



Flutter in cross platform development: Tools, performance and optimization

University of Oulu
Information Processing Science
Bachelor's Thesis
Teemu Väänänen
2025

Abstract

Mobile development has undergone changes in recent years, as companies aim to reach broader target audiences and expand their solutions. Cross-platform mobile application development has emerged as a solution, utilizing various frameworks and platforms. Cross-platform approaches are popular due to their ability to streamline development processes and create applications that more easily reach wider audiences.

The purpose of this study was to examine Flutter, Google's answer to cross-platform development. The study was conducted as a literature review where relevant academic sources were used to gather information on the tools and core concepts of Flutter, the performance of the applications developed with Flutter and possible optimization techniques for application performance.

The findings of the study revealed that various optimization strategies can be employed to enhance performance. The most critical strategies involve the efficient use of widgets—the building blocks of user interfaces—and careful handling and coding of the methods used to construct them. Regarding performance, the study found that Flutter applications achieve strong performance, primarily due to its rendering engine, which eliminates unnecessary intermediary layers between the application and the target device.

The study expressed Flutter's rapid evolution as a limitation. Newer versions may address some of the limitations highlighted, such as app size and memory use. Longitudinal studies examining the impact of these updates would provide valuable insights into Flutter's long-term adaptability, the suggested area for future research.

Keywords

Flutter, cross-platform development, mobile development

Supervisor

Lecturer Antti Juustila

Foreword

The rapid development of technology has equipped people of modern day with high performing devices ready to be used and deployed from one's pocket in an instant. Naturally, different components, making life more suitable, easier, and streamlined, called "mobile apps" create a vast market for developers to use to create new, more sophisticated, and computationally demanding solutions for people in need.

At the same time two operating systems (Android and iOS) for mobile devices compete together. These two platforms are also the main targets for mobile developers, and it is for certain, the who produces solutions for both, has advantages in the number of customers and can spread their product once for larger audience.

Different solutions are available to "code once, deploy everywhere". Flutter has emerged as a promising framework for this type of software development. It has also gained notable traction online by forming a large, diverse community of developers building applications. Personally, I came across Flutter during one of my optional university courses and was immediately impressed by how quickly one can make a user interface "come to life." Further researching and learning Flutter features such as "Widgets" and the programming language "Dart" made me more excited on learning and building little projects.

For new developers Flutter can be enjoyable with building little projects which can become a part of one's portfolio or can be used as tools for example. But, how about when a company such as BMW choose Flutter to build their mobile application and deploy it to millions of users? It raised a questions in my head. How will these applications perform? How does Flutter perform? Can you optimize performance if it does not exceed the goals?

With these questions in mind thoughts formed into a foundation of my bachelor's thesis.

I would like to extend my gratitude to Antti Juustila, who gave unquestionably valuable insight from the very beginning of forming the structure to actual writing of the thesis.

To conclude, I hope this thesis helps at guiding me towards the world of academics and research.

23.01.2025

Teemu Väänänen

Contents

Abstract	2
Foreword	3
Contents	4
1. Introduction	5
1.1 Background and motivation	6
1.2 Research Questions and Method	6
1.3 Research structure	7
2. Overview on Flutter and Cross-Platform Development	8
2.1 Introduction to Cross-platform Development	8
2.2 Overview on Flutter	9
2.2.1 Dart	10
2.2.2 Prebuilt widgets and UI components	10
2.2.3 Development tools	12
2.2.4 Community	12
3. Flutter's performance	13
3.1 Performance Features of Flutter	13
3.2 A brief comparison with another Framework	14
4. Performance Optimization for Flutter	16
4.1 Identifying performance bottlenecks	16
4.2 Key optimization techniques	17
5. Discussion	18
5.1 Tools and performance	18
5.2 Performance optimizations	19
6. Conclusion	20
6.1 Summarizing the key results	20
6.2 Implications for practical use	21
6.3 Limitations of this study	21
6.4 Recommendations for future research	22
References	23

1. Introduction

Mobile devices and mobile applications play a significant role in modern society, they have impacted on how people interact, work and carry out their daily tasks. Combining this with the vast availability of distinct kinds of devices, people now have access to a wide range of different services and functionalities from payment processing to social interaction and even remote work at their reach all the time anywhere in the world. (Souha et al., 2024.) This accessibility has made the daily routines of people more streamlined and also has impact on the productivity and even well-being of individuals.

Mobile applications can be principally categorized into two distinct types of applications “native” and “cross-platform” applications. Equally they have their own advantages and disadvantages. Native applications are developed for a specific operating system, offering advantages in performance and execution speed, as they are controlled and integrated within the platform-specific tools. These operating systems with the two most used being Android and iOS, dominate the mobile landscape, making native apps particularly effective on their platforms (Kishore et al., 2022). Often these types of applications rank higher in app stores due to their efficiency and provide users with the adherence to the native operating system’s look and feel. Additionally, application’s debugging process is easier, and the target device’s resource use is utilized better, ensuring high performance.

However, applications that run on a single platform have limited utility in today’s diverse platform ecosystem. Native applications, while optimized for their respective platforms, demand higher resources due to the need for distinct codebases and for example separate teams specializing in platform-specific operations when developing the same solutions to different native platforms. (Suri et al., 2022). This can lead to increased costs and challenges as updates and bug fixes must be implemented separately for each version of the application.

Cross-platform applications, by contrast, are aiming to tackle these limitations. Different frameworks such as Flutter (Flutter, n.d.) and React Native (React Native, n.d.) allow developers to target multiple operating systems when working from a single codebase, severely reducing development time and costs (Kishore et al., 2022). Cross-platform applications can reach a larger audience as the apps are compatible with various platforms. In spite of these remarkable advantages, performance challenges are often faced. These include slower execution compared to native apps and limitations in operating the platform-specific user experience tools, which could have impact on the overall user experience and satisfaction (Suri et al., 2022).

1.1 Background and motivation

This study aims to investigate how Flutter influences the performance of applications developed with it. Optimization strategies and other best practices that developers can use to further enhance the performance of applications are also explored and discussed,

With the growing demand for low-cost and efficient cross-platform applications many solutions have emerged from companies. Google's answer to this problem is Flutter which released officially in 2018. Its core idea was to lower costs and make cross-platform app development easier and more efficient (Alanazi & Alfayez, 2024). Flutter is a widely used tool and developers from around the world have adopted it building over 400 000 applications that have been distributed to hundreds of millions of devices impacting many people's lives. Notable examples include BMW and Alibaba Group within which Flutter has been utilized to make applications for the needs of the companies (Alanazi & Alfayez, 2024).

Having personally used Flutter in small scale projects, its ease of use, steep learning curve of the Dart programming language and the ability to instantly view changes in the user interface influenced the decision of purely focusing on Flutter. Understanding the performance of specifically Flutter, a relatively new framework, is also vital because more of the big tech companies are beginning to see it as a promising option in application development and the community around it is growing continuously (Novac et al., 2022).

As companies increasingly adopt Flutter for developing cross-platform mobile applications, it becomes essential to evaluate how this framework performs under real-world conditions and across various platforms, such as Android and iOS. Despite its growing popularity, there are still concerns regarding Flutter's ability to deliver the level of performance expected in high-demand, resource-intensive applications (Donglan et al., 2024).

1.2 Research Questions and Method

This thesis utilizes a literature review with the aim to answer one research problem and two main research questions:

- RP: How does Flutter's cross-platform development framework impact applications' performance and can the performance be optimized?
- RQ1: "How does Flutter's cross-platform development framework impact the performance of mobile applications?"
- RQ2: What performance optimization techniques and tools does Flutter provide?

With the help of the research problem and the two research questions the pre-existing studies, literature and Flutter's official documentation was analysed and a thorough search on Flutter and its performance was conducted. Two main platforms used to conduct the analysis were Institute of Electrical and Electronics Engineers database (IEEE explorer) and ACM digital library. The results found were mainly different conference papers regarding the forementioned research questions and a single book which is a comprehensive review of Flutter and its key components.

One key limitation parameter for this research was the year range. With Flutter being released in 2018 there was no use in searching for results prior to this year as there would not be any relevant performance related studies or relevant studies on Flutter at all.

Different search words such as: “Flutter,” “cross-platform development,” “performance”, “framework” was combined and for example with the year range set to 2018-2024 and searching with the term “Flutter and performance” in the IEEE explorer, the number of results was 153. However, some of the results were based on the term “flutter” which means to move back and forth in a rapid manner.

The whole of the references in this research is drawn from studies published from 2020 onward. This is mostly due to the fact that Flutter gained substantial traction in the developer community after its official release in 2018. As the framework evolved and matured, more comprehensive research, case studies, and performance benchmarks began to emerge.

1.3 Research structure

In chapter two an overview of Flutter and cross platform development is presented by examining the key components and tools of Flutter and giving a definition for cross-platform development.

Chapter three discusses the performance of Flutter and seeks for an answer for RQ1 by examining how the applications developed with Flutter perform and are there any problems or advantages in using Flutter. A brief comparison of other popular cross-platform development framework is also provided.

Chapter four focuses on performance optimization in Flutter, addressing RQ2 by exploring tools, techniques, and best practices for improving the performance of Flutter applications. This includes an in-depth analysis of optimization strategies, such as rendering improvements, memory management, and profiling, with real-world examples where possible.

The findings of chapters three and four are discussed in chapter five linking the performance and optimization strategies. Evaluation of Flutter as a cross-platform framework is prepared and an overall answer to the research problem is hence provided.

Chapter six concludes this study by summarizing the key findings of the thesis, revisiting the research questions, and providing insights into Flutter’s impact on application performance and performance optimization techniques. Suggestions for future research directions are also briefly discussed.

Finally the references used in this study are listed as their own dedicated chapter.

2. Overview on Flutter and Cross-Platform Development

This chapter describes briefly what is cross-platform development and how it is conducted in modern day software development. The chapter also explains why Flutter has emerged as one of the most popular frameworks for cross-platform development. A brief overview of Flutter and its key components such as the programming language Dart is provided, and the framework is also compared briefly with other common frameworks for cross-platform development.

2.1 Introduction to Cross-platform Development

The growth of mobile computing has been a transformative development in technology. Adoption of different devices such as smartphones, tablets, wearable devices, and other embedded technologies has expanded the initial scope and complexity of the applications in current market. Mobile applications are becoming increasingly sophisticated in the sense that the handling tasks require higher computational power and advanced user interfaces (UI), graphics and seamless user experience (UX). (Jia et al., 2018).

Simultaneously, the mobile platform ecosystem, essentially led by two platforms (Android and iOS), presents a key challenge for developers. Creating applications for multiple platforms often requires different codebases tailored to the specific need of the target platform. As a result, the demand for solutions that enable applications to run seamlessly across multiple platforms has grown significantly. (De Almeida et al., 2023; Jia et al., 2018)

Cross-platform development addresses this demand. Cross-platform development is the process of developing applications which can run expectedly on multiple different platforms across multiple devices (Alanazi & Alfayez, 2024). As a result businesses using this methodology can target larger audiences by providing sophisticated applications that can seamlessly run on web browsers and different mobile operating systems (De Almeida et al., 2023).

These frameworks use different methodologies to achieve the compatibility of multiple target platforms. Broadly categorized the methodologies are “compilation to native code” and “interpretation or runtime execution.” In compilation to native code the code written in the cross-platform framework is transformed into the target platform’s native code. The code is compiled ahead of time or just in time allowing the generation of native binaries that can be executed on the native platform. Alternatively, frameworks such as Facebook’s React Native and Google’s Flutter adopt an interpretation or runtime execution approach. Code written in a common language (e.g., JavaScript for React Native or Dart for Flutter) is interpreted or executed by the framework’s runtime environment. This environment acts as a bridge, connecting the cross-platform code with platform-specific APIs and UI components. (De Almeida et al., 2023).

Popular frameworks used nowadays among developers include Ionic, React Native Xamarin and Flutter for example (Fatkhulin et al., 2023).

2.2 Overview on Flutter

Flutter is an open-source software development kit that was built and released to public in 2018 by Google. It was created to address the demand for efficient and cost-effective solutions for cross-platform development. The primary goal of Flutter is to simplify the development process by allowing developers to use a single codebase to create applications across multiple platforms, including Android, iOS, web, and desktop. Since its launch Flutter has seen widespread adoption with over 400 000 applications created globally. Noticeable corporations have also applied Flutter in their own development processes and have created successful cross platform applications. Notable examples are Google itself, and Alibaba Group (Alanazi & Alfayez, 2024).

Flutter offers an entire software development kit to developers. With the software development kit developers are able to use Flutter's rendering engine, testing framework and other tools such as a vast set of UI components. (Windmill & Rischpater, 2020).

Flutter framework consist of four main components which work together to enable the efficient cross-platform development. At the core of the framework is the Dart programming platform which powers the framework with the programming language Dart designed for both ahead of time (AOT) and just in time (JIT) compilation. Another essential element of the Flutter framework is the rendering engine, which plays a vital role in the performance of the applications. Built using C++ and integrated with the "Skia graphics library," an open source 2D graphics engine. The engine is responsible for rendering widgets, handling gestures, and managing animations. The "foundation library" adds another layer of abstraction by providing developers with access to platform-specific APIs through a unified interface. This library simplifies tasks such as accessing device hardware or interacting with native features like cameras. Lastly the UI building blocks, widgets, encapsulate everything on the screen of a Flutter app. A key idea in understanding Flutter and the development process is knowing that "everything is a widget." (Cheon & Chavez, 2021; Windmill & Rischpater, 2020).

2.2.1 Dart

Flutter is built on top of a programming language called Dart. Developed by Google and released in 2011 and with the first stable release in 2013 it was originally created for web development and attempted to resolve some problems with JavaScript. Main goals for dart were to simplify development process and ensure high performance across different platforms. (Bailey & Biessek, 2021).

Dart's initial design offers several features. Key aspects include its productive tooling, which includes code analysis tools, integrated development environment (IDE) plugins, and access to a package ecosystem. Another notable feature of Dart is its garbage collection system, which manages memory deallocation. By automatically freeing up memory occupied by objects that are no longer in use, Dart reduces the chances of memory leaks and improves application stability. Optional type annotations add another layer of security and consistency for developers who want greater control over data in their applications. While Dart is a statically typed language, it allows type inference to analyse and ensure type safety at runtime, reducing the likelihood of bugs during code compilation.

Portability is another design feature of Dart. While it is known for its ability to be compiled into JavaScript for web development, it can also be natively compiled to ARM and x86 code, ensuring usability across various platforms and devices. (Bailey & Biessek, 2021).

Other key features of Dart include “just in time compilation (JIT)” and “ahead of time (AOT) compilation.” JIT-type of compilation is generally used during the development of an application. It allows developers to “hot-reload” an application instantly viewing the changes made without restarting or building the application from the beginning. In contrast, AOT compilation is utilized when deploying applications, converting Dart code into native machine code. This ensures that applications built with Flutter run with the high speed and efficiency expected from native apps. (Bailey & Biessek, 2021; Sattar et al., 2023).

2.2.2 Prebuilt widgets and UI components

Flutter's core idea when it comes to composing applications and the user interface in particular is using widgets to construct the layout. Widgets define both the structural and visual elements of an application. Every widget in Flutter is organized hierarchically in a widget tree, which is a central concept in Flutter's architecture. This tree-like structure allows each widget to inherit properties from its parent and influence its child elements. (Nanavati et al., 2024).

Widgets are different objects and classes created with the Dart programming language (Bailey & Biessek, 2021). Flutter offers an extensive set of prebuilt widgets that are ready for use to developers and are customizable for adapting to the needs of developers. These widgets include for example: “Button,” “Row” and “TextField.” If the prebuilt widgets do not fit in the context of the app, developers are able to create own custom widgets utilizing Dart (Alanazi & Alfayez, 2024).

Widgets are broadly categorized into “Stateful” and “Stateless” widgets (Sharma et al., 2022). This classification is central to how developers approach creating Flutter applications.

Stateless widgets represent immutable components of the UI. These widgets do not change their appearance or behaviour in response to user actions or application state changes. Since they are immutable, stateless widgets are used for parts of the interface that do not require frequent updates (see Figure 1 for details). (Bailey & Biessek, 2021; Sharma et al., 2022).

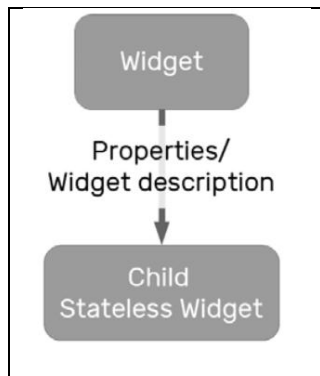


Figure 1. Representation of a Stateless Widget (Bailey & Biessek 2021).

In contrast Stateful widgets are dynamic and can rebuild themselves based on changes in the applications state. The changes can be for example interaction from the user, changes in the data of the application or device orientation change. Changes in this state trigger the rebuilding of the widget, allowing for dynamic interaction and updates (see Figure 2 for details). For instance, a form field that updates its appearance as the user types is a stateful widget. However, the way Flutter re-renders widgets can have performance implications. State management becomes critical in applications with intricate interactive components. Without a structured architecture, managing the states across the widget tree can become cumbersome and unmaintainable. (Boukhary & Colmenares, 2019).

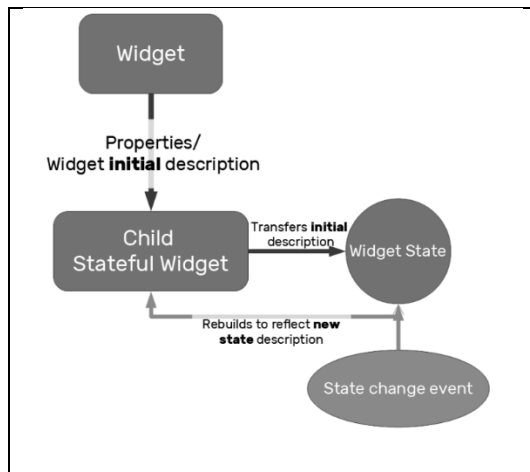


Figure 2. Representation of a Stateful Widget (Bailey & Biessek 2021).

2.2.3 Development tools

When considering application development with Flutter, developers need to ensure they have the right set of tools for the development process. The essentials include an Integrated Development Environment (IDE) and the Flutter Software Development Kit (SDK), both of which play a pivotal role in app creation (Souha et al., 2024). Flutter provides support for popular IDEs like Android Studio, IntelliJ IDEA, and Visual Studio Code (Alanazi & Alfayez, 2024).

The Flutter SDK comes equipped with developer tools designed to enhance productivity and simplify debugging. For instance, these tools allow developers to analyse application size, inspect UI layouts and states, and diagnose possible performance issues in the applications. Such functionalities ensure that developers can fine-tune their applications for optimal performance. Additionally, Flutter's developer tools integrate seamlessly with supported IDEs, providing a seamless and efficient workflow. (Flutter, n.d.).

2.2.4 Community

Flutter has established itself as a popular technology among developers worldwide. As described by Alanazi and Alfayez (2024), Flutter was ranked among the top ten most popular technologies in the 2021 annual developer survey conducted by Stack Overflow, which gathered responses from nearly 80,000 software developers.

A fundamental factor contributing to Flutter's popularity is its active community. This community provides developers with access to multiple resources, including comprehensive documentation, different tutorials, and forums. Platforms such as Stack Overflow, Reddit, and Discord host discussions where developers can seek advice when debugging their own projects or about using different tools (Souha et al., 2024). Flutter's official Twitter account, which has garnered approximately 240,000 followers, serves updates, tips, and interaction within the developer community. (Flutter, n.d.).

In addition to its online presence, the Flutter community organizes various events across the world. These events, which range from conferences to meetups and hackathons, offer opportunities for developers to network, exchange ideas, and collaborate on projects. (Flutter, n.d.).

Flutter's ecosystem also benefits from open-source contributions. Developers frequently create and share plugins, widgets, and packages, which can be integrated into projects, enhancing functionality and speeding up development. The platform's collaborative culture ensures that even niche problems often have solutions available within the community. (Flutter, n.d)

3. Flutter's performance

Performance is an essential consideration for cross-platform application development because it directly influences user experience and the retention rates of mobile applications especially. According to Mahendra & Anggorojati, (2020) mobile applications have a historical retention rate from ranges between 31% and 39%, emphasizing the fact that approximately one in three users may discontinue using an app after a single interaction. This statistic highlights the significance of delivering a seamless user experience and with that seeking to create a high performing app.

Measuring the performance of cross-platform applications and mobile applications especially has multiple different parameters to consider. These include “response time,” “memory usage” and “battery usage” (Wilcox et al., 2016). Among these, response time is influential regarding the user experience. As Wilcox et al. (2016) describe, response times encompass various metrics, such as “application launch time”, “page load time”, and the time required to navigate back to a previous page in the application.

This part of the thesis focuses on performance of Flutter and briefly compares the framework with other popular cross-platform frameworks to identify possible advantages or disadvantages.

3.1 Performance Features of Flutter

Flutter's architecture and the compilation process are designed to achieve high performance. The framework employs a layered architecture where each layer optimizes a specific part of the application to perform well. (Nanavati et al., 2024). Central to this architecture is the frameworks direct reliance on direct machine code compilation. What this means is that Flutter does not need any intermediary layers to utilize the target platform. Flutter applications can execute natively on different platforms such as iOS and Android without having to rely on intermediate layers like Java bytecode or JavaScript bridges. This gives Flutter a distinct performance advantage over frameworks like Jetpack Compose (compiled to JVM bytecode) and React Native (which uses JavaScript threads) (Kusuma et al., 2023; Rambhia et al., 2023)

The rendering process in Flutter is streamlined through its rendering engine, which leverages the Skia graphics library for direct rendering on the screen. This eliminates reliance on platform-specific UI elements, ensuring stable visuals and responsive interactions on multiple devices and platforms. The engine is built using C++ which further enhances its capability to manage complex animations and high-resolution interfaces with minimal overhead (Nanavati et al., 2024; Oliveira et al., 2023).

Dart, the programming language used to build Flutter applications, also influences Flutter's performance. Its features such as ahead-of-time compilation, generational garbage collection and asynchronous programming allows applications to execute effectively (Bailey & Biessek, 2021).

3.2 A brief comparison with another Framework

When comparing Flutter to other common cross-platform frameworks such as React Native different parameters involving performance and usability, emerge as factors on which most developers can consider when choosing the right framework.

Flutter’s use of Dart and its Ahead-of-Time (AOT) compilation process allows cross-platform applications to compile directly into native machine code for the target device, contributing to smoother runtime performance, reduced latency, and faster start-up times (Kusuma et al., 2023). Visual depiction of Flutter communication with the platform can be seen in Figure 3.

React Native, in contrast, leverages a “JavaScript-Bridge” to communicate with the operating system’s native components (see Figure 4 for visual representation). This could introduce delays during major operations such as rendering and hot reloading (Kishore et al., 2022).

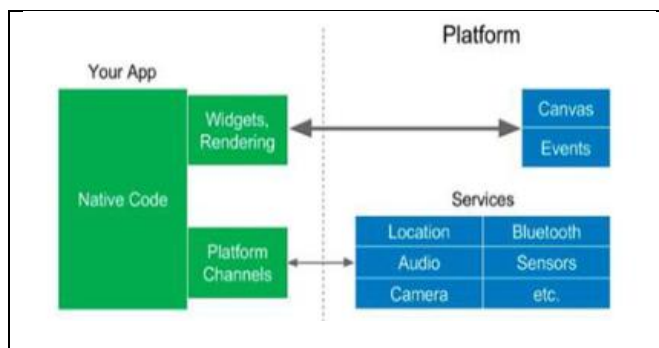


Figure 3. How Flutter interacts with the target platform (Ameen & Mohammed, 2022)

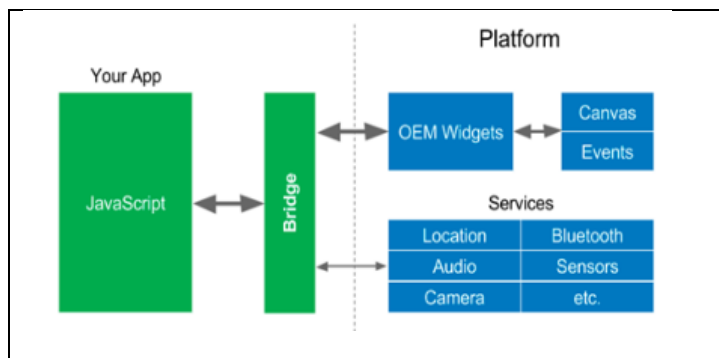


Figure 4. How React Native interacts with the target platform (Ameen & Mohammed, 2022)

Performance benchmarks show Flutter’s efficiency in terms of storage and battery usage. Testing reveals that React Native applications consume more storage and drain battery life more quickly across different devices (Wilcox et al., 2016). Additionally, React Native’s reliance on the JavaScript bridge increases CPU usage, making it less suitable for resource-constrained environments (Mahendra & Anggorojati, 2020).

In experiments comparing full-stack food delivery applications built with Flutter and React Native, Flutter consistently demonstrated better performance in start-up time, smoother interactions, and lower resource consumption (Kusuma et al., 2023). Findings by Oliveira et al., (2023) further demonstrate that Flutter outperforms React Native in execution time and also energy efficiency. However, it is noted by Oliveira et al., (2023) that Flutter’s memory usage is higher, which is a trade-off developers should consider when examining the application requirements.

4. Performance Optimization for Flutter

This part of the thesis focuses on identifying, understanding, and optimizing different performance drawback of Flutter applications. Even though the architecture of Flutter is designed for high performance some drawbacks in performance have been identified. By identifying and addressing known issues in performance of Flutter, developers can deliver efficient and scalable applications.

4.1 Identifying performance bottlenecks

When developing cross-platform applications with Flutter various potential bottlenecks can limit the applications' performance if they are not addressed properly. Common issues with performance in Flutter applications are often associated with rendering. Delays in rendering especially are a common issue, especially when using “vectored” images or animations. Delays effecting these components can result in lags in the user interface, disrupting the user experience that Flutter aims to provide. Such issues particularly occur in applications with challenging and dynamic visual content (Novac et al., 2022).

Another identified bottleneck concerns the management of “network images.” Flutter caches network images only in memory during their loading and display process. If the application is then terminated and reopened, the same images must be downloaded again, leading to unnecessary data consumption. This problem is worse in environment where there are slow network speeds or when the image file itself is large. Flutter provides cache management tools, but nevertheless, improper usage can result in downgraded performance (Donglan et al., 2024).

Furthermore, application size poses another bottleneck. Flutter applications are noted be larger, leading to compromises in the quality of images, animations, and libraries to maintain manageable application sizes. Memory overhead associated with Flutter applications requires careful consideration to prevent excessive consumption of resources. For example, experiments have shown that while Flutter is usually more efficient regarding it can impose higher demands in memory that can have major impacts on devices with limited resources (Oliveira et al., 2023).

Drawbacks can also arise from factors related to the typical application architecture and the implementation of the widgets and the “build”-method. Widgets play a crucial role in determining the app's responsiveness. The effectiveness of the “build”- method, which renders widgets to the screen, is equally vital. Poorly programmed build-methods can lead to performance issues such as user interface lag and unresponsive application states. Bad practices when programming the build method include handling large datasets or other computationally heavy operations directly inside the method. (Nanavati et al., 2024).

As an example, Nanavati et al., (2024) came across a performance issue when filtering large lists directly in the build method. Addressing this involved leveraging isolates—Flutter's mechanism for parallel code execution outside the main thread. This approach reduced the computational load on the primary UI thread, resulting in smoother interactions and better performance.

4.2 Key optimization techniques

Optimizing Flutter applications involves careful management of different components and resources of an application. While Flutter's architecture and tools provide a solid foundation for high performance, targeted optimizations can significantly enhance application efficiency.

One critical area of focus is widget- and state management and efficient UI rendering. For instance, replacing a “stateless widgets” with “stateful widgets,” when applicable, allows selective rebuilding of an UI, triggered by only specific state changes. This technique minimizes unnecessary rebuilding of widgets, which can lead to downgrades in applications’ performance. (Nanavati et al., 2024).

Flutter’s “build”-method is another essential component, where developers can optimize the performance. The method is responsible for rendering widgets on the screen and can easily become a bottleneck if not optimized. Optimization of the build method can be done by offloading computationally heavy tasks, such as filtering a large dataset, to a separate "isolate," enabling parallel execution of code outside the “main thread” resulting in a smoother user experience and better performance. (Nanavati et al., 2024).

“Caching” and “pre-loading” are other optimization techniques, particularly for applications dealing with copious amounts of data or media and which are relying on some third-party application programming interfaces (API’s). These techniques reduce the need for repeated network requests and speed up data retrieval, lowering energy footprints, especially for Progressive Web Apps (PWAs) or web-based applications with Flutter.(Huber et al., 2022).

Moreover, expensive operations, such as those involving complex visual effects or animations, should be optimized. Developers can explore alternative approaches or limit the frequency of these operations to ensure smoother rendering and reduced “junk.” Leveraging techniques such as “tree-shaking,” which removes unused code during compilation, also helps streamline app performance and reduce resource consumption (Flutter, n.d.).

How about ensuring the actual performance and optimizations?

Testing on actual target devices such as Android and iOS mobile phones and tablets can give valuable insights on the performance with real-world scenarios. Often Flutter applications can be tested with an emulator running on a computer, but often they may lack to represent actual behaviour of the application (Sattar et al., 2023). Testing on multiple devices across different hardware configurations and operating systems versions, developers can get most out of the actual performance and gain insight into optimizing the performance.

Profiling tools such as “Flutter DevTools” (Flutter, n.d.) further provide invaluable insights into bottlenecks by enabling developers to monitor app performance in real time. This toolset allows for detailed examination of CPU usage, memory allocation, and frame rendering times, making it easier to identify areas for refinement or inspect, if possible, optimizations have worked. For instance, developers can use these tools to detect and address excessive widget rebuilding or redundant operations during UI rendering (Sattar et al., 2023).

5. Discussion

In this part of the thesis the initial findings of the literature review are discussed, and the research questions are also answered.

5.1 Tools and performance

The tools and components on which Flutter is built is substantial. The most valuable attribute which Flutter provides is the use of a single codebase across different platforms and devices. This removes the need for redundant coding, testing, and deploying. Essentially developers get a full software development kit to experiment and build apps with.

Building applications with Flutter is relatively easy and streamlined. The wide availability of prebuilt widgets for both iOS and Android enhance productivity by allowing developers to quickly build user interfaces and user interface components without having to create interfaces from scratch.

Community support also contributes positively to Flutter's position in the software development markets. Flutter has a strong and growing community which members actively contribute to tutorials, plugins, and different packages. The open-source ecosystem enables developers to find solutions to usual challenges quickly and also asking for help if a solution has not previously been identified (Flutter, n.d.).

RQ1: "How does Flutter's cross-platform development framework impact the performance of mobile applications?"

The findings of this study indicate that due to Flutter's architectural design, particularly the use of the direct, high performing rendering engine contributes positively to application performance. Unlike other popular cross-platform frameworks such as React Native, which specifically relies on a JavaScript bridge to communicate between the app and the target device's native components Flutter eliminates this constraining layer and establishes a direct connection between the application and the device (Nanavati et al., 2024). This way the potential bottlenecks in UI rendering are reduced and the interface provides a smoother and faster user experience in response time and UI rendering which are crucial metrics when considering mobile application performance.

Another factor which contributes to Flutter high performance is the programming language, Dart. Results from this study highlight that Dart integration with Flutter is aligned with keeping simplicity and high performance as standards. By leveraging both ahead of time (AOT) and just in time (JIT) compilation, Dart ensures a balance between development efficiency and runtime performance. While JIT can make development processes more efficient with features such as "hot reload" AOT ensures that applications run at native speeds, eliminating runtime interpretation overhead (Bailey & Biessek, 2021).

Furthermore, other features of Dart such as asynchronous programming and the garbage collection system also align with Flutter's goals of responsive and stable applications. These features allow application to manage real-time data interactions and complex UI updates seamlessly.

Moreover, Dart's portability confirms that Flutter applications achieve same functionalities and attributes across various platforms, contributing to the goals of Flutter in cross-platform development.

5.2 Performance optimizations

During the literature review it was found that Flutter's architecture is overall designed and implemented with high performance in mind. However if developers want to achieve the most optimal results, thoughtful attention is needed to detect possible performance bottleneck and following best practices.

Flutter's frameworks flexibility can be both an advantage and a challenge. Without careful management, even minor inefficiencies especially in widget rebuilding, data processing, or UI rendering can impact performance in a negative manner.

RQ2: "What performance optimization techniques and tools does Flutter provide?"

Several optimization techniques were found during the study. Among them using Dart's asynchronous programming with keeping best practices in mind. These include caching mechanisms for network images, which can avoid repeated data requests, reducing latency and data consumption. However, as noted by Huber et al., (2022), relying strongly on caching can lead to storage issues. Thus, developers need to carefully assess balance between resources and performance.

Other performance optimization techniques include replacing "StatelessWidgets" with "StatefulWidgets" selectively. This ensures that only the necessary parts of an UI are updated when the application's state changes which can occur during user interactions or data processing. This lowers the amount of work on Flutter's rendering engine and improves responsiveness. Additional way of increasing applications performance with Flutter can be done with efficient use of the "build" method. Leaving heavy calculations and operations outside of the method can result in a better performance. (Nanavati et al., 2024).

Sattar et al., (2023) mentioned real-world testing on actual devices. Emulators can be unsuccessful to capture the performance behaviours seen on actual devices. This way of testing can ensure applications actual performance and responsiveness. Testing with different hardware and operating systems was also emphasized to get actual results.

During the actual development, using performance profiling tools like Flutter's "DevTools", can expose expensive operations such as excessive calls to "saveLayer" and enable developers to optimize them before further development and deployment of the application (Flutter, n.d.).

It is worth noting that techniques such as caching, profiling, and device-specific testing are not exclusive to Flutter but are standard industry practices across mobile development. However, Flutter's built-in tooling can simplify these processes, offering an integrated approach to identifying and mitigating performance bottlenecks.

6. Conclusion

With this chapter the thesis is concluded. The key findings are summarized and guidelines for practical use and future research are also reflected upon.

6.1 Summarizing the key results

The goal of this literature review was to find out the impact of Flutter when it comes to application performance and can optimizations be fitted for enhanced performance. The findings demonstrate that Flutter offers a strong solution for developers seeking to create high-performing applications across multiple different platforms effectively. Like other frameworks Flutter still has limitations that need to be accounted for when developers are seeking a fitting solution.

The most important takeaways from this study include several insights into how Flutter enhances the performance of the mobile applications developed with it, particularly in the terms of improving the UI performance. One of the key standout features of Flutter is the direct rendering engine, which significantly contributes to the excellent performance achieved by Flutter apps in general. Other components of the framework such as the Dart programming language further enhance performance and give developers a variety of tools to work with.

Frameworks such as React native, which rely on an intermediary component to bridge the communication between the native code of the target device which can lead to compensations during runtime of the app resulting in ticks in the performance. Flutter eliminates this problem with the rendering engine which communicates directly with the native code ensuring a smooth user experience and great performance.

For optimizing the performance of the applications, the study underscored the importance of techniques like efficient widget rebuilding, caching, and using Flutter's performance profiling tools. Such strategies can mitigate bottlenecks and improve runtime efficiency, especially for data-intensive and visually dynamic applications. Furthermore, Flutter's strengths in cross-platform performance, combined with its comprehensive toolkit and growing ecosystem, establish it as a competitive solution for modern mobile app development.

6.2 Implications for practical use

The findings of this study offer insights for developers and organizations considering taking Flutter as part of their software development projects. From a developer's viewpoint, Flutter's integrated development toolkit and prebuilt widgets can streamline the process of creating functional, visually appealing apps. By reducing redundancy in code development, testing, and deployment, Flutter allows for quicker turnaround times and cost-effective solutions.

The study also emphasizes areas where developers must exercise caution and get familiar with the best practices of Flutter. For example, the effective use of the "build"-method has effect on the performance of the application. Practical optimization strategies—such as caching and isolating heavy operations—are essential for ensuring stability and seamless user experience in the application. Furthermore, testing on actual hardware and software environments is crucial for identifying and addressing platform-specific- and possible performance issues.

Flutter's suitability for performance-critical applications makes it a strong choice for industries like e-commerce and entertainment, where responsiveness and user experience are often fundamental. However, developers must also navigate challenges like limited support for niche platforms (for example watchOS, Android Auto) and the potential for larger app sizes.

6.3 Limitations of this study

One key limitation to this research is that it primarily focuses on Flutter's performance and optimization on a general level. This approach allows for a general understanding but leaves room for more detailed explorations into specific technical metrics of application performance such as memory consumption, battery efficiency and data processing speed. Future studies could also significantly benefit from comparative analysis of these metrics with other cross-platform development frameworks such as React Native or Xamarin. This would help in contextualizing Flutter's strengths and weaknesses.

Another major limitation is the actual scope of availability of the current research based on Flutter. Flutter is relatively new to mainstream adoption among the major tech corporations and the development community. Because of this the foundation of longitudinal studies is not present and the lack of case studies in these big tech companies leaves an opening for in-depth exploration as more organizations begin adopting Flutter for more diverse projects.

Flutter is also constantly evolving and new features and versions are published frequently. Some issues identified in this study and the pre-existing literature can become obsolete when newer and better versions are put out.

Lastly generalizability could arise as an issue in literature and conducted studies on Flutter. Many of the studies could be focused on for example, specific types of applications such as business apps or prototypes and hence, may not reflect the true nature of Flutter's performance on different platforms.

6.4 Recommendations for future research

While this study provided a general analysis of the state of Flutter, its performance and optimization techniques, further research is yet needed. To explore its full potential, further studies are needed to especially determine the effectiveness in large-scale applications. As the complexity and size of an application grows, so do the demands on the framework, performance optimization, and maintainability. Researching how Flutter handles more complex projects, as well as its adaptability when working with large development teams or integrating with other enterprise-level systems, will be crucial in determining its suitability for large-scale applications.

Another promising area for investigation is Flutter's adoption in emerging technologies and platforms. For instance, evaluating Flutter's performance in fields like machine learning, augmented reality, mobile gaming and the Internet of Things could provide insights into its scalability and flexibility.

Additionally, the development and innovation within the Flutter community will likely discover additional features that could further boost its performance and usability. As the community surrounding Flutter continues to grow, more tools and best practices will emerge, offering opportunities for developers to maximize its capabilities. In light of these considerations, while Flutter has already demonstrated its impact for cross platform development, future research and exploration will benefit understanding its full capabilities as a cross-platform development framework.

References

- Alanazi, A., & Alfayez, R. (2024). What is discussed about Flutter on Stack Overflow (SO) question-and-answer (Q&A) website: An empirical study. *Journal of Systems and Software*, 215. <https://doi.org/10.1016/j.jss.2024.112089>
- Ameen, S. Y., & Mohammed, D. Y. (2022). Developing Cross-Platform Library Using Flutter. *European Journal of Engineering and Technology Research*, 7(2), 18–21. <https://doi.org/10.24018/ejeng.2022.7.2.2740>
- Bailey, T., & Biessek, A. (2021). *Flutter for beginners : an introductory guide to building cross-platform mobile applications with flutter 2. 5 and dart (Second edition)*. Packt Publishing, Limited.
- Boukhary, S., & Colmenares, E. (2019). A clean approach to flutter development through the flutter clean architecture package. *Proceedings - 6th Annual Conference on Computational Science and Computational Intelligence, CSCI 2019*, 1115–1120. <https://doi.org/10.1109/CSCI49370.2019.00211>
- Cheon, Y., & Chavez, C. (2021). Converting Android Native Apps to Flutter Cross-Platform Apps. *Proceedings - 2021 International Conference on Computational Science and Computational Intelligence, CSCI 2021*, 1898–1904. <https://doi.org/10.1109/CSCI54926.2021.00355>
- De Almeida, J. C., Brito E Abreu, F., & De Almeida, D. S. (2023). Cross-Platform Mobile App Development: The IscteSpots experience. *Proceedings - 2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops, ASEW 2023*, 11–16. <https://doi.org/10.1109/ASEW60602.2023.00006>
- Donglan, Z., Bin Darus, M. Y., & Ramli, A. B. (2024). Investigating Developer Experiences with UI Components in Flutter: Challenges and Implications. *2024 International Visualization, Informatics and Technology Conference (IVIT)*, 128–133. <https://doi.org/10.1109/IVIT62102.2024.10692626>
- Fatkhulin, T., Alshawi, R., Kulikova, A., Mokin, A., & Timofeyeva, A. (2023). Analysis of Software Tools Allowing the Development of Cross-Platform Applications for Mobile Devices. *2023 Systems of Signals Generating and Processing in the Field of on Board Communications, SOSG 2023 - Conference Proceedings*. <https://doi.org/10.1109/IEEECONF56737.2023.10092148>
- Flutter. (n.d.). Flutter Documentation. N.d. Retrieved 4.12.2024 from <https://doi.org/https://flutter.dev>
- Huber, S., Demetz, L., & Felderer, M. (2022). A comparative study on the energy consumption of Progressive Web Apps. *Information Systems*, 108. <https://doi.org/10.1016/j.is.2022.102017>
- Jia, X., Ebone, A., & Tan, Y. (2018). A performance evaluation of cross-platform mobile application development approaches. *Proceedings - International Conference on Software Engineering*, 92–93. <https://doi.org/10.1145/3197231.3197252>

- Kishore, K., Khare, S., Uniyal, V., & Verma, S. (2022). Performance and stability Comparison of React and Flutter: Cross-platform Application Development. *International Conference on Cyber Resilience, ICCR 2022*. <https://doi.org/10.1109/ICCR56254.2022.9996039>
- Kusuma, M., Rifani, A. H., & Sugiantoro, B. (2023). Comparison analysis of Jetpack Compose and Flutter in Android-based application development using Technical Domain. *2023 8th International Conference on Informatics and Computing, ICIC 2023*. <https://doi.org/10.1109/ICIC60109.2023.10381987>
- Mahendra, M., & Anggorojati, B. (2020). Evaluating the performance of Android based Cross-Platform App Development Frameworks. *ACM International Conference Proceeding Series*, 32–37. <https://doi.org/10.1145/3442555.3442561>
- Nanavati, J., Patel, S., Patel, U., & Patel, A. (2024). Critical Review and Fine-Tuning Performance of Flutter Applications. *Proceedings - 2024 5th International Conference on Mobile Computing and Sustainable Informatics, ICMCSI 2024*, 838–841. <https://doi.org/10.1109/ICMCSI61536.2024.00131>
- Novac, O. C., Novac, C. M., Ciora, B., Gordan, C. E., Gordan, M. I., & Bujdoso, G. (2022). The rise of mobile development: a comparison between Ionic and Flutter. *2022 14th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2022*. <https://doi.org/10.1109/ECAI54874.2022.9847460>
- Oliveira, W., Moraes, B., Castor, F., & Fernandes, J. P. (2023). Analyzing the Resource Usage Overhead of Mobile App Development Frameworks. *ACM International Conference Proceeding Series*, 152–161. <https://doi.org/10.1145/3593434.3593487>
- Rambhia, P., Shinde, P., & Bamane, K. (2023). Securing Flutter Applications: A Comprehensive Study. *2023 7th International Conference On Computing, Communication, Control And Automation, ICCUBEA 2023*. <https://doi.org/10.1109/ICCUBEA58933.2023.10392001>
- React Native. (n.d.). React Native Documentation. Retrieved 4.12.2024 from <https://doi.org/https://reactnative.dev>
- Sattar, A. Md., Soni, P., Ranjan, M. K., Kumar, A., Sahu, C., Saxena, S., & Chaudhari, P. (2023). Accelerating Cross-platform Development with Flutter Framework. *JOURNAL OF OPEN SOURCE DEVELOPMENTS*. <https://doi.org/10.37591/joosd.v10i2.580>
- Sharma, S., Khare, S., Unival, V., & Verma, S. (2022). Hybrid Development in Flutter and its Widgits. *International Conference on Cyber Resilience, ICCR 2022*. <https://doi.org/10.1109/ICCR56254.2022.9995973>
- Souha, A., Benaddi, L., Ouaddi, C., & Jakimi, A. (2024). Comparative analysis of mobile application Frameworks: A developer's guide for choosing the right tool. *Procedia Computer Science*, 236, 597–604. <https://doi.org/10.1016/j.procs.2024.05.071>
- Suri, B., Taneja, S., Bhanot, I., Sharma, H., & Raj, A. (2022, December 23). Cross-Platform Empirical Analysis of Mobile Application Development frameworks: Kotlin, React Native and Flutter. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/3590837.3590897>

Willox, M., Vossaert, J., & Naessens, V. (2016). Comparing performance parameters of mobile app development strategies. Proceedings - International Conference on Mobile Software Engineering and Systems, MOBILESoft 2016, 38–47. <https://doi.org/10.1145/2897073.2897092>

Windmill, E., & Rischpater, R. (2020). Flutter in action (1st edition). Manning Publications.