

Ministry of Education and Science of Ukraine
V. N. Karazin Kharkiv National University
Educational and Scientific Institute of Computer Sciences and Artificial
Intelligence

Recommended for defense:
minutes of the department meeting
No. ____ dated __.2025
Head, Dept. of CEST ____
_____Oleshko O.I.

Explanatory Note

for the Bachelor's Degree Project

on the topic “Development of an Information System for Interactive Text
Translation for Languages That Use Hieroglyphs.”

Defense at Exam. Comm. No. ____
protocol No. ____ dated _____.2025
Grade ____ / _____
Head of the SEC
_____D.Sc. (Tech), Prof. Fesenko H.V.

Prepared by:
4th-year student, group CS-41
Specialty: 122 – Computer Science
Kysil Hryhorii Andriiovych

Supervisor: S. V. Dyadun, Ph.D. (Tech),
Assoc. Prof.

Reviewer: M. M. Khruslov, Ph.D.
(Phys.–Math.), Assoc. Prof.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра інтелектуальних програмних систем і технологій

Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр

Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІПСіТ

_____ Олег ОЛЕШКО
підпис ім'я, прізвище

“ _____ ” _____ 2025 року

**З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Кисіль Григорій Андрійович

(прізвище, ім'я, по батькові студента)

1. Тема роботи: **«Розробка інформаційної системи інтерактивного перекладу текстів для мов, що використовують ієрогліфи.»/ «Development of an Information System for Interactive Text Translation for Languages That Use Hieroglyphs.»**

керівник роботи к.т.н., доцент Дядюн С. В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ _____ ” _____ 2025 року № _____

2. Строк подання студентом роботи _____

3. Перелік питань, які потрібно розробити:

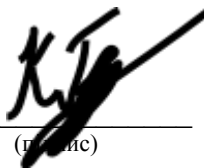
Ознайомитись з особливостями розробки крос-платформних застосунків, провести огляд існуючих методів розробки крос-платформних застосунків за допомогою Flutter framework, ознайомитися з алгоритмом роботи і методиками розробки застосунків за допомогою Flutter, створити модель для інтерактивного перекладу японських текстів, провести програмну реалізацію та експериментальне дослідження ефективності обраних методик отримання нових мов.

4. План роботи

№ з/ п	Назви етапів роботи
1	Формулювання мети та повного переліку завдань для виконання кваліфікаційної роботи (КР).
2	Пошук та аналіз інформаційних джерел за темою КР.
3	Визначення особливостей японської писемності та мобільних платформ.
4	Розробка моделей для створення, презентації та рекомендацій інтерактивних текстів.
5	Програмна реалізація застосунку для IOS та Android платформ.
6	Налаштування бекенду застосунку у вигляді хмарного сервісу
7	Тестування застосунку, підготовка до релізу в Play Store та App Store
8	Підсумок отриманих результатів та оформлення пакету документів для процедури захисту КР в екзаменаційній комісії ННІ КН та ШІ.

5. Дата видачі завдання _____

Студент


(підпис)

Кисіль Г.А.
(ініціали, прізвище)

Керівник роботи


(підпис)

Дядюн С.В.
(ініціали, прізвище)

ABSTRACT

Explanatory note to the diploma thesis: 4 chapters, 55 pages, 21 figures, 2 tables 15 sources.

Relevance of the topic: Development of the app that will be a platform for users to create and read interactive text to improve their knowledge about foreign languages.

Object: Process of development of the system that allows users to get access to interactive texts on foreign languages.

Subject: Tools of development for mobile cross platform applications such as Flutter framework, Android Studio, Xcode as well as cloud services to manage a backend for data storage.

Goal: Development of the cross platform mobile application that provides users the platform for creating and reading interactive text with languages that use kanji(hieroglyphs).

This work offers an overview of the development process of mobile cross-platform applications that allows users to create and read interactive texts as well as monitor their learning journey.

This note includes analysis of the problems and already existing solutions, description of tools that were used during development, description of development process, and the description of app functionality.

Keywords: MOBILE, CROSS-PLATFORM, DART, FLUTTER, FIREBASE, IOS, ANDROID, INTERACTIVE TEXT, COMPREHENSIBLE INPUT, LANGUAGES, JAPANESE

TABLE OF CONTENTS

TABLE OF CONTENTS.....	4
CONVENTIONAL SYMBOLS.....	6
INTRODUCTION.....	7
1 APP CONCEPT.....	8
1.1 Problem statement.....	8
1.2 Relevance objective and goals.....	9
1.3 Main concept.....	10
1.4 Interactive texts.....	11
1.5 Progress and words based recommendation.....	12
1.6 Analogues.....	12
1.6.1 Lingopie.....	12
1.6.2 ImmersionKit.....	13
1.6.3 Yomu Yomu.....	13
1.6.4 Hiragana times.....	13
1.6.5 Anki+Yomitan.....	13
1.6.6 Kanji Study app.....	14
2 REQUIREMENTS.....	15
2.1 Functional requirements.....	15
2.1.1 Find Texts.....	15
2.1.2 My Texts.....	16
2.1.3 Words.....	17
2.1.4 Account and Settings.....	17
2.2 Non-functional requirements.....	18
3 TOOLS AND TECHNOLOGIES.....	20
3.1 Tools.....	20
3.1.1 Android Studio, plugins and AVD.....	20
3.1.2 XCode and IOS Simulator.....	21
3.1.3 Git with Github.....	21
3.1.4 Firebase CLI and Flutterfire.....	22
3.2 Description of used technologies.....	22
3.2.1 Dart.....	22
3.2.2 Flutter.....	23
3.2.3 Flutter bloc.....	25
3.2.4 Firebase authentication.....	26

3.2.5 Firebase firestore.....	27
3.2.6 Firebase storage.....	28
3.2.7 Firebase Crashlytics and Analytics.....	28
3.2.8 Gemini API, GenKit and Ai logic.....	29
4 APPLICATION DEVELOPMENT.....	30
4.1 Cloud services and backend.....	30
4.1.1 Firebase Authentication.....	30
4.1.2 Firebase Firestore DB planing.....	31
4.1.3 Firebase storage.....	33
4.1.4 Firebase Firestore and Storage security rules.....	33
4.1.5 Analytics and Crashlytics.....	34
4.2 Mobile app development.....	35
4.2.1 Use cases.....	35
4.2.2 Project architecture.....	36
4.2.3 Authentication.....	39
4.2.4 Public texts UI and functionality.....	43
4.2.5 My texts UI and functionality.....	45
4.2.6 User's vocabulary UI and functionality.....	49
4.2.7 User's preference UI and functionality.....	49
4.2.8 Builds preparation.....	50
4.2.9 Testing and analytics.....	51
CONCLUSIONS.....	52
SOURCES.....	53

CONVENTIONAL SYMBOLS

UI - User interface.

IDE - Integrated Development Environment.

SDK - Software development kit.

App – application.

Learner - a person who gets the language by actively studying it.

Acquirer - person who gets the language by consciously focusing on understanding the input.

ToS - terms of service.

Kanji – hieroglyphs borrowed from Chinese that are used in the Japanese language.

Kana – script that represents syllabary.

Addon - extension for the functionality of the application

INTRODUCTION

In the modern world, interactions between individuals from diverse cultural and linguistic backgrounds are becoming increasingly common. However, several barriers can hinder effective communication during such interactions. Among these are differences in behavior, perspectives, interests, and most significantly, language barriers. This poses substantial challenges for people who try such interactions. It is inherently difficult to maintain engagement in a conversation with a person whose language one cannot fully understand.

Over recent years, numerous tools have emerged to address these challenges, including interactive translators (such as Google Lens and various real-time speech/text translators), spelling and grammar checkers (e.g., Grammarly), predictive text applications, and gamified language learning platforms like Duolingo. Despite the diversity of available tools, many individuals still perceive communication across linguistic boundaries as a formidable task. Moreover, these tools often lack the capacity for contextual language comprehension, which is crucial for conveying nuanced elements like humor or cultural references.

Additionally, not all languages are equally supported by these tools. For instance certain features inherent to the Arabic language family, Chinese, and Japanese can make some technological solutions ineffective or obsolete. As a result, language acquisition through traditional learning methods remains the most reliable approach for sustained and meaningful communication.

Learning a new language is a complex, resource-intensive process that demands considerable time and effort. For certain languages, achieving proficiency can require thousands of hours of dedicated practice. During this extensive learning period, maintaining motivation and interest is crucial. However, successful learning also depends on access to appropriate materials, such as texts, dialogues, grammar guides, and other instructional resources.

1 APP CONCEPT

1.1 Problem statement

Let's discuss the problems that language learners encounter while learning languages that use hieroglyphs(logographic characters) based on examples of Japanese language. In Japanese primary method of forming words is straining together morphemes that represent some grammatical meaning, and modified by addition of particles. Writing system consists of 3 different scripts which are used simultaneously during writing. Each of the scripts has its purpose and provides some context.

- Hiragana(平仮名, ひらがな) - is script that represents Japanese syllabary. The word itself means plain/common). The characters in this script represent simple sounds that can be encountered in the language. During writing it's usually used to form the ending of the words and particles, alongside some other grammatical functions. However it is becoming more common nowadays to see some words written in hiragana as well.
- Katakana(片仮名/カタカナ) - the script that mirrors hiragana. All characters in hiragana have a representation in katakana but the usage is a bit different. Katakana is never used for grammatical purposes, instead it is usually used for writing (nouns?) personal names, botanic/animal names and words borrowed from another language(like テレビ - 'terebi' - tv, television).
- Kanji(漢字) - the script that was borrowed from Chinese writing system but undergone some significant changes during the centuries. This script contains about 50 thousand characters however in common communications only about 3 thousand characters are used regularly. Each of these characters have their own meaning and usually several reading: on'yomi(音読み) - reading borrowed from Chinese writing system(however it is not always exact the same as in modern Chinese) and kun'yomi(訓読み) - original

Japanese reading. Considering the amount of reading, in modern texts rare kanjis are written with the furigana(reading of the character in this context written in hiragana above or right to it).

- Rōmaji (romanization of Japanese) - the latin script that used to write kana(Hiragana and Katakana) representation of words in latin characters. It's rarely used outside of language learning.

In addition to the complexity of the scripts, spaces are almost allways absent in japanese sentences, making it even more difficult for learners to navigate.

Apart from the writing system, the Japanese language has somewhat different from most of the other languages' word order in the sentence. For example, time is usually stated first in the sentence and the verb is usually placed by the end. Let's translate the Japanese sentence "I like when it rains". The result will be "私は雨が降っているのが好きです。" which if we translate it word by word would mean "As for me, the time when rain is falling is liked.". Moreover, noun represented by "私は" or "as for me" in the sentences, usually omitted when it supposed to be obvious to the listener.[1]

1.2 Relevance objective and goals

Relevance: In the modern world interaction between people from different cultures and countries is becoming more common, the problem of communication arises. Language learning platforms are becoming increasingly popular attracting millions of users. However commonly used tools are not equally suited for every language, some of them require additional attention to their features. This work is dedicated to the languages that use hieroglyphs in their writing system (mainly Japanese).

Objective of this work: Creating a system that helps users to interact with scripts that use hieroglyphs in its writings. Helping them to better understand the

messages written in the text without providing complete direct translation. However we will be scaling down the subject to concrete(Japanese) language.

We need to develop a working mobile application that helps language learners to interact with complex writing systems(such as Japanese in our case) and enrich their vocabulary using these interactions. Based on this description our goals will be:

- Analyze the problems that new language learners face while studying languages that use hieroglyphs in its writing system.
- Analyze existing market of solutions, research the best practices out here, look at gaps that are not covered and which we can cover.
- Considering the existing market of solutions, find a way to cover the problems that learners face.
- Research the tools and technical possibilities of different approaches(such as web, mobile app, cloud services, custom api).
- Create an application that helps users to interact with complex writing systems.
- Testing and polishing the application in preparation for release.

1.3 Main concept

The main idea behind the application is based on the “Input hypothesis” offered by Stephen D Krashen in his work “Principles and Practice in Second Language Acquisition” [2]. According to this hypothesis natural progress acquiring language happens when an acquirer that is currently on stage i of foreign language acquisition receives and understands the input from stage $i+1$ of the given language. By understanding we meant getting the meaning of the message, not necessarily its form or structure. Simplifying the statement above we can say that acquisition of language happens when we understand a little more than we currently know. It’s possible due to additional context, prior knowledge about the subject or/and extra-linguistic information.

As for more recent studies in their 2018 study, Kazuya Saito and Keiko Hanzawa [3] examined how varying amounts of English input influence the development of oral proficiency among first-year Japanese university students over an academic year. The research focused on key aspects of language acquisition, including fluency, vocabulary, grammar, and pronunciation.

The findings revealed that increased exposure to English—both through classroom instruction and usage outside the classroom—significantly enhanced students' fluency and lexicogrammatical abilities, particularly during the first semester. This improvement aligns with Krashen's Input Hypothesis, which emphasizes the importance of comprehensible input in language learning.

Pronunciation development presented a more complex picture. Prosodic features, such as intonation and rhythm, showed gradual improvement over two semesters, while segmental features, like individual sounds, exhibited limited change within the study's timeframe. These results suggest that while comprehensible input is crucial, certain aspects of language, such as pronunciation, may require more targeted or prolonged exposure to achieve noticeable improvement.

Based on that hypothesis we can suggest a system that will provide users with language materials that is slightly beyond the user's current knowledge, allowing them to acquire new vocabulary and grammar constructions. The process of interacting with materials should be as smooth as possible to keep the users engaged

1.4 Interactive texts

One of the ways to present materials to people who try to acquire new language is in the way of interactive text. The concept by itself is pretty common - an application provides the user with a view with text, settings to it and ability to click for words to get a translation. As a result of this setup our user will be able to read the text and translate the unknown words in it without big interruptions in reading flow like going to vocabulary, or searching the meaning somewhere else.

This approach, however, usually assumes that the user will know how to read the text which might not be true for some languages that use hieroglyphs. Japanese language, for example, have more than 3 thousand commonly used kanji each of which in average have more than 2 readings, excluding exceptions and specific readings for personal names. Therefore in our case interactive text must account for the complex writing system and specifics of readings. Additionally it might be a good idea to add some optional grammar explanations and the ability to transform/modify complex or unknown kanji to their phonetic reading.

1.5 Progress and words based recommendation

A feature that will bridge the gap between input theory and interactive texts is a word-based recommendation system. Combined with user progress tracking. It will lay a foundation that allows us to personalize materials for every individual. The core concept behind this is to monitor each user's learning journey, generating personalized suggestions on the most suitable material to study next. By doing so, we can ensure that learners are consistently engaging with content that is understandable to them yet introduces new vocabulary or grammar structures to expand their knowledge.

Moreover, this system will potentially allow us to adapt the interactive texts dynamically, tailoring it based on individual progress. This approach supports incremental learning and helps maintain motivation by providing material that feels relatively easy to understand yet still has something new for users to learn.

1.6 Analogues

In the area of language learning, competition is really intense. However most of the app are trying to get user attention with easy, gamified experience or robust grammar-based courses. Let us discuss apps that closely resemble the system we are developing and specify the differences between them and our system.

1.6.1 Lingopie

Lingopie - is an interactive system consisting of a site, browser add-on and a mobile app. The main idea of the app is to provide users with a place to gather new words and help users learn them. The main downside of this app is that it is expensive to use. It costs around 20 USD per month (with ability to buy lifelong licence for 170 USD). This app doesn't have a free version. And in addition to this, most services it interacts with (like Netflix or Apple TV) are also paid services.

1.6.2 ImmersionKit

ImmersionKit - app that provides a variety of sentences and examples from different media. The problem is that most of these examples are incoherent and the app itself is quite buggy and unintuitive to use.

1.6.3 Yomu Yomu

Yomu Yomu - The closest resemblance of the system we are building. It provides a variety of texts for different levels, allowing users to read them, and when needed, get a translation for the specific sentences or words. It also provides some grammar lessons, and vocabulary exercises. The difference is that this app does not have a personalized recommendation system, and it's not doing much to actually simplify the reading process.

1.6.4 Hiragana times

Hiragana times - most renowned app to start practicing reading in Japanese. It provides full translation of the text first and then Japanese text for users to read. Depending on the user's settings they may turn on hiragana/romaji transcription, word by word translation, spaces between words. It also has text to speech functionality. This app contains large libraries of texts for users to read, which was accumulated over more than 400 issues of the magazine.

Compared to the system we are trying to develop it provides very limited capabilities in terms of how users want to read the text. All elements are static and may be turned on/off for the whole text.

1.6.5 Anki+Yomitan

Anki+Yomitan - two different apps that in combination can achieve something similar to what we want to do in our app. Yomitan is a browser extension that allows users to translate words that they encounter on different web pages, while Anki is a flash cards app, designed to help them memorize words in foreign languages. Yomitan's integration with Anki, it's extremely easy to add translated words to collections. However, this setup is a bit clunky to use and doesn't have any recommendation system to find new materials.

1.6.6 Kanji Study app

Kanji Study app - its reading addon was one of the sources of inspiration for this app. The app itself focuses on helping users to memorize separate kanji, and uses several methods for achieving this. One of the paid extensions is a reading addon which gives users examples of sentences for each kanji. The main feature is - these sentences are only using kanjis that the user is already learned - everything else is kana. With this approach the system progressively introduces users to more and more kanjis. The problem however is that this system does not account for words users know or any grammar. So sentences are often a bit confusing and contain a lot of unknown words that will not be repeated or introduced anytime soon.

2 REQUIREMENTS

This chapter will be describing the list of requirements that our application will need to meet. Functional requirements will describe and specify the functionality of the app while non-functional requirements will be about general specification of the application.

2.1 Functional requirements

Upon launching the app, the user will be greeted with a splash screen. After a brief delay, which allows the app to initialize main classes and load resources, the user will be redirected to the main dashboard screen.

The dashboard screen consists of four distinct tabs. The tab bar is located at the bottom of the screen, while the app bar is positioned at the top. Each tab is represented by a unique icon and title. The app bar displays the app's name, a vocabulary button for quick word search, and, if the user is logged in, their username's first letter. If the user is not logged in, this space will display a "Sign In" button.

2.1.1 Find Texts

The first tab is called "Find Texts." This tab is designed to help users discover texts for study. At the top of the tab, there is a search bar. When the user taps on the search bar, the remaining page will be dimmed, and search results will appear after the user types a query, with a debounce time of 1 second to improve performance. The result list is scrollable and includes texts that match the search term either in the title or the tags. The search bar also features a suffix icon that allows users to clear the text in the field, remove focus from the bar, and return to the content of the tab.

Beneath the search bar, there is a small list of recommended tags tailored to the user. Tapping on any tag will lead the user to a screen displaying a list of texts related to that particular tag.

Below the tag recommendations, the user will find a list of the newest texts available in the app. Tapping on any text will take the user to the text study screen.

The text study screen includes an app bar, a small settings panel at the top, and the body of the text itself. Tapping on a sentence will open a bottom sheet displaying information related to that sentence. A long press on any word will trigger a bottom sheet that provides details about the word, with the option to add it to the personal dictionary.

2.1.2 My Texts

The second tab is called "My Texts," and it is dedicated to texts that the user has studied as well as texts that they have created. This tab is divided into two sub-tabs located at the top of the screen. The first sub-tab is called "Studied Texts." At the top of this tab, the user will see a section featuring texts that they are currently studying and have not yet completed (i.e., they have not clicked the "I understood everything" button). Below this section, there will be a history of all texts the user has studied, which are still available in the app.

The second sub-tab is titled "Add Text." This section displays a list of texts that the user has created, along with a circular "+" button in the bottom left corner of the tab. Pressing on a text from the list will open the "Edit Text" screen with that specific text already loaded. Pressing the "+" button will open the "Edit Text" screen with a blank text field, allowing the user to create a new text.

There are several steps on the 'Edit Text' screen. Initially, users can type or paste their text into a text field. Once text is entered into the field, the user can press "Continue" to move on to the second step.

At this stage, users can no longer edit the source text, but they can select portions of the text, mark them as sentences (for the text with punctuation missing) and enter translations and additional information. Each part of the text can belong to only one sentence. Afterward, users can move on to the next step, where they can add translations and additional information for individual words and phrases.

This flow can be simplified by using the AI assistance button after entering the text on the first step. This will parse the text automatically using the AI model, leaving for users only to review parsed sentences and words.

Once these steps are completed, the text can be saved and prepared for publishing.

2.1.3 Words

The third tab is called "Words", and it allows users to view words they have added from texts, as well as search for words within their personal vocabulary or the app's default vocabulary. Pressing on any word will open a dialog with detailed information of that word, with options to add it to the user's personal vocabulary or, if the word is already in their vocabulary, delete it. In future versions, users will also be able to track their progress in learning specific words.

2.1.4 Account and Settings

The final tab is called "Account and Settings", which is where users can manage their account information, app settings, and access the Terms of Service (ToS) and Privacy Policy.

At the top of this tab, logged-in users will see their email address, along with buttons to sign out or delete their account. Pressing either of these buttons will trigger a confirmation pop-up, after which the corresponding action will be executed. If the user is not logged in, they will see a message indicating that they are not signed in, along with options to sign in or sign up. Each option will lead the user to the appropriate screen for signing in or creating an account.

Beneath this section, users will find app settings.

At the bottom of the screen, users will see links to the Terms of Service and Privacy Policy. Tapping on these links will open a new screen displaying the corresponding information.

2.2 Non-functional requirements

The mobile application will need to meet defined performance standards. It should load the home screen within two seconds on common mobile devices and support up to 100 concurrent users while maintaining a stable response time.

The application should follow design guidelines specific to each platform—Material Design for Android and Human Interface Guidelines for iOS. This will ensure consistency and ease of use across devices.

The app should handle network interruptions by informing the user of the issue and returning to its previous state. This will help minimize user disruption.

Application should scale horizontally and handle increased user load without requiring downtime or major changes to the infrastructure.

User data will be protected by encrypting all transmissions using HTTPS with TLS 1.2 or higher. Authentication will be implemented through OAuth 2.0. The system should comply with applicable data protection regulations such as GDPR and CCPA.

Maintainability will be achieved through a modular codebase that follows clean architecture principles. This structure will support future updates, testing, and development without unnecessary complexity.

The app will be portable across major platforms. It will run on Android 10 and later, and iOS 13 and later. At launch, it will support the English language, with a framework in place to add additional languages when needed.

Table 2.1 - Requirements Summary

Functional	Non-functional
Splash screen is shown on app launch	Home screen loads within 2 seconds
Dashboard with 4 tabs,	Supports up to 100 concurrent users
Find Texts tab that allows user to look for new texts	Follows platform design guidelines (Material/iOS)

Table 2.1 - Continuation

Text study screen that present interactive text	Handles network interruptions
My Texts tab for studied and user-created texts	Data encrypted
Text editor with AI-assisted parsing	OAuth 2.0 authentication
Words tab for personal vocabulary	GDPR and CCPA compliant
Account & Settings for user and app management	Modular and maintainable codebase
Authentication flow (sign in/up, logout)	Cross-platform: Android 10+, iOS 13+
	English support with multi-language framework

3 TOOLS AND TECHNOLOGIES

3.1 Tools

3.1.1 Android Studio, plugins and AVD

It's IDE based on IntelliJ IDEA Community Edition with addition of tools necessary for android development such as Android Gradle plugin, Layouts and resource editor, emulator and virtual device manager, keytool to form signature files for apps and Android centric UI elements and windows. Base version of Android Studio do not support neither Flutter toolkit or Dart analyzer therefore we need to install Dart and Flutter plugins before we can start working with the project. After installation, the option to create a new Flutter project will be added to the file context menu alongside numerous tools(like Flutter inspector, hot reload button or new options for selecting run targets) that are useful when developing an app.

After setting up plugins we will need to set up an android toolchain and Android virtual device. First we will need to setup android SDK, which we can select at corresponding menu (fig. 3.1).

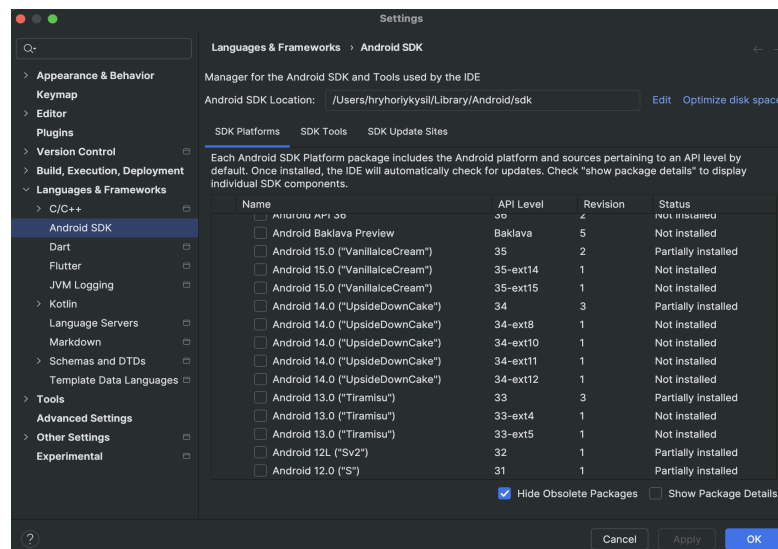


Figure 3.1 - Variety of SDKs for installation

After this we need to set up an emulator(optional, we can always use a real device) and our Android setup is done.

3.1.2 XCode and IOS Simulator

Now for us to be able to ship an app on IOS we will need to download and install XCode, IDE and toolkit developed for working with apps for MacOS or IOS. To get these tools we will need to create an account on AppStore and specify our billing details(required, despite the IDE being free). After this we simply download XCode on our MacOS, all installation happens automatically. We won't be writing code there but will be visiting from time to time to make builds, add some IOS specific configurations(capabilities, deeplinks, etc.) and set up Apple certificates for app signing(on release stages). After xcode is set we will need to download the IOS simulator tool. Process yet again automatic.

After full setup we now can reopen android studio and from target selector we will be able to select IOS simulator as a build target (fig. 3.2).

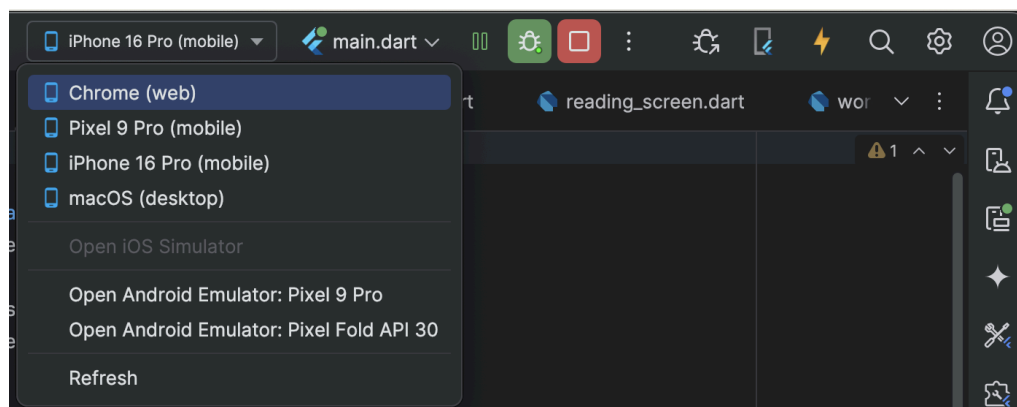


Figure 3.2 - Device selection

3.1.3 Git with Github

After all main platforms are set we can start writing our app. However for the safety of the project and as a general good practice we will need to setup version control system. Android Studio. We will be using Git as it is something that is natively supported by Android Studio and the author is most familiar with,

however we still need to select a service. As we don't need complex CI/CD flows, or corporate level security and management we can settle with the simplest solution - GitHub. The main advantage is the free model and relative simplicity.

We can connect our IDE to a github account via security token and then create a remote repository for our project (fig. 3.3). Now we can manage our commits and branches right from our IDE.

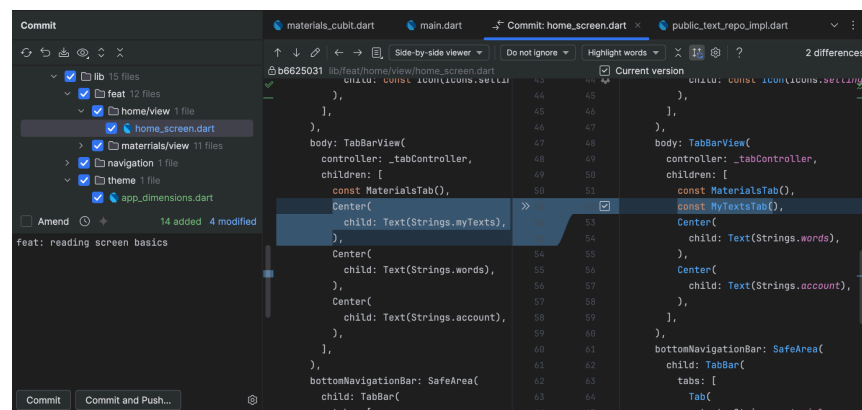


Figure 3.3 - Example of working with git

3.1.4 Firebase CLI and Flutterfire

After IDE, SDKs and version control is set, we will need to set up one more thing - connect firebase to our project, and install FirebaseCLI and FlutterFire tools for easier interactions with the firebase console. These tools allow us to manage our project from our command line without needing to constantly go to the console. It will also automates setup of the apps for our projects. [5]

3.2 Description of used technologies

3.2.1 Dart

Dart is a modern, object-oriented, class-based, garbage-collected programming language developed by Google. Designed for building fast and scalable applications, Dart is especially known for its seamless support for both front-end and back-end development. Its syntax is clean and familiar to those who

have used Java or JavaScript, making it accessible yet powerful. Dart powers Flutter, a popular framework for creating cross-platform mobile, web, and desktop apps from a single codebase. With features like sound null safety and just-in-time as well as ahead-of-time compilation, Dart balances developer productivity with high performance.

3.2.2 Flutter

Flutter is a cross platform framework that was developed by Google and first introduced in May 2017. Since then it has grown rapidly building a community of like minded developers and reaching first stable version 1.0 in December 2018.

The main idea behind this framework was for developers to think not “On which platform I want to build things?” but “What I want to build?”, therefore focus was on supporting a multitude of platforms and archiving competitive performance on every platform, while keeping the “native feeling” of the platform user is working on.

Version 1.0 supported only mobile platforms but already had extensive community support. While focus has remained on mobile platforms, the framework since then Flutter has expanded to a multitude of platforms(Windows, Linux).

Flutter’s pros and cons are listed in the table 3.1. So we can get a better understanding on how it fits for our purposes:

Table 3.2 - Flutter pros and cons

Pros	Cons
Build once and run everywhere	Code base require more frequent maintenance
Wide selection of community built tools	Bigger app size
Fast development with “Hot reload” feature	Challenging in achieving “pixel perfect” UI

Table 3.2 - Continuation

High performance	New platform-specific features usually integrated with delay
Open source and free	

Plenty of community built packages start appearing, and google created pub.dev repository and support for external packages. Which has helped to speed up the development process even more.

This allowed flutter to keep up to the competition and get a favour of developers and brands. Google's own "Google cloud" mobile app developed with flutter. According to JetBrains State of development ecosystem survey flutter was the most used cross platform framework since 2021[11]. And according to Apptopia[2] 28% of new apps in AppStore are written in flutter in 2024.

Flutter uses a declarative approach to build UI classes. UI is built by nesting widgets into a huge widget tree that is then laid out on the device screen. The optimization algorithms keep keeping performance in check

On the figure 3.4 we can see an example of a simple widget that will be used as a part of a larger widget tree.

```

3  class TextFile extends StatelessWidget {
4    const TextFile({
5      super.key,
6      required this.title,
7      required this.tags,
8      required this.onPress,
9    });
10
11    final String title;
12    final List<String> tags;
13    final void Function() onPress;
14
15    @override
16    Widget build(BuildContext context) {
17      return ListTile(
18        title: Text(title),
19        subtitle: Text(
20          tags.join(', '),
21        ), // Text
22        onTap: onPress,
23      ); // ListTile
24    }
25  }

```

Figure 3.4 - Example of a simple widget

This particular widget is responsible for displaying list tiles in list view, it accepts a title, tags array and the closure function as required parameters and composes them into list tile representation, where title string will be title of the list, tags array will be converted to string with comma separator and presented as a subtitle and the closure will be called when user taps on any place inside this tile.

3.2.3 Flutterr bloc

Flutter bloc(business logic of class) is a state management package that allows us to keep the state of the app separate from the UI classes and update UI asynchronously when something is changed in the state. Package contains two classes `Bloc<Event,State>` and `Cubit<State>`, that can manage states, they both extend the `BlocBase<T>` class but have a slightly different implementation on how they interact with UI. They both use Streams(observable sequence of data) to emit a new state for UI element(s) that listen to it. However in Bloc we have an input Stream that listens to the event that is UI sending to this bloc, and Bloc's methods shouldn't be called directly from UI. On the other hand in Cubit the event stream is absent and UI interacts with it using its public methods (fig. 3.5). Bloc provides better encapsulation of inner logic, by using an additional level of abstraction in forms of events while Cubit is faster to implement, and does not require nuanced Event structure(which may be unnecessarily tedious in smaller scale projects).

As for method to UI for listen blocs we have 3 options:

- `BlocBuilder<Bloc,State>` - widget is used to rebuild part of the UI that is placed in its builder when new state is emitted.
- `BlocListener<Bloc,State>` - widget that has a `Function(BuildContext, State)` parameter that is called when a new state is emitted. While the child remains static.
- `BlocConsumer<Bloc,State>` - widget that combines functionality of both `BlocBuilder` and `BlocListener`.

To pass the Bloc down the widget tree we use a Block Provider widget that is built based on Flutter's native Inherited Widget. It will make a class passed there available to read from BuildContext in any place down in the widget tree. [9]

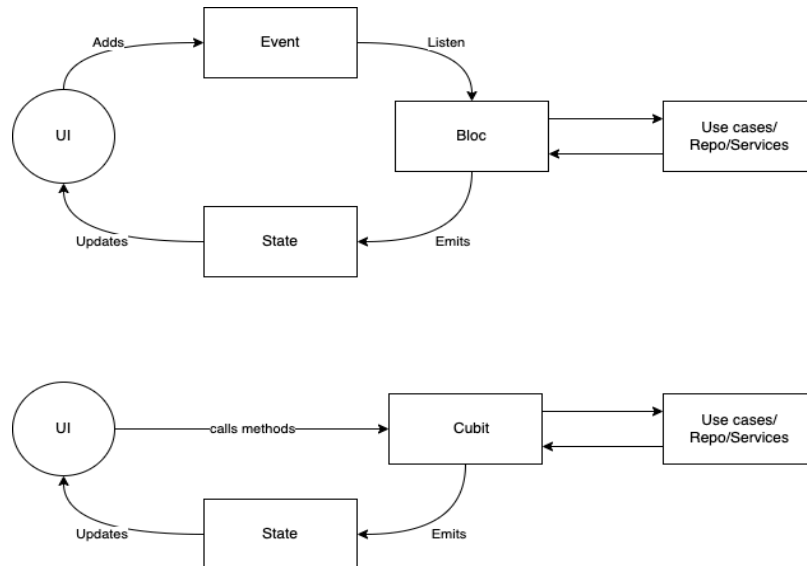


Figure 3.5 - Schema of bloc and cubit workflow

3.2.4 Firebase authentication

Firebase authentication is an identity management service provided by Firebase platform. It allows a variety of authentication methods like email+password, phone number, OAuth providers like Google or Facebook, etc. It also allows anonymous logins for guest sessions.

It is a solution to handle Sign In and Sign Up flows while ensuring state persistence across multiple applications launches. Complex backend logic like handling and refreshing tokens, validating credentials and securely store user data is handled by the service itself.

For our project however the main benefit will be the integration of Firebase authentication with other firebase services. We can seamlessly use data from firebase auth while writing security rules for Firestore, Firebase storage and cloud function. This will make it easier for us to manage user access to different types of data available in our app, and improve security of our app.

Another distinct feature of Firebase authentication is its support on mobile platforms. Packages for Android, IOS and Flutter simplifying development process and service integration in the app allowing to get results faster.

Another advantage of this service lies in its pricing model. The base service is completely free up until 50000 monthly users, email messages with validation and password reset are not billed separately. The only feature that requires payment to start using it is - phone authentication, which is billed per SMS send.

3.2.5 Firebase firestore

Cloud Firestore is a NoSQL database from Firebase that stores data in the cloud and can keep it in sync across devices in real time. It uses documents and collections to organize data, being a shameless database content of the document might vary inside 1 collection. It also supports real-time updates, so whenever data changes, all users that are subscribed to given data see the latest version instantly. It can handle complex queries on the same level as more light one thanks to a complex indexing system. As another notable feature Firestore connects smoothly with Firebase Authentication and other Firebase tools.

Firestore Constraints and Considerations

When designing with Firestore, it's essential to work within its architectural and pricing constraints:

Document and Data Structure Constraints:

- Maximum document size: 1 MB
- Maximum fields per document: 200,000
- Only full-document reads (no partial field reads)
- Each document read counts as a billing event
- Subcollections are not automatically fetched with documents
- Collections can only contain documents; documents may contain fields and subcollections

Query Limitations:

- No native OR logic; must use in or array-contains-any

- Only one inequality filter per query
- Compound queries require indexed fields
- No built-in full-text search (use Algolia or ElasticSearch if needed)
- Maximum of 30 values in in or array-contains-any filters
- Queries with inequalities must use orderBy on the same field
- Pagination must use document snapshots or field values—offsets are not supported

These limitations must be factored into the data model to ensure both performance and scalability. Additionally, the pay-as-you-go pricing model means that structuring our request for minimal document reads and writes is critical for cost-efficiency. [7]

3.2.6 Firebase storage

Firebase storage is a cloud solution for storing different types of files. It's built on Google Cloud Storage therefore inherited its benefits like scalability, flexible access control, and security. It also has features like real time file monitoring during upload, potentially allowing us to implement smoother user experience, and resumed uploads which might be especially useful on mobile platforms.

As is the case for other Firebase services it is tightly integrated with the Firebase ecosystem, which makes it easier for us to work with. It also has its own SDK for different platforms (like important for us - Android, IOS, and Flutter) that simplify our work on the front end.

3.2.7 Firebase Crashlytics and Analytics

Firebase Crashlytics and Analytics will be helpful to get insight on how users interact with our application and what problems they encounter.

Firebase Crashlytics will be a useful tool to track app crashes and unhandled exceptions even in production environments. It will provide us with logs that can help us understand how often problems happen, what is happening specifically and what is causing the problem. It will also provide us with information about device

that had a problem and the time when it happened - creating an easier time for us to act on feedback from people who are testing the app.

As for Firebase Analytics - is a tool that gathers information about user interactions like: screen had been visited, what content had been viewed, and other. This data will help us to improve user experience further, and get insight about what users actually want from our app. This tool will be especially helpful for further improving navigation structure inside the application and making it more efficient.

3.2.8 Gemini API, GenKit and Ai logic

Gemini API, GenKit, and Firebase AI Logic form a great toolkit for creating logic that uses generative AI in the apps. The Gemini API is a direct link to Google's AI models, it allows applications to use it in various forms for generating different types of content. It provides a powerful engine that can be a core for the AI-powered features in our application.

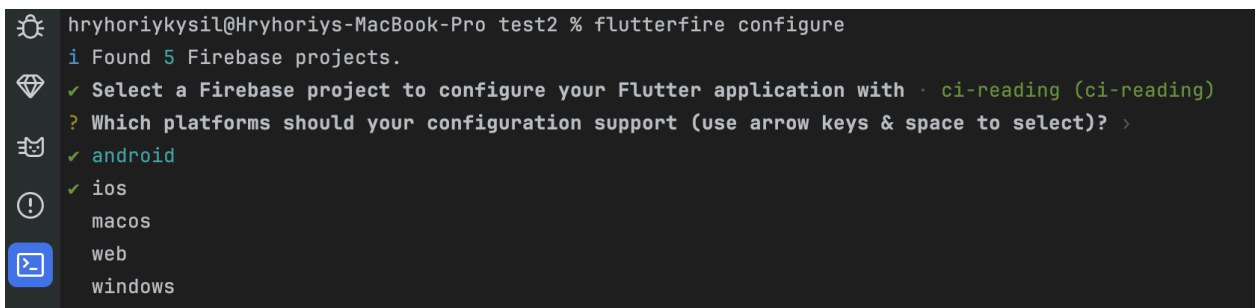
GenKit acts as a framework that helps streamline the development lifecycle of the applications. It simplifies the integration of various AI models, and offers a structured approach to managing AI workflows in our application. It has robust debugging tools, local development capabilities, and integrated observability. It also helps to ensure that AI features are producing correct results and can be easily managed.

Finally, Firebase AI Logic helps establish a link between our client-side applications and the Gemini models, while ensuring secure and scalable access. It provides SDKs for major platforms (including Flutter), allowing for direct interaction with AI models. It also can handle aspects like user authentication and app verification for API security which might be hapefull on later stages of development.

4 APPLICATION DEVELOPMENT

4.1 Cloud services and backend

First we will need to set up a firebase project in the firebase console. We can do this by entering a project id(“ci-reading” in our case) and selecting tools that we will need for development. In the next step we create the app(IOS and Android), it can be done manually, but as we have flutterfire installed, we can use it to automate the process. By using “*flutterfire configure*” command and following instructions(selecting projects, platform, etc.) (fig. 4.1).

A terminal window on a dark background showing the execution of the 'flutterfire configure' command. The prompt is 'hryhoriykysil@Hryhoriys-MacBook-Pro test2 %'. The output shows it found 5 Firebase projects, selected 'ci-reading', and asked for platforms. 'android' and 'ios' are selected with green checkmarks, while 'macos', 'web', and 'windows' are listed without. A blue cursor icon is visible on the left side of the terminal.

```
hryhoriykysil@Hryhoriys-MacBook-Pro test2 % flutterfire configure
i Found 5 Firebase projects.
✓ Select a Firebase project to configure your Flutter application with · ci-reading (ci-reading)
? Which platforms should your configuration support (use arrow keys & space to select)? >
✓ android
✓ ios
macos
web
windows
```

Figure 4.1 - Example of configuration

This will automatically create profiles for the applications on our firebase console based on our project content. The SHA256, and SHA1 fingerprints along with IOS certificates must be uploaded manually. These are required for some functionality to work, but can be skipped at this point.

4.1.1 Firebase Authentication

We will start from adding authentication service to our project and selecting sign in methods (fig. 4.2).

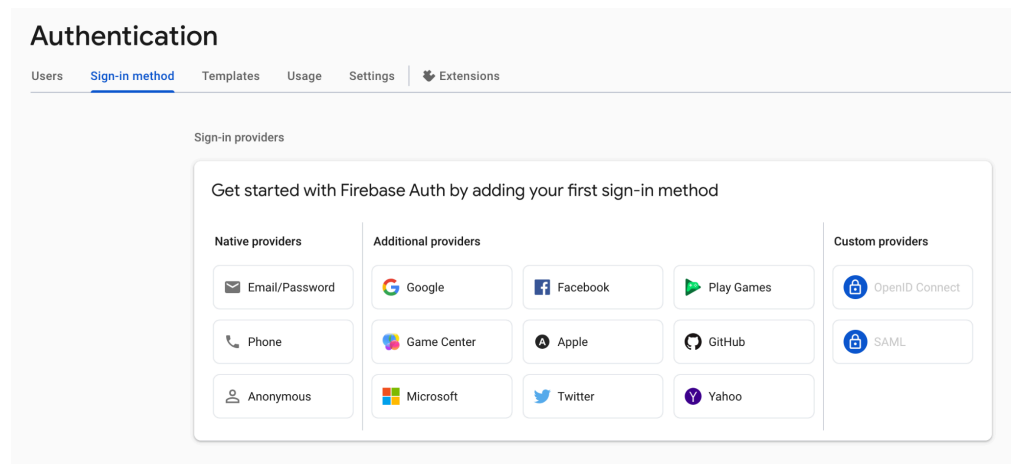


Figure 4.2 - Available authentication methods

At first we will be using only email providers so we will select it. After adding the first provider we can manage numerous settings available on the console. We can edit account verification and password reset emails' content, password requirements, etc.

At this point all cloud services are set for handling authentication and we can proceed to next step.

4.1.2 Firebase Firestore DB planing

We plan to structure the database around two primary top-level collections.

user_data Collection

Each document in this collection corresponds to a single user, that can be identified by their Firebase Auth UUID. Access is restricted, each user can only access their own document.

Each user document may contain:

- A read_texts collection: storing metadata of texts the user read
- A private_texts collection: storing data texts added by the user
- A progress map: storing the user's learning data and progress

This setup keeps user-specific data isolated and secure while allowing flexibility in customization and tracking.

texts Collection

This collection stores metadata for public texts. Each document contains:

- A concise summary (e.g., title, difficulty, author ID, average review score)
- A subcollection (e.g., content) containing the full text split into structured units (paragraphs, sentences, etc.)

The separation of metadata and content enables efficient loading: only the summary data is fetched during text selection, and full content is retrieved only when necessary (e.g., when opening the text) (fig. 4.3).[4]

To simplify *read* operations, each document is structured as a custom map, allowing full content retrieval in a single *read*. However, this imposes a practical limit of around 4,000–4,900 words per text due to Firestore's field limit.

As an alternative, full text data could be stored in .json files within Firebase Storage. This increases document size and field count limits and can reduce Firestore usage costs. The downside is the loss of queryability on text content. However, if content-level queries are not essential, this may be a viable and cost-effective solution.[6]

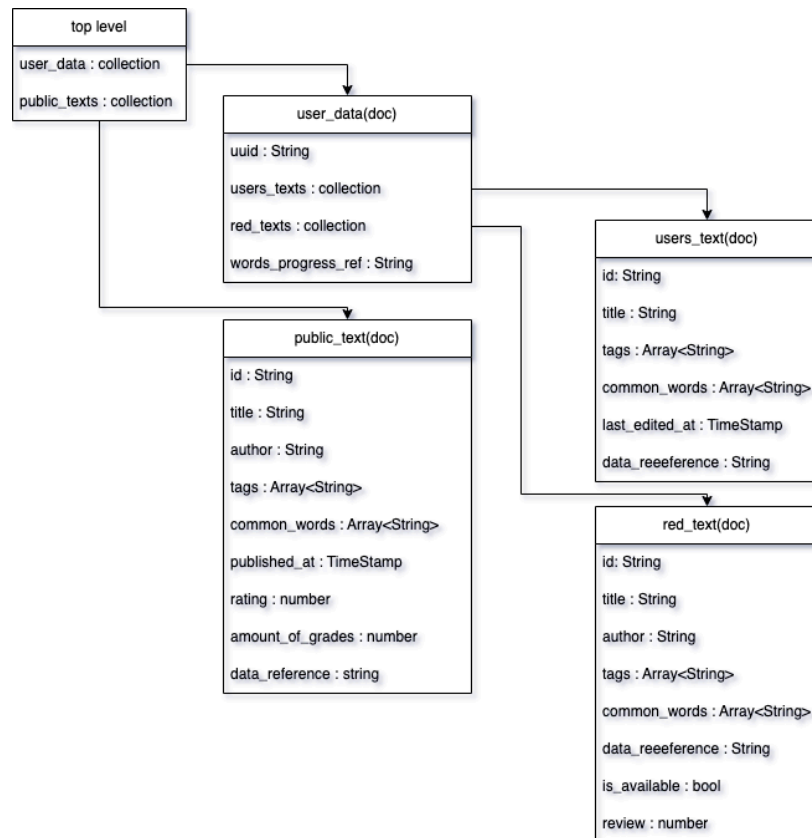


Figure 4.3 - Firestore structure

4.1.3 Firebase storage

In the same way as Firestore, when setting up Firebase Storage, we need to select a region where our data will be stored. Region also might heavily affect pricing as well as access speed. For this particular storage was selected Warsaw region because of combination of pricing and proximity to our initial audience. The region will not be the only similarity with Firestore. Firebase storage requires us to think about security rules and access limitations when setting up its structure. Correct file structure will simplify access to different parts of the file and potentially save us some Firestore *reads*.

The structure on the figure 4.4 will allow us to easily separate access to private data of each user while keeping the public text available. It also accounts for the possibility of future media, or other complementary files attached to the text.

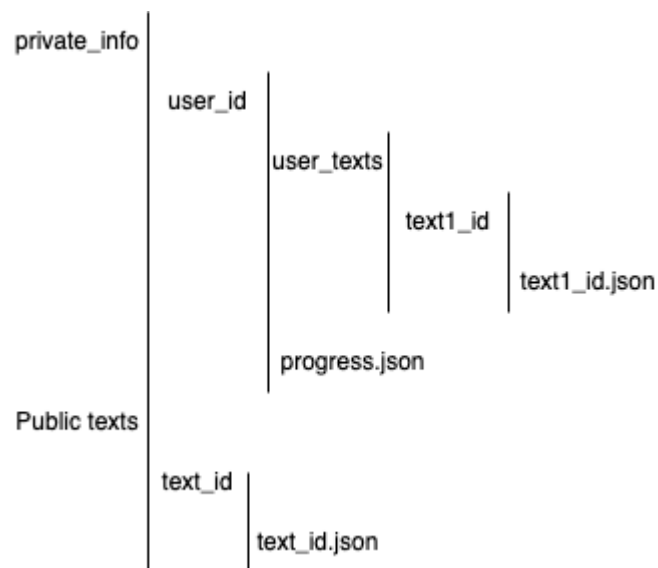


Figure 4.4 - Storage schema of Firebase storage

4.1.4 Firebase Firestore and Storage security rules

Given the sensitivity of user data and our intention to minimize data collection, strict access controls will be implemented:

- user_data documents:

Only accessible and editable by the authenticated user (verified via UUID).

- Documents with text info:

Publicly readable, but only editable or deletable by the original author. Each document stores the author field to verify ownership.

- Review System:

Users can submit review scores on texts via Cloud Functions. Each review updates the document's aggregated score and count. Once a review is submitted, the text is marked as "read" and added to the user's `read_texts` subcollection.

Privacy and Data Minimization

We adhere to a minimal data collection policy. No unnecessary personal data is stored. All user-specific data originates from and is limited to what Firebase Authentication provides. This eliminates the need for a separate user profile collection.

Structure summary

This database design ensures:

- Security: Fine-grained access control
- Scalability: Designed for mobile usage and cost
- Flexibility: Easily adaptable data model due to NoSQL nature
- Performance: Performant reads, minimal data transfer
- Privacy-first: No excessive user data collection

As the project evolves, the structure can be adjusted with minimal disruption thanks to Firestore's schema-less model.

4.1.5 Analytics and Crashlytics

We also will want to set up Crashlytics and Analytics to our project (fig. 4.5). Both will help us monitor app health and adjust the app structure to fit farther user needs. No special setup for this is required on the Firebase console and the data will be accessible automatically after the client app is set.

```
FlutterError.onError = FirebaseCrashlytics.instance.recordFlutterFatalError;
PlatformDispatcher.instance.onError = (error, stack) {
  FirebaseCrashlytics.instance.recordError(error, stack, fatal: true);
  return true;
};
FirebaseAnalytics.instance.logEvent(name: 'app_launch');
```

Figure 4.5 - General setup for Crashlytics and Analytics in Flutter app

4.2 Mobile app development

4.2.1 Use cases

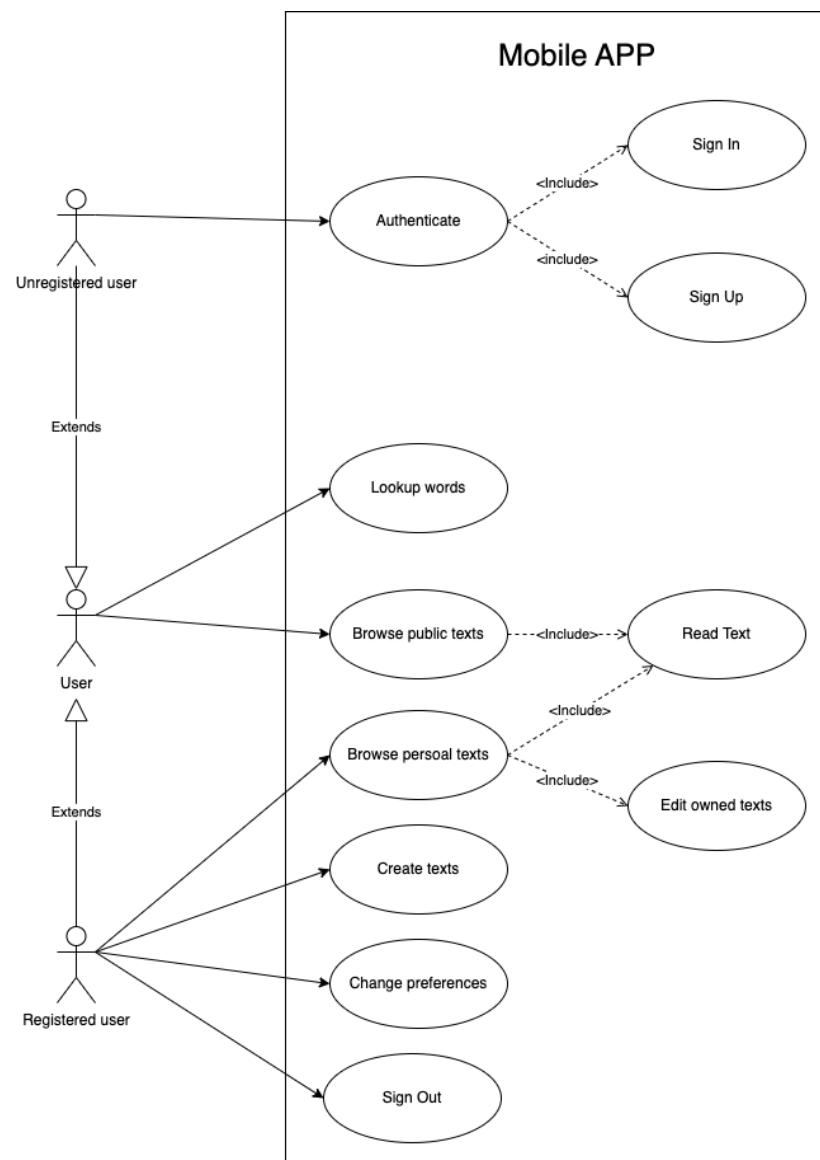


Figure 4.6 - Use case diagram of mobile app

As we can see from the use case diagram figure 4.6 we should have a bit different app behaviour depending on if the user is authenticated or not. Because a not authenticated user can't have any data connected to their account.

4.2.2 Project architecture

Proper architectural planning is essential for this project, as we are developing a complex system with multiple features and components that must function across various platforms—specifically iOS and Android. There is also a problem of code maintenance, as most developers spend a lot of time not on creating new features but maintaining old ones[13]. Therefore a correctly planned architecture can save a lot of development and maintenance time in the future.

To ensure a well-structured and maintainable codebase, it is important to apply software design principles such as Object-Oriented Programming (OOP) and the SOLID principles. Below is a brief overview of these concepts, followed by a proposed high-level architecture [14].

Object-Oriented Programming (OOP) Principles

1. Encapsulation

Groups related data (variables) and methods (functions) into a single unit called a class. It hides internal details and only exposes what is necessary through public methods.

2. Abstraction

Hides complex implementation details and displays only the essential features of an object. This reduces complexity and improves code clarity.

3. Inheritance

Allows a class (child or subclass) to inherit properties and behaviors from another class (parent or superclass), encouraging code reuse and logical structure.

4. Polymorphism

Enables objects to be treated as instances of their parent class, allowing different implementations of the same interface or method.

Solid principles will help to ensure stability of the application and modular structure.

SOLID Design Principles

1. Single Responsibility Principle (SRP)

A class should have only one responsibility or reason to change.

2. Open/Closed Principle (OCP)

Code should be open to extension but closed to modification. New functionality should be added without altering existing code.

3. Liskov Substitution Principle (LSP)

Subclasses should be replaceable by their parent classes without affecting the program's correctness.

4. Interface Segregation Principle (ISP)

Clients should not be forced to depend on interfaces they do not use. It is better to have several small, specific interfaces.

5. Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules. Both should rely on abstractions.

Given the need for flexibility, modularity, and maintainability, Clean Architecture is recommended. This approach divides the project into well-separated layers and organizes features in a scalable way (fig. 4.7)[15].

Project Structure [7]

1. Core/Common Module.

Contains shared components, utilities, and abstractions used across the entire application (e.g., error handling, base use cases).

2. Feature Modules

Each specific feature (e.g., authentication, reading, editing text) has its own folder, including three internal layers:

- Presentation Layer (UI Layer)

Includes the user interface and classes for managing UI state (e.g., ViewModels, Cubits, Controllers).

- Domain Layer

Contains business logic, use cases, and abstract definitions. It is independent of external frameworks and libraries.

- Data Layer

Handles data sources, services, and repository implementations. It defines models retrieved from APIs or local storage, which may differ from domain models.

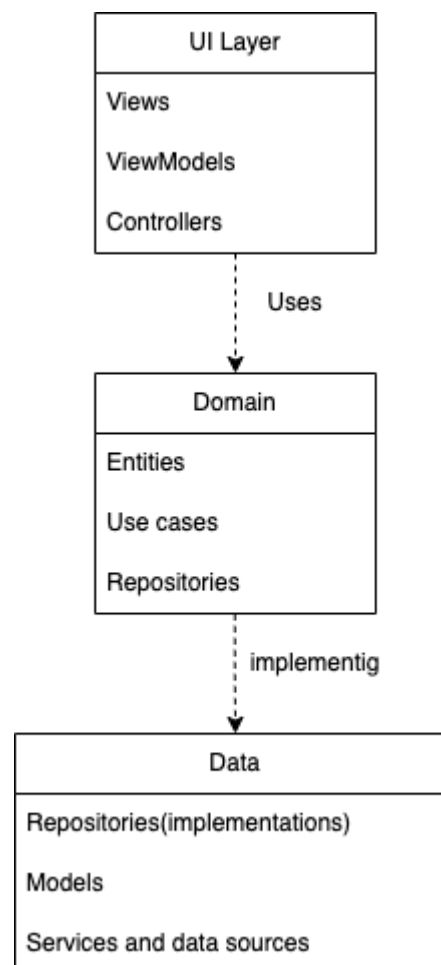


Figure 4.7 - Clean architecture feature scheme

3. Utility Modules

Folders such as navigation, localization, and theming provide app-wide support functions.

4.2.3 Authentication

Detailed class listing for authentication feature and its requirements can be found in Appendix A.

As we can see a huge part of our functionality is locked behind authentication. We also shouldn't allow users to get access to authentication functionality after the user is already signed in. So we need to create a system that manages authentication and update some UI elements according to the current status of the user. We will have an AuthBloc as a bloc responsible to handle authentication state inside our app. This bloc will have 3 states that will determine behaviour of the app:

- Initial state of the created bloc state for the moment of it creation, it will lasts only until the app manages to check if we can access user data from AuthRepo.
- SignedOut - state that is emitted when AuthRepo does not have any user's data. During this state all elements that contain user data will be replaced with placeholders and display messages asking user to sign in or create an account.
- SignedIn - state that emitted when AuthRepo was able to get information about a user. It also contains the user information provided by AuthRepo that allows us to build UI elements according to the user's current data.

On figure 4.8 we can see how the elements in our authentication are interacting during the sign in process.

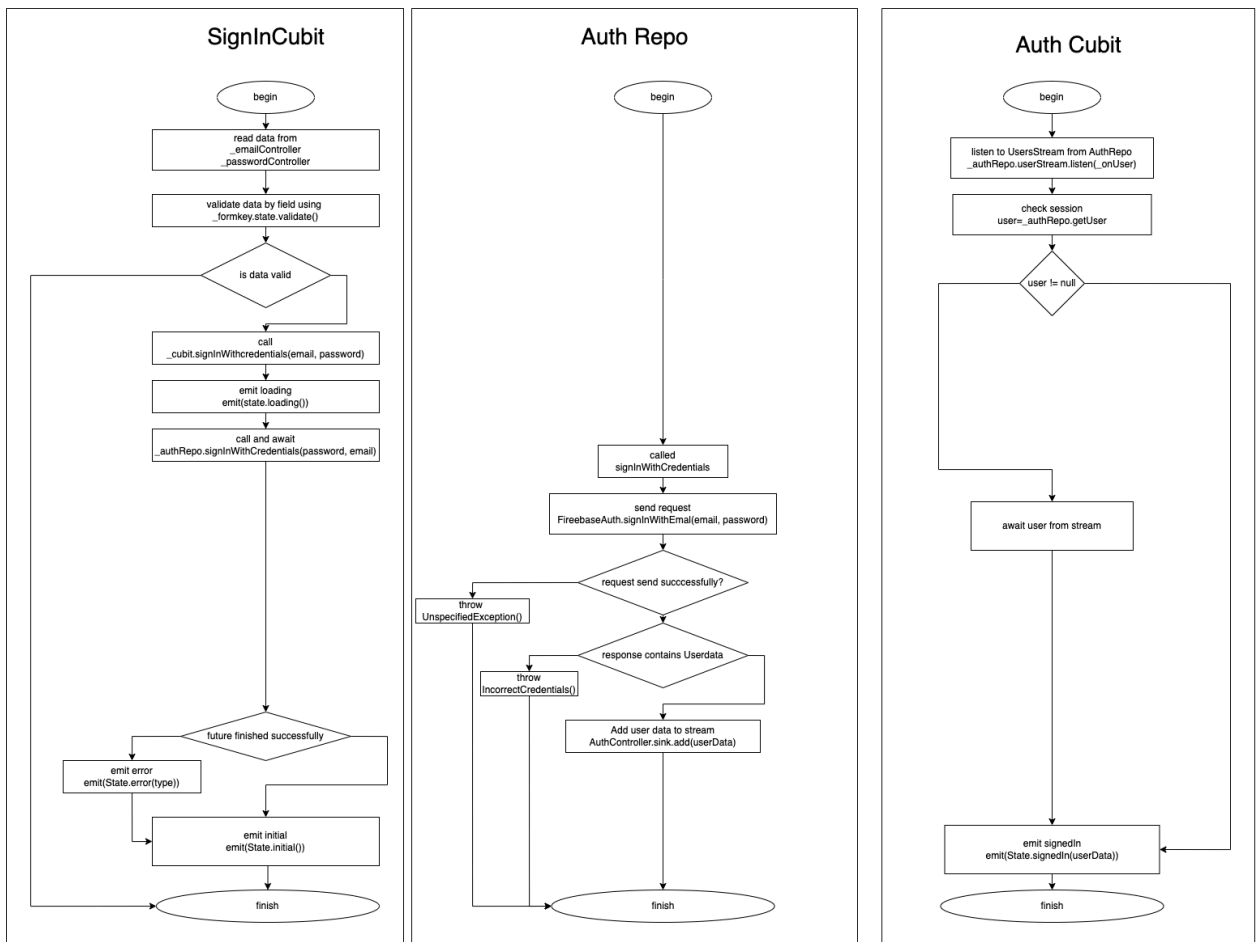


Figure 4.8 - Authentication via email full flow

SignIn cubit is responsible for gathering data from the UI elements, calling repo methods and handling the exceptions that might occur during this process. AuthRepo is the data-level class that handles requests. However in this case AuthRepo will not return data from the response to the SignIn cubit, instead it will only return a success event. The actual data will be put to the stream that AuthCubit is already listening to. If the authentication is successful AuthCubit will emit the SignedIn state and listeners connected to it will navigate us to corresponding stream, while its builders will update elements of the UI connected to the state of the user's authentication.

On the figure 4.9 we can see how the sign in UI is looking in our app. The UI contains *Email* field, *Password* field, button *Forgot password* that help user restore their password and *Continue* button. As we can see *Email* and *Password* fields have validation which is performed before we send any requests as well as

additional functionality and redirections. All of the validation in the application setup using regular expressions and based on common practices in mobile app development. If both fields passed validation, user can press Continue button and it will send them to the main screen of the app. Detailed class listing can be found in Appendix A.4-A.6.

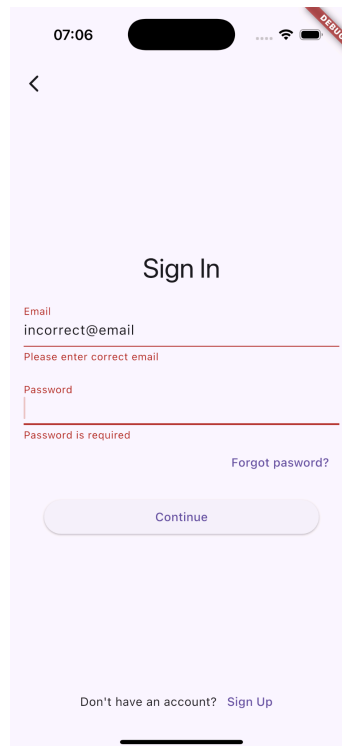


Figure 4.9 - Sign In UI

On the figure 4.10 we can see a sign up form of our application. As we follow guidelines about data privacy we are trying to collect as little as possible user's private information.

We ended up settling on collecting only the user's email - something connected to the user which allows them to keep access to the account even after forgetting password, and username - nickname that user is setting for themselves. It's needed to represent the author of the text as well as making the account a bit more personalized.

Figure 4.10 - Sign Up screen

After filling up this form request to Firebase Authentication server will be sent and if successful the user will be logged in automatically. Later we might also want to add email validation to make sure that the data user entered is valid and avoid possible problems in the future. Class listing provided in Appendix A.9 - A1.

Forgot password screen (fig. 4.11) is allowing users who forgot their password restore the access to their account using an email. After sending this request the user will receive an email with the link to a webpage on which they can re enter a new password. Detailed code listing for forgot password can be found in Appendix A13-A15.

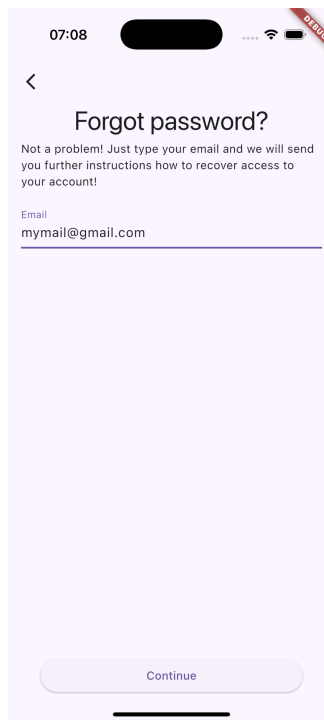


Figure 4.11 - Forgot password screen

4.2.4 Public texts UI and functionality

On the figure 4.12 we can see a UI for the first tab of our dashboard. This tab allows users to select texts to read. On the top of the screen we have a search bar that allows users to search through public texts based on text title. Next we have a tag menu that can be used to search texts by a specific tag. This menu will also have an 'Explore' button which redirects users to a more advanced tag menu.

For authorized users there will be a section with recommended text. This recommendation will be based on current user progress and contain 1 text based on the current data. Section is also has an 'More recommendations' button which will allow users to browse a selection of text that based on their current progress will be most fitted for them to study.

For unauthenticated users we will have a new text section, which contains the list of latest published text, and the tags this text has. When pressed on the text user will be redirected to the Reading screen where they can read the selected text.

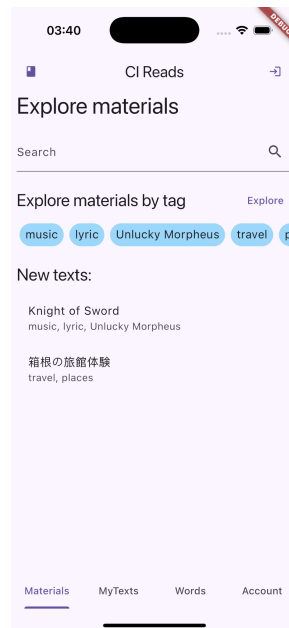


Figure 4.12 - Dashboard UI(Study materials tab)

Reading screen contains text that the user can read (fig. 4.13). Name of the text is showed at the top of the screen. The bar with sentence details is located at the bottom and opens when the user taps on any word or sentence.

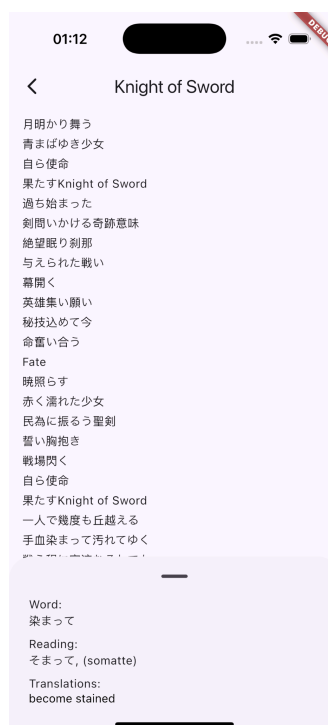


Figure 4.13 - Reading Screen UI

Depending on user settings some or all kanji(hieroglyphs that have morphologic meaning) can be displayed with furigana(reading helper above them) or entirely in kana(phonetic script). There is also a function to add spacing between the words to simplify reading for new learners.

On Reading Screen UI (fig. 4.13) users can tap on any word or sentence from the text. By simple tap the bottom bar with sentence details will be opened. There we can see the whole sentence as well as its translations and some grammar points related to it. By long tapping on any word, the bottom sheet will be opened. It will contain information about the word, its reading in kana(phonetic japanese script), romaji(latin script for representing reading) and its translation. There is also a button “Study” at this panel allowing users to add the word for studying manually. In the future updates will be added the function to listen the word’s pronunciation (using text to speech tools) as well as import this word for studying to outside apps like Anki.

4.2.5 My texts UI and functionality

On the figure 4.14 we can see a UI for the “My texts” tab.

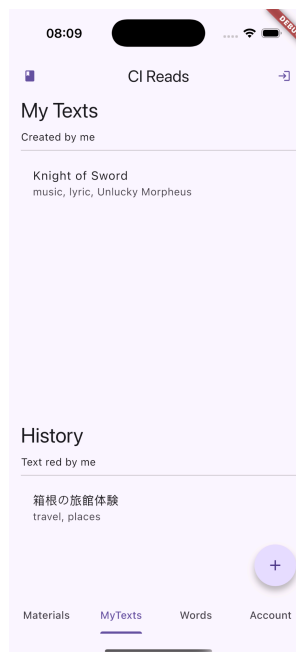


Figure 4.14 - My texts UI

This tab allows users to interact with text they created as well as text they read previously. In the upper section of the tab the user can see a list with texts they created previously, in the lower section they can view texts they studied. Dynamic and animated UI changes based on the section that users interacts with making it bigger and easier to navigate.

On this tab we also have a floating action button with ‘+’ icon. Pressing this button will navigate us to the start of the process of creating interactive text. Let’s first look at the algorithm of creating new text.

This tab is only available for users that are authenticated, otherwise content will be replaced with a placeholder we can see at figure 4.15. This placeholder contains 2 buttons that will lead to the Sign In or Sign Up screen accordingly.

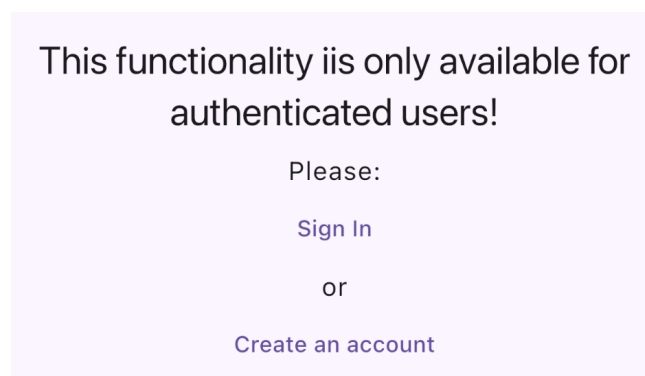


Figure 4.15 - Unauthorized placeholder

As we can see from figure 4.16 the text creation will be a multistep process, some of the steps are relatively independent, and some based on the result of the others. Therefore we will be splitting the process in 3 separate steps:

1. Entering base line text, and it’s metadata(currently only tags).
2. Splitting text into sentences and adding translation and comments for each one.
3. Splitting sentences into words, adding reading to each word, and individual translations.

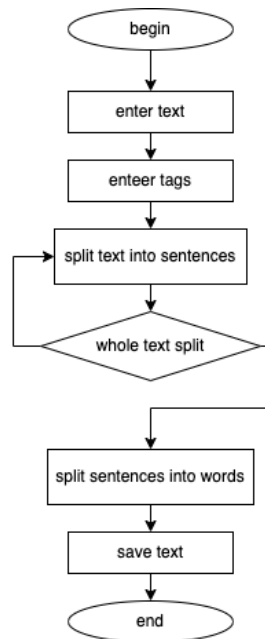


Figure 4.16 - Text creation algorithm

As we can see on figure 4.17 interface is displaying the step we are on, so users can have a distinct feeling of progress. Also there is a field to enter whole text and tags on this display.

The screenshot shows a mobile app interface for creating text. At the top, the status bar displays the time 11:12 and battery level. The app title is 'Create text' and it is 'Step 1 of 3'. The main text area contains several lines of Japanese lyrics. Below the text area is a 'Tags' section with a text input field containing 'music, lyrics, Wagakki band'. At the bottom is a 'Continue' button.

11:12

<

Create text

Step 1 of 3

謎々かぞえて遊びましょう
 タタタラタタララッタッタ
 何故何故此处で踊っているでしょう

簡単なことも解らないわ あたしって何だっ
 け
 それすら夜の手に絆されて 愛のように消え
 る
 さようならも言えぬ儘 泣いたフォニイ
 嘘に絡まっているあたしはフォニイ

何時しかその音がいやに鳴り合って
 色の目があなたを溶いている
 鏡に映るあたしを欠いて誰もが見間違った
 虚像
 如何して愛なんてものに群がりそれを欲して
 生きるのだ
 今日も泳いでいる夜の電車が通り去っていく
 踊り明かせよ

Tags

music, lyrics, Wagakki band

Continue

Figure 4.17 - Text entering screen

The field for text supports scrolling, as the text might have various lengths, allowing users to type freely and easily navigate the text. Tags can be entered in the lower field separated by the comma, later they will be parsed, new ones will be added to Firestore.

The second step of creating text will be splitting it into sentences (fig. 4.18).

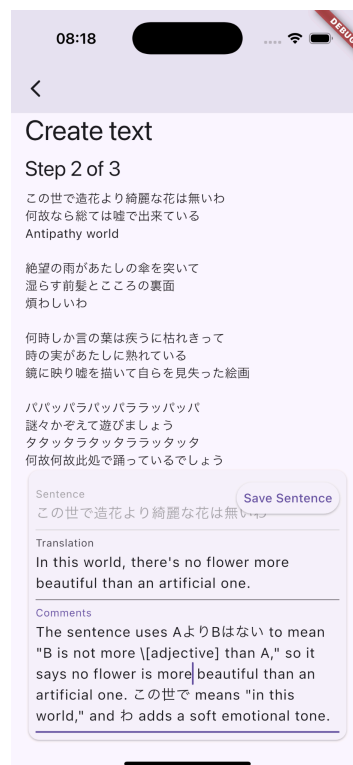


Figure 4.18 - Text parsing screen

User can select parts from the text that he's just entered. Upon selection the bottom sheet will appear which allows the user to add translation to the selected part, comment or grammar note and save it as a separate sentence. After this it will be displayed in a small list above the text, when the selected part will be removed from the original text. Users can remove items from the list to return it back to the original text if needed. This process will continue until all sentences are parsed.

Step 3 will repeat the algorithm for step 2, with the only difference that now the user should select a word from the existing sentences and fill up data regarding these words.

After this step users can save their text in Firebase storage and publish it or keep it private with an ability to publish it later.

4.2.6 User's vocabulary UI and functionality

The vocabulary page (fig. 4.19) provides us with the ability to review words that the user is currently studying as well as familiar kanjis (complex hieroglyphs). Two buttons on the top will redirect users to screens dedicated to respected sections while lists below show users' latest words/kanji addition to their collection.

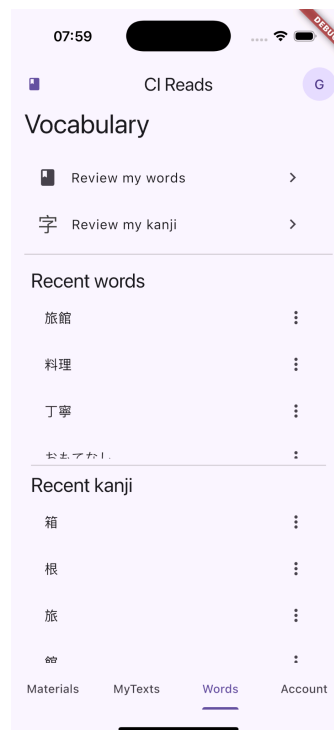


Figure 4.19 - Vocabulary UI

4.2.7 User's preference UI and functionality

The preference screen (fig. 4.20) allows users to review information about their account, manage the current session, and delete account. It also provides user with the link to resources connected to the service like ToS, privacy policy, link to the support page, and info about developer and representative contact info.

This page also has a menu of additional settings that allows users to customize app behaviour according to their need. Text settings help user customize fonts on the reading screen according to their preferences. It also provides users with an ability to change some default settings in text display like (furigana, spacing between words, translation).

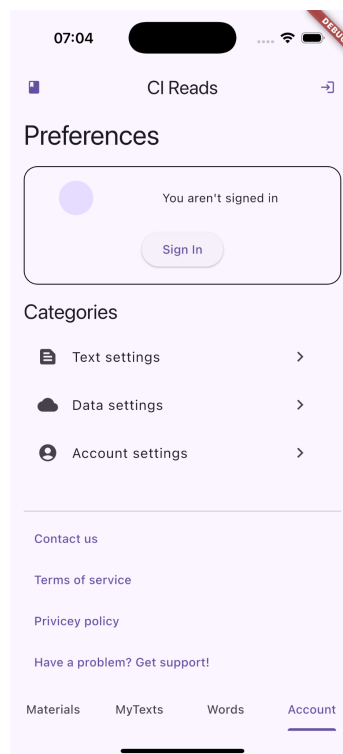


Figure 4.20 - Preference UI

4.2.8 Builds preparation

After the main functionality is finished we should start to think about preparing the first build for devices and testing our application. As the current state of the application represents proof of concept we could start with a local build for IOS without any specific signature and APK build for Android to save money or initial stores' fees. IOS builds with signature will only run on the devices which it is built for, there will be no sense to publish this build for others. As for android we will be able to use Firebase Hosting to store our builds history until we start deployment to the Play Store. Firebase Hosting allows us to easily add notes for

every build that is visible for testers, gather feedback, and manage the access across different tester groups, providing a flexible solution for distributing the app to testers.

4.2.9 Testing and analytics

After the application was deployed it was tested by several people across multiple device types. The main goal of this initial test was to gather information about how users interact with an application for us to be able to adjust data structures and UI, find problems in UX as well as catch bugs and crashes.

On the figure 4.21 we can see the percentage of crash-free sessions for every version of our app. As we can see the overall trend has a positive dynamic, but some versions can add additional issues that may negatively affect the stability of the app.

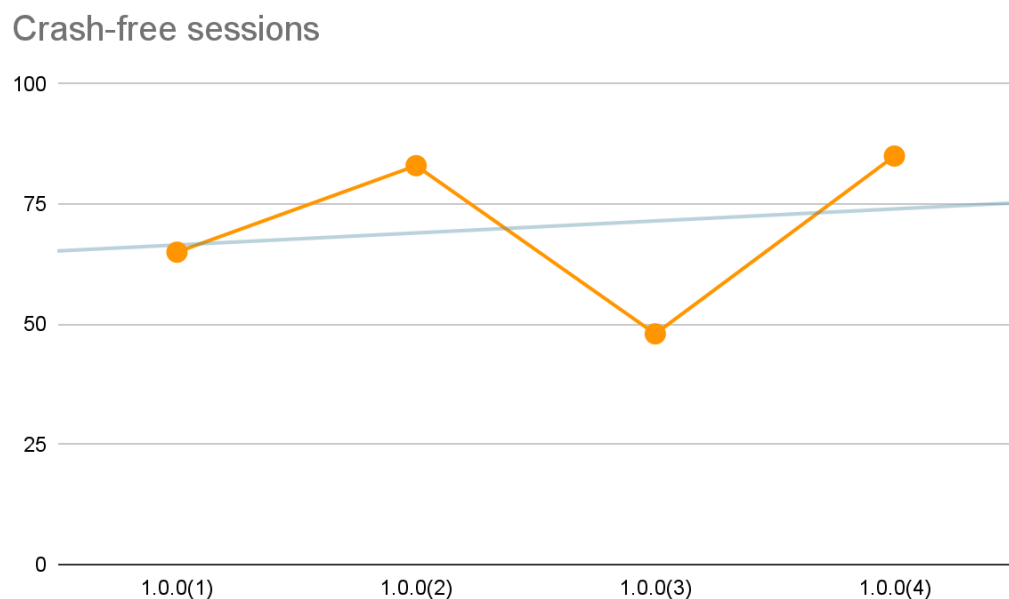


Figure 4.21 - Crashlytics stats and trend

CONCLUSIONS

During this work a mobile application was created for IOS and Android platforms. Based on the requirements analysis, we can say that the initial goal of the project was achieved. We also found crucial information about customers' interaction with the app that needs to be implemented before the application can be released to the end users. Further adjustments are required to the structure of several vital features (such as reviews, create text) as well as testing and bug fixing across the whole app.

Flutter framework proved to be easy to use and reliable. The app looks exactly the same on both Android and IOS platforms, without any difference in behavior. Choosing Dart and Flutter also offers the possibility to expand the app to support web and desktop platforms in the future, using the already existing code base.

Firebase proved to be a reliable tool in the app development and heavily simplified work with the backend of the app. It provided us with all necessary tools for authentication, data storage and a huge amount of tools that speed up development and help with analytics of the current application. However before releasing the app to the end client we still need to run a couple of closed tests to gain deeper insights and estimate costs of using it.

Results of the work were introduced at the International scientific and practical conference «Application of information technologies in the preparation and operation of law enforcement forces».

SOURCES

1. Banna R., Ikeda Y., Ohno Y., Shinagawa C., Takashiki K. Genki. 2nd ed. – Tokyo : Japan Times Publishing, 2011. – P. 20–30.
2. Krashen S. D. Principles and Practice in Second Language Acquisition. – University of Southern California. – P. 22–29.
3. In'nami Y., Koizumi R. The role of input in second language oral ability development in foreign language classrooms: A longitudinal study [Electronic resource] // ResearchGate. – Access mode: https://www.researchgate.net/publication/309284933_The_role_of_input_in_second_language_oral_ability_development_in_foreign_language_classrooms_A_longitudinal_study – Accessed: 14 March 2025.
4. Structure Your Data [Electronic resource] // Firebase Documentation. – Access mode: <https://firebase.google.com/docs/firestore/manage-data/structure-data> – Accessed: 16 April 2025.
5. Firebase for Flutter [Electronic resource]. – Access mode: <https://firebase.flutter.dev/docs/overview> – Accessed: 16 March 2025.
6. Firebase Pricing [Electronic resource]. – Access mode: <https://firebase.google.com/pricing> – Accessed: 3 April 2025.
7. Cloud Firestore Overview [Electronic resource]. – Access mode: <https://firebase.google.com/docs/firestore> – Accessed: 3 April 2025.
8. Flutter Clean Architecture & TDD [Electronic resource] // Reso Coder. – Access mode: <https://resocoder.com/flutter-clean-architecture-tdd/> – Accessed: 14 March 2025.
9. BLoC Library [Electronic resource]. – Access mode: <https://bloclibrary.dev/> – Accessed: 14 March 2025.
10. S.O.L.I.D. – The First Five Principles of Object-Oriented Design [Electronic resource] // DigitalOcean. – Access mode:

- <https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design> – Accessed: 14 March 2025.
11. Flutter Documentation [Electronic resource]. – Access mode: <https://flutter.dev/> – Accessed: 18 March 2025.
 12. The State of Developer Ecosystem 2024 [Electronic resource] // JetBrains. – Access mode: <https://www.jetbrains.com/lp/devecosystem-2024/> – Accessed: 25 May 2025.
 13. McConnell S. Code Complete. 2nd ed. – Microsoft Press, 2004.
 14. Shvets A. Dive Into Design Patterns. – Refactoring.guru, 2020.
 15. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2017.
 16. Cloud Firestore Pricing [Electronic resource] // Google Cloud. – Access mode: <https://cloud.google.com/firestore/pricing> – Accessed: 15 May 2025.
 17. Kysil H.A., Dyadun S.V. Development of an information system for interactive text translation for languages which use hieroglyphs // International scientific and practical conference «Application of information technologies in the preparation and operation of law enforcement forces». – Kharkiv, 2025, p. 85-86.

Appendix A

Authentication feature and core files

Listing A.1 main.dart

```
import 'dart:ui';

import 'package:ci_reading/di.dart';
import 'package:ci_reading/firebase_options.dart';
import 'package:ci_reading/navigation/app_router.dart';
import 'package:ci_reading/theme/theme.dart';
import 'package:firebase_analytics/firebase_analytics.dart';
import 'package:firebase_core/firebase_core.dart';
import
'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:flutter/material.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  setupDi();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );

                                FlutterError.onError                                =
FirebaseCrashlytics.instance.recordFlutterFatalError;
  PlatformDispatcher.instance.onError = (error, stack) {
    FirebaseCrashlytics.instance.recordError(error, stack,
fatal: true);
    return true;
  };
  FirebaseAnalytics.instance.logEvent(name: 'app_launch');

  runApp(MyApp());
}

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

  // This widget is the root of your application.
  final AppRouter _appRouter = AppRouter();

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      title: 'socket_practice',
      theme: AppThemes.light,
      routerConfig: _appRouter.config(),
    );
  }
}
```

```

    );
  }
}

```

Listing A.2 auth_wrapper.dart

```

import 'package:auto_route/auto_route.dart';
import 'package:ci_reading/di.dart';
import
'package:ci_reading/feat/auth/domain/repo/auth_repo.dart';
import
'package:ci_reading/feat/auth/view/forgot_password/cubit/forgot_p
assword_cubit.dart';
import
'package:ci_reading/feat/auth/view/sign_in/cubit/sign_in_cubit.da
rt';
import
'package:ci_reading/feat/auth/view/sign_up/cubit/sign_up_cubit.da
rt';

import 'package:ci_reading/navigation/app_router.gr.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import 'cubit/auth_cubit.dart';

@RoutePage()
class AuthWrapper extends StatelessWidget implements
AutoRouteWrapper {
  const AuthWrapper({super.key});

  @override
  Widget build(BuildContext context) {
    return const AutoRouter();
  }

  @override
  Widget wrappedRoute(BuildContext context) {
    return BlocListener<AuthCubit, AuthState>(
      listener: (context, state) {
        final result = switch(state){
          SignedIn()=> context.navigateTo(const HomeRoute()),
          _ => null,
        };
      },
      child: MultiBlocProvider(
        providers: [
          BlocProvider(create: (context) =>
SignUpCubit(getIt.get<AuthRepo>()))),
          BlocProvider(create: (context) =>
SignInCubit(getIt.get<AuthRepo>()))),
          BlocProvider(create: (context) =>
ForgotPasswordCubit(getIt.get<AuthRepo>()))),
        ],
        child: this,
      ),
    ),
  ),

```

```

    );
  }
}

```

Listing A.3 page_is_unavailable.dart

```

import 'package:auto_route/auto_route.dart';
import 'package:ci_reading/localization/strings.dart';
import 'package:ci_reading/navigation/app_router.gr.dart';
import 'package:ci_reading/theme/app_dimensions.dart';
import 'package:flutter/material.dart';

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

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return SafeArea(
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            Text(
              Strings.signInFilterMessage,
              style: theme.textTheme.headlineSmall,
              textAlign: TextAlign.center,
            ),
            const SizedBox(height: AppDimensions.paddingS),
            Text(
              Strings.please,
              style: theme.textTheme.bodyLarge,
            ),
            TextButton(
              onPressed: () {
                context.pushRoute(const SignInRoute());
              },
              child: Text(Strings.signIn),
            ),
            Text(
              Strings.or,
              style: theme.textTheme.bodyLarge,
            ),
            TextButton(
              onPressed: () {
                context.pushRoute(const SignUpRoute());
              },
              child: Text(Strings.createAnAccount),
            ),
          ],
        ),
      ),
    );
  }
}

```

```

    }
  }

```

Listing A.4 sign_in_screen.dart

```

import 'package:auto_route/auto_route.dart';
import 'package:ci_reading/core/helpers/app_reg_exp.dart';
import 'package:ci_reading/feat/auth/cubit/auth_cubit.dart';
import
'package:ci_reading/feat/auth/view/sign_up/cubit/sign_up_cubit.da
rt';

import 'package:ci_reading/navigation/app_router.gr.dart';
import 'package:ci_reading/theme/app_dimensions.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

@RoutePage()
class SignInScreen extends StatefulWidget {
  const SignInScreen({super.key});

  @override
  State<SignInScreen> createState() => _SignInScreenState();
}

class _SignInScreenState extends State<SignInScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return BlocListener<AuthCubit, AuthState>(
      listener: (context, state) {
        final result = switch(state){
          SignedIn() => context.navigateTo(const
HomeRoute()),
          _ => null,
        };
      },
      child: Scaffold(
        appBar: AppBar(
          leading: BackButton(
            onPressed: () {
              context.navigateTo(const HomeRoute());
            },
          ),
        ),
        body: BlocConsumer<SignUpCubit, SignUpState>(
          listener: (context, state){
            final result = switch (state) {
              Error() =>
                showDialog(

```



```

    }
    return null;
  },
),
const SizedBox(height:
AppDimensions.paddingM),
TextFormField(
  obscureText: true,
  controller: _passwordController,
  keyboardType:
TextInputType.visiblePassword,
  decoration: const InputDecoration(
    labelText: 'Password',
  ),
  validator: (value) {
    if (value == null || value.isEmpty)
    {
      return 'Password is required';
    }
    if
    (!AppRegExp.passwordValidation.hasMatch(value)) {
      return 'Password must contain at
least 8 characters, '
      ' 1 lower case character, 1
upper case and 1 number';
    }
    return null;
  },
),
Row(
  mainAxisAlignment:
MainAxisAlignment.end,
  children: [
    TextButton(
      onPressed: () {
        context.pushRoute(const
ForgotPasswordRoute());
      },
      child: const Text('Forgot
password?'),
    ),
  ],
),
const SizedBox(height:
AppDimensions.paddingM),
Container(
  width: double.infinity,
  padding: const EdgeInsets.symmetric(
    horizontal: AppDimensions.paddingL,
  ),
  child: ElevatedButton(
    onPressed: () {

```

```

(_formKey.currentState?.validate() ?? false) {
    //do sign upnb
  },
  child: const Text('Continue'),
),
),
const Spacer(),
Row(
    mainAxisAlignment:
MainAxisAlignment.center,
    children: [
      const Text('Don\'t have an
account?'),
      TextButton(
        onPressed: () {
          context.pushRoute(const
SignUpRoute());
        },
        child: const Text('Sign Up'),
      ),
    ],
  ),
),
),
),
),
);
},
),
);
}
}

```

Listing A.5 sign_in_cubit.dart

```

import 'package:bloc/bloc.dart';
import
'package:ci_reading/feat/auth/domain/repo/auth_repo.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

part 'sign_in_state.dart';
part 'sign_in_cubit.freezed.dart';

class SignInCubit extends Cubit<SignInState> {
  SignInCubit(this._authRepo) : super(const
SignInState.initial());

  final AuthRepo _authRepo;

  Future<void> signIn(String email, String password) async{

```

```

    try{
      emit(const SignInState.loading());
      await _authRepo.signIn(email, password);
      emit(const SignInState.success());
    }on Exception catch(e){
      emit(const SignInState.error());
    }finally{
      emit(const SignInState.initial());
    }
  }
}

```

Listing A.6 sign_up_state.dart

```

part of 'sign_in_cubit.dart';
@freezed
sealed class SignInState with _$SignInState {
  const factory SignInState.initial() = Initial;
  const factory SignInState.loading() = Loading;
  const factory SignInState.success() = Success;
  const factory SignInState.error() = Error;
}

```

Listing A.7 auth_repo.dart

```

import 'package:firebase_auth/firebase_auth.dart';

abstract class AuthRepository {
  Stream<User?> get statusStream;

  Future<User> get user;

  Future<void> logInWithCredentials(String email, String
password);

  Future<void> signUp(String email, String password, String
nickname);

  Future<void> logOut();

  Future<void> requestPasswordReset(String email);

  Future<void> resetPassword(String code, String
newPassword);

  void close();
}

```

Listing A.8 firebase_auth_repo.dart

```

import 'dart:async';

```

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import
'package:ci_reading/feat/auth/domain/repo/auth_repo.dart';

class FirebaseAuthRepository extends AuthRepository {
  final FirebaseAuth firebaseAuth;
  final StreamController<User?> _statusController =
StreamController();
  late final StreamSubscription _authStateSubscription;

  FirebaseAuthRepository({
    required this.firebaseAuth,
  }) {
    _authStateSubscription =
firebaseAuth.authStateChanges().listen((event) {
      _statusController.add(event);
    });
  }

  User? get user => firebaseAuth.currentUser;

  @override
  Future<void> loginWithCredentials(String email, String
password) async {
    final result = await
firebaseAuth.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
  }

  @override
  Future<void> logout() async {
    await firebaseAuth.signOut();
  }

  @override
  Stream<User?> get statusStream => _statusController.stream;

  @override
  Future<void> signUp(String email, String password, String
nickname) async {
    final result = await
firebaseAuth.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );
    result.user?.updateDisplayName(nickname);
  }

  void dispose() {
    _authStateSubscription.cancel();
  }

```

```

    _statusController.close();
  }

  @override
  Future<void> requestPasswordReset(String email) async {
    await firebaseAuth.sendPasswordResetEmail(email: email);
  }

  @override
  void close() {
    _authStateSubscription.cancel();
  }
}

```

Listing A.9 sign_up_screen.dart

```

import 'package:auto_route/auto_route.dart';
import 'package:ci_reading/core/helpers/app_reg_exp.dart';
import 'package:ci_reading/feat/auth/cubit/auth_cubit.dart';
import
'package:ci_reading/feat/auth/view/sign_up/cubit/sign_up_cubit.da
rt';
import 'package:ci_reading/navigation/app_router.gr.dart';
import 'package:ci_reading/theme/app_dimensions.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

@RoutePage()
class SignUpScreen extends StatefulWidget {
  const SignUpScreen({super.key});

  @override
  State<SignUpScreen> createState() => _SignUpScreenState();
}

class _SignUpScreenState extends State<SignUpScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _nicknameController = TextEditingController();
  final _repeatPasswordController = TextEditingController();
  late final SignUpCubit _cubit;

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    _cubit = context.read<SignUpCubit>();
  }

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Scaffold(

```

```

    appBar: AppBar(
      leading: const BackButton(
        // onPressed: (){
        //   context.pop();
        // },
      ),
    ),
    body: BlocConsumer<SignUpCubit, SignUpState>(
      listener: (context, state) {
        final result = switch (state) {
          Error() =>
            showDialog(
              context: context,
              builder: (context) =>
                AlertDialog(
                  title: const Text('Error'),
                  content: const Text('Something went
wrong'),
                  actions: [
                    TextButton(
                      onPressed: () {
                        context.pop();
                      },
                      child: const Text('Ok'),
                    ),
                  ],
                ),
          _ => null,
        };
      },
      builder: (context, state) {
        return SafeArea(
          child: Padding(
            padding: const EdgeInsets.symmetric(
              horizontal: AppDimensions.paddingM,
            ),
            child: Form(
              key: _formKey,
              child: Column(
                mainAxisAlignment:
CrossAxisAlignment.start,
                crossAxisAlignment:
MainAxisAlignment.center,
                children: [
                  Text(
                    'Sign Up',
                    style: theme.textTheme.headlineLarge,
                  ),
                  const SizedBox(height:
AppDimensions.paddingM),
                  TextFormField(
                    controller: emailController,

```

```

                                decoration: const
InputDecoration(labelText: 'Email'),
                                validator: (value) {
                                    if (value == null || value.isEmpty) {
                                        return 'Email is required';
                                    }
                                }
                                if
(!AppRegExp.emailValidation.hasMatch(value)) {
                                    return 'Please enter correct
email';
                                }
                                return null;
                            },
                        ),
                                const SizedBox(height:
AppDimensions.paddingM),
TextFormField(
    controller: _nicknameController,
    decoration: const InputDecoration(
        labelText: 'Username',
        errorMaxLines: 2,
    ),
    validator: (value) {
        if (value == null || value.isEmpty) {
            return 'Username is required';
        }
    }
                                if
(!AppRegExp.passwordValidation.hasMatch(value)) {
                                    return 'Username must consist from
letters and numbers, '
                                    'and be at least 4 character
long';
                                }
                                return null;
                            },
                        ),
                                const SizedBox(height:
AppDimensions.paddingM),
TextFormField(
    obscureText: true,
    controller: _passwordController,
    decoration: const InputDecoration(
        labelText: 'Password',
        errorMaxLines: 2,
    ),
    validator: (value) {
        if (value == null || value.isEmpty) {
            return 'Password is required';
        }
    }
                                if
(!AppRegExp.passwordValidation.hasMatch(value)) {
                                    return 'Password must contain at
least 8 characters,'

```

```

' 1 lower case character, 1
upper case and 1 number';
    }
    return null;
  },
),
                                const SizedBox(height:
AppDimensions.paddingM),
TextFormField(
  obscureText: true,
  controller: _repeatPasswordController,
  decoration: const InputDecoration(
    labelText: 'Repeat password',
  ),
  validator: (value) {
    if (_repeatPasswordController.text !=
      _passwordController.text) {
      return 'Passwords don\'t match';
    }
    return null;
  },
),
const Spacer(),
Row(
                                mainAxisAlignment:
MainAxisAlignment.center,
    children: [
      Checkbox(
        value: true,
        onChanged: (value) {},
      ),
      const Text('I agree to'),
      TextButton(
        onPressed: () {},
        child: const Text('Term of
Service'),
      ),
    ],
  ),
                                const SizedBox(height:
AppDimensions.paddingM),
Container(
  width: double.infinity,
  padding: const EdgeInsets.symmetric(
    horizontal: AppDimensions.paddingL,
  ),
  child: ElevatedButton(
    onPressed: () {
                                if
    (_formKey.currentState?.validate() ?? false) {
      _cubit.signUp(
        _emailController.text,
        _passwordController.text,

```

```

        _nicknameController.text,
      );
    },
    child: const Text('Continue'),
  ),
),
],
),
),
),
),
);
},
),
);
}
}

```

Listing A.10 sign_up_cubit.dart

```

import 'package:bloc/bloc.dart';
import
'package:ci_reading/feat/auth/domain/repo/auth_repo.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

part 'sign_up_state.dart';
part 'sign_up_cubit.freezed.dart';

class SignUpCubit extends Cubit<SignUpState> {
  SignUpCubit(this._authRepo) : super(const
SignUpState.initial());

  final AuthRepo _authRepo;

  Future<void> signUp(String email, String password, String
username) async{
    try{
      emit(const SignUpState.loading());
      await _authRepo.signUp(email,password, username);
      emit(const SignUpState.success());
    }on Exception catch(e){
      emit(const SignUpState.error());
    }finally{
      emit(const SignUpState.initial());
    }
  }
}

```

Listing A.12 sign_up_state.dart

```

part of 'sign_up_cubit.dart';

```

```

@freezed
sealed class SignUpState with _$SignUpState {
  const factory SignUpState.initial() = Initial;
  const factory SignUpState.loading() = Loading;
  const factory SignUpState.success() = Success;
  const factory SignUpState.error() = Error;
}

```

Listing A.13 forgot_password_screen.dart

```

import 'package:auto_route/annotations.dart';
import 'package:auto_route/auto_route.dart';
import 'package:ci_reading/core/helpers/app_reg_exp.dart';
import
'package:ci_reading/feat/auth/view/forgot_password/cubit/forgot_p
assword_cubit.dart';
import 'package:ci_reading/navigation/app_router.gr.dart';
import 'package:ci_reading/theme/app_dimensions.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

@RoutePage()
class ForgotPasswordScreen extends StatefulWidget {
  const ForgotPasswordScreen({super.key});

  @override
  State<ForgotPasswordScreen> createState() =>
  _ForgotPasswordScreenState();
}

class _ForgotPasswordScreenState extends
State<ForgotPasswordScreen> {
  final _emailController = TextEditingController();
  final _validationKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Scaffold(
      appBar: AppBar(),
      body: BlocConsumer<ForgotPasswordCubit,
ForgotPasswordState>(
        listener: (context, state) {
          final result = switch (state) {
            Error() =>
              showDialog(
                context: context,
                builder: (context) =>
                  AlertDialog(
                    title: const Text('Error'),
                    content: const Text('Something went
wrong'),

```

```

        actions: [
          TextButton(
            onPressed: () {
              context.pop();
            },
            child: const Text('Ok'),
          ),
        ],
      ),
    ),
    Success() => () {
      context.navigateTo(const SignInRoute());
    },
    _ => null,
  };
},
builder: (context, state) {
  return SafeArea(
    child: Padding(
      padding: const EdgeInsets.symmetric(
        horizontal: AppDimensions.paddingM,
      ),
      child: Column(
        children: [
          Text(
            'Forgot password?',
            style: theme.textTheme.headlineLarge,
          ),
          const SizedBox(height:
AppDimensions.paddingXS),
          const Text(
            'Not a problem! Just type your email and
we will send you '
            'further instructions how to recover
access to your '
            'account!',
          ),
          const SizedBox(height:
AppDimensions.paddingM),
          Form(
            key: _validationKey,
            child: TextFormField(
              controller: _emailController,
              decoration: const InputDecoration(
                labelText: 'Email',
              ),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Email is required';
                }
              },
            ),
          ),
        ],
      ),
    ),
  );
}
if (!AppRegExp.emailValidation.hasMatch(value)) {

```

```

                                return 'Please enter correct
email';
                                }
                                return null;
                                },
                                ),
                                ),
                                const Spacer(),
                                Container(
                                  width: double.infinity,
                                  padding: const EdgeInsets.symmetric(
                                    horizontal: AppDimensions.paddingL,
                                  ),
                                  child: ElevatedButton(
                                    onPressed: () {
if
(_validationKey.currentState?.validate() ?? false) {
//do sign up
}
},
                                  child: const Text('Continue'),
                                ),
                                ),
                                ],
                                ),
                                ),
                                ),
                                );
                                },
                                ),
                                );
                                }
                                }

```

Listing A.14 forgot_password_cubit.dart

```

import 'package:bloc/bloc.dart';
import
'package:ci_reading/feat/auth/domain/repo/auth_repo.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

part 'forgot_password_state.dart';
part 'forgot_password_cubit.freezed.dart';

class ForgotPasswordCubit extends Cubit<ForgotPasswordState>
{
  ForgotPasswordCubit(this._authRepo) : super(const
ForgotPasswordState.initial());

  final AuthRepo _authRepo;

  Future<void> resetPassword(String email) async{

```

```

    try{
      emit(const ForgotPasswordState.loading());
      await _authRepo.forgotPassword(email);
      emit(const ForgotPasswordState.success());
    }on Exception catch(e){
      emit(const ForgotPasswordState.error());
    }finally{
      emit(const ForgotPasswordState.initial());
    }
  }
}

```

Listing A.15 forgot_password_state.dart

part of 'forgot_password_cubit.dart';

```

@freezed
sealed class ForgotPasswordState with _$ForgotPasswordState
{
  const factory ForgotPasswordState.initial() = Initial;
  const factory ForgotPasswordState.loading() = Loading;
  const factory ForgotPasswordState.success() = Success;
  const factory ForgotPasswordState.error() = Error;
}

```