Mobile Application for Learning Japanese Alphabet

Ivan Goriakin

Bachelor's thesis
2024

Tomas Bata University in Zlín
Faculty of Applied Informatics

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Ivan Goriakin**
Osobní číslo:	**A21236**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Mobilní aplikace pro učení japonské abecedy**
Téma práce anglicky:	**Mobile Application for Learning Japanese Alphabet**

Zásady pro vypracování

1) Popište současný stav technologií pro vývoj multiplatformních aplikací.
2) Zaměřte se především na framework Flutter.
3) Zpracujte přehled problematiky japonské abecedy.
4) Navrhněte aplikaci a zvolte vhodný framework.
5) Na základě návrhu aplikaci vytvořte a popište její klíčové části.
6) Demonstrujte výsledky a formulujte závěr.

Forma zpracování bakalářské práce: **tištěná/elektronická**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

1. Flutter documentation. Google Docs [online]. Oficiální dokumentace od Google[cit. 2023-11-09]. Dostupné z: https://docs.flutter.dev/
2. Dokumentace k Dart. Dart Dev [online]. Oficiální dokumentace od Google [cit. 2023-11-09]. Dostupné z: https://dart.dev
3. NHK WORLD-JAPAN. Easy Japanese. NHK WORLD-JAPAN [online]. Bezplatné lekce japonštiny od NHK [cit. 2023-11--09]. Dostupné z: https://www.nhk.or.jp/lesson/english
4. BANNO, Eri et al. Genki: An Integrated Course in Elementary Japanese. Tokyo: The Japan Times, 2011. ISBN 978--4789014403.
5. MIOLA, Alberto. Flutter Complete Reference 2.0: The ultimate reference for Dart and Flutter. [vyd. 1.]. [Místo vydání není známo]: Alberto Miola, 2023-05-20. ISBN 979-8394957390.

Vedoucí bakalářské práce: **doc. Ing. Petr Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

I hereby declare that:

- I understand that by submitting my Bachelor´s Thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Bachelor´s Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Bachelor´s Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlín, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Bachelor´s Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlín has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Bachelor´s Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlín with a view to the fact that Tomas Bata University in Zlín must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Bachelor´s Thesis include the use of software provided by Tomas Bata University in Zlín or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Bachelor´s Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Bachelor´s Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

I herewith declare that:

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.

In Zlín; 10.05.2024: .....Ivan Goriakin v.r.....

Student´s Signature

ABSTRAKT

Tato diplomová práce zkoumá vývoj mobilní aplikace zaměřené na výuku japonské abecedy, se zaměřením na využití frameworku Flutter pro multiplatformní funkčnost. Studie primárně prozkoumává pokroky v technologiích multiplatformního vývoje, s důrazem na možnosti Flutteru. Poskytuje podrobný pohled na složitosti japonského písemného systému a detaily procesu designu a implementace aplikace. Project zahrnují návrh uživatelského rozhraní a tvorbu interaktivních modulů usnadňujících výuku abecedy. Výběr architektury aplikace a technologií je řízen jejich potenciálem zlepšit zkušenost s učením jazyka.

Klíčová slova: Flutter framework, Dart, Multiplatformní vývoj, Android, IOS, Japonská abeceda

ABSTRACT

This diploma thesis explores the development of a mobile application aimed at learning the Japanese alphabet, with a focus on utilizing the Flutter framework for cross-platform functionality. The study primarily investigates the advancements in multiplatform development technologies, emphasizing the capabilities of Flutter. It provides an in-depth look at the intricacies of the Japanese writing system and details the design and implementation process of the application. Project include user interface design and the creation of interactive modules to facilitate alphabet learning. The selection of the application's architecture and technologies is driven by their potential to enhance the language learning experience.

Keywords: Flutter framework, Dart, Multiplatform development, Android, IOS, Japanese alphabet:

ACKNOWLEDGEMENTS

Acknowledgements, motto and a declaration of honour saying that the print version of the Bachelor's/Master's thesis and the electronic version of the thesis deposited in the IS/STAG system are identical, worded as follows:

I hereby declare that the print version of my Bachelor's/Master's thesis and the electronic version of my thesis deposited in the IS/STAG system are identical.

CONTENTS

INTRODUCTION

In the digital age, the demand for educational applications that are accessible, efficient, and capable of supporting continuous learning on multiple devices is increasingly prominent. This reality necessitates the selection of robust and versatile development frameworks that can cater to diverse user needs across various platforms. This thesis explores the theoretical and practical elements of multiplatform application development, with a particular focus on the challenges and solutions associated with creating language learning tools.

Japanese, a language rich with three distinct alphabets—Hiragana, Katakana, and Kanji—presents unique challenges to learners, especially those accustomed to Roman alphabetic systems. Addressing this, the thesis introduces a mobile application designed to facilitate the learning of the Japanese alphabet.

Central to this thesis is the evaluation and utilization of Flutter, a modern framework developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. The thesis examines Flutter's architecture, its benefits over other cross-platform technologies, and its specific advantages for educational applications. By leveraging Flutter, the application aims to provide a seamless and productive learning experience across multiple platforms without compromising on performance or user experience.

In the practical section, the thesis will detail the design and development process of a "Japanese Learning App," which utilizes multimedia resources to enhance the learning process. The application's development will serve as a case study for applying theoretical principles in real-world software engineering.

The primary goal of this thesis is to deliver insightful analysis into the application of multiplatform development frameworks particularly through the lens of Flutter. It aims to demonstrate how such technologies can be effectively utilized to meet the educational needs of users while accommodating the broad compatibility and performance demands of modern software development.

I. THEORY

1 MOBILE DEVELOPMENT ECOSYSTEM

Mobile development ecosystem that includes a wide range of technologies and frameworks designed to develop software for smartphones and tablets running on wireless computing devices. But this ecosystem is a constantly evolving and dynamic one, driven by improvements in both hardware and operating system features and user preferences. But in order to create interesting and user-friendly applications, developers need to navigate this complex landscape.

1.1 Overview of mobile operating systems

Mobile operating systems (OS) are the foundation of the mobile development ecosystem, providing the needed platform for applications to run. These operating systems manage the device's hardware resources and provide the necessary services for mobile applications. Some of the most well-known mobile operating systems are Android, iOS and, most notably historically, Windows Mobile, each of which provides unique development environments and user experiences. [1]

1.1.1 Android

One of the most widely used operating systems for mobile devices is Android, developed by Google. As of May 2022, 71.4% of the smartphone market, is occupied by Android. Android is open source, making it a cheaper solution for many developers. Also due to the popularity of the system, it has a large user base and a wide range of compatible devices. The system is very flexible and allows for more precise customization. The Android OS itself is implemented on Linux, by a developer named Andy Rubin. Because Android is implemented on Linux, it allows Android to be easily integrated into many smart devices such as smart watches, TVs and even refrigerators. [1, 2]

1.1.2 IOS

The creation of one of the leading mobile operating systems, IOS, began in the garage, with no one at that time unknown to Steve Jobs. He had an idea to make a touch screen for a computer. As the system developed, as well as with the appearance of the first prototype, it was decided to use the touch screen for cell phones, so in 2007, along with the first iPhone appeared the first working version of IOS written specifically for Apple mobile devices.

However, at first it was called iPhone OS and only with the release of iPhone 4 was renamed to the familiar name IOS.
The basis for IOS was the operating system for computers, MacOS at the time it was still called "OS X", which was installed on company computers and laptops. In fact, IOS received a new interface and trimmed functionality from MacOS, at the same time, the core they had almost identical. [1, 3]

1.1.3 Windows Mobile

Windows OS was released by Microsoft in 2010. The first phone to receive Windows OS was the HTC 7 Mozart. The system was not badly made and was quite safe (compared to the Android system). But did not take root among users, third-party developers did not hurry to develop under the new system, which led to problems for users of Windows OS, often users simply could not find the application they need in the application store. At the same time, Microsoft itself provided free access to some of its applications, which of course was no compensation for the inability to use the usual applications. Now a bit of history, the project started under the name "Photon", in 2004, but was extremely slow and as a result it was closed. Later, in 2008, Microsoft decided to reassemble the development team and start working on a new Windows Mobile operating system. The first public announcement of the new mobile operating system was on February 15, 2010, at Mobile World Congress 2010, when Steven Ballmer first announced the new Windows Phone 7 mobile system. The first official version was released on October 10, 2010. [4]

1.2 History of mobile operating systems

The evolution of mobile operating systems (OS) has been a fascinating journey that shows how quickly technology is changing. At first, basic OSs ran on simple cell phones with limited features. But in the late 1990s and early 2000s, more sophisticated systems like Palm OS and BlackBerry OS emerged, which introduced features like email and basic web surfing. This led to the creation of smartphones.[5]

The release of Apple's iOS in 2007 and Google's Android in 2008 marked a turning point in the mobile OS landscape. These platforms revolutionised smartphones, turning them into versatile devices capable of performing a variety of tasks such as navigation, gaming, shopping, and banking. This shift has also led to the decline of older systems such as Windows Mobile and Symbian, which were unable to adapt to the touchscreen paradigm.[5]

Right now, Android and iOS are the top dogs in the global market, and they're pushing the development and design of mobile apps. They're always evolving because people want more secure, user-friendly, and feature-rich environments that let them create sophisticated apps to meet different user needs. [1][5][6]

1.3 Understanding mobile devices' capabilities and limitations.

Mobile devices have become very popular, offering tons of convenience and functionality. However, their design and how they are used are influenced by a few key things that determine what they can and can't do. For example, screen size affects how content is displayed and interacted with, so designers need to make everything simple and easy to use.[1]
Battery life is another important thing to think about because the portability of mobile devices depends on whether they can work for a long time. This means that apps need to be designed to consume as little power as possible so that they don't drain the battery.[1, 2]
The amount of processing power and memory available also affects how complex tasks can be. Today's smartphones are getting more powerful, but developers still need to think across a wide range of different devices to make their apps accessible to as many people as possible. To solve these problems, developers often use offline functionality or data caching. This ensures that users get a good experience even if they are offline or have slow internet.[1]
It's important to understand these limitations when you're developing a mobile app. They affect the design of the app, how users interact with it, and its success in the market.

2 MULTIPLATFORM DEVELOPMENT TECHNOLOGIES

Multi-platform mobile development has become one of the most popular trends in software development in recent years. This approach allows the same code to be used on different platforms, saving money, time and effort. According to We Are Social Inc. and Hootsuite Inc. in their Digital 2020 report, the number of internet users worldwide is growing at a rate of 9 people per second, which means that more than 800,000 people join the online world every year. A day via desktop or mobile devices. Interestingly, the latter option is becoming more popular every month. Globally, the penetration of mobile devices in everyday life is driving the overall growth of the mobile market. It can be said that by 2024, three quarters of con- sumers worldwide will be using cell phones. According to StatCounter, the share of desktop computers will drop to 45.66%. The simplest explanation is changes in our life-styles. We are using the internet more and more. Almost all of us have a cell phone or tablet. Consider the fact that the average user spends almost 7 hours a day on the internet, and more than half of that traffic comes from our phones. Hence, this is actively driving the growth of the mobile app market. The experience of using mobile apps is irresistible. Global mobile app revenue was $461 billion in 2019, and revenue from paid downloads and advertising could exceed $935 billion by 2023, according to a Statista report published last year. [6, 7]
Multi-platform app for both systems. Cross-platform refers to the ability of software (in this case, a mobile app) to run on multiple platforms. Cross-platform development for mobile devices allows you to use the same code on both iOS and Android operating systems. This approach eliminates the need to write code in a low-level programming language and pro-vides an almost natural user experience through the use of a visual interface and customiza-ble controls. This makes mobile app development easier and cheaper, which is no small part for any business. Many companies are now using cross-platform solutions, and some are already seriously considering switching to them in the foreseeable future. This applies not only to solution providers such as Facebook and its React Native, on which Facebook and Instagram apps run, but also to other important market players who already have products on Flutter, such as Alibaba, Philips Hue, Hamilton. Tencent, Grab, Groupon and others.
Below we will look at a few of the platform's main solutions for managing cross-platform development.[11]

2.1 Brief introduction to various multiplatform technologies

As we have already realized, multi-platform frameworks allow us to reach a wider audience at a lower cost. The benefit is that these frameworks translate applications into their original code. This means that we don't need to write an app for each platform - we can create one app with the help of a framework, and the framework will take care of the rest, compiling the code for each platform.

Every business wants to save money, and multi-platform frameworks can help us do just that. We don't need to hire different developers for each platform - one app can be used on multiple platforms thanks to the framework. Let's take a look at some of the most popular mobile development frameworks: Flutter from Google and Xamarin and MAUI from Microsoft. These frameworks are used by some of the biggest and most well-known companies, such as Google Pay, BMW, eBay and Alibaba, who have mobile applications built with these frameworks. [15]

For example, Google Pay uses Flutter to create its mobile payment app. BMW also uses Flutter for its mobile app, as does eBay. Alibaba has also used Flutter to develop its mobile apps. [15]

On the other hand, MAUI, developed by Microsoft, is also used by several well-known companies. For instance, NBC Sports Next and Escola Agil use MAUI to create their mobile apps. [16]

2.2 Key features and architectural principles.

One of the main advantages of using multi-platform tools is their ability to address each platform's native APIs. This allows developers to take advantage of features specific to each platform, such as access to camera, location, and sensors. This ensures optimal performance and a seamless user experience.

In addition, multi-platform frameworks provide a consistent coding experience across platforms, reducing development time and effort. This is achieved by utilizing common code bases and libraries that are shared across platforms.

Another benefit of using such frameworks is the ability to develop more scalable and maintainable applications. With a common code base, developers can easily update and maintain the application on different platforms. Another significant advantage of using a common code base is that it allows applications to run on different platforms simultaneously, making

it easier to maintain and update them. This reduces the overall time and effort required to manage applications because changes can be made and tested in a centralized location.
With CI/CD, developers can automate the testing and deployment process, ensuring that applications are always up to date and functioning properly. This leads to a smoother and more efficient workflow, as bugs are quickly identified and fixed, and changes are implemented quickly and easily.

3 COMPARATIVE ANALYSIS OF MULTIPLATFORM FRAMEWORKS

This section aims to provide a comprehensive analysis and comparison of several leading multi-platform development frameworks. We will evaluate their performance, user interface, user experience features, development environment, access to native application programming interfaces, and level of community support.

3.1 Flutter

Flutter is a free and open-source framework for mobile user interface development, was developed by Google in May 2017. It is also a multi-platform framework for mobile application development, based on a single programming language - Dart. Flutter is also aimed at working with complex user interfaces, animations, perhaps at the moment it is one of the leading multi-platform frameworks for working with mobile applications, as it has great flexibility. Flutter targets two important areas: [9, 12]

SDK: A set of tools that facilitates app development, including compilers that convert your code into native machine code for iOS and Android.[13]

Framework: A widget-based UI library that includes functional UI elements such as buttons, text boxes, and sliders that can be customized.[12]

Flutter uses Dart to write the code, which was also developed by Google in October 2011 and has been greatly improved in recent years. Dart originally focused on web development, but it can also be used to create mobile and web applications.

Flutter supports working with systems such as:

Android, IOS, Web, Windows OS, MacOS, Linux, Embedded devices [14]

Architectural layers

Flutter is built in such a way that it consists of several independent libraries, where each layer relies on the functionality of the one below it. This architecture ensures that no layer has privileged access to the layer below it, which encourages modular and customizable design. Each component of the framework is made optional and replaceable, allowing developers to tailor the system to their specific needs or replace parts of the framework without breaking the overall architecture. This design philosophy allows for extensive customization and scalability, facilitating the development of a wide range of applications that can take advantage of Flutter's capabilities. [9, 12]

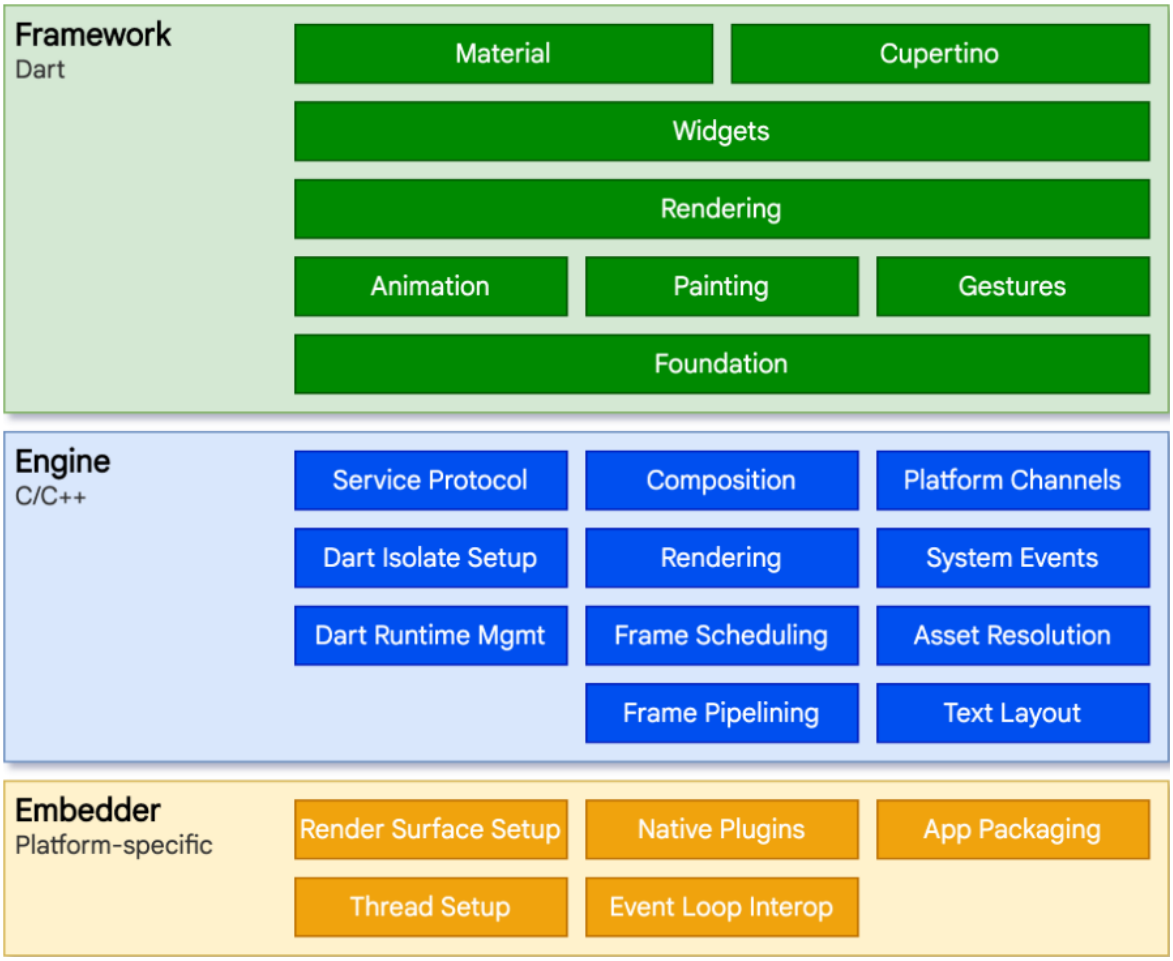

Figure 1 Flutter architectural layers [9]

Flutter design includes a well-defined layered architecture that makes it easy to create responsive and dynamic user interfaces. Here's an exploration of its key concepts and components: [9, 12]

Layer Model: Flutter is built using a hierarchical layer model, where each layer consists of modular independent libraries. These layers are layered sequentially on top of each other, with no layer having special access to the layer below, allowing for high customization and scalability. [9]

Reactive user interfaces. Flutter's UI design philosophy is based on a reactive approach to user interfaces. This model ensures that any changes to the application's state are immediately reflected in the user interface, providing a dynamic and interactive user experience. [9]

Introduction to widgets. Widgets are the basic building blocks of Flutter user interfaces. They define the structural elements, stylistic elements, and layout aspects that serve as the primary method for developers to interact with the platform. [9]

Rendering Process: Flutter converts the UI code into pixels on the screen using the rendering process. This requires taking widget descriptions and turning them into a series of rendering commands that render the user interface in detail and fluidly. [9]

Overview of embedded platforms. Built-in platforms are very important for running Flutter applications on various operating systems such as iOS, Android, Windows and others. They handle important tasks such as surface rendering, input handling, and integration with other native code. [9]

Integrate Flutter with other code: Flutter provides various methods of integration with other codebases and libraries, allowing you to reuse existing code and introduce new features. These include using platform channels to interact with native code and using packages to enable external functionality. [9]

Web support. Flutter web support allows you to deploy apps in the browser using the same basic principles and components of Flutter. This support ensures that Flutter apps can maintain consistent behavior and performance across desktop, mobile, and web platforms.

Each of these components plays an important role in making Flutter an effective tool for building modern, multi-platform applications. [9]

Flutter applications targeting underlying operating systems are compiled in the same way as traditional native applications. Each platform-specific module acts as a gateway, facilitating interaction with the operating system for functions such as UI rendering, adaptive access and data entry, and message management and event handling. These platform modules are built using native languages: Java and C++ for Android, Objective-C and Objective-C++ for iOS and macOS, and C++ for Windows and Linux. This allows Flutter to be integrated as a module into existing applications or used as the core content of an application, supported by a variety of embeddable modules adapted to common platforms. [9]

Central to Flutter's operation is its engine, written primarily in C++, which performs important tasks such as rasterising scenes for frame rendering. It provides fundamental support for Flutter applications, including graphics processing (currently via Impeller on iOS and soon on Android, as well as Skia on other platforms), text creation, file and network operations, accessibility features, plugin architecture, and the Dart runtime and compilation environment. [9]

The engine interacts with the Flutter framework through the dart:ui library, which wraps basic C++ code in Dart classes, making lower-level operations such as input control and graphics rendering available. [9]

Developers interact with Flutter through its framework, a modern reactive system developed in the Dart language. This framework consists of several layers: [9]

Base Classes and Services: These include basic functionality such as animation, drawing, and gesture recognition, offering fundamental abstractions over the platform infrastructure. [9]

Rendering Layer: This layer abstracts layout management, allowing developers to create and modify an object tree that is dynamically updated to reflect any changes. [9]

Widget Layer: In this layer, each rendering object is associated with a widget class, making it easy to define and reuse combinations of classes in a reactive programming environment. [9]

Material and Cupertino libraries: These libraries provide a complete set of controls that conform to Material Design or iOS aesthetics using the compositional primitives of the widget layer. [9]

In addition, the Flutter framework remains relatively compact, and many advanced functionalities are provided through external packages. These packages offer both platform-specific plugins, such as for camera or webviews, and platform-independent features, such as networking and animation, built on top of the underlying Dart and Flutter libraries. [9]

3.1.1 Detailed exploration focusing on performance, UI/UX capabilities, and developer ecosystem.

Flutter architecture achieves superior productivity by compiling Dart code directly into native machine code, bypassing the need for intermediate interpretation.

This compilation approach improves execution efficiency and accelerates application responsiveness. Consequently, applications developed with Flutter run faster and are more resource-efficient than those that rely on a virtual machine for interpretation.

In terms of UI and UX, Flutter provides developers with a complete set of tools to create highly responsive and visually appealing applications. Its extensive widget library supports the creation of customizable and dynamic interfaces that easily adapt to different screen sizes and device orientations, providing a seamless and engaging user experience across all platforms.

From a developer's perspective, Flutter creates a beneficial ecosystem that significantly reduces development challenges. Its unified codebase approach allows developers to write once and deploy across multiple platforms such as iOS, Android, Windows and the web. This cross-platform capability not only simplifies the development process but also speeds

up the time to market for apps. Moreover, the strong support from Flutter's vibrant community and Google's continued investment in the framework provides developers with a wealth of resources, best practices, and opportunities to collaborate, creating a dynamic and supportive development environment.

This review highlights Flutter's performance advantages, its ability to create versatile user interfaces, and the benefits of its developer-friendly ecosystem that contribute to its popularity and effectiveness as a development platform. [17, 18]

3.2 React Native

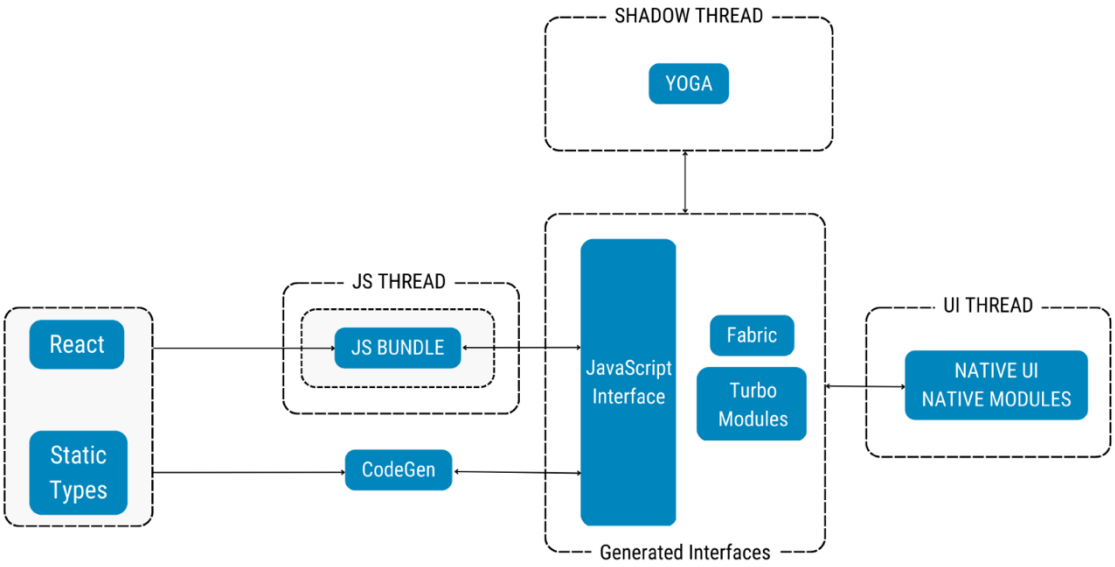

Figure 2 React Native new Architecture [8]

React Native is a very popular JavaScript-based framework used to create mobile applications for iOS and Android. This allows developers to write the same code for both platforms, thereby saving time and resources. With React Native, you can create apps that run on both iOS and Android without creating two separate apps. Facebook launched React Native as an open-source project in 2015, and since then it has become a popular choice for mobile app development. It is used by many large companies such as Instagram, Facebook and Skype.

The main reason for the success of React Native is that it allows you to write code once and use it on iOS and Android devices. This saves a lot of time and effort during the development process. It is also based on the React JavaScript library that most developers know. This means web developers can easily transition to building mobile apps using React Native without having to learn a whole new language.

To understand where React Native comes from, we need to remember when Facebook tried to create a mobile website. They decided to use HTML5, but it didn't work the way they wanted. Then they realized they needed a better solution and React Native was born. In 2012, Facebook CEO Zuck realized he was too reliant on HTML5 and needed to try something new. This is how React Native was born. It was first released in 2014 and has been a game changer ever since. Allows you to create mobile applications using JavaScript.

Initially, React Native was only for iOS, but later they added support for Android. In 2015, everyone used it. Now this is one of the most popular photographs in the world.

The impact of React Native has been huge. This not only changed the way apps are built, but also impacted the entire industry. It was the second most popular project on Github in 2018, and its popularity continues to grow. In 2019 it took sixth place in popularity. Advantages of local reaction: [19]

React Native is implemented on the React Framework and does not require WebView or HTML to work.

React Native solves the problems of its predecessors; it is designed to work with an adaptive interface, that is, previously developers had to come up with algorithms to display the interface correctly on different devices. React Native, in turn, has this right. hood.

React Native doesn't use DOM API.

It is designed for creating mobile applications based on Android and IOS.

It allows you to work with complex designs and animations.

It is used to create universal apps for Android, iOS, tvOS, macOS, Android TV, Windows, web and Windows platform.

It is compatible with Android, iOS, Android TV, macOS, tvOS and Microsoft Windows [20].

3.2.1 Analysis of development efficiency, access to native APIs, and community support.

Through enabling the creation of both iOS and Android applications from a single JavaScript codebase, React Native has altered the trajectory of app development. This approach consolidates the coding effort, significantly cutting down on the time and resources traditionally spent on maintaining distinct codebases for various platforms, thus making React Native an efficient and economical option for developers aiming for multi-platform reach.

React Native is acclaimed for its seamless interaction with native APIs, granting developers the latitude to maximize the performance and features of core platforms, breaking free from

the usual restrictions of frameworks aimed at multiple platforms. By weaving JavaScript with native elements, the user experience approaches the quality and fluidity of native apps. The framework enables the integration of bespoke modules, tapping into the innate characteristics and functions of the operating systems to craft compelling and interactive app experiences.

The vibrant community that has rallied around React Native constitutes a substantial portion of its appeal. A robust, engaged collective has organically emerged, giving rise to a wealth of React Native-specific libraries, tools, and frameworks. Participation in this proactive open-source environment goes beyond code contributions; it extends to problem-solving, feature advancements, and the invention of new functionalities within React Native, offering developers a rich repository of knowledge and tools that facilitate and enhance the app creation process.

This community-focused nature of React Native places developers at the vanguard of mobile tech innovation. The framework's open-source ethos encourages a spectrum of developer contributions, fostering a culture of shared inventive problem-solving. This dynamic ensures that React Native stays abreast of technological advancements, supporting a wide range of devices and platforms, which now includes not just mobile but also web, tvOS, macOS, Android TV, and Windows.

To conclude, React Native epitomizes a model of effective and progressive development practices, fusing direct access to native APIs with the strength of a supportive community, thereby continuously broadening the horizons of what can be accomplished in the realm of cross-platform mobile application development. [22]

3.3 Xamarin (. NET MAUI)

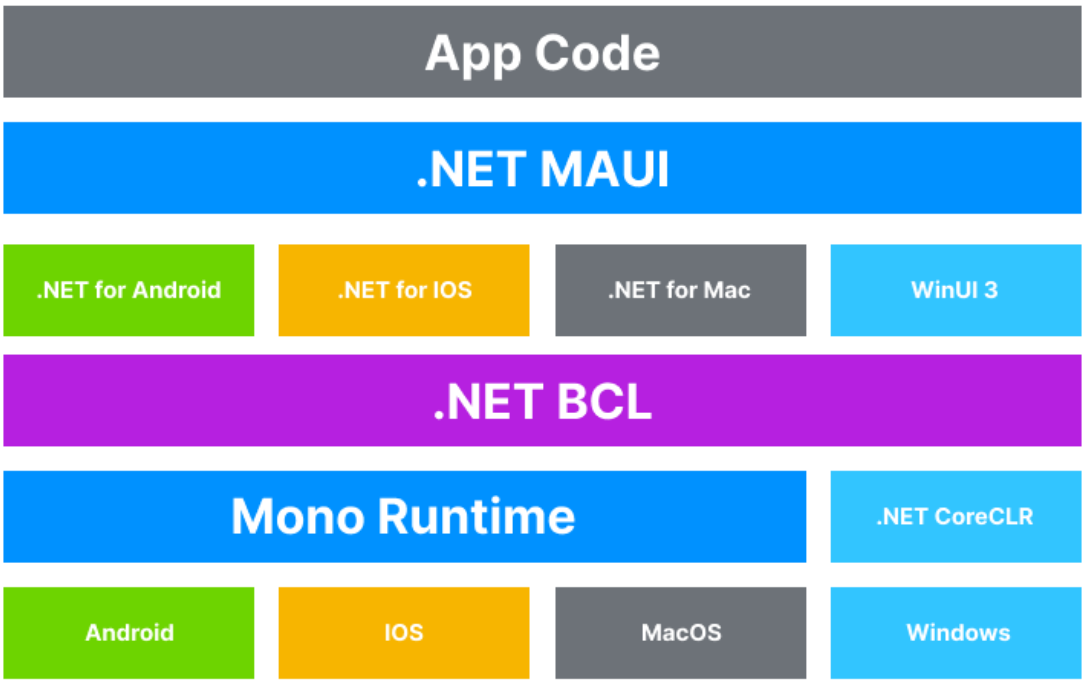

Figure 3 .NET MAUI Architecture [23]

.NET MAUI marks a watershed moment for developers by eliminating the need to create multiple versions of an application for different platforms. This innovation allows developers to write code just once and deploy it across multiple platforms, resulting in notable savings in time and financial resources.

With .NET MAUI, developers have access to state-of-the-art rendering engines that provide smooth animations and reduced load times, as well as improved memory management for increased application stability.

Moreover, the versatility of .NET MAUI allows applications to run on a variety of platforms including iOS, Android, macOS and Windows, increasing potential market reach and device compatibility.

One of the hallmarks of .NET MAUI is its ability to facilitate the creation of attractive and responsive user interfaces that easily adjust to different screen sizes and orientations, providing an optimal look and feel on any device. Tools from the .NET ecosystem, such as Xamarin, simplify the process of creating applications with .NET MAUI. Integrated development environments such as Visual Studio and Visual Studio Code provide robust platforms for coding, troubleshooting, and application refinement.

Backed by a robust Microsoft infrastructure and reinforced by a vibrant open-source community, .NET MAUI takes advantage of shared knowledge and collective support, offering developers a treasure trove of resources to improve application development.
In addition, .NET MAUI's compatibility with the entire .NET stack allows developers to use familiar libraries and tools, speeding up the development process and fostering more robust applications.
In essence, .NET MAUI provides a unified solution for cross-platform application development by providing all the necessary components in one consolidated package, allowing you to extend application functionality without spending too much development time.[24, 25]

3.3.1 Examination of cross-compatibility, integration with Microsoft technology stack, and development tools.

.NET MAUI simplifies app creation by allowing developers to use a single code base for apps across multiple platforms, facilitating cross-compatibility. This unified approach eases workflow and extended the reach of apps by allowing them to serve users on iOS, Android, macOS, and Windows in a single development effort.
The use of .NET MAUI is tightly integrated with Microsoft's technology stack, which is part of the extensive .NET framework. This allows developers to leverage Microsoft's well-known collection of tools and resources, ensuring robust application performance and a consistent development experience across all Microsoft offerings.
In terms of development tools, .NET MAUI comes with support for powerful IDEs such as Visual Studio and Visual Studio Code, which create an optimised environment for writing, troubleshooting and testing code. This favourable environment improves the efficiency of the development process and helps you build high-quality applications faster.
The relationship between .NET MAUI and the .NET framework provides developers with a flexible and rich development path. They benefit from an established ecosystem that provides a broad set of features, reducing the need to start from scratch and accelerating production timelines.
What's more, the .NET MAUI's developer community and open-source code, backed by Microsoft's support, promise continuous improvement in the platform's capabilities and user experience. This collaborative progress solidifies .NET MAUI's position as a technology leader, offering developers a cost-effective and reliable solution for building complex, scalable applications across multiple platforms. [23, 26]

3.4 Other Notable Frameworks

3.4.1 Ionic

Ionic so presents itself as a multi-platform solution implemented on web technologies. As in the above described frameworks allows you to develop for such platforms as Android, IOS, as well as for windows 10 and above. At some point in its existence, Ionic even planned to release its own IDE, but in 2020, after a failed attempt to implement the IDE, the company decided to abandon its development and focus on improving the framework itself. [27, 28]

3.4.2 NativeScript

Another multi-platform framework, the main idea of which is to combine the framework with popular web frameworks such as Angular and Vue.js allowing developers to use familiar tools and practices when creating mobile applications. In addition, NativeScript provides access to each platform's API, allowing you to utilize all native features such as camera, geolocation and more.

NativeScript is supported and promoted by Progress and a community of developers who contribute to the development of the framework and the creation of additional modules and plugins. [29]

4 CHALLENGES IN MOBILE APPLICATION DEVELOPMENT

After evaluating various multiplatform frameworks, I have chosen the Flutter framework for developing my mobile application. My decision to opt for Flutter is influenced by its demonstrated foresight and leadership in the realm of multiplatform frameworks. Flutter's robust capabilities and strategic advances in facilitating cross-platform application development have significantly contributed to its selection.

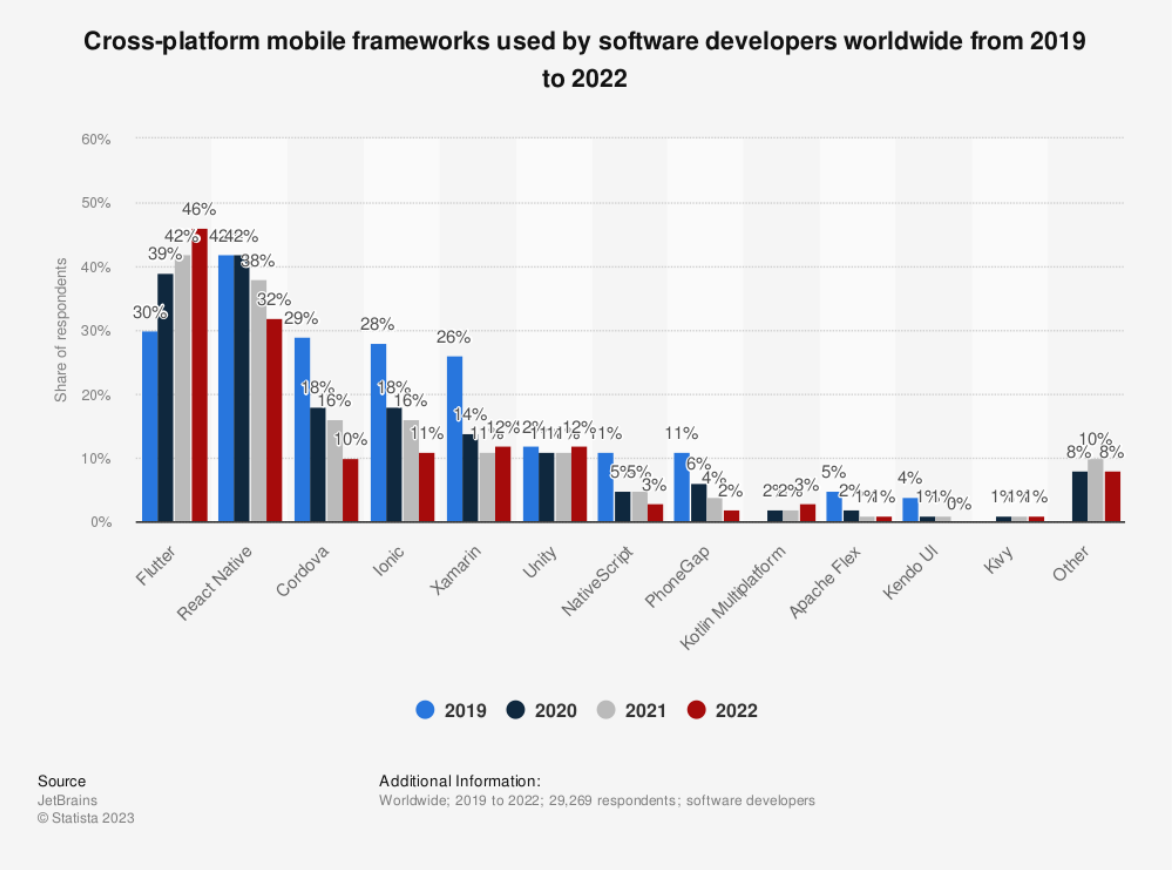

Figure 4 Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022 [31]

4.1 Addressing the intricacies of developing for multiple platforms.

In light of my analytical assessment and previous experiences, it is imperative to acknowledge that the supportability of a framework is critical, particularly in an environment where technological advancements are rapid, and devices are regularly updated, necessitating ongoing support from the framework. For instance, in the event that Android or iOS introduces a new feature that influences core application functionalities, the robust and actively engaged community surrounding Flutter is well-positioned to ensure that essential tools and updates are expeditiously incorporated into the framework.

In my previous experience with the Xamarin framework during the development of an educational application for the Japanese alphabet [30], I faced many insurmountable challenges. The importance of robust community support and the prevailing popularity of the framework are integral elements; however, in the case of Xamarin, I encountered notable shortcomings, including its limited adaptability to new features of mobile operating systems and incomplete support for the capabilities of modern devices. In contrast, Flutter is different in that it provides an extensive set of tools and a comprehensive library of widgets, making it noticeably easier to develop universal applications for a variety of devices. This adaptability of Flutter is not just theoretical, it is validated by an active ecosystem that is constantly updated and keeps it relevant and effective.

4.2 Handling diverse device capabilities, screen sizes, and user interaction modes.

Developing applications that perform effectively across a variety of devices, with different screen sizes and user interaction methods, requires a well-thought-out approach. Flutter excels in this area thanks to its flexible architecture:

Responsive Design: With Flutter's extensive set of widgets and layout mechanisms, I can easily create flexible and responsive designs. This ensures that applications look and function perfectly across devices of all screen sizes, from the smallest smartphones to the largest tablets.

Device Capability Utilization: Flutter enables developers to access and utilize the native features and capabilities of operating systems, ensuring that applications can fully leverage the potential of the hardware they run on. This includes everything from camera functions to accelerometer data.

User Interaction Adaptability: Flutter supports various user interaction modes, including touch, swipe, and tap, as well as traditional keyboard and mouse inputs. This adaptability is ideal for ensuring a seamless user experience in a multi-platform environment.

Accessibility and Localization: Flutter's commitment to providing accessible and internationally adaptable applications is evident. It supports straightforward localization and internationalization options, making it possible to cater to users from diverse linguistic and cultural backgrounds without extensive modifications to the core application.

5 JAPANESE LANGUAGE

Japanese is the main language of the Japanese language group. It is spoken by about 120 million people, mostly in Japan, where it is the official language. It is also widely used by the Japanese diaspora around the world. [32]

The scope of Japanese language includes the predominant Japanese dialect as well as various Ryukyuan languages. The origins of Japanese go back to Proto-Japanese, the common ancestor of modern Japanese and the Ryukyuan languages, which is believed to have been brought to the islands from the Korean peninsula in the early to mid-fourth century BC.[32]

5.1 Japanese writing system

The Japanese writing system is unique and complex, employing three main types of scripts that make it challenging for learners, particularly those not native to logographic systems. [33]

5.1.1 Hiragana

Hiragana is one of two kana systems in Japanese, a syllabary used predominantly for native Japanese words, grammatical elements, and inflections. It consists of 46 characters, each representing a distinct sound or mora. Children in Japan typically begin their education by learning Hiragana. It's also used to write words for which there is no Kanji, or in conjunction with Kanji to indicate grammatical conjugations. [32, 33]

5.1.2 Katakana

Katakana, the other kana system, mirrors Hiragana in phonetic structure but is used primarily to transcribe foreign words, loanwords, scientific names, and to provide emphasis or modern flavor to text. Like Hiragana, it consists of 46 characters and serves a distinct function in written Japanese by distinguishing foreign terms from native ones, which can be crucial in understanding context and nuance. [32, 33]

5.1.3 Kanji

Kanji are logographic characters derived from Chinese. In modern Japanese, over 2,000 Kanji are in regular use. Each Kanji has one or several meanings and can be pronounced in multiple ways depending on its context. Learning Kanji is considered one of the most challenging aspects of mastering the Japanese language due to its complexity and the sheer

volume of characters. Kanji conveys meaning visually and are crucial for understanding deeper nuances in texts, making them indispensable despite their difficulty. [32, 33]

5.2 Challenges in Learning Japanese Scripts

5.2.1 Adaptation for Mobile Learning

Adapting Japanese writing for use in a mobile app poses several unique challenges. The app will focus on two main features: flashcard memorization and writing tests. These functions are designed to help students learn characters effectively.

The app will introduce you to the most basic kanji characters.

For hiragana and katakana, the app will use flashcards to promote character recognition and memorization. Written tests will complement these by requiring users to practice writing the characters on their own, which promotes muscle memory and memorization.

This approach aims to build a fundamental understanding of Japanese writing, ensuring that students understand the basics of reading and writing, as well as the cultural context of the language. Through this focused method, the app aims to make the first steps in learning Japanese accessible and rewarding.

II. ANALYSIS

6 APPLICATION DEVELOPMENT STRATEGY

When planning out the development of the mobile application for learning the Japanese alphabet, I decided to go with Flutter, given its wide array of advantages. One of the main reasons for choosing Flutter was its capability to manage both iOS and Android development from a single source code. This not only makes the development cycle more efficient but also helps in conserving resources. Flutter is known for its vast selection of widgets that can be tailored to create detailed and interactive interfaces, which are key for educational apps like this one. Additionally, Flutter's direct compilation to native ARM code guarantees that the app performs as well as any native application, ensuring a smooth user experience. Supported by a strong community and backed by Google, Flutter provides extensive documentation and resources that facilitate troubleshooting and functional enhancements. Features such as Hot Reload, which allows developers to see changes instantly without a restart, significantly speed up the development process. Ultimately, Flutter's comprehensive features make it the top choice for building efficient, high-quality educational apps across various platforms.

6.1 Stack

For my project, I selected the following technology stack:

- Dart
- Flutter
- Json Serializable

The Dart Build System includes builders designed to handle JSON. These builders automatically produce code when they encounter members marked with annotations from the json_annotation package. To enable JSON serialization and deserialization for a class, add the JsonSerializable annotation to it. This annotation accepts arguments that allow you to tailor the resulting code. Additionally, you can modify specific fields by using the JsonKey annotation with appropriate arguments. For more information on the values of these annotations, consult the table below. Furthermore, to create a Dart field containing data from a JSON file, apply the JsonLiteral annotation. [34]

7 IMPLEMENTATION OF THE MOBILE APPLICATION

As the technologies had been selected, it became imperative to acquire a comprehensive understanding of the initiation process for a new technology, including the determination of an appropriate starting point. Concurrently, the decision regarding the choice of an Integrated Development Environment (IDE) was necessary. Flutter applications are commonly developed using Android Studio, VSCode, or IntelliJ. Given its simplicity and robust functionality, coupled with my previous experience, I opted for VSCode. The specifications of the computer used for development are detailed in Figure 5:

Figure 5 MacBook Characteristics

7.1 Flutter Development Environment Setup

Preparing to work with the Flutter framework began with an initial review of the official documentation. This step is extremely important when starting out with any framework, as it provides authoritative guidance and basic knowledge. The documentation first describes the system requirements needed for development, as shown in Figure 6.

Requirement	Minimum	Recommended
CPU Cores	4	8
Memory in GB	8	16
Display resolution in pixels	WXGA (1366 x 768)	FHD (1920 x 1080)
Free disk space in GB	44.0	70.0

Figure 6 Hardware requirements. [35]

Next, due to the peculiarities of my operating system, it is necessary to install Rosseta 2 using the command you can see in Figure 7.

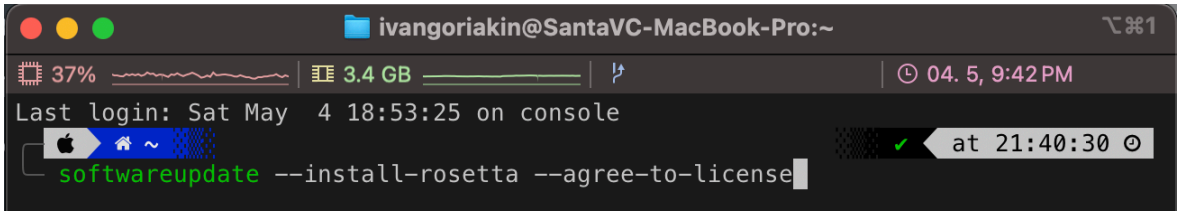

Figure 7 Install Rosseta 2.

After Rosseta 2 has been successfully installed, we will proceed to install the components for working with Flutter. Now it is necessary to download Flutter itself. Since my operating system is MacOS, I use the package manager Homebrew. Using the command you can see in Figure 8, we install Flutter on the local machine.

Figure 8 Homebrew install Flutter. [36]

After a successful installation we need to find out what further components we need to work with Flutter using the flutter doctor command you can see in Figure 9.

```
❯ flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.19.6, on macOS 14.4.1 23E224 darwin-arm64, locale
    en-CZ)
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] Xcode - develop for iOS and macOS (Xcode 15.3)
[✓] Chrome - develop for the web
[✓] Android Studio (version 2023.3)
[✓] VS Code (version 1.89.0)
[✓] VS Code (version 1.87.2)
[✓] Connected device (3 available)
[✓] Network resources

• No issues found!
```

Figure 9 Flutter Doctor.

Also in Figure 9 we can see all the components that are required and their status. In my case, I already have each required component installed. However, I will demonstrate the installation of each of them:

Flutter: Already downloaded you can see Figure 8.

Android toolchain: I use ToolBox by JetBrains for downloading Android Studio Figure 10. [37]

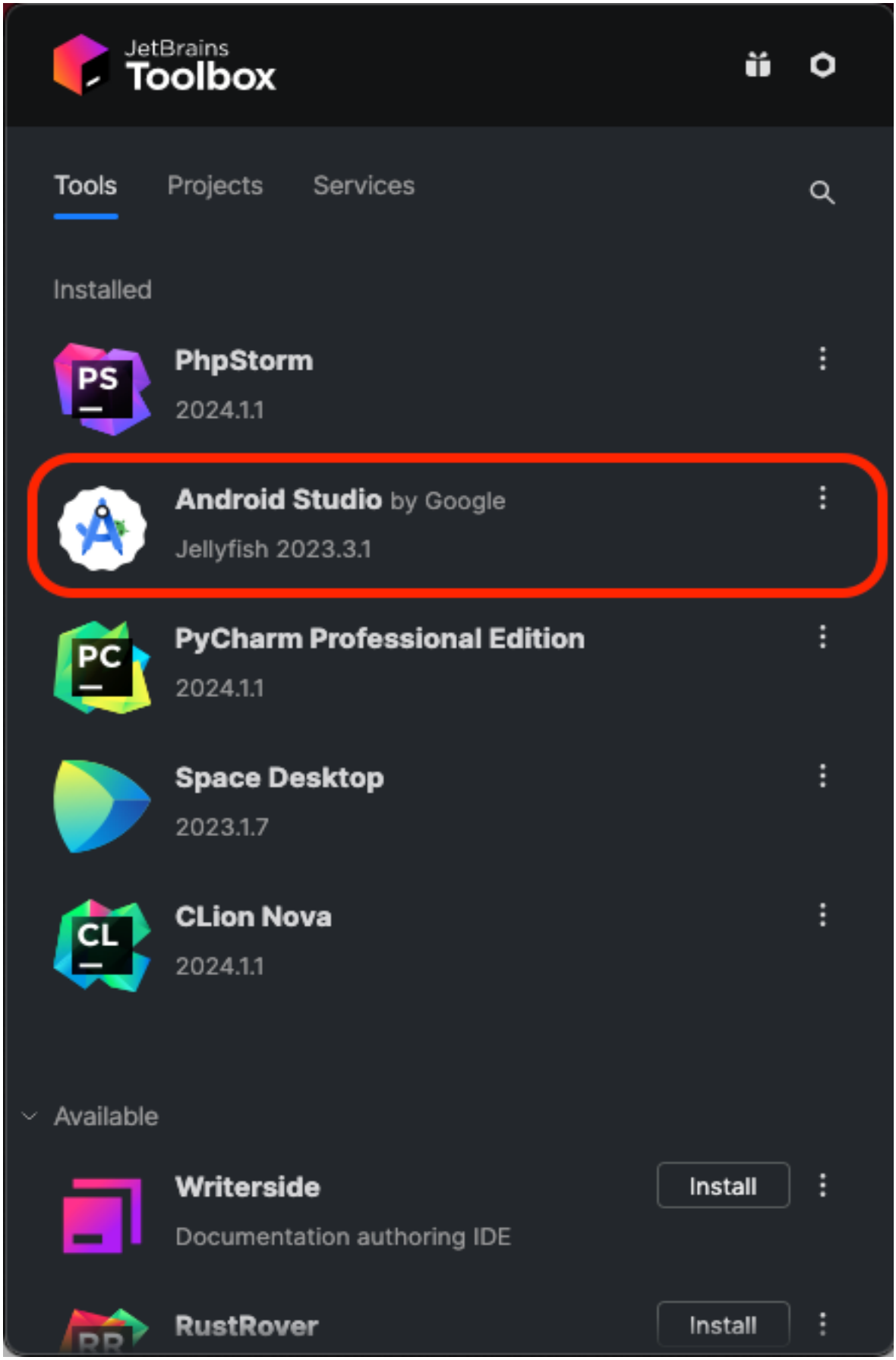

Figure 10 JetBrains Toolbox Android Studio.

XCode: For install XCode I use AppStore, default application in MacOS. You can see install in Figure 11.

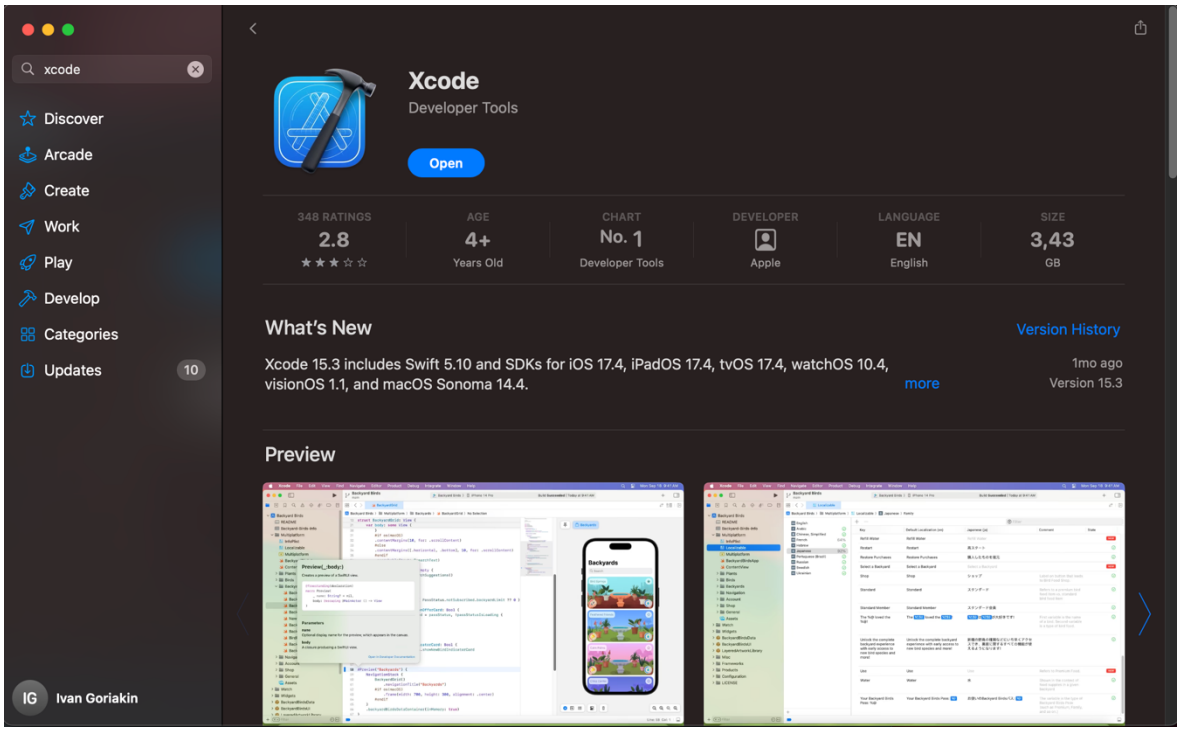

Figure 11 XCode install.

Chrome: For install Chrome go to official Google Chrome website you can see in Figure 12. [38]

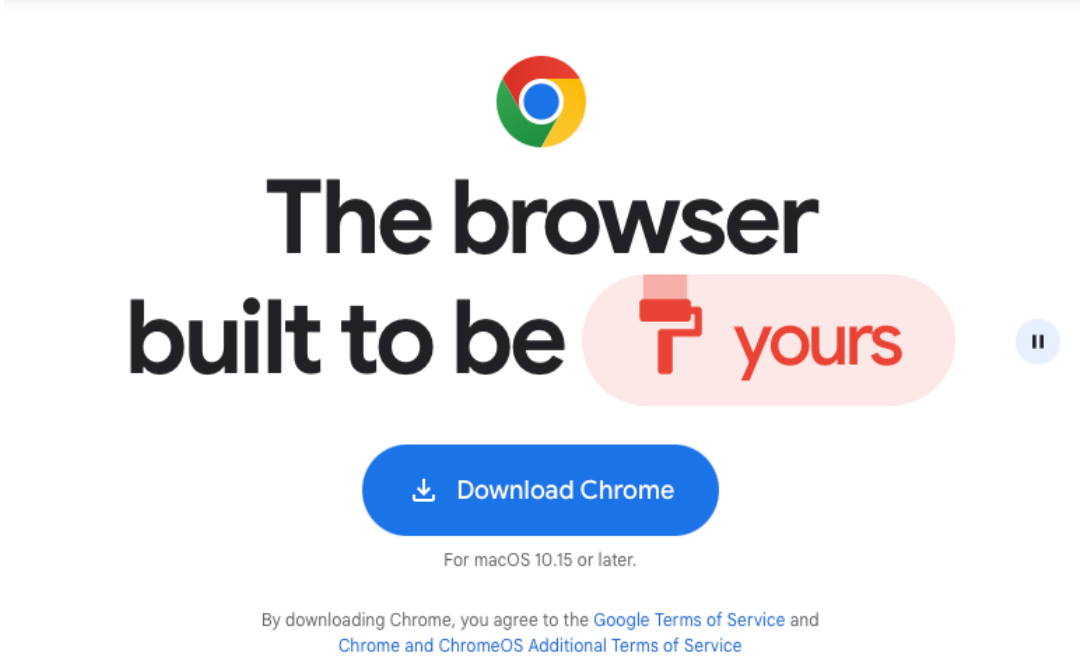

Figure 12 Install Chrome. [38]

VSCode: For install VSCode go to official website can see in Figure 13. [39]

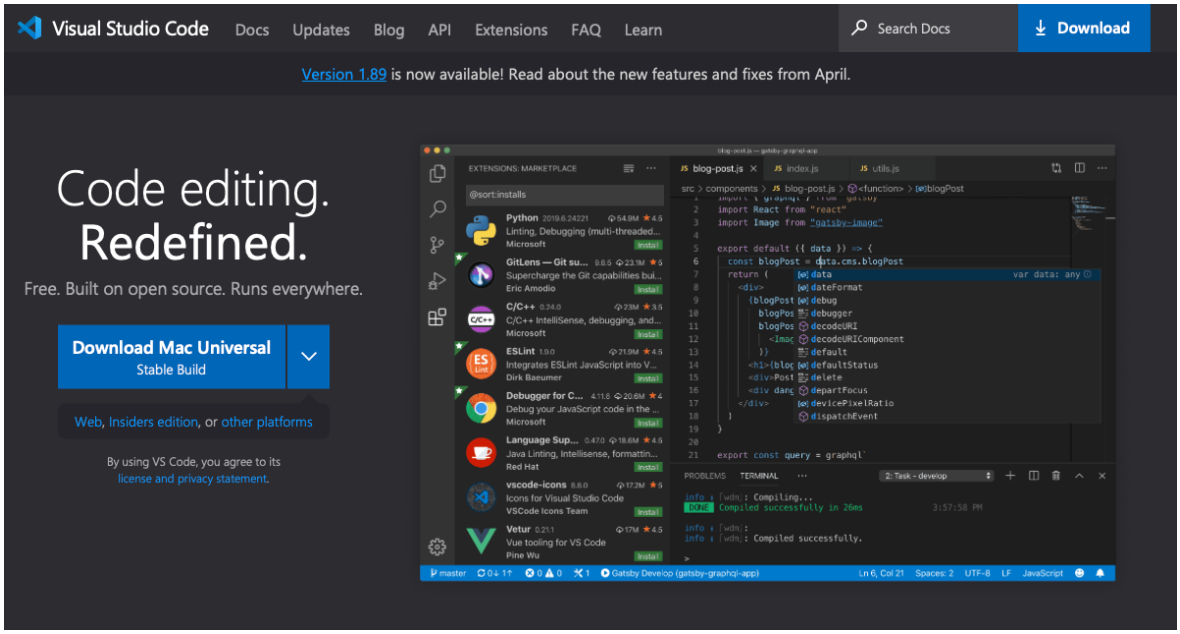

Figure 13 Install VSCode. [39]

After successfully installing the required components, we can see the tests of the flutter doctor command passing successfully can see Figure 9.

Now we can start preparing vscode for flutter development. The extensions only recommended for comfortable development.

Extensions: Flutter, Dart, Flutter Color, Flutter Tree, Flutter-Stylizer, Json to Dart Model, Prettier, Sort Lines, VSCode-Icons.

7.2 Building the Application

7.2.1 Solution Structure

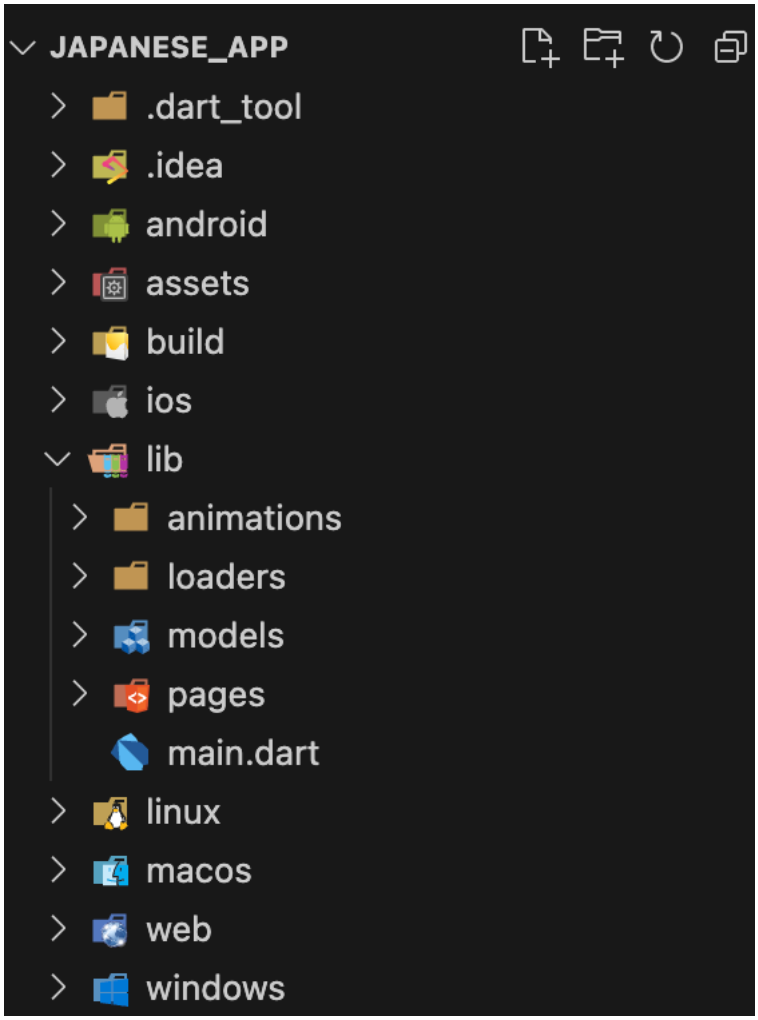

Figure 14 Solution Structure.

.dart_tool: This directory is used by the Dart programming language for various SDK-related files. It's generally auto generated and used internally by Dart.

.idea: Contains project settings specific to JetBrains IDEs, like IntelliJ IDEA or Android Studio. It includes configurations such as code styles, project profiles, etc.

.vscode: Stores configurations for Visual Studio Code, such as settings for the editor, extensions, and debug configurations.

android: Contains all the necessary files to build and run the app on Android devices, including Gradle scripts and the Android manifest.

assets: A directory to store assets like images, audio, video and data files which are bundled with the application when built.

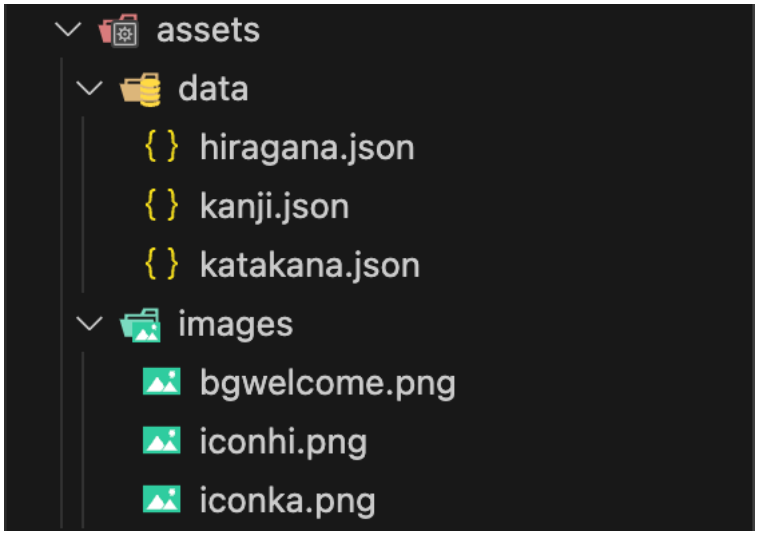

Figure 15 Assets Structure.

In my case, I use the assets directory to store images and json data for Japanese alphabets Figure 15.

build: Contains the output files from the build process, including intermediate files. This directory is automatically generated when you build the application.

ios: Similar to the android directory but specifically for iOS. It contains the Xcode project and configurations needed to build the iOS version of the app.

lib: The main directory for Dart code:

- loaders: Contains files that responsible for loading data or resources dynamically.
- models: Files defining the data model of the application.
- pages: Contain files for different screens/pages of the app.

linux, macos, windows: These directories contain platform-specific code to compile the app on desktop operating systems using Flutter's desktop support.

web: Holds files that enable building the application for web browsers, leveraging Flutter's web support.

This organized for a Flutter project, separating platform-specific code and assets, while maintaining a clear path for main development in the `lib` directory. Each directory serves a clear function, supporting multiple platforms and facilitating both development and testing across them can see in Figure 14.

7.2.2 Pages Structure

In 7.2.1 I already touched on the Lib directory, now let's look at it in more detail.

Let's start with the fact that the main logic and implementation of the application is mostly done in the Lib directory. For structured work, I have divided functions and modules into separate directories, it allows better orientation in the project, as well as in the case of using code by third-party developers, speeds up the process of their integration into the project Figure 16.

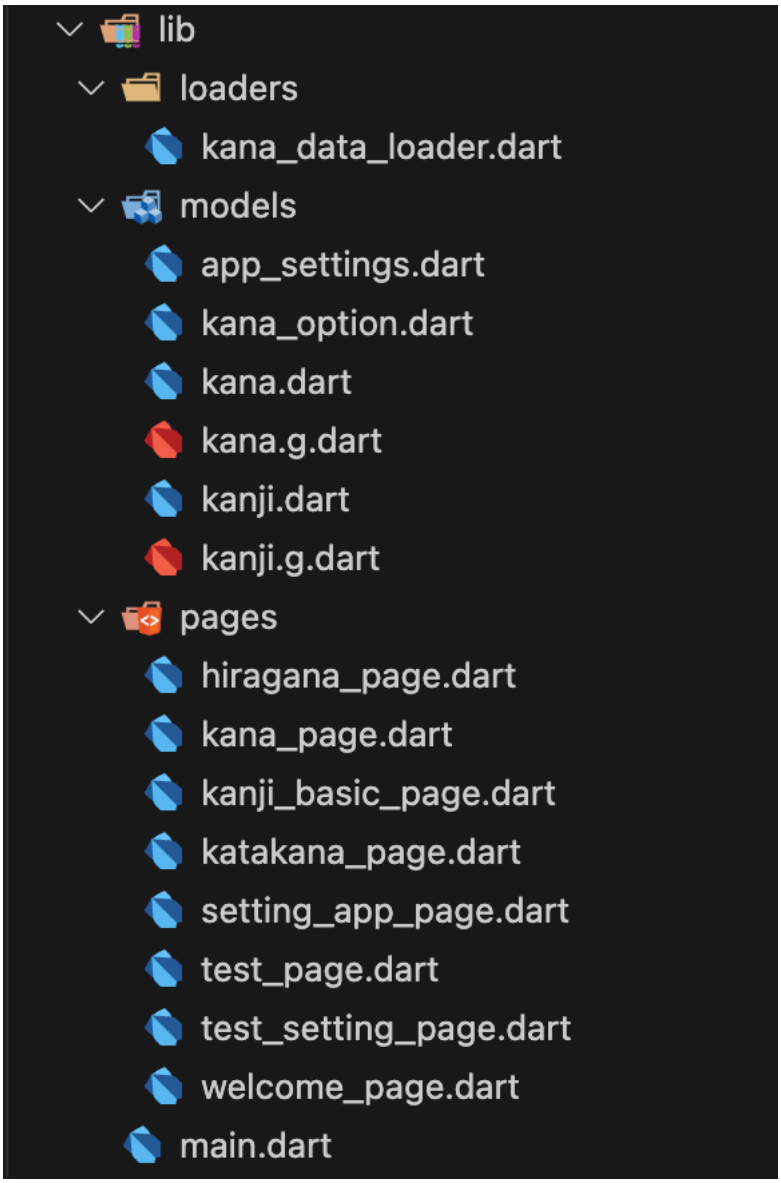

Figure 16 Lib Directory.

The main file of any project is usually Main.dart (As part of Flutter development), and because of we`ll start from Main.dart.

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'pages/welcome_page.dart';
import '../models/app_settings.dart';

void main() {
```

```
  WidgetsFlutterBinding.ensureInitialized();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider<AppSettings>(
      create: (_) => AppSettings(),
      child: MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'Japanese Learning App',
        theme: ThemeData(
          colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),
          useMaterial3: true,
        ),
        home: WelcomePage(),
      ),
    );
  }
}
```

Imports:

- `flutter/material.dart`: This import includes Material Design widgets that are visually consistent and customizable.
- `provider/provider.dart`: This import allows the application to manage state using the Provider package.
- `pages/welcome_page.dart`: Imports the WelcomePage widget, the first screen that users see when launching the application.
- `../models/app_settings.dart`: Imports the AppSettings class, which, is used to store and manage the settings of the entire application.

Main function:

- main(): The main entry point of any Dart program. It calls WidgetsFlutterBinding.ensureInitialised() to initialize the framework bindings before running the application, which is necessary for the plugins to be configured before `runApp()` is executed.
- runApp(MyApp()): This function inflates the given widget and binds it to the screen. It is the root of the widget tree that will be built.

MyApp class:

- MyApp: A widget that extends `StatelessWidget`, which means that it does not manage any internal state changes.
- It returns a ChangeNotifierProvider<AppSettings>, which sets the provider for the AppSettings object. This allows child widgets to access AppSettings and respond to changes to it throughout the application.
- The MaterialApp widget is used as a child of the ChangeNotifierProvider. It includes several widgets that often need to be accessed within the application, such as for navigation and theming.
- debugShowCheckedModeBanner: Set to false to remove the debug banner that appears in the top right corner in debug mode.
- title: A string describing the application, used by the device to identify the application by a user-friendly title.
- theme: Defines the visual design of the application. ThemeData is set to a blue colour scheme and uses Material Design 3 features (`useMaterial3: true`).
- home: The first screen displayed in the application, configured with WelcomePage().

The next step was to create the pages of the app. Usually the work of Frontend and Backend developers is divided into roles, but in the case of my application, I acted as a fullstack developer. The interface of the pages will be presented below.

7.2.2.1 *Welcome Page:*

```
class WelcomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    Size screenSize = MediaQuery.of(context).size;

    return Scaffold(
      body: Stack(
        children: [
          Container(
            width: screenSize.width,
            height: screenSize.height,
            child: Image.asset(
              'images/bgwelcome.png',
              fit: BoxFit.cover,
            ),
          ),
          positionedButton(context, 'KANA', KanaPage(), 0.2, false),
          positionedButton(
              context, 'KANJI BASIC', KanjiBasicPage(), 0.35, true),
          positionedButton(context, 'SETTING', SettingAppPage(), 0.5, false),
```

```
        ],
      ),
    );
  }

  Widget positionedButton(BuildContext context, String title, Widget page,
      double top, bool isLeftAligned) {
    BorderSide borderSide = BorderSide(color: Colors.black, width: 2);
    Size screenSize = MediaQuery.of(context).size;

    double fontSize = screenSize.width * 0.06;
    double padding = screenSize.width * 0.08;

    return Positioned(
      top: screenSize.height * top,
      left: isLeftAligned ? null : null,
      right: isLeftAligned ? null : 0,
      child: Container(
        decoration: BoxDecoration(
          border: Border(
            top: borderSide,
            bottom: borderSide,
            left: isLeftAligned ? BorderSide.none : borderSide,
            right: isLeftAligned ? borderSide : BorderSide.none,
          ),
        ),
        child: Material(
          color: Colors.transparent,
          child: InkWell(
            onTap: () => Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => page),
            ),
            child: Container(
              padding:
                  EdgeInsets.symmetric(horizontal: padding, vertical: padding),
              child: Text(
                title,
                style: TextStyle(
                  color: Colors.black,
                  fontSize: fontSize,
                ),
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

WelcomePage Class

- Extends StatelessWidget: Indicates that this widget does not manage any internal state.
- build Method: Constructs the UI elements to be rendered by this widget.
- Size screenSize: Retrieves the size of the screen to adjust the layout dynamically based on the device's screen size.

Scaffold Widget

- Used as the primary visual structure for the page, which scaffolds various UI components together.
- Body: Composed of a Stack widget, allowing overlay of elements.

Children of Stack

1) Background Container:
    - Covers the entire screen (width and height set to match the screen dimensions).
    - Displays an image (images/bgwelcome.png) as a full-screen background with the image scaled to cover the entire viewing area.
2) Positioned Buttons:
    - Three calls to a custom method positionedButton to create buttons that navigate to different pages (KanaPage, KanjiBasicPage, SettingAppPage).
    - Buttons are positioned at different vertical levels (top is a proportion of the screen height) and are designed to toggle alignment (left or right) through the isLeftAligned parameter.
    - Each button has a dynamic size and font based on screen width, ensuring scalability across different device sizes.

positionedButton Function

- Parameters: BuildContext context, String title, Widget page, double top, bool isLeftAligned.
- Returns a Positioned widget that places a clickable button at the specified vertical position.
- Container Decoration:
    - A border that changes based on whether the button is left or right aligned, making the border dynamic and visually indicating the alignment.

- Material & InkWell:
    - Used for visual effects on touch such as ripples and to handle tap gestures which trigger navigation using a MaterialPageRoute to the respective pages.

The implemented interface can be seen in Figure 17.

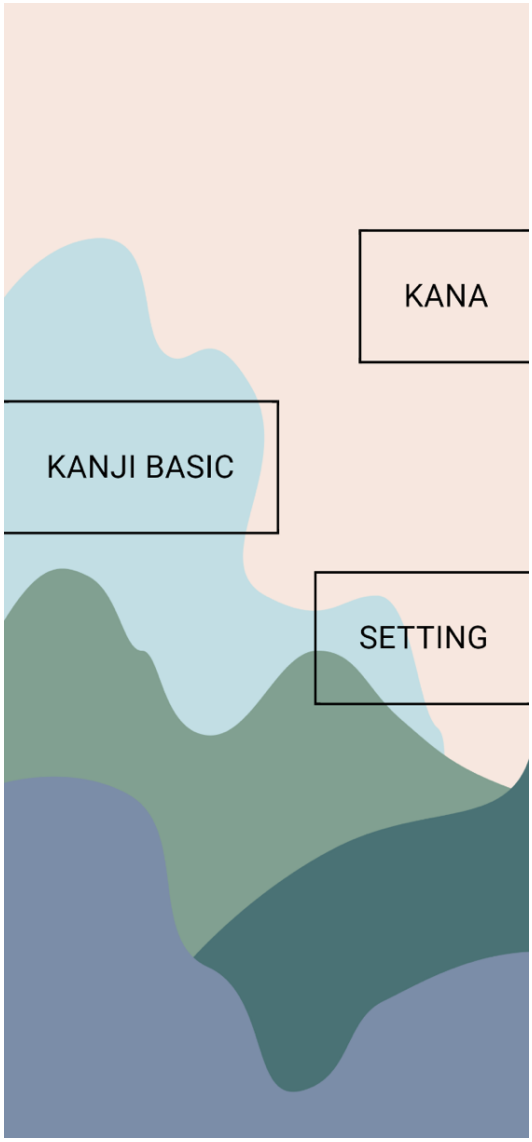

Figure 17 Welcome Page.

7.2.2.2 *Kana Page:*

```
class KanaPage extends StatefulWidget {
  @override
  _KanaPageState createState() => _KanaPageState();
}

class _KanaPageState extends State<KanaPage> {
  int _selectedIndex = 0;
  final PageController _pageController = PageController(keepPage: true);
```

```
  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
      _pageController.animateToPage(index,
          duration: Duration(milliseconds: 300), curve: Curves.easeInOut);
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Kana Learning')),
      body: PageView(
        controller: _pageController,
        onPageChanged: (index) {
          setState(() {
            _selectedIndex = index;
          });
        },
        children: [
          HiraganaPage(),
          KatakanaPage(),
        ],
      ),
      bottomNavigationBar: BottomNavigationBar(
        items: <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Image.asset('images/iconhi.png', width: 24, height: 24),
            label: 'Hiragana',
          ),
          BottomNavigationBarItem(
            icon: Image.asset('images/iconka.png', width: 24, height: 24),
            label: 'Katakana',
          ),
        ],
        currentIndex: _selectedIndex,
        selectedItemColor: Colors.amber[800],
        onTap: _onItemTapped,
      ),
    );
  }
}
```

Class Definition

- KanaPage: A StatefulWidget that will rebuild when state changes, due to user interactions or internal events.
- _KanaPageState: The state class for KanaPage, which holds the logic and mutable state for the widget.

State Management

- _selectedIndex: A variable to keep track of the currently selected tab or page.
- _pageController: Manages the page being displayed in a PageView. The keepPage flag is set to true to maintain the page state when switching between tabs.

Navigation and Page Control

_onItemTapped: A function that is triggered when a BottomNavigationBar item is tapped. It updates _selectedIndex and animates the PageView to the selected page.

Widget Build Method

- Scaffold:
    - AppBar: Displays a simple app bar with a title "Kana Learning".
    - Body: Uses a PageView widget for swipeable pages between Hiragana and Katakana.
        - onPageChanged: Callback that updates _selectedIndex when pages are swiped manually.
        - Children:
            - HiraganaPage(): A widget showing content related to Hiragana.
            - KatakanaPage(): A widget for learning Katakana.
- BottomNavigationBar:
    - Items: Defines two items with icons and labels for Hiragana and Katakana.
        - Hiragana item uses an image asset 'images/iconhi.png'.
        - Katakana item uses an image asset 'images/iconka.png'.
    - currentIndex: Indicates the currently selected item based on _selectedIndex.
    - selectedItemColor: Specifies the color of the selected item to enhance visibility, here using Colors.amber[800].
    - onTap: Connects to _onItemTapped to handle item selection.

The implemented interface can be seen in Figure 18, Figure 19.

7.2.2.3 *Hiragana Page:*

```
class HiraganaPage extends StatefulWidget {
  @override
  _HiraganaPageState createState() => _HiraganaPageState();
}
```

```
class _HiraganaPageState extends State<HiraganaPage>
    with AutomaticKeepAliveClientMixin {
  final HiraganaDataLoader _dataLoader = HiraganaDataLoader();
  late Future<List<List<Kana>>> _dataFuture;
  late List<bool> checkedStatus;

  @override
  bool get wantKeepAlive => true;

  @override
  void initState() {
    super.initState();
    _dataFuture = _dataLoader.loadKanaData();
    _dataFuture.then((data) {
      checkedStatus = List.filled(data.length, false);
    });
  }

  @override
  Widget build(BuildContext context) {
    super.build(context);
    return Scaffold(
      body: FutureBuilder<List<List<Kana>>>(
        future: _dataFuture,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Error: ${snapshot.error}'));
          } else if (snapshot.hasData) {
            var floatingActionButton;
            if (shouldShowButton(snapshot.data!)) {
              floatingActionButton = FloatingActionButton(
                heroTag: 'hiraganaFAB',
                onPressed: () {
                  Set<int> selectedIndexes = Set<int>();
                  for (int i = 0; i < checkedStatus.length; i++) {
                    if (checkedStatus[i]) selectedIndexes.add(i);
                  }

                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => TestSettingsPage(
                        selectedIndexes: selectedIndexes,
                        dataLoader: _dataLoader,
                      ),
                    ),
                  );
                },
```

```
              child: Icon(Icons.navigate_next),
            );
          }
          return Scaffold(
            floatingActionButton: floatingActionButton,
            body: ListView.builder(
              itemCount: snapshot.data!.length,
              itemBuilder: (context, index) {
                return CheckboxListTile(
                  title: buildKanaGroup(snapshot.data![index]),
                  value: checkedStatus[index],
                  onChanged: (bool? value) {
                    setState(() {
                      checkedStatus[index] = value!;
                    });
                  },
                  controlAffinity: ListTileControlAffinity.leading,
                );
              },
            ),
          );
        } else {
          return Center(child: Text('No data available'));
        }
      },
    ),
  );
}

bool shouldShowButton(List<List<Kana>> data) {
  int countLessThanFour = 0;
  int totalSelected = 0;

  for (int i = 0; i < checkedStatus.length; i++) {
    if (checkedStatus[i]) {
      int numChars = data[i].where((kana) => kana.kana.isNotEmpty).length;
      totalSelected++;
      if (numChars < 4) {
        countLessThanFour++;
      }
    }
  }
  bool shouldShow =
      (totalSelected - countLessThanFour > 0) || (countLessThanFour >= 2);

  return shouldShow;
}

Widget buildKanaGroup(List<Kana> kanaGroup) {
  return Container(
    padding: EdgeInsets.symmetric(vertical: 10, horizontal: 20),
```

```
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: kanaGroup
                .map((kana) => Text(kana.kana,
                    style:
                        TextStyle(fontSize: 24, fontWeight: FontWeight.bold)))
                .toList(),
          ),
          SizedBox(height: 5),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: kanaGroup
                .map((kana) => Text(kana.roumaji,
                    style: TextStyle(fontSize: 18, color: Colors.grey[600])))
                .toList(),
          )
        ],
      ),
    );
  }
}
```

Class Definitions

- HiraganaPage: A StatefulWidget which creates its state from _HiraganaPageState.
- _HiraganaPageState: Manages the state of the HiraganaPage, including data loading and user selections. It implements AutomaticKeepAliveClientMixin to keep the state alive when switching tabs, ensuring that the state isn't disposed of when the user navigates away.

Data Handling

- HiraganaDataLoader: Responsible for loading Hiragana character data, from a local json file.
- _dataFuture: A Future that holds the asynchronously loaded data from _dataLoader.
- checkedStatus: A list that tracks which items (groups of Kana characters) are selected by the user through checkboxes.

Lifecycle Methods

initState(): Initializes the _dataFuture and sets up checkedStatus based on the loaded data.

UI Build Process

- The build method uses FutureBuilder to handle the asynchronous nature of the data loading:
    - ConnectionState.waiting: Shows a loading spinner while data is being loaded.
    - hasError: Displays an error message if the data fails to load.
    - hasData: Renders the data as a list of checkbox list tiles, allowing users to select different groups of Kana characters.
- A floating action button is conditionally rendered if the method shouldShowButton returns true, indicating that the criteria for showing the button (based on selected items and character counts) are met.

Navigation

The floating action button uses Navigator.push to navigate to the TestSettingsPage when pressed, passing along the indexes of the selected Kana groups and the _dataLoader for further use.

Helper Methods

- shouldShowButton: Determines whether the floating action button should be shown based on the number of selected items and their character counts.
- buildKanaGroup: Constructs a widget that displays a group of Kana characters, both in Hiragana script and their Romaji (Latin script).

The implemented interface can be seen in Figure 18.

Figure 18 Hiragana Page.

7.2.2.4 *Katakana Page:*

```
class KatakanaPage extends StatefulWidget {
  @override
  _KatakanaPageState createState() => _KatakanaPageState();
}

class _KatakanaPageState extends State<KatakanaPage>
    with AutomaticKeepAliveClientMixin {
  final KatakanaDataLoader _dataLoader = KatakanaDataLoader();
  late Future<List<List<Kana>>> _dataFuture;
  late List<bool> checkedStatus;

  @override
  bool get wantKeepAlive => true;

  @override
```

```
void initState() {
  super.initState();
  _dataFuture = _dataLoader.loadKanaData();
  _dataFuture.then((data) {
    checkedStatus = List.filled(data.length, false);
  });
}

@override
Widget build(BuildContext context) {
  super.build(context);
  return Scaffold(
    body: FutureBuilder<List<List<Kana>>>(
      future: _dataFuture,
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return Center(child: CircularProgressIndicator());
        } else if (snapshot.hasError) {
          return Center(child: Text('Error: ${snapshot.error}'));
        } else if (snapshot.hasData) {
          var floatingActionButton;
          if (shouldShowButton(snapshot.data!)) {
            floatingActionButton = FloatingActionButton(
              heroTag: 'katakanaFAB',
              onPressed: () {
                Set<int> selectedIndexes = Set<int>();
                for (int i = 0; i < checkedStatus.length; i++) {
                  if (checkedStatus[i]) selectedIndexes.add(i);
                }

                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => TestSettingsPage(
                      selectedIndexes: selectedIndexes,
                      dataLoader: _dataLoader,
                    ),
                  ),
                );
              },
              child: Icon(Icons.navigate_next),
            );
          }
          return Scaffold(
            floatingActionButton: floatingActionButton,
            body: ListView.builder(
              itemCount: snapshot.data!.length,
              itemBuilder: (context, index) {
                return CheckboxListTile(
                  title: buildKanaGroup(snapshot.data![index]),
                  value: checkedStatus[index],
```

```
                  onChanged: (bool? value) {
                    setState(() {
                      checkedStatus[index] = value!;
                    });
                  },
                  controlAffinity: ListTileControlAffinity.leading,
                );
              },
            ),
          );
        } else {
          return Center(child: Text('No data available'));
        }
      },
    ),
  );
}

bool shouldShowButton(List<List<Kana>> data) {
  int countLessThanFour = 0;
  int totalSelected = 0;

  for (int i = 0; i < checkedStatus.length; i++) {
    if (checkedStatus[i]) {
      int numChars = data[i].where((kana) => kana.kana.isNotEmpty).length;
      totalSelected++;
      if (numChars < 4) {
        countLessThanFour++;
      }
    }
  }
  bool shouldShow =
      (totalSelected - countLessThanFour > 0) || (countLessThanFour >= 2);

  return shouldShow;
}

Widget buildKanaGroup(List<Kana> kanaGroup) {
  return Container(
    padding: EdgeInsets.symmetric(vertical: 10, horizontal: 20),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: kanaGroup
              .map((kana) => Text(kana.kana,
                  style:
                      TextStyle(fontSize: 24, fontWeight: FontWeight.bold)))
              .toList(),
        ),
```

```
          SizedBox(height: 5),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: kanaGroup
                .map((kana) => Text(kana.roumaji,
                    style: TextStyle(fontSize: 18, color: Colors.grey[600])))
                .toList(),
          )
        ],
      ),
    );
  }
}
```

The Katakana page, has the same logic and structure as the Hiragana page, except for the sub-characters.

The implemented interface can be seen in Figure 19.

Figure 19 Katakana Page.

7.2.2.5 *Test Setting Page*

```
class TestSettingsPage extends StatefulWidget {
  final Set<int> selectedIndexes;
  final KanaDataLoader dataLoader;

  const TestSettingsPage(
      {Key? key, required this.selectedIndexes, required this.dataLoader})
      : super(key: key);

  @override
  _TestSettingsPageState createState() => _TestSettingsPageState();
}

class _TestSettingsPageState extends State<TestSettingsPage> {
  String testType = 'card'; // card or write
  String direction = 'kanaToRomanji'; // kanaToRomanji or romanjiToKana
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Test Settings'),
    ),
    body: Center(
      child: SingleChildScrollView(
        child: ConstrainedBox(
          constraints: BoxConstraints(maxWidth: 600),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Padding(
                padding: const EdgeInsets.all(8.0),
                child: Text('Select Direction:',
                    style: Theme.of(context).textTheme.headline6),
              ),
              ToggleButtons(
                children: <Widget>[
                  Padding(
                    padding: const EdgeInsets.symmetric(horizontal: 16),
                    child: Text('Kana to Romanji'),
                  ),
                  Padding(
                    padding: const EdgeInsets.symmetric(horizontal: 16),
                    child: Text('Romanji to Kana'),
                  ),
                ],
                isSelected: [
                  direction == 'kanaToRomanji',
                  direction == 'romanjiToKana'
                ],
                onPressed: (int index) {
                  setState(() {
                    direction =
                        index == 0 ? 'kanaToRomanji' : 'romanjiToKana';
                  });
                },
                color: Colors.grey,
                selectedColor: Colors.white,
                fillColor: Theme.of(context).primaryColor.withOpacity(0.7),
                borderColor: Theme.of(context).primaryColor,
                selectedBorderColor: Theme.of(context).primaryColor,
                borderRadius: BorderRadius.circular(8),
                borderWidth: 2,
                constraints: BoxConstraints(minHeight: 40),
              ),
              Padding(
```

```
                padding: const EdgeInsets.all(8.0),
                child: Text('Select Test Type:',
                    style: Theme.of(context).textTheme.headline6),
              ),
              ToggleButtons(
                children: <Widget>[
                  Padding(
                    padding: const EdgeInsets.symmetric(horizontal: 16),
                    child: Text('Card Test'),
                  ),
                  Padding(
                    padding: const EdgeInsets.symmetric(horizontal: 16),
                    child: Text('Write Test'),
                  ),
                ],
                isSelected: [testType == 'card', testType == 'write'],
                onPressed: (int index) {
                  setState(() {
                    testType = index == 0 ? 'card' : 'write';
                  });
                },
                color: Colors.grey,
                selectedColor: Colors.white,
                fillColor: Theme.of(context).primaryColor.withOpacity(0.7),
                borderColor: Theme.of(context).primaryColor,
                selectedBorderColor: Theme.of(context).primaryColor,
                borderRadius: BorderRadius.circular(8),
                borderWidth: 2,
                constraints: BoxConstraints(minHeight: 40),
              ),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => TestPage(
                          selectedIndexes: widget.selectedIndexes,
                          dataLoader: widget.dataLoader,
                          testType: testType,
                          direction: direction,
                        ),
                      ));
                },
                child: Text('Start Test'),
              )
            ],
          ),
        ),
      ),
    ),
```

```
    );
  }
}
```

Class Definition

- TestSettingsPage: A StatefulWidget that receives a set of selected indexes and a data loader object through its constructor. These are used to configure and load the necessary data for the test.
- _TestSettingsPageState: Maintains the state of the TestSettingsPage, managing the test settings such as type and direction.

Constructor Parameters

- selectedIndexes: A set of integers representing the selected Kana groups for the test.
- dataLoader: An instance of KanaDataLoader used to load the specific Kana data for the test.

State Variables

- testType: A string that toggles between "card" and "write" to determine the type of test. "Card" imply a flashcard-style test, while "write" suggests a input-based test.
- direction: A string toggling between "kanaToRomanji" and "romanjiToKana" to set the direction of question syllable in the test.

Build Method

- Scaffold:
    - AppBar: Contains a simple title, "Test Settings".
    - Body: A centered SingleChildScrollView that allows vertical scrolling when the content exceeds the screen size, making it responsive to various device sizes.
        - ConstrainedBox: Constrains its child widget to a maximum width of 600 pixels, centering the content in larger screens.
        - Column: Arranges its children vertically. It contains the UI elements to select test settings:
        - Text: Displays labels such as "Select Direction:" and "Select Test Type:".

- ToggleButtons: Allows the user to select between options for direction and test type. Configured with visual properties like color, selectedColor, fillColor, etc., enhancing the UI consistency with the app's theme.
- ElevatedButton: Triggers the start of the test by navigating to TestPage with the current settings.

Functionality

- The ToggleButtons widgets let the user choose the test configuration. Pressing these updates the state variables (testType and direction) and visually reflects the selection.
- The ElevatedButton at the bottom initiates the test by passing the configured settings along with the selectedIndexes and dataLoader to TestPage, which presumably handles the test execution.

Navigation

Uses Navigator.push to transition to the TestPage, passing along necessary parameters to conduct the test based on user preferences.

The implemented interface can be seen in Figure 20.

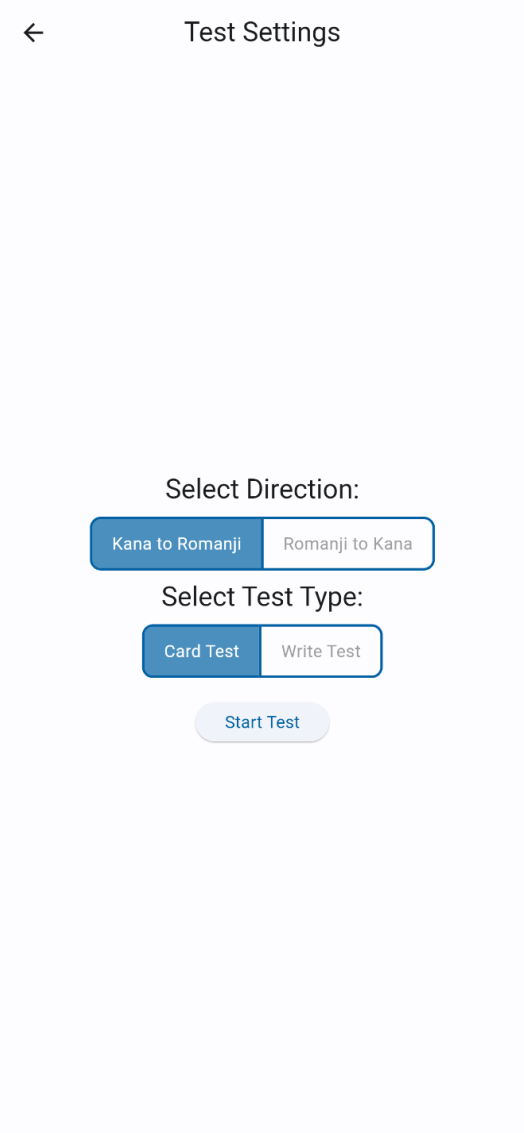

Figure 20 Test Setting Page

7.2.2.6 *Kanji Basic Page*

```
class KanjiBasicPage extends StatelessWidget {
  Future<List<Kanji>> loadKanjiData() async {
    final String jsonString = await rootBundle.loadString('data/kanji.json');
    List<dynamic> kanjiDataJson = jsonDecode(jsonString);
    List<Kanji> kanjiData =
        kanjiDataJson.map((item) => Kanji.fromJson(item)).toList();
    return kanjiData;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Kanji Basic Page')),
      body: FutureBuilder<List<Kanji>>(
        future: loadKanjiData(),
```

```
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.done) {
            if (snapshot.hasError) {
              debugPrint('Snapshot Error: ${snapshot.error}');
              return Center(
                  child: Text("Error loading kanji data: ${snapshot.error}"));
            }

            return ListView.builder(
              itemCount: snapshot.data!.length,
              itemBuilder: (context, index) {
                Kanji kanji = snapshot.data![index];
                return Padding(
                  padding: const EdgeInsets.symmetric(
                      horizontal: 8.0, vertical: 4.0),
                  child: Card(
                    child: Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: Row(
                        crossAxisAlignment: CrossAxisAlignment
                            .center,
                        children: <Widget>[
                          Container(
                            width: 80,
                            height: 80,
                            alignment: Alignment
                                .center,
                            decoration: BoxDecoration(
                              color: Colors.grey[200],
                              borderRadius: BorderRadius.circular(8),
                            ),
                            child: Text(
                              kanji.kanji ?? "N/A",
                              style: TextStyle(
                                  fontSize: 32, fontWeight: FontWeight.bold),
                            ),
                          ),
                          SizedBox(
                              width:
                                  16),
                          Expanded(
                            child: Column(
                              crossAxisAlignment: CrossAxisAlignment
                                  .start,
                              children: <Widget>[
                                Text(
                                    'Meaning: ${kanji.meaning ?? "Not availa-
ble"}',
                                    style: Theme.of(context)
                                        .textTheme
                                        .titleMedium),
```

```
                                Text(
                                    'Reading:\nOnyomi: ${kanji.onyomi}\nKunyomi:
${kanji.kunyomi}',
                                    style:
                                        Theme.of(context).textTheme.bodyMedium),
                                ],
                              ),
                            ),
                          ],
                        ),
                      ),
                    ),
                  );
                },
              );
            } else {
              return Center(child: CircularProgressIndicator());
            }
          },
        ),
      );
  }
}
```

KanjiBasicPage Class

This class is a StatelessWidget, meaning it doesn't hold any state changes internally after it's built. The kanji data loading is managed through a Future to handle asynchronous data fetching.

Methods

- loadKanjiData: This asynchronous method loads a JSON file from the app's resources, decodes it into a list of dynamic objects, and then maps these to a list of Kanji model instances. This operation is performed using:
    - rootBundle.loadString: To asynchronously load the JSON data from the filesystem.
    - jsonDecode: To convert the JSON string into a list of maps.
    - Kanji.fromJson: A factory method of the Kanji class to instantiate objects from the JSON data.

Widget Build Method

- Scaffold:

- AppBar: Contains a title, "Kanji Basic".
- Body: Implements a FutureBuilder that handles the future returned by loadKanjiData().
  - ConnectionState.done: Checks if the asynchronous operation is complete.
  - hasError: Displays an error message if there was an error loading or parsing the kanji data.
  - hasData: If data is present, it renders a ListView.builder that dynamically creates a list of kanji entries.
  - Each kanji is presented in a Card, which displays the kanji character prominently and provides its meaning and readings (Onyomi and Kunyomi).
  - The layout uses Row and Column to organize the kanji character and its details nicely.

Error Handling and Debugging

Errors in data fetching or parsing are handled by checking snapshot.hasError, and if true, logging the error and displaying an error message in the UI.

ListView.builder

- This widget is used to create a scrollable list of kanji cards. Each card contains:
- A container that visually highlights the kanji character.
- Text widgets that display the kanji's meaning and its readings.

The implemented interface can be seen in Figure 21.

Figure 21 Kanji Basic Page.

7.2.2.7 *Setting App Page*

```
class SettingAppPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    var appSettings = Provider.of<AppSettings>(context);
    return Scaffold(
      appBar: AppBar(title: Text('Setting App')),
      body: ListView(
        children: [
          SwitchListTile(
            title: Text('Enable Confetti Animation'),
            value: appSettings.enableConfetti,
            onChanged: (bool value) {
              appSettings.toggleConfetti();
            },
          ),
```

```
        ],
      ),
    );
  }
}
```

SettingAppPage Class

Extends StatelessWidget: This means the widget does not have mutable state internally. Any state changes are handled by the AppSettings model through the Provider.

Widget Build Method

- Scaffold:
    - AppBar: Displays a simple AppBar with the title "Setting App".
    - Body: Uses a ListView which can easily accommodate more settings in the future:
        - SwitchListTile:
        - Title: Labeled "Enable Confetti Animation", indicates the purpose of the switch.
        - value: The current state of the confetti animation setting, retrieved from appSettings.enableConfetti, where appSettings is an instance of the AppSettings model obtained from the Provider.
        - onChanged: A callback that gets triggered when the user toggles the switch. It calls appSettings.toggleConfetti(), which toggles the enableConfetti boolean value in the AppSettings model.

Provider and AppSettings

- The Provider.of<AppSettings>(context) call fetches the AppSettings instance from the nearest ancestor Provider in the widget tree, which manages the state of settings like the confetti animation.
- This approach ensures that any changes to the settings are centrally managed and can trigger UI updates where necessary. The Provider package helps manage state across different parts of the app more cleanly and efficiently, reducing the need for callbacks and manually passing data down the widget tree.

Functionality

The SwitchListTile widget provides a toggle switch for the user to enable or disable the confetti animation feature. This interaction is simple and user-friendly, providing direct feedback on the state of the setting through the UI.
The implemented interface can be seen in Figure 22.

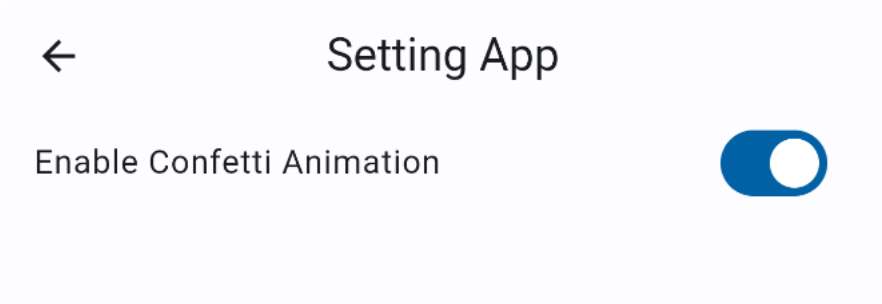

Figure 22 Setting App Page.

7.2.3 Models Structure

7.2.3.1 *App Settings*

```
class AppSettings extends ChangeNotifier {
  bool enableConfetti = true;

  void toggleConfetti() {
    enableConfetti = !enableConfetti;
    notifyListeners();
  }
}
```

AppSettings Class

- enableConfetti (Boolean Variable): A boolean property that determines whether confetti animations are enabled or disabled in the app. It is initially set to true, indicating that confetti animations are enabled by default.
- toggleConfetti (Method):
    - This method toggles the state of the enableConfetti boolean. It achieves this by setting enableConfetti to the opposite of its current value.
    - notifyListeners(): After changing the value of enableConfetti, this method is called. It's a part of the ChangeNotifier mixin and notifies any listeners that a change has occurred. This is crucial in Flutter for triggering UI updates when the underlying model changes.

7.2.3.2 *Kana Option*

```
class KanaOption {
  Kana kana;
  bool isEnabled;
  Color? color;

  KanaOption({
    required this.kana,
    this.isEnabled = true,
    this.color,
  });
}
```

KanaOption Class

Properties:

- kana (Kana): An instance of the Kana class. This property stores the specific Kana character associated with this option. The Kana class is not defined in this snippet but would contain attributes related to the Japanese Kana characters, such as the character itself, its pronunciation, and possibly other related data.
- isEnabled (bool): A boolean that indicates whether the Kana option is enabled or not in the application's context. It defaults to true, meaning the Kana character is enabled by default when an instance of KanaOption is created.
- color (Color?): An optional Color property that can be used to customize the appearance of the Kana character in the UI, such as changing its color when displayed. The property is nullable (Color?), indicating that it does not need to be set upon instantiation of the class.

Constructor

KanaOption Constructor:

The constructor is defined with named parameters, requiring an instance of Kana and allowing optional overrides for isEnabled and color.

Parameters:

- kana: Mandatory parameter. Every instance of KanaOption must be initialized with a Kana object.
- isEnabled: Optional parameter with a default value of true.
- color: Optional parameter, nullable, with no default value. This allows for customization of the Kana display without enforcing a default color.

7.2.3.3 *Kana*

```dart
part 'kana.g.dart';

@JsonSerializable()
class Kana {
  String kana;
  String roumaji;
  String type;

  Kana({required this.kana, required this.roumaji, required this.type});

  factory Kana.fromJson(Map<String, dynamic> json) => _$KanaFromJson(json);
  Map<String, dynamic> toJson() => _$KanaToJson(this);
}
```

Part directive

part 'kana.g.dart': This directive links this file with a part file kana.g.dart that is generated by running the build runner. This generated file contains part of the code necessary to serialize and deserialize the data, specifically the implementation of the fromJson and toJson methods.

JsonSerializable Annotation

@JsonSerializable(): This annotation from json_annotation marks the Kana class as a target for JSON serialization code generation. It tells the code generator to build the serialization functionality automatically into the part file.

Kana Class

Properties:

- kana (String): Holds the Kana character. In Japanese, Kana refers to characters in both the Hiragana and Katakana scripts.
- roumaji (String): Contains the Romaji (Latin script transcription) representation of the Kana character.
- type (String): A string to categorize the Kana, possibly distinguishing between "Hiragana" and "Katakana" or other types.

Constructor:

The constructor for Kana is defined with named parameters, each marked as required. This ensures that an instance of Kana cannot be created without these essential values.

Factory Constructor fromJson:

A factory constructor named fromJson is defined, which uses the function _$KanaFromJson. This function is expected to be in the generated file and is used to create a new instance of Kana from a JSON map.

Method toJson:

The toJson method returns a map that represents the current state of the Kana instance. It uses the _$KanaToJson function, which is also generated by the code generator.

7.2.3.4 *Kana.g.dart. Part of Kana.dart*

```dart
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'kana.dart';

// **************************************************************************
// JsonSerializableGenerator
// **************************************************************************

Kana _$KanaFromJson(Map<String, dynamic> json) => Kana(
      kana: json['kana'] as String,
      roumaji: json['roumaji'] as String,
      type: json['type'] as String,
    );

Map<String, dynamic> _$KanaToJson(Kana instance) => <String, dynamic>{
      'kana': instance.kana,
      'roumaji': instance.roumaji,
      'type': instance.type,
    };
```

Part Directive

part of 'kana.dart';: This directive specifies that the code belongs to a part file linked to kana.dart, meaning that it is a part of the same library and shares its scope.

JsonSerializable Generator Comment

Comment: The comment // GENERATED CODE - DO NOT MODIFY BY HAND and the subsequent comment block indicate that the content is generated automatically by the Json-SerializableGenerator. It is standard practice not to manually modify generated code, as any changes will be overwritten when the code is regenerated.

Functions Generated

_$KanaFromJson(Map<String, dynamic> json):

- A factory constructor function that creates a new instance of the Kana class from a JSON object.

- It reads the kana, roumaji, and type fields from the JSON map, explicitly casting them to the String type to match the properties of the Kana class.
- This function is called during JSON deserialization, converting JSON data retrieved from local storage into usable Kana objects.

_$KanaToJson(Kana instance):

- A function that converts a Kana instance back into a JSON map.
- It takes a Kana object and constructs a map containing its properties (kana, roumaji, type), mapping them to the corresponding JSON keys.

7.2.3.5 *Kanji*

```dart
part 'kanji.g.dart';

@JsonSerializable()
class Kanji {
  final int number;
  final String? kanji;
  @JsonKey(defaultValue: '')
  final String onyomi;
  @JsonKey(defaultValue: '')
  final String kunyomi;
  final String? meaning;

  Kanji({
    required this.number,
    this.kanji,
    this.onyomi = '',
    this.kunyomi = '',
    this.meaning,
  });

  factory Kanji.fromJson(Map<String, dynamic> json) => _$KanjiFromJson(json);
  Map<String, dynamic> toJson() => _$KanjiToJson(this);
}
```

Part Directive

part 'kanji.g.dart'; This line indicates that the kanji.g.dart file is a part of this Dart file. The kanji.g.dart file to contain the generated code for JSON serialization and deserialization functions.

Annotations

- @JsonSerializable(): This annotation on the Kanji class enables the automatic generation of serialization and deserialization logic, which will be part of the kanji.g.dart file. It simplifies the conversion process between a JSON map and a Kanji instance.
- @JsonKey: Used to specify configuration on how individual fields are handled during serialization and deserialization. For instance, default values for fields can be defined here.

Kanji Class Definition

Properties:

- final int number; - An identifier for the Kanji, representing an order or an index.
- final String? kanji; - The actual Kanji character. It's nullable, indicating that the Kanji character might not always be provided.
- @JsonKey(defaultValue: '') final String onyomi; - Represents the Onyomi reading (Chinese reading) of the Kanji. The defaultValue attribute ensures that if onyomi is not present in the JSON, it defaults to an empty string.
- @JsonKey(defaultValue: '') final String kunyomi; - Represents the Kunyomi reading (Japanese reading) of the Kanji. Similarly, it defaults to an empty string if not provided.
- final String? meaning; - The meaning of the Kanji character. It's nullable.

Constructor

Kanji constructor: Initializes a Kanji object with required and optional parameters. Required parameters must be passed during object creation, while optional ones have default values or are nullable.

Factory Constructor and Serialization Methods

- factory Kanji.fromJson(Map<String, dynamic> json) => _$KanjiFromJson(json); - A factory constructor that creates an instance of Kanji from a JSON map. The actual function _$KanjiFromJson is generated by the code generation package.
- Map<String, dynamic> toJson() => _$KanjiToJson(this); - Serializes the Kanji instance into a JSON map. The function _$KanjiToJson handles the serialization process and is also generated.

7.2.3.6 *Kanji.g.dart. Part of Kanji.dart*

```
// GENERATED CODE - DO NOT MODIFY BY HAND
```

```dart
part of 'kanji.dart';

// **************************************************************************
// JsonSerializableGenerator
// **************************************************************************

Kanji _$KanjiFromJson(Map<String, dynamic> json) => Kanji(
      number: (json['number'] as num).toInt(),
      kanji: json['kanji'] as String?,
      onyomi: json['onyomi'] as String? ?? '',
      kunyomi: json['kunyomi'] as String? ?? '',
      meaning: json['meaning'] as String?,
    );

Map<String, dynamic> _$KanjiToJson(Kanji instance) => <String, dynamic>{
      'number': instance.number,
      'kanji': instance.kanji,
      'onyomi': instance.onyomi,
      'kunyomi': instance.kunyomi,
      'meaning': instance.meaning,
    };
```

Part Directive

part of 'kanji.dart';: This line indicates that the provided code is part of the kanji.dart file, linking it back to the main library that defines the Kanji class.

Generated Code Comments

// GENERATED CODE - DO NOT MODIFY BY HAND: This comment warns developers not to edit the file manually because any changes will be overwritten when the code is re-generated. This ensures the integrity and functionality of the automatically generated serialization code.

JsonSerializableGenerator

Comment Block: Describes the tool used (JsonSerializableGenerator) and marks this section of the code as part of the serialization infrastructure.

Functions Generated

_$KanjiFromJson(Map<String, dynamic> json):

- A factory constructor function that creates a Kanji instance from a JSON object.
- Fields from the JSON object are parsed and appropriately cast to match the data types defined in the Kanji class.
- Conversion Details:
- The number field is parsed as a number and converted to an integer.

- The kanji, onyomi, kunyomi, and meaning fields are nullable strings. The code checks for null values and provides default values for onyomi and kunyomi if they are absent (" indicates an empty string if the field is not present).

_$KanjiToJson(Kanji instance):

- Converts a Kanji instance into a JSON map.
- Each field of the Kanji class is transformed into a key-value pair in the map. This function ensures that all relevant properties of a Kanji instance are included in the resulting JSON object, ready for serialization.

Structure:

It maps each property of Kanji to JSON keys. This structure directly corresponds to the fields defined in the Kanji class, ensuring that all information is retained in the serialized form.

7.2.3.7 *Kana Data Loader*

```
import 'dart:convert';
import 'package:flutter/services.dart' show rootBundle;
import '../models/kana.dart';

class KanaDataLoader {
  final String jsonFilePath;

  KanaDataLoader({required this.jsonFilePath});

  Future<List<List<Kana>>> loadKanaData() async {
    final String jsonString = await rootBundle.loadString(jsonFilePath);
    List<dynamic> kanaDataJson = jsonDecode(jsonString);
    List<Kana> kanaData =
        kanaDataJson.map((item) => Kana.fromJson(item)).toList();
    return _createKanaGroups(kanaData);
  }

  List<List<Kana>> _createKanaGroups(List<Kana> kanaData) {
    List<List<Kana>> groups = [
      kanaData.sublist(0, 5), // a, i, u, e, o
      kanaData.sublist(5, 10), // ka, ki, ku, ke, ko
      kanaData.sublist(10, 15), // sa, shi, su, se, so
      kanaData.sublist(15, 20), // ta, chi, tsu, te, to
      kanaData.sublist(20, 25), // na, ni, nu, ne, no
      kanaData.sublist(25, 30), // ha, hi, fu, he, ho
      kanaData.sublist(30, 35), // ma, mi, mu, me, mo
      kanaData.sublist(35, 38), // ya, yu, yo
      kanaData.sublist(38, 43), // ra, ri, ru, re, ro
      [
        kanaData[43],
```

```
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[45],
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[44]
    ], // wa, wo, n
    kanaData.sublist(46, 51), // ga, gi, gu, ge, go
    kanaData.sublist(51, 56), // za, ji, zu, ze, zo
    kanaData.sublist(56, 61), // da, dji, dzu, de, do
    kanaData.sublist(61, 66), // ba, bi, bu, be, bo
    kanaData.sublist(66, 71), // pa, pi, pu, pe, po
    [
      kanaData[72], // kya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[73], // kyu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[74] // kyo
    ],
    [
      kanaData[75], // sha
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[76], // shu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[77] // sho
    ],
    [
      kanaData[78], // cha
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[79], // chu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[80] // cho
    ],
    [
      kanaData[81], // nya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[82], // nyu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[83] // nyo
    ],
    [
      kanaData[84], // hya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[85], // hyu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[86] // hyo
    ],
    [
      kanaData[87], // mya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[88], // myu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[89] // myo
```

```
    ],
    [
      kanaData[90], // rya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[91], // ryu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[92] // ryo
    ],
    [
      kanaData[93], // gya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[94], // gyu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[95] // gyo
    ],
    [
      kanaData[96], // ja
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[97], // ju
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[98] // jo
    ],
    [
      kanaData[99], // bya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[100], // byu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[101] // byo
    ],
    [
      kanaData[102], // pya
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[103], // pyu
      Kana(kana: '', roumaji: '', type: ''),
      kanaData[104] // pyo
    ]
  ];

  List<int> specialRows = [35];

  for (var groupIndex = 0; groupIndex < groups.length; groupIndex++) {
    List<Kana> group = groups[groupIndex];
    if (specialRows.contains(groupIndex * 5)) {
      List<Kana> newGroup =
          List.filled(5, Kana(kana: '', roumaji: '', type: ''));
      for (var item in group) {
        if (item.roumaji.startsWith('y')) {
          int index = 'aiueo'.indexOf(item.roumaji[1]);
          newGroup[index] = item;
        }
      }
```

```
        groups[groupIndex] = newGroup;
      } else {
        while (group.length < 5) {
          group.add(Kana(kana: '', roumaji: '', type: ''));
        }
      }
    }
    return groups;
  }
}

class HiraganaDataLoader extends KanaDataLoader {
  HiraganaDataLoader() : super(jsonFilePath: 'data/hiragana.json');
}

class KatakanaDataLoader extends KanaDataLoader {
  KatakanaDataLoader() : super(jsonFilePath: 'data/katakana.json');
}
```

KanaDataLoader Class

Properties:

- jsonFilePath: A string that stores the path to the JSON file from which Kana data should be loaded.

Constructor:

Requires a jsonFilePath to be provided, ensuring each instance of KanaDataLoader is associated with a specific JSON data file.

loadKanaData Method:

- Asynchronously loads Kana data from the specified JSON file.
- Uses rootBundle.loadString to read the JSON file content as a string.
- Parses the JSON string into a dynamic list using jsonDecode.
- Converts this list into a list of Kana objects by mapping each item through the Kana.fromJson factory method.
- Calls _createKanaGroups to organize these Kana objects into logical groups.

_createKanaGroups Method:

- Takes a list of Kana objects and organizes them into groups, which can be used to structure learning sessions or UI displays.
- The grouping logic is hardcoded to segment the Kana into traditional groupings (like vowels, k-group, s-group, etc.) and special combinations (like kya, kyu, kyo).

- It also handles special cases where some Kana characters need to be replaced or re-positioned, especially for syllables that involve diacritics or are combinations.

HiraganaDataLoader and KatakanaDataLoader Classes

- These classes are specific implementations of KanaDataLoader tailored for loading Hiragana and Katakana data, respectively.

Constructors:

They initialize the superclass KanaDataLoader with predefined paths to their respective

7.3 Interactive Learning Modules

Let's move on to the final point. Implementation of tests for learning the Japanese alphabet is a complex file that includes the logic of test generation, checks and calculation of results.

Test Page:

```
import 'package:flutter/material.dart';
import 'dart:math';
import 'dart:async';
import 'dart:collection';
import '../loaders/kana_data_loader.dart';
import '../models/kana.dart';
import '../models/kana_option.dart';
import 'package:confetti/confetti.dart';
import 'package:provider/provider.dart';
import '../models/app_settings.dart';

class TestPage extends StatefulWidget {
  final Set<int> selectedIndexes;
  final KanaDataLoader dataLoader;
  final String testType;
  final String direction;

  const TestPage({
    super.key,
    required this.selectedIndexes,
    required this.dataLoader,
    required this.testType,
    required this.direction,
  });

  @override
  _TestPageState createState() => _TestPageState();
}

class _TestPageState extends State<TestPage> with TickerProviderStateMixin {
  late List<List<Kana>> testData;
```

```
late KanaOption correctAnswer;
late KanaOption questionSymbol;
List<KanaOption> currentTestOptions = [];
Queue<String> lastThreeSymbols = Queue<String>();

bool isLoading = true;
bool isAnswered = false;
bool correct = true;
int? shakeIndex;
int correctAnswersStreak = 0;

final TextEditingController _controller = TextEditingController();
late AnimationController _shakeController;
late Animation<Offset> _offsetAnimation;
late AnimationController _bounceController;
late Animation<double> _bounceAnimation;
late ConfettiController _confettiController;

@override
void initState() {
  super.initState();
  loadTestData();

  _confettiController =
      ConfettiController(duration: const Duration(milliseconds: 500));
  _bounceController = AnimationController(
      duration: const Duration(milliseconds: 500), vsync: this);
  _shakeController = AnimationController(
      duration: const Duration(milliseconds: 300), vsync: this);

  _bounceAnimation =
      Tween<double>(begin: 0.0, end: -50.0).animate(CurvedAnimation(
    parent: _bounceController,
    curve: Curves.easeInOut,
  ));
  _offsetAnimation = Tween<Offset>(
          begin: const Offset(-0.02, 0.0), end: const Offset(0.02, 0.0))
      .animate(CurvedAnimation(
    parent: _shakeController,
    curve: Curves.elasticInOut,
  ));

  _bounceController.addStatusListener((status) {
    if (status == AnimationStatus.completed) {
      _bounceController.reverse();
    }
  });
  _shakeController.addStatusListener((status) {
    if (status == AnimationStatus.completed) {
      _shakeController.reverse();
    }
```

```
  });
}

@override
void dispose() {
  _shakeController.dispose();
  _bounceController.dispose();
  _confettiController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Test'),
      actions: <Widget>[
        Center(
          child: Padding(
            padding: const EdgeInsets.only(right: 16.0),
            child: Text('Streak: $correctAnswersStreak',
                style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
          ),
        ),
      ],
    ),
    body: Stack(
      children: [
        Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: [
              Expanded(
                child: AnimatedBuilder(
                  animation: _bounceAnimation,
                  builder: (context, child) {
                    return Transform.translate(
                      offset: Offset(0, _bounceAnimation.value),
                      child: child,
                    );
                  },
                  child: Container(
                    child: Center(
                      child: Text(
                        widget.direction == 'kanaToRomanji'
                            ? questionSymbol.kana.kana
                            : questionSymbol.kana.roumaji,
                        style: const TextStyle(
                            fontSize: 120, fontWeight: FontWeight.bold),
                      ),
```

```
                    ),
                  ),
                ),
              ),
              widget.testType == 'card'
                  ? buildOptionGrid()
                  : buildInputField(),
            ],
          ),
        ),
        Align(
          alignment: Alignment.topCenter,
          child: ConfettiWidget(
            confettiController: _confettiController,
            blastDirectionality: BlastDirectionality.explosive,
            shouldLoop: false,
            colors: const [
              Colors.green,
              Colors.blue,
              Colors.pink,
              Colors.orange,
              Colors.purple
            ],
            numberOfParticles: 20,
            gravity: 0.3,
          ),
        ),
      ],
    ),
  );
}

Widget buildOptionGrid() {
  return GridView.builder(
    shrinkWrap: true,
    physics: const NeverScrollableScrollPhysics(),
    gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 2,
      childAspectRatio: 1.5,
      crossAxisSpacing: 15,
      mainAxisSpacing: 15,
    ),
    itemCount: currentTestOptions.length,
    itemBuilder: (BuildContext context, int index) {
      var option = currentTestOptions[index];
      bool isCorrectOption = widget.direction == 'kanaToRomanji'
          ? option.kana.roumaji == correctAnswer.kana.roumaji
          : option.kana.kana == correctAnswer.kana.kana;
      return AnimatedBuilder(
        animation: _shakeController,
        builder: (context, child) {
```

```
          final offset = shakeIndex == index
              ? sin(_shakeController.value * pi * 1.2) * 7
              : 0.0;
          return Transform.translate(
            offset: Offset(offset, 0),
            child: child,
          );
        },
        child: ElevatedButton(
          onPressed: !isAnswered && option.isEnabled
              ? () => checkAnswer(
                  index,
                  widget.direction == 'kanaToRomanji'
                      ? option.kana.roumaji
                      : option.kana.kana)
              : null,
          style: ElevatedButton.styleFrom(
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(10),
            ),
            padding: const EdgeInsets.symmetric(vertical: 16.0),
            disabledBackgroundColor:
                isCorrectOption ? Colors.green : Colors.red.withOpacity(0.5),
            foregroundColor: Colors.black,
          ),
          child: Text(
            widget.direction == 'kanaToRomanji'
                ? option.kana.roumaji
                : option.kana.kana,
            style: const TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),
        ),
      );
    },
  );
}

Widget buildInputField() {
  return SlideTransition(
    position: _offsetAnimation,
    child: TextField(
      controller: _controller,
      decoration: InputDecoration(
        labelText: 'Input your answer',
        border: OutlineInputBorder(
          borderSide: BorderSide(
              color: !correct ? Colors.red : Colors.grey, width: 1.0),
        ),
        focusedBorder: OutlineInputBorder(
          borderSide: BorderSide(
              color: !correct ? Colors.red : Colors.blue, width: 2.0),
```

```
          ),
        ),
        onSubmitted: (submittedAnswer) =>
            checkAnswerForTextField(submittedAnswer),
      ),
    );
  }

  void checkAnswerForTextField(String submittedAnswer) {
    String correctAnswerText = widget.direction == 'kanaToRomanji'
        ? correctAnswer.kana.roumaji.toLowerCase()
        : correctAnswer.kana.kana.toLowerCase();

    submittedAnswer = submittedAnswer.trim().toLowerCase();

    bool isCorrectAnswer =
        submittedAnswer == correctAnswerText && widget.testType == 'write';

    if (isCorrectAnswer) {
      _bounceController.forward(from: 0.0);
      _playConfetti();
      setState(() {
        isAnswered = true;
        correct = true;
        _controller.clear();
      });
      Future.delayed(const Duration(seconds: 1), () {
        if (mounted) {
          generateTest();
          setState(() {
            isAnswered = false;
          });
        }
      });
    } else {
      setState(() {
        correct = false;
      });
      _shakeController.forward(from: 0.0);
    }
  }

  void loadTestData() async {
    var allData = await widget.dataLoader.loadKanaData();
    testData = widget.selectedIndexes.map((index) => allData[index]).toList();
    isLoading = false;
    generateTest();
    setState(() {});
  }

  void _playConfetti() {
```

```
    var settings = Provider.of<AppSettings>(context, listen: false);
    if (settings.enableConfetti) {
      _confettiController.play();
    }
  }

  void generateTest() {
    final random = Random();
    List<Kana> allKanas = testData.expand((group) => group).toList();

    List<Kana> filteredKanas = allKanas
        .where((kana) =>
            !lastThreeSymbols.contains(widget.direction == 'kanaToRomanji'
                ? kana.roumaji
                : kana.kana) &&
            kana.roumaji.isNotEmpty &&
            kana.kana.isNotEmpty)
        .toList();

    if (filteredKanas.isEmpty) {
      lastThreeSymbols.clear();
      filteredKanas = allKanas
          .where((kana) => kana.roumaji.isNotEmpty && kana.kana.isNotEmpty)
          .toList();
    }

    if (filteredKanas.isEmpty) {
      throw Exception('No valid kanas available to generate a test.');
    }

    Kana questionKana = filteredKanas[random.nextInt(filteredKanas.length)];
    correctAnswer = KanaOption(kana: questionKana);
    questionSymbol = correctAnswer;

    updateLastSymbols(widget.direction == 'kanaToRomanji'
        ? questionKana.roumaji
        : questionKana.kana);

    Set<String> seen = {
      widget.direction == 'kanaToRomanji'
          ? questionKana.roumaji
          : questionKana.kana
    };
    List<String> options = [seen.first];

    List<Kana> optionCandidates = allKanas
        .where((kana) =>
            !seen.contains(widget.direction == 'kanaToRomanji'
                ? kana.roumaji
                : kana.kana) &&
            kana.roumaji.isNotEmpty &&
```

```
          kana.kana.isNotEmpty)
      .toList();

  while (options.length < 4 && optionCandidates.isNotEmpty) {
    Kana option = optionCandidates[random.nextInt(optionCandidates.length)];
    String optionText =
        widget.direction == 'kanaToRomanji' ? option.roumaji : option.kana;

    if (!seen.contains(optionText) && optionText.isNotEmpty) {
      options.add(optionText);
      seen.add(optionText);
    }
  }

  if (options.length < 4) {
    throw Exception('Not enough valid options to form a test.');
  }

  options.shuffle();
  currentTestOptions = options
      .map((optionText) => KanaOption(
          kana: widget.direction == 'kanaToRomanji'
              ? Kana(kana: '', roumaji: optionText, type: '')
              : Kana(kana: optionText, roumaji: optionText, type: '')))
      .toList();
}

void updateLastSymbols(String newSymbol) {
  if (lastThreeSymbols.length >= 3) {
    lastThreeSymbols.removeFirst();
  }
  lastThreeSymbols.addLast(newSymbol);
}

void checkAnswer(int index, String selectedOption) {
  String correctAnswerText = widget.direction == 'kanaToRomanji'
      ? correctAnswer.kana.roumaji
      : correctAnswer.kana.kana;

  if (selectedOption == correctAnswerText) {
    _bounceController.forward(from: 0.0);
    _playConfetti();
    setState(() {
      isAnswered = true;
      correct = true;
      correctAnswersStreak++;
      for (var option in currentTestOptions) {
        option.isEnabled = false;
      }
    });
    Future.delayed(const Duration(milliseconds: 1000), () {
```

```
        if (mounted) {
          generateTest();
          setState(() {
            isAnswered = false;
            shakeIndex = null;
            for (var option in currentTestOptions) {
              option.isEnabled = true;
            }
          });
        }
      });
    } else {
      setState(() {
        shakeIndex = index;
        currentTestOptions[index].isEnabled = false;
        correct = false;
        correctAnswersStreak = 0;
      });
      _shakeController.forward(from: 0.0);
    }
  }
}
```

Class Definition:

StatefulWidget Properties: Includes properties for selected Kana indices, data loader, test type, and test direction, all required to configure and render the test page appropriately.

State Management in _TestPageState

Animation Controllers:

- Confetti, Bounce, and Shake Animations: Set up to visually respond to user interactions—celebrating correct answers with confetti, providing bounce feedback on correct responses, and shaking for incorrect ones.

Initial Setup in initState():

- Loading Test Data: Asynchronously loads data using dataLoader based on selectedIndexes to prepare for the test.
- Animation Initialization: Configures animations for different UI effects based on user interactions.

Widget Building in build():

UI Composition: Uses a Stack to layer different UI components like the testing area and confetti effects. Depending on the test type (card or write), it dynamically adjusts the display to either show multiple-choice options or an input field for text answers.

Test Generation and Management

Avoiding Repetition with lastThreeSymbols Queue:

- Queue Management: Stores the last three symbols used in tests to prevent their immediate repetition. This is managed by adding new symbols to the queue and removing the oldest when updating.

Dynamic Test Generation in generateTest():

- Random Selection: Selects a random Kana for the question from a list that excludes recent symbols, ensuring a diverse range of characters in tests.
- Error Handling: Checks if enough valid Kana characters are available for testing, throwing an exception if the data set is insufficient.

Answer Checking and UI Updates:

- Interaction Handling: Depending on the user's response, updates UI elements like animations and text displays. Correct responses trigger positive animations and increase the streak count, while incorrect responses provide immediate visual feedback through shaking animation.

Disposal of Resources in dispose():

Ensures proper cleanup of animation controllers and other resources to avoid memory leaks. The implemented interface can be seen in Figure 23, Figure 24, Figure 25, Figure 26 and Figure 27.

8 USER MANUAL

Upon launching the application, users arrive at the Welcome page, depicted in Figure 17. This page presents three buttons: "Kana", "Kanji Basic", and "Settings". Clicking on the "Kana" button directs users to the Kana page, showcasing the hiragana and katakana alphabets in Figure 18 and Figure 19, respectively.

At the Kana page, users can select specific rows of alphabets to practice. Upon selecting at least one row, a "Next" button appears, leading to the Test Settings Page (Figure 20). This page allows users to customize their Kana test. Options include converting Kana to Romanji, where the question is displayed in the Japanese alphabet and answers are provided in the Latin alphabet, or Romanji to Kana, where the process is reversed.

Users can choose between a Card Test and a Write Test. In the Card Test, as shown in Figure 23, multiple-choice answers are available. Selecting the correct option triggers a celebratory animation featuring the jumping symbol and confetti. The correct answer turns the button green, incorrect ones turn red, and a counter tallies consecutive correct responses (Figure 25). An incorrect selection results in the button turning red and resets the correct answers counter, as seen in Figure 24.

In the Write Test, depicted in Figure 27, users type their answers without given options. Correct responses also trigger the bouncing symbol and confetti animation. Incorrect answers cause the input box's outline to turn red, and the consecutive correct answers counter resets, as shown in Figure 26.

Returning to the Welcome Page and navigating to the "Kanji Basic" page, users encounter the first 80 basic Kanji characters, complete with their English translations and pronunciations (Figure 21). Heading back to the Welcome Page and accessing the "Settings" page allows users to toggle the confetti animation on correct test answers. This settings interface is illustrated in Figure 22.

Figure 23 Test Card Page.

Figure 24 Test Card Page Incorrect Answer.

Figure 25 Test Card Page Correct Answer.

Figure 26 Test Write Test Incorrect Answer.

Figure 27 Test Write Test Correct Answer.

CONCLUSION

This thesis has meticulously examined the challenges and opportunities associated with the development of multiplatform educational applications, with a particular focus on language learning. A thorough review of contemporary application development frameworks has underscored the capabilities and potential of Flutter. This modern framework is adept at crafting high-quality, natively compiled applications across mobile, web, and desktop platforms from a singular codebase.

The practical component of this thesis centered on the development of a "Mobile Application for Learning Japanese Alphabet" designed to facilitate the mastery of the Japanese alphabet—including Hiragana, Katakana, and introductory Kanji—through interactive learning experiences. This application capitalizes on Flutter's extensive feature set, including its rich widget library and responsive framework, to develop a dynamic user interface that accommodates various learning styles and paces.

The development process was elaborately described, highlighting the critical considerations essential for crafting educational software that is both engaging and effective. This encompassed the design of intuitive navigation, the incorporation of interactive testing mechanisms, and the employment of multimedia elements to augment memorization and recall. Fundamental functionalities, such as the test generation algorithm, which guarantees a comprehensive and non-repetitive learning experience, and the integration of animations to provide feedback, were meticulously implemented to enhance user engagement and educational outcomes.

Overall, this thesis has successfully demonstrated the efficacy of cross-platform development, specifically through the use of Flutter, in addressing distinct educational challenges. The "Japanese Learning App" not only meets the essential requirement of providing a functional tool for mastering the Japanese alphabet but also illustrates the extensive potential of multiplatform applications to revolutionize educational methodologies and accessibility.

The outcomes of this project lay a robust foundation for future advancements, suggesting that subsequent enhancements could include the incorporation of adaptive learning technologies, integration with more extensive Kanji databases, and the application of artificial intelligence to customize learning trajectories for individual users. The adept implementation of these technologies within educational applications could profoundly transform language learning in an increasingly digital-first environment.

In conclusion, the research and development conducted in this thesis provide critical insights into the application of contemporary software development techniques within the realm of educational technology, underscoring substantial opportunities for innovation and enhancement in language learning applications.

BIBLIOGRAPHY

[1] Mobile Operating System [online]. [cit. 2024-04-12]. Available from: https://www.toppr.com/guides/computer-science/computer-fundamentals/operating-system/mobile-operating-system

[2] Android operating system [online]. [cit. 2024-04-12]. Available from: https://www.britannica.com/technology/Android-operating-system

[3] iOS Explained: Apple's operating system version history, features, and iPhone capabilities [online]. [cit. 2024-04-12]. Available from: https://www.businessinsider.com/apple-ios

[4] Windows Mobile [online]. [cit. 2024-04-12]. Available from: https://microsoft.fandom.com/wiki/Windows_Mobile

[5] History and timeline of mobile operating systems [online]. [cit. 2024-04-12]. Available from: https://www.techtarget.com/searchmobilecomputing/definition/mobile-operating-system

[6] Multiplatform mobile development [online]. [cit. 2024-04-12]. Available from: https://habr.com/ru/articles/491926/

[7] Global Digital Overview [online]. [cit. 2024-04-12]. Available from: https://datareportal.com/reports/digital-2020-global-digital-overview

[8] React Native What is it [online]. [cit. 2024-04-12]. Available from: https://habr.com/ru/articles/596183/

[9] Flutter Architectural overview [online]. [cit. 2024-04-12]. Available from: https://docs.flutter.dev/resources/architectural-overview

[10] What is cross-platform mobile development [online]. [cit. 2024-04-12]. Available from: https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-mobile-development.html

[11] The Challenges of Cross-Platform Architecture [online]. [cit. 2024-04-12]. https://ankocorp.com/blog/the-challenges-of-cross-platform-architecture#:~:text=Cross%2Dplatform%20architecture%2C%20also%20known,multiple%20operating%20systems%20and%20devices.

[12] Flutter (software) [online]. [cit. 2024-04-12]. Available from: https://en.wikipedia.org/wiki/Flutter_(software)

[13] What is an SDK [online]. [cit. 2024-04-12]. Available from: https://aws.amazon.com/what-is/sdk/#:~:text=does%20AWS%20provide%3F-,What%20is%20an%20SDK%3F,operating%20system%2C%20or%20programming%20language.

[14] Dart (programming language) [online]. [cit. 2024-04-12]. Available from: https://en.wikipedia.org/wiki/Dart_(programming_language)

[15] Flutter apps in production (programming language) [online]. [cit. 2024-04-12]. Available from: https://flutter.dev/showcase

[16] .NET MAUI customers showcase (programming language) [online]. [cit. 2024-04-12]. Available from: https://dotnet.microsoft.com/en-us/platform/customers/maui

[17] What is the Future of Flutter App Development (programming language) [online]. [cit. 2024-04-12]. Available from: https://medium.com/@glyphstergo/flutter-app-development-in-2024-74655ccb7b2d

[18] Building Responsive UIs in Flutter: Tips and Best Practices [online]. [cit. 2024-04-12]. Available from: https://www.linkedin.com/pulse/building-responsive-uis-flutter-tips-best-practices-steve-johnson-elq5f/

[19] What is a React Native[online]. [cit. 2024-04-12]. Available from: https://habr.com/ru/articles/596183/

[20] What is a React Native? Complex Guide for 2024 [online]. [cit. 2024-04-12]. Available from: https://www.netguru.com/glossary/react-native

[21] The new React Native architecture - what changes does it bring? [online]. [cit. 2024-04-12]. Available from: https://www.linkedin.com/pulse/react-native-new-architecture-what-changes-brings-react-poland/

[22] Top features of React Native application development [online]. [cit. 2024-04-12]. Available from: https://medium.com/@anuj.tomar11/top-features-of-react-native-app-development-355eff652c00

[23] What is .NET MAUI [online]. [cit. 2024-04-12]. Available from: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0

[24] Cross-platform application development with .NET MAUI [online]. [cit. 2024-04-12]. Available from: https://arpitkulsh.medium.com/cross-platform-app-development-with-net-maui-5a2d39a9cda

[25] Revolutionizing Mobile App Development with .NET MAUI [online]. [cit. 2024-04-12]. Available from: https://www.linkedin.com/feed/update/urn:li:activity:7112198490583158784/

[26] What is .NET MAUI And Should You Use It [online]. [cit. 2024-04-12]. Available from: https://assemblysoft.com/blog/post/what-is-net-maui-and-should-you-use-it

[27] Ionic (mobile app framework) [online]. [cit. 2024-04-12]. Available from: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework)#cite_note-14

[28] Ionic Studio is Dead [online]. [cit. 2024-04-12]. Available from: https://www.reddit.com/r/ionic/comments/jkk45i/comment/gamii1b/

[29] NativeScript [cit. 2024-04-12]. Available from: https://en.wikipedia.org/wiki/NativeScript

[30] Kana Practice [cit. 2024-04-12]. Available from: https://play.google.com/store/apps/details?id=com.yukiprojects.kanapractices

[31] Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022 [cit. 2024-04-12]. Available from: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[32] Japanese language [cit. 2024-04-12]. Available from: https://en.wikipedia.org/wiki/Japanese_language

[33] Basic of Hiragana, Katakana and Kanji in Japanese [cit. 2024-04-12]. Available from: https://dzen.ru/a/ZbFqssDKaxT00G6t

[34] json_serializable [cit. 2024-04-12]. Available from: https://pub.dev/packages/json_serializable

[35] Start building Flutter Android apps on macOS [cit. 2024-04-12]. Available from: https://docs.flutter.dev/get-started/install/macos/mobile-android

[36] Homebrew install Flutter [cit. 2024-04-12]. Available from: https://formulae.brew.sh/cask/flutter

[37] Toolbox JetBrains App [cit. 2024-04-12]. Available from: https://www.jetbrains.com/toolbox-app/

[38] Google Chrome [cit. 2024-04-12]. Available from: https://www.google.com/chrome/

[39] Visual Studio Code [cit. 2024-04-12]. Available from: https://code.visualstudio.com

LIST OF ABBREVIATIONS

API	Application Programming Interface
HTML	Hypertext Markup Language
DOM	Document Object Model
MAUI	Multi-platform App User Interface
SKD	Software Development Kit
UI	User Interface
UX	User Experience
iOS	iPhone Operating system
.NET	Network Enabled Technology
OS	Operating System
TV	Television
APP	Application
CI	Continuous Integration
CD	Continuous Deployment
CEO	Chief Executive Officer
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
ARM	Advanced RISC Machine

LIST OF FIGURES

APPENDICES

Appendix P I: CD with the source code of the application.