specifically hot reload and not hot restart, strengthens the verification of hot reload being a beneficial tool during development.

Another bachelor thesis conducted by Lifh & Lidholm, (2018) looked at the portability potential of taking an already existing fitness application and turn the preexisting application into a single codebase using the cross-platform tool React Native. Despite using another cross-platform tool the result in this paper correlates strongly with the data collected from the interviews in this thesis. Everything ended up working with just a single codebase except for handling Bluetooth as a background task on Android. (Lifh & Lidholm, 2018). This led to different Bluetooth implementations between iOS and

Android which goes against the purpose of utilizing a cross-platform development tool (Biørn-Hansen, 2019).

The verdict from the drawbacks collected in the interview, together with the similar limitations with cross platform competitors' points towards this being a limitation not only in Flutter, but also for cross platform tools in general when trying to handle very specific tasks that can differ between operating systems.

6.4 Design

Flutter has a complete set of unique widgets. With a UI being completely built up by different widgets, this allows reuse of widgets and code and improves the time required to develop apps (Björemo, 2020).This is especially true when creating your own custom widgets where separation has been done of the UI and its business logic (Mehta, 2020). This enables the widgets to be reused and customized without being duplicated in development. The approach with widgets seems, both from the experiment and secondary research, to make the code more purposeful and the reusability of code gets higher and this speeds up the development.

With declarative UI the focus is on what should be accomplished rather than *how* it shall be accomplished with imperative UI (Kayere, 2021). This approach abstracts away details of a complex UI and allows the code to visually convey the structure of the UI. Benefits of developing in a declarative way is that it improves the speed of development, and it also reduces the lines of code needed and makes the code cleaner (Guster, 2020).

All data gathered about declarative UI point toward it being the faster and cleaner way of creating UIs. Despite the hurdles of getting into it for the first time as mentioned in the results chapter.

For native mobile development declarative UI has been implemented by Apple for iOS with SwiftUI as of 2019 (SwiftUI, n.d.) and with Jetpack Compose which is currently in beta for Android (Jetpack compose | Android developers, n.d.).

The design and UI created in the experiment application, became almost identical and the UX of the application on all the different platforms became uniformed without needing additional work for specific platforms. There are still some variations of the UX between the platforms that are caused by the native functionality of the different operating systems. For example, the back button on OS-level is placed differently on a browser vs an Android mobile device.

6.5 Test

The lead developer spoke of the importance with regression testing during the development process. Testing in general seems to be an important step for any type of

software development. In fact, “50% of a project’s cost is spent on software testing and 80% of this amount is consumed by regression testing” (Kazmi, Jawawi, Mohamad, & Ghani, 2017) Regression testing ensures that already tested implementations of a feature does not get affected by the introduction of new changes within the system (Kazmi, Jawawi, Mohamad, & Ghani, 2017).

The interview data points towards testing being a cost and time intensive process even for them, despite utilizing Flutter. This points towards Flutter not being different in this regard compared to other software development kits.

The experiences and struggles described in the results chapter regarding unit testing in Flutter led us to analyze this section a bit more afterwards. Further research on unit testing in general, did not portrait it like a tedious and difficult task to achieve on simple logic. This led us to believe that the specific problem was something unique to the test case built in this experiment and the Mockito package utilized to achieve unit testing.

6.6 Deployment

From the interviews an important point was raised regarding launching a new app as well as updates for an application. Because a Flutter project is written with a single codebase, all target devices can receive the same updates and features within the same time frame.

This ensures that users of different platforms can experience the same updated version of the application simultaneously, which gives a unified user experience. The benefits of a shared UX design are talked about more in the design phase above.

6.7 Maintenance

The maintenance work gets the same general benefits as every other phase, due to Flutter being a cross-platform development kit. A codebase can be maintained with just a single developer, while bug-fixes and new features just needs to be implemented once. Unless the bug is an OS specific bug, then in some cases it must be solved natively for that target device. This requires experience of writing native code for the device.

As referenced in the problem statement, maintaining several codebases results in more time when bug-fixing and implementing new features (Biørn-Hansen, 2019). This further indicates that maintaining a Flutter application is easier and faster than native applications with several codebases, at least for multi-platform solutions that do not require specific native features.

7 Discussion

7.1 Result discussion

The purpose of this study was to evaluate Flutter in the context of the software development process when creating multi-platform solutions. The study highlighted some potential differences between working with Flutter vs conventional development.

The interviews with the Google developer expert and the lead developer gave valuable insight into how the software development process is affected when Flutter is used. The interviews were semi-structured which allowed the respondents to talk in an open and unrestricted direction of what their thoughts were on Flutter, why they chose to use it, and what its benefits, and drawbacks were.

Most of the answers to the questions received in the interview process was in relation to how conventional developing differs. For instance, it is hard to talk about how declarative UI is better, worse, or more suitable without implying how imperative UI compares. Due to this we got a lot of insight into how the respondents worked with Flutter and how that compares to traditional development.

The experiment of creating a multi-platform application in Flutter gave us firsthand experience of working with Flutter. The work was documented in a diary that followed a similar structure to the software development process, which aided the analyzing process of the data.

The results of this study can inform of how some of the aspects of working with Flutter can look like, a few differences as well as some benefits and drawbacks are examined as well.

7.2 Limitations

Despite reaching out to approximately twenty companies and developers with the goal of conducting interviews we only got 2 interviews. it is therefore a bit hard to generalize the results from the interviews.

Secondary research was done with aim of giving more context of how the different phases of the software development process can look like in Flutter which gives more strength to the findings of the study.

The lack of studies where SDKs and frameworks have been qualitatively evaluated for Flutter in the context of the software development process as well as evaluation of how working in Flutter can look like, led us to find additional secondary sources besides studies in the form of videos and articles of how Flutter development looks like. The authors of these sources were verified experienced developers. The process of verifying them was to both take into consideration the channel to host the articles as well as looking up the authors on social medias as LinkedIn and Twitter. The criteria to be used

as a source was to have at least a few years in the industry as a developer and that the host of the website is a credible outlet.

The experiment was conducted with the aim of gaining firsthand insight into the software development process. But due to our inexperience with Flutter much of the focus of the work was on the technical details of creating our application, instead of the actual software development process. This made it difficult to draw detailed and specific conclusions.

The observations from the experiment were also based on our own experience and opinions which could make them biased as well as hard to replicate since it highly depends on the individual

8 Conclusions and further research

8.1 Conclusions

The question that was answered in this study:

- How can a development process in Flutter look like?

This was answered through the form of collecting secondary research, interviewing developers with experience in Flutter and an experiment phase of developing an application Flutter.

The second question that was answered:

- What are the differences between using Flutter vs conventional development in the context of a developer's perspective?

This question was answered through analyzing the interview data, experiment data and previous research to conclude how a development process can look like in Flutter. To compare this with conventional development more secondary research was collected and some of the more generalized interview data was analyzed for this part.

Some conclusions regarding Flutter vs conventional development can be established through the results part laid out in chapter 5 together with secondary research in chapter 6.

The results point towards some general benefits of using Flutter when creating multi-platform solutions. These benefits are only applicable when one is to create a product that shares the same user interface and have no OS specific features that requires need for deep integration into the targets device system.

Only having a single codebase seems to affect several phases of the development process. The work being done is effectively cut down from three codebases to one. Fewer developers can achieve the same work compared to conventional development. This also means the developer does not need as many different areas of expertise to multitask between.

Technical aspects of Flutter, like hot reload and declarative UI and good documentation also seem to have a positive impact on the development process.

8.2 Implications

The results from the study indicate that using Flutter to develop multi-platform solutions speeds up the development in almost every phase of the development process. At least where the functionality does not rely too heavily upon the target device's hardware.

8.3 Further research

To give more legitimacy and validity to this study more research could be conducted on how Flutters different components can affect the entire software development process. A broader survey or more interviews with experienced and non-experienced developers in Flutter could be conducted to gain more generalizable conclusions.

A deeper look into each part of the software development process would be interesting. To describe and categorize the work being done within them.

9 References

Addy Osmani, I. G. (2019, September 23). *Speed is now a landing page factor for Google search and ads | Web.* Retrieved from Google Developers: https://developers.google.com/web/updates/2018/07/search-ads-speed

Apoorva Mishra, D. D. (2013). A comparative study of different software development life cycle models in different scenarios. *International Journal of Advance Research in, 1*(5), 64-69. doi: 10.17148/IJARCCE.2016.5246

Arroyo, P. P. (2020). PWA and TWA: Recent Development Trends. In V. D. Aguirre, *Computer Science – CACIC 2019* (pp. 205-214). Switzerland: Springer International Publishing. doi:10.1007/978-3-030-48325-8_14

Aycock, J. (2003). A brief history of just-in-time. *ACM Computing Surveys,, 35*(2), 97–113. doi:https://doi.org/10.1145/857076.857077

Bengtsson, M. (2016). How to plan and perform a qualitative study using content analysis. *Elsevier Ltd, 2*, 8-14. doi:10.1016/j.npls.2016.01.001

Biørn-Hansen, A. G. (2019). A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development. *ACM computing surveys, 51*(5), 1-34. doi:10.1145/3241739

Björemo, C. (2020). *A qualitative study about Flutter.* Karlstad : Karlstads Universitet .

Boukhary, S. &. (2019). A Clean Approach to Flutter Development through the Flutter Clean Architecture Package. *2019 International Conference on Computational Science and Computational Intelligence (CSCI),* (pp. 1115–1120.). Texas: IEEE Electronic Library. doi:10.1109/CSCI49370.2019.00211

C.R, K., & Thomas, S. M. (2011). Requirement Gathering for small Projects using Agile. *Computational Science - New Dimensions & Perspectives*, 122-128.

Carrillo-Tripp, M. A.-R.-R.-T.-R.-V.-Z.-V.-V.-R.-V.-V. (2018). HTMoL: full-stack solution for remote access, visualization, and analysis of molecular dynamics trajectory data. *computer-aided molecular design*, 869-876. doi:10.1007/s10822-018-0141-y

Chadha, S. B. (2017). Facilitating the development of cross-platform software via automated code synthesis from web-based programming resources. *Computer languages, systems & structures, 48*, 3-19. doi:10.1016/j.cl.2016.08.005

Dart. (n.d.). *Dart2js: Dart-to-JavaScript compiler*. Retrieved 05 23, 2021, from Dart programming language | Dart: https://dart.dev/tools/dart2js

Dart. (n.d.). *Dartdevc: FAQ*. Retrieved 05 23, 2021, from Dart programming language | Dart: https://dart.dev/tools/dartdevc/faq

Dart. (n.d.). *dartdevc: The Dart dev compiler*. Retrieved 05 23, 2021, from https://dart.dev/tools/dartdevc

Dart overview. (n.d.). Retrieved May 08, 2021, from Dart: https://dart.dev/overview

Dawson, C. &. (2014). Software Development Process Models: A Technique for Evaluation and Decision-Making: Software Development Process Models. *Knowledge and Process Management, 21*(1), 42–53. doi:10.1002/kpm.1419

Dhillion, S. M. (2015). An evaluation framework for cross-platform mobile application development tools. *Software: Practice and Experience*, 1331-1357. doi:10.1002/spe.2286

Flutter . (2020, March 19). *web*. Retrieved from Flutter: https://flutter.dev/web

Flutter. (2021, 02 27). *Flutter*. Retrieved from FAQ: https://flutter.dev/docs/resources/faq

Flutter. (n.d.). *Building a web application with flutter*. Retrieved 05 23, 2021, from Flutter - Beautiful native apps in record time: https://flutter.dev/docs/get-started/web

Flutter Team. (2019, May 7). *Flutter: A portable UI framework for mobile, web, embedded, and desktop.* Retrieved from Google Developers Blog: https://developers.googleblog.com/2019/05/Flutter-io19.html

Flutter. (n.d.). *Testing flutter apps*. Retrieved 05 23, 2021, from Flutter - Beautiful native apps in record time: https://flutter.dev/docs/testing

Flutter. (n.d.). *Web renderers*. Retrieved 05 23, 2021, from Flutter - Beautiful native apps in record time: https://flutter.dev/docs/development/tools/web-renderers

Guster. (2020, 05 29). *Declarative UI rolling into Mobile 2020 and beyond*. Retrieved 05 23, 2021, from Medium: https://medium.com/@gusterwoei/declarative-ui-rolling-into-mobile-and-beyond-966f49d055f4

Hammond, M. W. (2020). *Research Methods: The Key Concepts.* (2. edition, Ed.) London: Taylor and Francis.

HTTP Archive. (2021, April 30). Retrieved from HTTP archive: Page weight: https://httparchive.org/reports/page-weight?start=2018_01_01&end=latest&view=list

Jetpack compose | Android developers. (n.d.). Retrieved 05 30, 2021, from Android Developers: https://developer.android.com/jetpack/compose

Kayere, P. (2021, 04 19). *Declarative vs Imperative UI in Android*. Retrieved 05 23, 2021, from Section: https://www.section.io/engineering-education/declarative-vs-imperative-ui-android/

Kazmi, R., Jawawi, D., Mohamad, R., & Ghani, I. (2017). Effective Regression Test Case Selection: A Systematic Literature Review. *ACM Computing Surveys, 50*(2), 1–32. doi:https://doi.org/10.1145/3057269

Lifh, O., & Lidholm, P. (2018). *Recreating a Native Application in React Native - Feasibility of Using React Native With Bluetooth & Background Processing.* Karlskrona: Faculty of Computing Blekinge Institute of Technology.

Manhas, J. (2014). Comparative Study of Website Page Size as Design Issue in Various Websites. *International Journal of Information Engineering and Electronic Business, 6*(6), 33-39. doi:10.5815/ijieeb.2014.06.04

Mehta, A. (2020, 05 22). *Flutter: Reusable Widgets*. Retrieved 05 23, 2021, from Medium: https://medium.com/flutter-community/flutter-reusable-widgets-38e270846d59

Mercado, I. M. (2016). The impact of cross-platform development approaches for mobile applications from the user's perspective. *WAMA 2016: Proceedings of the International Workshop on App Market Analytics*, 43-49. doi:10.1145/2993259.2993268

Mockito | Dart package. (n.d.). Retrieved 05 23, 2021, from Dart packages: https://pub.dev/packages/mockito

Pham, T. (. (2019, October 16). *6 stages for software development procedure you need to know*. Retrieved June 8, 2021, from Saigon Technology - 6 Stages for Software Development Procedure You Need to Know:

https://saigontechnology.com/blog/6-stages-for-software-development-procedure-you-need-to-know

Poeplau, S. &. (2020). Symbolic execution with SymCC: Don't interpret, compile! *2020 USENIX Security Symposium*, 181-198.

Rodríguez, D., Sicilia, M., & E. García, R. H. (2012). Empirical findings on team size and productivity in software development. *Journal of Systems and Software, 85*(3), 562-570. doi:https://doi.org/10.1016/j.jss.2011.09.009.

Ruparelia, N. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes, 35*(3), 8-13. doi:10.1145/1764810.1764814

Saldaña, J. (2011). *Fundamentals of qualitative research.* New York : Oxford University Press.

Sauma, R., & Ziai, M. (2020). *Komparativ studie mellan React-Native och Flutter med avseende på utvecklarens produktivitet.* Bachelor Thesis, Jönköping University, Computer Science and Informatics , Jönköping. Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-49913

Skia. (n.d.). *CanvasKit - Skia + WebAssembly*. Retrieved 05 23, 2021, from Skia: https://skia.org/docs/user/modules/canvaskit/

Skia. (n.d.). *Documentation*. Retrieved 05 23, 2021, from Skia: https://skia.org/docs/

Solovei, V. O. (2018). THE DIFFERENCE BETWEEN DEVELOPING SINGLE PAGE APPLICATION AND TRADITIONAL WEB APPLICATION BASED ON MECHATRONICS ROBOT LABORATORY ONAFT APPLICATION. *Automatizácia Tehnologičeskih i Biznes-Processov, 10*(1). doi:10.15673/atbp.v10i1.874

Stewart, D. W. (1993). *Secondary research information sources and methods.* (2nd ed ed.). London: SAGE Publications, Inc.,.

SwiftUI. (n.d.). Retrieved 05 30, 2021, from Apple Developer: https://developer.apple.com/xcode/swiftui/

10 Appendixes

10.1 Appendix 1 (Email sent out to participants)

Inquiry to you who are a developer.

The product development process is a generalized view of how a company works with software solutions from start to finish. The process is intended to cover the entire development of a web-based product. This is an overarching view of the production of the product. One part of the product development process is a development process, which is a more detailed look at the work process being undertaken on a more technical level for the development and interplay among team members. The reason why we want to conduct research in this area, as well as get a better idea of how to develop web solutions, is because we want to see the possibilities of being able to apply such a process in Flutter. The purpose of the study is therefore to investigate the possibilities of how a development process in Flutter can differ from traditional development. Traditional development refers to developing solutions natively. So, for websites this refers to using HTML, CSS, and JavaScript together with multiple associated frameworks and technologies. While for mobile using Kotlin for android and Swift for IOS respectively.

The reason why we want to interview you is because you work with Flutter development. Participation is completely voluntary, and you will at any time during the interview process be able to cancel your participation if desired. The interview will be recorded if there is consent from you. the material we collect will only be processed by us and no one will subsequently be able to identify that you are the one who answered this. The interview is estimated to take about 1h and will be done orally. If you do not have any preference, the interview will be conducted via Microsoft teams.

It would be beneficial if you would consider being prepared to draw a picture during our interview that visualize phases and sub-parts of the development process. This can be done either on a whiteboard, on paint or in any other method of your choice.

The results from this interview will be the basis for defining how the development process can look like when developing in Flutter. This resulting process will be emulated by ourselves as we develop our own web solution in Flutter. Therefore, we need as detailed of a description as possible of the development process to be able to replicate it and for its conclusions to be of value. The results that this interview will provide, will be presented in our bachelor thesis.

10.2 Appendix 2 (English version of the interview process)

Questions:

1. How long have you been working in software development?
2. What kind of Flutter project are you working on?
3. What is the structure of the teams working on a project? And what are their roles?
4. Where is your place in this team structure?
5. If you divide the entire overview of the product development process from start to finish into different phases / steps, what could these be like for you? (could you draw this for us?)
6. When the project ends up at your team, how would you describe the development process for you?
7. How does the interplay between the different parties involved in the development process look like? I.e., developer, designer, team leader, clients
8. If we go more into detail of the development of the codebases, how does the structure of the work process look like when you start coding on the project? (what components does the coding have? in what order are they made? When is it all put together?)
9. Which additional framework/technologies do you use besides Flutter to develop web-solutions? (and why)
10. Before Flutter, which frameworks/technologies did you use?
11. How does it affect your work when you use Flutter instead of several different framework and technologies? For example, synchronization between the different components?
12. In your experience, how have Flutter affected your work process compared to when you used to develop websites/apps traditionally.
13. Explain how things could be easier. (why are you choosing flutter over traditional development)
14. What else do you think is relevant for us to know when we are to simulate/replicate your process when developing in Flutter?

Follow up questions:

- what do you mean?
- can you expand on this?
- can you give further examples of this?
- why/why not?

10.3 Appendix 3 (Interview with Google developer expert)

Talking about development process in flutter:

(application) Are now working for multiple platforms, but it really doesn't matter. It's still mostly a team size between two and five people. Everything is the same.

Its still a client-side application, we don’t write any server-side code as such. So, the client-side application at the end we bundle our application for the multiple stores, and we publish it.

work experience: (Q1)

Uh, my finger round about 10 years. So, when I was in University I was also working in parallel. Then I did six years of native development and I'm now so three years University with our where I worked as a front-end developer. With Flash and then I switched to Android for six years and now I'm doing flutter for over 2 years now.

Current project:

Uhm, multiple ones. So, I'm leading an agency and we have multiple projects from mobile projects, pure Android and iOS to also desktop first applications which we publish to WAP Windows and Mac OS.

Strucutre of the teams

It really depends some. Usually there's like two to three people per project and they work on multiple features. But on other projects it's more depends on. The kind of projects or the project itself. Then it depends on the technology, so it's as I said, it's the same for native development. We have the same. Team sizes for the development team as we did with native development. It's all the same round about between two and five people, but it really depends on how much workload we have per project.

where is your place in these team structure?

I'm still part of the development team, but sometimes clients call me solution architect so I'm not working on features myself. I help the whole team to deliver the features and organize how we work with multiple back end teams and things like that. So, making sure that we have all the requirements and everything is delivered on time so that we can actually start developing.

And if you divide the entire overview overview of the product development process into different phases, how would they look like?

There's one phase which is development. Maybe beforehand there is a small phase of sales, so we are an agency. We do not develop our own products. When clients come to us, they want us to immediately start developing the application and we tend to do so right away. So, we start building the application and handle all the issues which pop up or all the missing requirements that happens on the fly, but it's still part of the development process. We cannot say like we are working in Scrum and Kanban. Depending on the project and there is no like. OK, first we do three months of requirements. Then we do two months of design, then we do six months of development. It doesn't work like that. Everything works in parallel.

And then when the project ends up at your team, how would you describe the work process? Of the development process.

I mean yeah, when we start, we First off look at the first features we want to implement. Do this in some kind of refinement, which is a scrum process, and then we plan ahead so we have a project manager which defines the tickets we want to work on, and then we go one by one ticket after the other. Uhm, we have continuously, uh, refinements So we refine our tickets in our backlog salty priority with our clients together. And depending on that work on the tickets with the highest priority.

How does the interplay between different parties involved in the development look like?

So, we work all together. We are a big team, or we have big teams which all work together. There are no clear roles. Of course, we have designers which are only do design. We have developers which only do development, but from the developers it really doesn't matter if you are working on code or you're working with the back-end team to define an API, it's all part of the development process. So, development is not just writing code, it's also like. Solution architecture and things like that. Uhm, and if we go more into the details about developing the actual code bases and stuff, how is the structure of the work process? There is no structure that it really depends on the client and on the tickets, we want to work on. So, if we do say we want to work on that feature, we start with that part of. But there's nothing like OK before when we start on. From a project, we first implement the database. It doesn't work like that. We don't divide it into layers. We work by feature and we do everything to make the whole feature work.

additional framework and technologies beside flutter you usually use in a product?

That also highly depends on the kind of project. So, if we work on a project which involves Bluetooth, we have some Bluetooth libraries there. It highly depends on the back end to vacuuming technologies. We have projects which work solely with Firebase. Other projects have a whole back end completely written in Node or Alex ear or whatever. We as in flutter Agency we focus on purely flutter and we try to build everything in flutter where it makes sense and where it doesn't make sense. We I don't have partners or other agencies which help us building the other stuff like the backends for example. Sometimes we've built some basic web application in Vue JS because it was faster than or it was more appropriate because it was a pure website. Flutter would not be the right choice. But there's no such thing as if we build a flutter application. We also use 100% rife, for example, which is the animation library. It really depends on the project.

Why are you using flat though? (what's the benefit)

take away is that I as developer think that flutter is superior of building user interfaces to traditional Android and iOS. Speaking of Android and iOS, as before, Swift UI and before cuddling compose Jetpack Compose was released because declarative building UI is in the declarative way. As we are used to it by React and Vue is so much better and this really convinced me to switch. Over to flutter and that's why I'm ended up doing flutter and now that the technologies like Swift UI, so the declarative UI is also in native development, there's not. So, this doesn't count anymore, but on the other hand, we are able to build. Applications for multiple platforms. Identical applications for multiple platforms were for single codebase and not only that. We also can build applications or maintain applications with just a single developer, which is crucial when we come to a bug fixing phase, because in a traditional setup. We would have to hire at least three developers, one for Android, One for iOS, one for Web. To basically build the whole applications or to ship back fixes to all free applications. There are rarely developers doing native Android native iOS and web together. You always need a free person, but with flutter everything can be done with a single by a single person.

why is it beneficial to have just one codebase for the maintenance mode?

And that's mostly because bug fixes have only to be implemented once, features even. Even if you try to share code between Swift and Kotlin in on the native side, you can do that with. JavaScript with some Kotlin native, for example with C++. I tried all of that in my native career, and although we are able to share some code, most of the code is not actually the logic, it's actually the user interface, and being able to also share the user interface. Saves a ton of time. And that's why a single codebase makes a lot of sense, because in most apps most code is shared between the platforms. Now if you look at applications today, they look very similar and behave very similar. There's rarely a difference between multiple platforms, and yeah, so why implement the same thing? Multiple times if we can just share it.

And if we were to follow your development process when we, uh, code in flutter?

Even if you have zero knowledge in flutter, you can get started in one to two weeks with flutter and you are a good developer already whereas we. With native development, you really need to have like a year or two years of experience to build the same quality of applications. So, flutter really removes all hurdles for newcomers to build applications. It is sometimes unusual for people to build applications in a declarative way, that's something. People must get used to, especially if they have a background in C or Java. But once it makes click, it's so much faster to build applications and the only thing which is compatible is building a React application. It's the same thing, but it's. Not multi-platform. So, I would always recommend building applications in flutter, whatever it is. Expect for web applications, so if you want to build an app just for web, maybe investigate Vue JS or next JS or whatever.

And why is that?

If you just focus on your app, you mostly want to focus on web API's and it's often easier to connect them to directly to JavaScript or in JavaScript. We had one project for example, where we wanted to include JavaScript applications. Or I think it was. Push Firebase push notifications thing something like that. Anyways, it was not already wrapped in a dart package, so we manually had to connect it to come to our flutter application, which is absolutely doable, but maybe not as a beginner, and especially not if it's your first or second week in flutter.

Uh, is there any disadvantages of using flutter?

there can be disadvantages so you move away from the platform and all you get is basically drawing on a canvas. That's all you get with a flutter application. So, flutter is yes, it is an SDK for building applications, but mostly it's a UI toolkit, so whenever you want to Draw Something. That's easy and you get access to the native platforms by using plugins and things like. That the problem is if you want to use latest or very specific native features. For example, you want to show a specific application specific notification on Android, including a picture and things like that. If that's not 100% supported by the plugin, you may have a hard time calling these API's. And you have to go manually in Android platform channels and things like that, so it's very easy to build an application which draws on the screen and it's interactive, but it might be hard to build an application which is really hard to connect it to the operating system, like doing notifications. Doing Widgets on the home screen, for example. That's also not native flutter. If you want to have background jobs, that's very hard to do in flutter. So, for example, you want to scan for Bluetooth devices in the background while the app is not active, and that's harder. And if you want to integrate a lot of web views. So, think about the payment process where you must type in your credit card number and then you have to show a website for them for the 3D Secure check. And showing the website in flutter in line is not good a good experience because the keyboard behaves a bit differently than comparing. Showing a WebView in a native application. But there are ways around that and that's the only thing you must solve so we. Tell our clients. Yes, we can do everything in flutter, but if we run into a place where we cannot do it in flutter, we will do it still negatively and this might require some changes to the UI. But we had this client which wanted us to show our inline web view in one of the websites. So just a small part and that we can type in some code there some code from a lottery ticket. And we were not able to do this reliably on. I think it was iOS, but I can't remember and instead what we did, we just showed the button and when the user tap that button, we showed the native UI where the user could type in that thing in the WebView. So, there are some work arounds which you must be aware and which clients. Also, must be aware and then it works. But in the end, you can do everything with flutter and if it doesn't work, you always have the option to go into the native part and do it there.

there is no real difference for the actual development process. for the teams to work. Since you skip a few technologies in HTML, CSS and how we cooperate around those areas, is there any?

Only if you think about building multiple applications at once. So if you have a team of like one person and you want to build a native application for Android and iOS, that person has to know native Android and native iOS of course, and then you save a lot of time because that person only has to learn flutter ones. Sure, but if you think about enterprise scale where you anyways have like 5 people working on a product, you can divide them by native Android and native iOS developers and there's no difference in that sense. But with Flutter you most likely will be faster. But if it's just UI. you are faster by doing it in flutter because now five people can work on five features, whereas if you split the team you only have two persons per platform and then they can only work on 2 features at the same time. So of course, you will be faster with flutter, except if you run into. Problems where you must write native code at one point and then you might be even slower. I rarely see that happen because most applications really have a lot of UI and a lot of business logic which can be 100% written in flutter. But some of our clients really had a hard time to understand that showing a building like 5 screens in a row was as easy as building a single screen with a web view, because now suddenly we had to support that revenue on three different platforms. So we had to build 3 different implementations for that. whereas the other screens were pure flutter, which were much faster.

Yeah, you meant you talked earlier about it, that it is fast to get into flutter that it only required like 2 weeks. (why is that)

because flutter is using declarative UI, which abstracts away a lot of things so. Speaking of Android, for example, if you want to build a user interface in Android which is bit interactive, you already must understand the whole application lifecycle. How on resume works, how Unstart works and things like that. That's a solves flutter entirely for you, you don't. Even must think about it. All you must think about is how are my buttons on the screen layout it and what should happen if I click on it. You don't have to think about. What if my app isn't background water? If it my app is in foreground, you don't have to think about threading anymore on Android and iOS, you cannot run HTTP request on the main thread with flutter HTTP requests automatically run on a different thread. You are not even able to switch to thread to the wrong one and. There are so many there, there's a big layer on top of the operating system. You don't have to think about anymore. That's why you are much faster.

10.4 Appendix 4 (Interview with lead developer)

Now we started roughly two years ago from now to rewrite it in flutter out before it was written in React Native, but we have some huge maintenance issues that we maybe get into later, so we rewrote it and then we rolled it out like country by country. And this is now currently that like the production one and it has about 7,000,000 Monthly users to buy something?

So yeah, that's just like the scale of the app and developer wise.

Like at any time like 5 to 6 developers work on it and then two QA people and one or two project managers.

Yeah, they just.

Basically scope there.

Uhm, how long have you been working in software development?

It's 11 years or something. Yeah, I started when I was like first semester University and then for like different technology stacks like initially desktop applications then web. Yeah and since flutter like a super recent technology like nobody will have. Done it like much longer I guess than our team because before it was like still beta and yeah.

And then, what is the structure of the team that you're working on currently, and what are their rules?

Yes, so as I said, it's like 5 developers. Round about like in our structure in general.

For other software we do on Company X, there is always one lead developer who is. I'm basically working more on the upcoming features, like talking to external stakeholders. OK, you want to do this? OK, what does it entail? Like? What kind of API's do we need? How long will it roughly take us? And then like the rest of the developers get like the ready to work on tickets? And yeah they implement them. Like in the day to day. Yeah, then we have. As I said, we have like 2 project managers there or like 1 1/2 always like one intern or something like that for like some menial tasks so so they do like the really heavy lifting. What would the stakeholder communication right so that all the developers can focus on the technical topics and then? Two QA for like the parts that still need like manual testing which for us is like important things like for example the purchase flow. Yeah, does it still work across all countries and that kind of stuff.

Yeah, and I think the general process is like for us at least. It's like 2 week sprints, so like every two weeks, we also want to push to the App Store. So this is like a relatively high frequency I think for like an app project, but for us it's like super important that like individual app releases. Small right, because on the app, so you can't basically take it back. If you make a mistake compared to web. So it's really important that the quality is always like solid and that we can push to this door and then we'll find that millions of people will get it like in the neck. Two days a week. Two days. Uhm, yeah and so so. Therefore we use like normal Sprint style.

And, uh, where is your place in this team structure?

What are you working with?

Yeah I I'm this like lead developer. So this just means that I don't get to write as much code as I used to, so it's it's more like communication with like people. OK, How could we do something like? What approach would we take? Or this kind of stuff? Or like guiding? New hires, because we're also constantly get like new people or the team like there's yeah, always something like a turnover. Especially since we have some. And we always have like some freelance people, right? So they have like six months. Maybe one year contracts. So yeah, there's constantly and like an influx of new people and then people leaving of course. Otherwise we would go way beyond the six we currently have there.

If you divide the entire overview of the product development process from start to finish into different phases or steps, what could these be like for you?

Yes, so as I said, like the whole pipeline, we basically have to be, uh, considering it's usually a like a stakeholder request. So this just means like like somebody from inside the company. Design is like a feature, right? I mean we also have some internal technical stuff, but I think they. Just for example, we want to do like a specific refactoring or cleanup or whatever, but they are just like a part of this whole process, so maybe we can start with this. so one feature we're currently working on.

It's like a development like kinda confidential, but it comes out in like a week so I can tell you it's for example a product reviews, right it's. It's very I.

Mean it's it's easy to grasp, right?

So on each product, like after you've purchased this, you now get ask hey? How how did you like the quality? Give your opinion what was was it fitting like you expected or is it like rather big or small or whatever so this is for example a feature that did somebody from the outside wondered, right? So it's not very. Technical yet it's just yeah OK. This is nice. Other online shops seem to have it. It seems to help consumers make like a purchase decision. Yeah. Also we can like reduce or returns. If people are more happy with their like products that they receive, they are less likely to return it. So this is like good and like overall. It's always like. Associated by business value. So I I guess for some features it you you really can't make like a. You can't put like a good figure on it, like how much more money will you make or how much money will you save in returns. But they always try to to estimate something though. So how important is it to to basically rank it against all the other features? Right? And and So what once it's decided OK? Like yeah, we want to do this like from a really high level.

For example in this case. Then we debated. OK, so basically what you need for this is you need like a database to store all the reviews, right? But reviews also have this problem. That's some Consumers will use like profanity in their reviews, right? So they will use swear words or whatever. So you have to filter that out. Uhm, so this for us then. Early on, there was a decision. OK, we don't want to deal with this. We don't want to read all the reviews, so. Uh, we decided to use like an external partner to validate the reviews right and and also store them for us, right? So all the reviews get basically sent to their database and they take care and they have other logic already in place. Yeah, that is like a really a helpful review at the end right? And then like for example, my role as like lead developer and product management. Management, so this is like. A D and PM for us was. First of all, I had communicating with the external partners. And then check out their API which is an and have this like high level discussions but which are

already like a little bit more technical right? So, like very initially, probably at this point, before here like product management. And and say it. Like make a deal basically and say OK, hey, do you offer this? Yes, we offer this. How much is it and all that kind of stuff right? I I wasn't involved with it so when I get into the process Yeah, this is already the stage so they say hey we would like to work with this partner. Do you think it's like a good fit technically and then yeah, we we check out the API. We have some cards with them. How do we integrate all that kind of stuff? For example, in this process that went well. So said yeah, OK, your API is like good enough, we can work with. That and then. And then the next step was, uh, then, internal topics, which started right. So like the external part we we know, OK, we can work with you, and then we do the internal one. And and for like. Our app specifically that means designs, so this is, uh. Already going on in parallel right as soon as you start have like a high level idea or I want to have this feature. As somebody is already working or designs OK, how can it look right?

And then you kind of iterate like what is really the scope or do we want initially so designs are like nearly finished already once we start but then they get like finalized. Uh, once the external. Dependencies are clear, right dependencies, so we know exactly OK. What can this? Partner provide, for example, do they have profile pictures for the users or not right? So this of course affects our designs. If we can show them or not. And then the next thing we do is like. Because most of this stuff. We do features always need some new data, right? It's never something or we just visually changed something in the app. So we always define like the API. Like let's say define internal API. Yeah, so for for us we we use like a GRPC service and there if if you want to get into the technology which is like their counterparts to the flat front end.

Transcript

These service there if if you want to get into the technology which is like their counterparts to the flat front end. Uh, and then from like. Designs plus API. This in theory will be built by another team, the API just so you're clear. So we basically spec. How should the API look? What kind of data we do? I want to post for each review. What do I want to receive as reviews, but let's just assume it's built by, like, you know. So yeah, if we have designs at API, then uh. Like PM, writes the tickets so. Yeah, yeah, just your normal like sprinting it. These mentioned the goals. API is to call it like mentioned designs. Design. And then implementation pins and edge cases. So yeah, so usually for the ticket, because like all the other developers, they might never have heard of the feature so far, right? So they get all this. Information is now transformed to the ticket, so we explain OK.

You were talking about API's or something I believe.

Yeah, but which I mean.

It's always about API's.

Yeah, OK, so yeah so so they don't like we write the ticket and so we reiterate. OK, like what is the overall goal? What are the APIs they should be called, right? It's the one we defined earlier here. Then, like reference the designs and then also give like already like OK what are like expected edge cases. Water like something to be away. Yeah, so that like if the developer gets to the ticket, this should be everything they need, right? Of course, there's always like a easy escape hatch if you have a question you can just ask some of the people who were involved earlier. But like ideally you want to get that down to like a minimum so that they can just start working on it. And then. Or you have. Uh, I mean, you

know, like the normal Sprint stuff, so ticket gets in the backlog and it's prioritized, blah blah blah. At some point this Sprint starts and then it's it's, uh, it's in the. Within this Sprint, so uhm, so for sprints. So we use like Kanban boards. With like the following columns, so it's like basically ready for dev, so this is something that any developer can just pick, right? We don't. Decide that like up front. It's very free, but of course sometimes people are more suited for like a specific task than others, right? And there's always like a prior label which means pick these first, right? This is like. And the column is ordered anyway. So it's just OK. Through these first, the rest is like nice to have, then you have like in depth or one specific person who's working on it. And then you have code review which is like probably. Yeah. So let's say. This I mean, sometimes it takes longer but like the. Uh, the target is like 12312 to three days. It should take for an implementation of a ticket, right like? Uh, yeah. Sometimes it takes longer, but then it's usually like a problem and if you have like a dependency or like if you have something super huge, it's always the idea to spread it up right. For example, front end, back end. So then the back end might take two days and the front end takes like an additional 3 days, but it shouldn't be one big ticket that takes like then five to six days in total because that's just like a mess. Because yeah. And that would be hard. So then code reviews like one to two days. It's mostly. Uh, I think our code review is very in depth, so we check everything. We like the feature but also for example, the automated tests to see that it's like tested well and but a lot of the time caught like even though the time for food is not that long for each ticket is maybe like one to two hours. People don't start right away because they have their own tickets.

And and and F is like two or three days. And as I said, code review is like very in depth. Share your screen again. Uh, I want to do that. And as I said like it, it only will take like one to two hours in practice. But since people are working on their own ticket at the same time, it's like a lot of the time. It just doesn't get. I have prioritized earlier, right? So people want to finish their own stuff as well, so it's always like a. A drawback, but yeah, this is probably also one reason why it makes sense that the other tickets are small so that people. Yeah, they they can focus on their stuff, but maybe just for two days Max or something. And then they should really see again with the team. OK, like what's important for others like where can I help so that they don't work endlessly just on their own stuff? In isolation, all the time. Yeah, then there's like a Q. 8 which is like specifically. They just have like a look on the on the overall feature, like from the user perspective. So not anymore the code or like on the. Automation site.

Part 2

Like OK after everything is merged together, so just see OK. So yeah, then then everything is like bundled together, gets tested again and then. And Oh yeah, then, uh, these regression tests. So this is like the most important functionality, uh, that that always gets tested like again and again for each release, even though it like didn't change, right? So like login is usually the same or whatever, but of course that needs to be tested that it didn't inadvertently changed for any side effect, yeah? Yeah, and then we pushed to the app source. Yeah we do it like 250 and then 100% or something. Yeah you just to see if anything comes up in the field that we didn't find in testing, right? So? It's like 2 weeks and then yeah, the that's like OK and then. It's it's finally done so, so this is like per ticket and then pre release there's. Uh, all tickets get tested again. It's probably not too bad, but we have quite some. Amount of code which is like a country. So yeah, it's it's really hard to test them like show how many countries you even have, but it's more than 20. So yeah, everything that might happen in the wild, it's really important

that we that we have a close look here. So I think this describes like the over all Process there quit well. To give an initial idea.

And then if we go more into the development, how does the?

Uh, basically here I missed something at the start of this Sprint. There there's always like an an intro where each tick or like make intro with the major tickets get get explained. So like even more background, everybody has seen it already. You go together through the designs. Give it like feedback from the developers. Say what would you think. How could we make it, for example, easier to implement or like even asking? Is it like super complicated to implement? Yeah, and give us some feedback. And then like like. Like this is the main development part, right? So on this like if you have any questions. Depending on the nature of the question, you either go directly to the other Department involved, right? So you talk to the guys who did the back end. You talk maybe to another Tech Department inside the company because they have the connections, or if it's anything that would affect like the. Like what the customer sees. Basically the scope of the feature or the visual stuff that always gets run by the product manager. So it's just this divide OK like technical stuff, you can figure out internally if you don't need to change the scope of the feature. But if for whatever reason you want to change the scope and making it like different visually or also just easier to implement or whatever, and then you would run that by the PM. Yeah, nothing important gets missed out right? Because it might be that if you don't do like a certain thing that the feature is basically worthless. You can't like benefit as expected.

So, and which additional frameworks or technologies to use beside flutter when developing your application?

("Goes on to explain dependencies used in the project")

So before you started using flutter, which framework or technology did you use previously in your application?

Before it was react native or with like TypeScript.

So how come you transitioned from React native?

So first of all, like like two things. With react native on the website and also react on the website. So we were like heavily invested in like React and also TypeScript. But we also didn't like a lot of it right. I mean if you did something in express or something like on the node side you you don't have all this pain that that the front end web developers have right? Which is yeah, you have to bundle the application. You have to code, spread it blah blah blah on the web so it's super painful. It's just like 2 days ago I was speaking with exact developers and I left. Like the team like two years ago and they are still having the same pain, which is like insane. They still have like explosions of their JavaScript size if they pull in certain dependencies and then they can't seem to find a way to fix it. So it's like really, really bad developer experience. And even though on other things it's also super great, right? Like all the interactivity you have in the browser is like amazing that you can't just like toggle some CSS flag and then see how it looks and. But yeah, but but basically our learning was like with like React native and then also redux for data storage that it doesn't really work on on an app this size on a team this size. Also sometimes if you have not if you have like. People who are not super familiar with everything. It's really easy to make a mistake, so either they don't know. The React Redux structured very well or also

yeah untypically about the the product itself. It's really hard to figure that out. And also with React native. Even though it wasn't too bad with our app, you still have this. Like the native part, right? So it's it uses some native components, so it might behave differently between iOS and Android. Which is like a huge development effort to really check. Does it behave as you want on both platforms? while in flutter you basically never have this problem, at least not in the kind of app we're on. Because you always know. OK, it gets good start on screen but flatter. It will look exactly the same. And the device specific bugs we had were like very few. And then it's really more like bugs and he worked with the framework to get them fixed. And it's never like that normal code would behave differently on the two platforms.

Yeah, OK, so you list there a lot of pros and benefits. What would you say are the core drawbacks or negative parts about using flutter?

It it, it really depends. We didn't get into all the benefits even. Yeah, I think like like for us it's really great. It makes a lot of sense. If you want to do certain things you you you can't do it well, uh, for example, on on Android you usually have like a ton of activities and like for each page. So to say right? And but on flatter you just have one with the whole app. So for us it doesn't really matter because like we don't want to integrate into the system deeply or we don't want to like integrate with other apps. But if you wanted to do this kind of integration, you would have a hard time do. Bring it also, for example, on iOS you have like a lot of native platform native things and you just can't do that with flutter, right for example. Widgets you can only write them in like Swift UI. You can't do it any other way, but since we don't want to do that, it doesn't really matter. Then like design is like a huge thing, right? So this is basically. They just simplified that because they say, OK, we we want to do this one style it. I mean, you can see the screenshots on the link I sent you earlier and also it should go to play store it looks exactly the same or not on Android. So it is kinda material style Google style. In black and white, and yeah, if you don't want to have the platform style then it's totally fine. Right? If you were to want the platform style then you're like. Out of luck like a little bit, because now you would have to program in two different models, right? And say OK even though I'm using a cross platform technology. You would do certain things like completely different between Android and iOS, so like normally if you wanted that there will be like a huge drawback, but since the company says no, it's totally fine. We just do one side and it does. Doesn't have to look either like Android or iOS. Then it's like a no issue.

You said that iOS and Android are your target devices. Uh, but you have a website as well. You don't create the website in flutter that then?

No no.

OK, so it's an entirely separate entity?

Yeah, this is still also react TypeScript. Based mostly yeah, yeah, primarily. When we were always eager to try it since they announced like Flutter Web support but. I think just like maybe four weeks ago there was this like flat at 2.0 event. Have you seen the screencast?

Yeah, it was a big thing that was on the web blah blah blah but. But the sample page they had online, I think it was like 50 megabytes, which is already like a no go for like a mobile thing, right? And it was like really limited in what it can do and like I don't know. So for us we always worry that it will be just way too big. Yeah. Especially for like a shop which would, which yeah needs like fast initial loading time and it's really not that complex, right? It's it's not really. You don't really need an app for for the core

features for like just the buying right. It's just OK like show some images, show the price added to the basket, buy it. There's nothing rich about it, so yeah. Then we we would think on the other side with like a lot of the dependencies. At least at the current stage, this is probably quite tricky to get, like the the more native dependencies to get them like cross platform so that they would also work on the app because they are the integration is often completely different right? So for example like uh, log in is not like a native SDK you call, but rather you have to you open like a pop up or like an I frame and and yeah it will probably.

Also, be like quite a. A decent amount of work to do this to even try it out to have something running. Yep, so yeah, but mostly currently like the block size size, otherwise it would be super cool if the download size weren't that huge. But yeah, it currently is.

Yeah, yeah, what else do you think is relevant for us to know when we are to stimulate your development process in flutter like?

Yeah, like like the hardest part. Sometimes it's like this code review part here, and this is probably also really hard to to emulate. Uh, so it it it. It's like really intense like how it gets verified and. Like initially, if. If we have new people onboarding, they are sometimes like a little demoralized because it it can take like days of discussion. Like just explaining. Why certain approaches should be done? How certain approaches might be useful? But I mean how? How would you run this if you have no experts on the team, right?

So so so this is probably like a huge problem. So initially when the when it was started we had like two people working on it and then it was 3-4 but uhm. Yeah, there will always make some way more experienced people than others on it, but of course like we also learned like a good bit on the job. So I guess like the the kind of feedback shifts it or like.

What we would recommend as best practices shifted completely from like maybe the first year to the second, but there were always like a lot of people doing it, and I think initially we also did it with more, just more ice so that everything was at least looked at by two persons.

If you are not super confident how stuff should work in it. But it takes like a lot longer because you you maybe look things up. Look for like you know, alternative approach. And yeah, I guess it is hard to imitate. For example, initially when when like dependencies are introduced right? So you in instead of just doing it during development and say here's like a new dependency. You would really think about. OK, like, what do we? But what do we want to do and then, like as a team like one or two people go out and then say OK, like how can we achieve this? Whether it's like the best practice. So like I think initially we were looking far more to the outside and then after like maybe half a year or something you know and you have your basic structure in place and then you're looking more on the inside. Make. Uhm? When you say. Uh, what are the approaches we use in different kinds of areas already in the code and then the feedback is mostly. And for these architectural things, hey, in another place we use this other approach. So do you think we should align your thing in that direction? Or maybe the other thing in your direction? If somebody has like a new approach and just just get like a discussion going that way?

OK, is there any other benefits to using flutter that you haven't mentioned?

I mean that. The language is also quite nice, so it's probably like one of the better languages. I mean TypeScript was also supervised. I think if you if you use it correctly because type sleeping basically go from like no guarantees, you could just write it like JavaScript to like super strict guarantees if you write it like we did an. And yeah, just the whole development. Like it's it's even more interactive than any type of web development I've seen on this like scale. Because yeah, you you can just hot reload everything basically and it just works and it's way better and way faster than what you have in the browser. Yeah, the hot reload seems nice I I've been using it. Recently Yeah, even like at at like a huge app it like doesn't breakdown it like it still works like just like on the first demo or something. That's really impressive.