Haaga-Helia

A flashcard mobile application development with Flutter

Tung Huynh

Haaga-Helia University of Applied Sciences

Bachelor's Thesis

2021

Bachelor of Business Administration

Abstract

Author(s)
Tung Huynh

Degree
Bachelor of Business Administration

Report/thesis title
A flashcard mobile application development with Flutter

Number of pages and appendix pages
30 + 0

This product thesis is an overview of the development process of a flashcard mobile application with Flutter. Flutter is a framework that help developers to develop applications that can run on mobile, web and desktop from one codebase. The goal of this thesis is to develop a minimum viable product of the Memoa flashcard application and use it as the foundation for future improvement. This thesis will discuss the theoretical background of the technologies, algorithms, concepts and methodologies used in the project. An Agile model is applies as the project management methodology for this project.

The thesis consists of five chapters. First is the introduction of the thesis as well as the project, then the theoretical framework is discussed in the second part. Next, the author explains the methodological choice and the implementation plan of the project in the third chapter. Then the fourth chapter describes the key features of the practical implementation of the project, from preparation of the tools, database design, backend implementation, Flutter development, to the release of the test version on Google Play Store. Finally, the last chapter discusses the thesis result, evaluation of the author performance as well as his own personal development throughout the project.

In the end, a test version of the application was published to Google Play Store as planned. The test version was stable enough to be the foundation for future development and this is considered as an initial success of the project.

Keywords
Flutter, Firebase, Mobile Application, Flashcard

Table of contents

1 Introduction

Learning a new language is not easy, especially for university students and workers who do not have much free time to go to language classes. People with this background usually resort to online courses, and it happens that the knowledge they learn during these courses' sessions does not have a reasonable retention rate. People progress through online courses at their own pace and tend to put off learning.

One reason is that people usually lack the motivation to learn or simply do not remember to. This project aims to change that by proposing a mobile application that encourages users to learn and offers them a playful experience. First, the learning mechanism is integrated with spaced repetition algorithm to remind users to review what they learned at an appropriate interval. Users can navigate through the app with ease as it is designed to optimize the user experience. Finally, a virtual pet will accompany the users as they progress with their learning.

1.1 Project objectives

This project aims to create and promote a Flashcard mobile application with gamification features optimized for user experience, achieved by a beautiful user interface and science-backed learning algorithm. This project is carried out using Flutter as the primary technology for creating the Memoa flashcard mobile application. The application needs to run on multiple platforms with comparable performance with natively developed applications.

The measurable result would be a functional minimum viable product (MVP) that can present the test users with a smooth experience to collect user test results. The gathered results will be used to improve and develop the product for public releases in the future. The result would include a high-fidelity design of the application and a functional Flutter application with core features uploaded to Google Play Store for internal testing.

The project creates an opportunity for the author to learn about Flutter and its programming pattern, project management skills, and teamwork.

1.2 Out of scope

This thesis will only discuss the project's technical aspects, such as the database design, programming patterns, data structures, and algorithms. Some elements regarding user experience, user interface, or style guide could be mentioned, but the designs and concepts which are not related to the technical side will not be included.

1.3 Thesis structure

In this thesis, the theoretical framework will be covered first to provide the readers with a sufficient understanding of the terminologies and the technologies used. Then the project background is covered in the next part. After that, the practical implementation of the product will be discussed. Finally, the thesis concludes with the project result and some reflection on the overall process.

The theoretical framework chapter includes the definition of flashcard, gamification, and an overview of the technical infrastructure. The technical infrastructure subchapter will provide the reader with background information on the technologies used in the project, such as the Flutter framework, Dart programming language, and Google Firebase services.

The project background chapter explains the author's decision to choose the project management model and plan to implement the mobile application according to the product requirements.

The fourth chapter describes the development process of the application using Flutter and other necessary services. Its subchapters will cover the tools used, steps to designing the database, backend and frontend development, and the release of the test version to Google Play Store.

Lastly, the result of the project is discussed to evaluate the project outcomes and the author's professional improvement.

2 Theoretical framework

In this chapter, the author provides the theories that support the foundation of the project. The theoretical framework includes research of flashcards, spaced repetition techniques, gamification, and technical infrastructure.

2.1 Flashcards

Flashcards are a popular study tool that helps learners test and improves their memory when learning new information. A typical flashcard has two sides, with a question written on one side and the corresponding answer on the other side. For instance, the card's front side may ask, "What is Flutter," the backside provides the answer "UI framework for cross-platform application creation." Flashcards are more likely to come as a deck (a group of several cards) that store information in the same category. For example, learners may have different categories, such as Finnish language, finance terms, or body parts, thereby having various flashcards at hand. (University of Southern Maine, 2021.)

Flashcards are effective as they stimulate "active recall" when learners attempt to remember the answer on the backside. Instead of reading from textbooks and trying to memorize all information, flashcards "isolate" learners from supporting sources and ask them to retrieve the knowledge from scratch. The more learners use flashcards for self-test purposes, the stronger neuron connections will become. A research conducted in 2008 found that the repeated retrieval practice helps improve long-term retention up to 150%. Besides, flashcards constantly ask learners to compare their answers with written answers on flashcards, which forms constant self-reflection. This strategy helps learners focus on their weaknesses and study better next time. (Brainscape 2021.)

2.2 Spaced repetition and Leitner system

Before discussing spaced repetition in detail, it is beneficial to understand the forgetting curve as it has a close relationship with how spaced repetition technique can help learners remember longer. The for-getting curve was discovered by Herman Ebbinghaus - a German psychologist - in the late 19th century. He found that the information learned on the first day will disappear over time at a remarkable rate. Some attributes to this phenomenon are the memory's strength and the time passed since the information was first learned. (eLearning Industry 2018.)

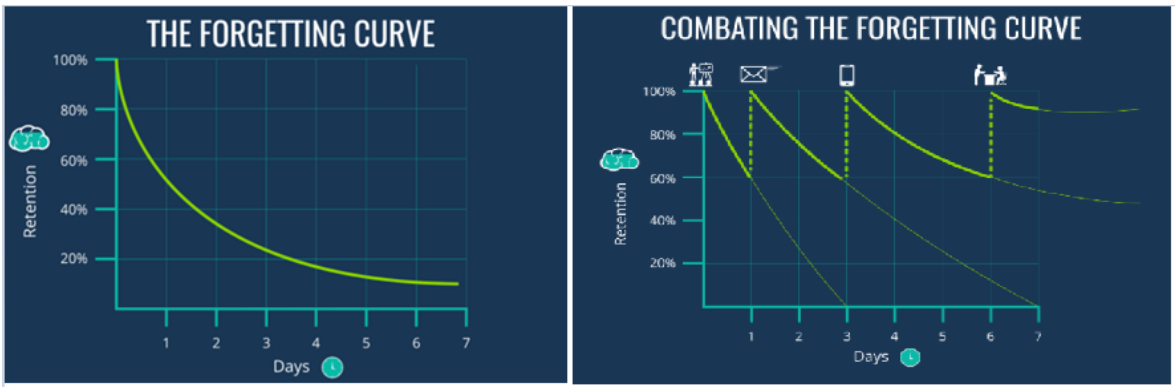

Figure 1. The forgetting curve (Ebbinghaus' Epiphany) and regular active recalls applied

One simple solution to combat the forgetting curve is to regularly recall the knowledge learners already know by testing learners' memory. The more frequently the information is retrieved, the stronger the memory will be, so the decline of the curve will be reduced. This is the foundation to apply spaced repetition to improve one's memory.

Spaced repetition is the technique that learners study complex concepts more often than easy ones. Once the information is memorized, the intervals between review sessions will be longer. This strategy allows learners to focus on the knowledge that is harder to remember, thereby improving the effectiveness of the learning process. In a nutshell, spaced repetition technique helps learners memorize concepts by reviewing the information before they forget. There are various spaced repetition algorithms, such as the Leitner system, Neural networks-based system, SM-family of algorithms, and Fibonacci sequence. This project's application uses the Leitner system for scheduling spaced repetition intervals.

Leitner system was proposed in the 1970s by a German science Journalist, Sebastian Leitner. This method uses boxes to sort flashcards in terms of the learner's familiarity level with the cards. In Figure 2, five Leitner learning boxes are chosen to indicate five proficiency levels. In the beginning, all flashcards will be kept in box number 1. When the learner manages to remember a card, he will move the card to the next box. If the learner fails to recall the answer, the card will be moved back to box number 1 no matter which box the card is currently located. The intervals between retrieval practices are different as follows (KCG College 2021):

- Box 1 – Every day
- Box 2 – Every 2 days
- Box 3 – Every 4 days
- Box 4 – Every 9 days
- Box 5 – Every 14 days

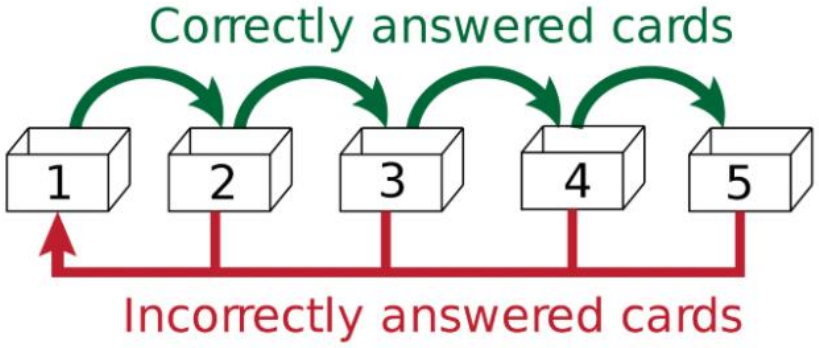

Figure 2. The Leitner system

Traditionally, spaced learning following Leitner System uses paper flashcards and physical boxes. The system works just fine if the learner only has a small number of flashcards. However, the more cards he has, the more complicated the scheduling will be as the review times are different.

2.3 Gamification

Gamification is the act of presenting people with practical challenges, organizing increasing difficulty levels for them to progress through, having them emotionally engaged, and encouraging them to perform at their best. It is a way to motivate people to be engaged in particular activities and help them achieve their goals in that particular field. (Burke 2014.)

Gamification can be applied to many fields, such as health, marketing, and education. Headspace is an excellent example of using gamification to motivate people to master different levels of meditation. Nike rewards customers when they run and compete against each other. Kahoot turns a classroom into a fun competition among students by displaying the leaderboard.

The purpose of using gamification features in this project is to provide users with a more exciting learning environment. The progress circle allows learners to track their progress and motivate them to move forward. Besides, users have a variety of quiz modes to choose from to test their knowledge. The instant grading feature integrated into the app helps learners monitor their performance and earn rewards for other activities.

Another gamification feature in the app Memoa is the virtual cat. Users take care of their cats with points earned by taking quizzes. The intention of including the cat is to encourage learners to use the application frequently, hence establishing their study habits. Ac-

cording to Burke (2014), this aspect of gamification is referred to as Mastery, which enhances the users' engagement with the mobile application and encourages them to perform better with their learning.

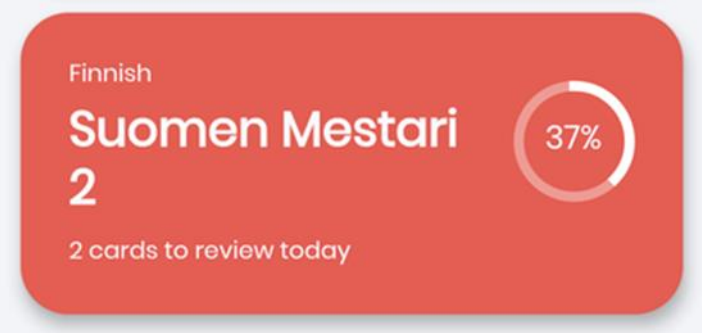

Figure 3. Progress circle as a gamification feature

2.4 Technical infrastructure

2.4.1 Flutter and Dart

Flutter is a UI framework for creating cross-platform applications with only one codebase. It was first released in 2017 by Google and had constantly been developing ever since. Flutter mainly comprises two factors: the Software Development Kit (SDK) and the UI framework. The SDK is a collection of tools that assist in application development, including tools to compile high-level programming languages into native machine code. The UI framework provides reusable UI elements such as buttons, containers, app bars, and dialogs that are highly flexible for personalization. (Thomas 2019.)

Flutter is powered by Dart, a programming language created by Google. Dart's goal is to provide a productive language for multi-platform development, complemented by a flexible execution runtime platform for app frameworks. (Dart 2021.) The most concise definition of Flutter can be found on its website:

> "Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase."

As of 3 March 2021, Google released Flutter 2. They proudly announced that this version of Flutter enables developers to build beautiful and fast applications native to five operating systems: iOS, Android, Windows, macOS, and Linux; it also offers optimized web experiences for four web browsers: Chrome, Firefox, Safari, and Edge. All of them can be built from only one codebase. (Flutter Team 2021.)

Every Flutter application composes of widgets as its building blocks. Widgets are built in a hierarchical structure where each child widget has a parent widget, and multiple levels of widgets form a widget tree. Child widgets can access the context passed down by their parents and either use it or continue passing it down to their descendants. (Flutter Documentation 2021.)

A widget can be used to display some UI elements on the screen or group multiple widgets as its children and apply some particular logic onto them. Flutter provides many UI widgets that follow Material UI and Cupertino (iOS-style) UI design. These widgets make Flutter applications natively appear as if they were developed separately on different platforms. Figure 4 demonstrates a button, a slider, and a progress indicator built by Flutter to run on iOS and Android.

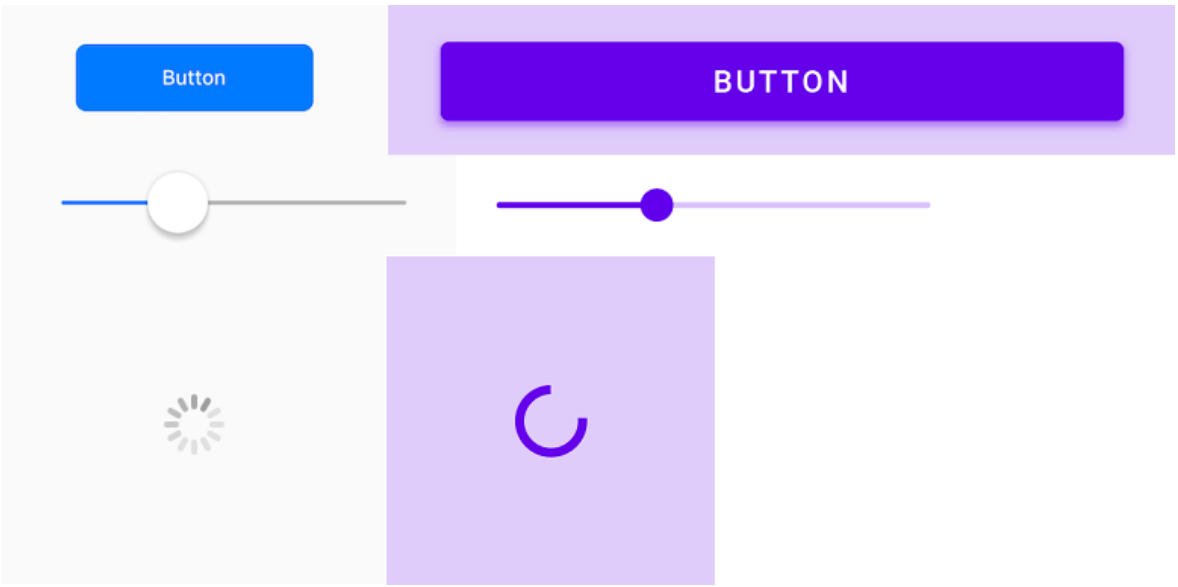

Figure 4. Flutter Cupertino (left) and Material widgets (right)

2.4.2 Google Firebase

Firebase is a mobile application development platform. It provides services for building, developing, and scaling applications. Firebase offers the possibility to manage necessary but tedious services, such as authentication, databases, or analytics, and allows developers to focus on the app experience itself. (Stevenson 2018.)

In this project, several Firebase services will be used – Firebase Authentication, Cloud Firestore, and Cloud Storage. Firebase Authentication manages user registration and authentication. It is also an essential factor for configuring other services such as the Cloud Firestore since it ensures that access to other services is requested from verified sources.

Firebase Authentication provides secured authentication methods while offers the possibility to register with numerous identity providers such as Facebook, Twitter, Google, and GitHub. (Stevenson 2018.)

Cloud Firestore offers a real-time, cloud-based, scalable database service (Stevenson 2018). It ensures data synchronization between client applications and web and mobile applications' responsiveness if the Internet connection is not available. Furthermore, Cloud Firestore is a non-relational, document-oriented database for storing and syncing data. Data is stored in collections and documents instead of tables and rows. Each document is a set of key-value pairs that can represent a real-life entity. Every document belongs to a collection and has a unique ID. For nested data structures, subcollections and nested objects can be added to a document for easier access. (Firebase Documentation 2021.)

On the other hand, Cloud Storage is a powerful and simple solution for storing user-generated content quickly and securely. It offers robust operations regardless of network quality. It is integrated with Firebase Authentication to restrict unauthorized accesses, and the storage grows along with the product development seamlessly. (Firebase Documentation 2021.)

3 Project background

This section will explain the choice of software development model and the implementation plan. Possible choices of software development models will be covered and compared to arrive at this project's suitable choice. Finally, the project implementation plan will be discussed in detail in the second section.

3.1 Methodological choice

Every software program possesses a development lifecycle. Disregarding a project's size, it has eight steps to go through. They are Conception, Requirement analysis, Design, Coding - Debugging, Testing, Release, Maintenance, and Retirement. How these steps are carried out can be classified into two fundamental classes of models: plan-driven models and Agile development models. Plan-driven models are suitable for large projects with well-defined requirements. The phases are distinctly separated from each other with clear outcomes in between and require extensive documentation.

On the other hand, Agile development models promote iteration and accumulation. They are more suitable for small- and medium-sized projects. These models emphasize focusing resources on the product itself rather than on operational processes such as documentation. Hence, they encourage small but frequent increments to a product rather than larger but less frequent ones since it is easier to maintain trimmer, incremental contributions. Design, Coding – Debugging, and Testing steps are usually iterated through many times in these models. Those steps do not happen one after another but take place in parallel and complement each other. (Dooley 2017.)

By taking the characteristics of the two models into account, it appears that the Agile development models would be an excellent match for this project as the team is small and the application is still at the beginning of its lifecycle. It is anticipated that the requirements will be constantly updated as the development team progress. Working in iterations also enables the team to learn, review current performance and improve on the shortcomings, which is highly beneficial as it consists of university students only. Furthermore, as features and tasks are divided into manageable pieces, it would be easier for the backlog to be logically organized and problems to be detected and resolved.

Typical Agile development models are eXtreme Programming (XP), Scrum, and Kanban. Among the choices, Kanban methodology indicates visualizing the workflow into a Kanban board. The board helps keep track of the team's progress and measure the team's productivity by "lead time" – the amount of time to get one task from developing state to

finish state. It also maximizes the team performance by distributing the resource, avoiding multitasking, and keeping individuals focus on their task. (Dooley 2017.)

It appears that the best choice for the project is Kanban. Its principles are the best guide for the team to manage the workflow, keep tasks in a manageable number, and progress well within the time scope.

3.2 Implementation plan

Some initial concepts and requirements are already available when the project kickstarts. The development team then analyses the requirements and sketches out the first few features and designs. More detailed requirements are made, and a user flow is created. The analysis results help both the designer and the developer produce a low-fidelity design and a starting codebase of the upcoming mobile application. Then the project backlog is made by analyzing the design and dividing features into manageable pieces.

From this point, the team works on tasks one by one, moving each one from the backlog to the finish state on the Kanban board, as mentioned in the previous section. During this time, the team follows the Agile principles to keep track of the overall progress. The designer proposes the color palette as well as refines the design with the proposed set of colors. Simultaneously, the developer converts the design into working UI components and implements the underlying functionalities.

After the backlog is cleared, the app should be thoroughly tested in a beta testing phase. The bugs found during this phase will need to be categorized into severity levels and fixed accordingly. When the app runs with stability, it is ready to be released to the app stores. Finally, the project moves to the maintenance phase, where bugs are fixed, and user feedback is actively taken.

4 Implementation

This section will discuss the process of building and releasing the flashcard application Memoa on the Android platform. First, the necessary tools will be discussed, then how the database is designed and developed using Firebase. The application infrastructure and the algorithms used will be covered next. Finally, the release of the application to the app store is to be described.

4.1 Tools

4.1.1 Flutter and dependencies installation

Flutter can be installed by cloning the repository from GitHub. The version can also be selected by indicating the branch to clone. For instance, stable and beta are possible selections, depending on the developer's needs.

```
PS C:\framework> git clone https://github.com/flutter/flutter.git -b stable
```

Figure 5. Cloning Flutter repository from GitHub

The stable channel is the recommended choice for people who are starting to work with Flutter as it is thoroughly tested and can ensure performance. On the other hand, Flutter is constantly updated, and new features are frequently pre-released on the beta channel for developers to try integrating the new improvements into their projects.

Flutter provides a command to check if the environment for developing Flutter applications is running correctly. By running "flutter doctor" in the command line, Flutter will check for possible new updates, IDE compatibility, and mobile phone (if the computer is connected to one) or simulator compatibility.

```
PS C:\dev\memoa> flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.0.1, on Microsoft Windows [Version 10.0.19042.804], locale vi-VN)
[√] Android toolchain - develop for Android devices (Android SDK version 30.0.3)
[X] Chrome - develop for the web (Cannot find Chrome executable at .\Google\Chrome\Application\chrome.exe)
    ! Cannot find Chrome. Try setting CHROME_EXECUTABLE to a Chrome executable.
[√] Android Studio (version 4.1.0)
[√] VS Code (version 1.54.1)
[√] Connected device (1 available)

! Doctor found issues in 1 category.
```

Figure 6. Running "flutter doctor"

As can be seen from the above image, flutter doctor detects that Chrome is not installed on the machine, which will affect the development of Flutter Web. Since the application is for mobile platforms, Chrome is not necessary.

Flutter and other programs powered by Dart programming language can take advantage of pub.dev – the Dart programming language package manager. Packages on pub.dev are reviewed thoroughly by the community and rated by the Flutter Ecosystem Committee for their quality. Various packages can be installed by adding the package's name and the minimum version number to the pubspec.yaml file. Then run "flutter pub get" in the command line. The following code snippet demonstrates how to include pub.dev packages in the project.

```
22
23  dependencies:
24    flutter:
25      sdk: flutter
26    firebase_core: ^1.0.0
27    firebase_auth: ^1.0.0
28    cloud_firestore: ^1.0.0
29    animations: ^2.0.0
30    provider: ^5.0.0
31    carousel_slider: ^4.0.0-nullsafety.0
32    percent_indicator: ^3.3.0-nullsafety.1
33    flutter_slidable: ^0.6.0-nullsafety.0
34    step_progress_indicator: 1.0.0-nullsafety.0
35
36    # The following adds the Cupertino Icons font to your application.
37    # Use with the CupertinoIcons class for iOS style icons.
38    cupertino_icons: ^1.0.2
39
```

Figure 7. Project dependencies

Like any other programming language, using ready-made libraries instead of putting effort into doing everything in-house will significantly reduce developing time; thus, having a trusted platform to import high-quality packages would be significantly cost-efficient.

4.1.2 Other necessary tools

A few tools are always needed when developing applications, such as the IDE and the application simulator. Flutter and Dart are supported on popular IDEs such as Android Studio and Visual Studio Code. It depends on the developer's preferences regarding which IDE to use since they have different strengths and drawbacks. Either IDE can be installed by going to their websites and installing the latest stable version.

Android Studio provides a built-in Android Virtual Device (AVD) Manager from which Android device simulators can be created. A virtual device is automatically detected by Flutter so that when a Flutter project is run, it will install the project to the device and run the app on the virtual device's screen. On the other hand, Flutter projects can also be run on

physical devices by connecting them to the machine via USB cables. These devices will also be detected by Flutter automatically.

4.2 Database design

The first step to designing a database is to analyze the actions users will perform on this system and the necessary data to support its users efficiently. The analysis result helps ensure that the problem is understood correctly. (Churcher & Faroult 2012.)

The project has some initial requirements and visions for the MVP that can act as a reference for designing a suitable database system. The requirements are as follow:

- User can create decks and flashcards; optionally, a user can classify a deck to a topic
- The data that connects to the user are the topics, decks, and flashcards they created, their result with flashcard decks after every review session, their score after every test, and information related to their virtual pet, such as level and status.
- A deck has a name, topic, timestamp, flashcards, and last reviewed date.
- A flashcard has the following fields: terminology, definition, example usage, other additional text, image, the box number which the flashcard is currently in, and the date at which the flashcard needs to be reviewed.
- The user can review the flashcards past the review date and evaluate whether they remember those cards.
- The app can generate multiple test types for the user to evaluate their memory on a particular deck. Possible test types are multiple-choice, matching, fill-in-the-blank, and mixed. The test score can be converted into points which can be used to interact with the virtual pet.
- The app requires the user to register an account and be signed in when using the app. By doing so, the user's data is constantly saved and backed up to the cloud, and synced across multiple devices. Users can sign in to the app using their registered email and password, Google account, or Facebook account.
- The virtual pet acts as a motivator, and the app creates push notifications to remind the user to review their decks at the preset time of day.

From the requirements listed above, it can be seen that data revolve around the user. A user can create multiple decks of flashcards, assign them to multiple topics for filtering purposes, study them and attempt tests to earn points for taking care of their virtual pet. Thus, virtual pet information, decks, flashcards, topics, and learning progress data belong to the user only. A use case diagram can be used to help understand the problem better in this scenario.

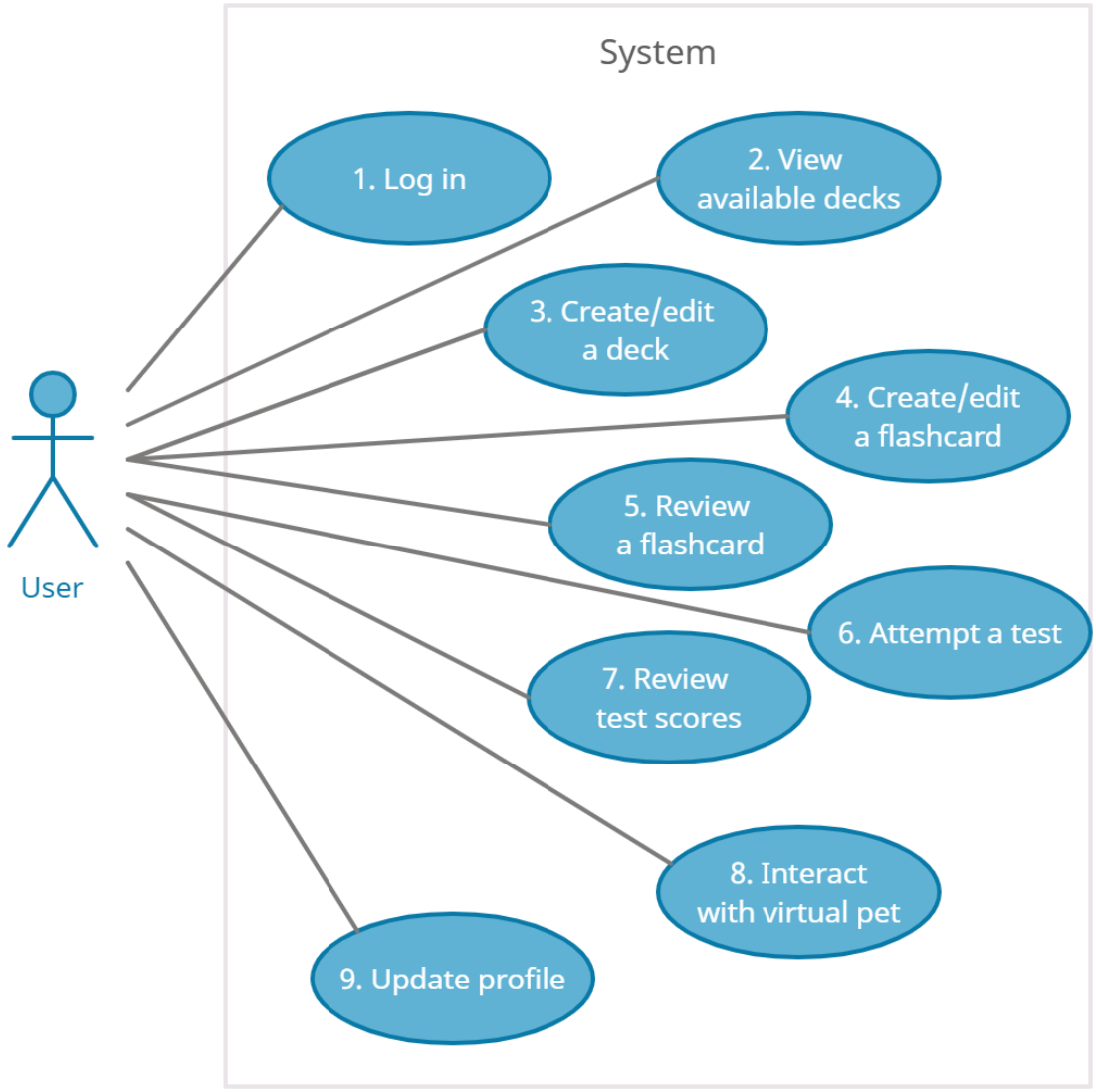

Figure 8. Use cases for Memoa application

The diagram helps determine the relationship between the system's entities and the users. It separates actions and other elements that were mentioned in the requirements. A table can be constructed to visualize user actions and the related data by combining this visualization and the initial requirements (Table 1).

Table 1. User tasks and related data

Task	User Actions	Related data
1	Log in to the app	Email, password
2	View the available decks	Data of user's decks and flashcards
3	Create/edit a deck	Deck name, topic, deck color, timestamp
4	Create/edit a flashcard	Term, definition, example usage, additional text, image
5	Review a flashcard	Box number, review date

6	Attempt a test	Test score
7	Interact with the virtual pet	Pet level, status
8	Review test scores	Test scores from previous attempts
9	Update user profile	Profile image, user's name, password

At this point, sufficient analysis has been accumulated to construct a data model. As stated by Churcher & Faroult (2012), "A data model is a precise description of the data stored for a real-world problem, in much the same way that a mathematical equation describes a real-world physical event or an architectural drawing describes the plan of a building."

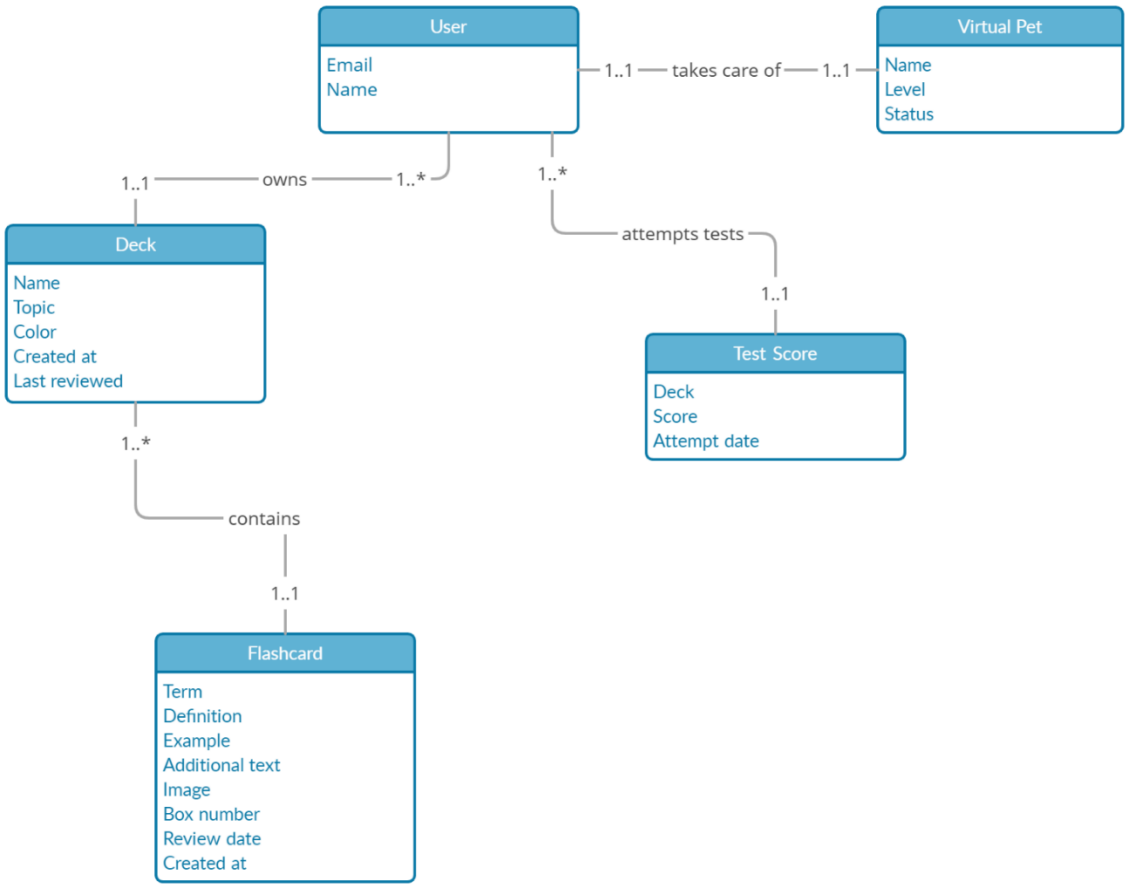

Figure 9. Memoa app data model

With a data model, it is easier for developers to understand the relationships among the data items and prepare for the upcoming development steps. The following diagram illustrates the relationships between the Memoa application entities and how they interact with each other.

The next step is to translate this data model into a database system on Cloud Firestore, which will be discussed in the next section.

4.3 Backend – Firebase services

4.3.1 Initialize Firebase project

Initially, a Firebase project needs to be created from the Firebase console. This project will be the container for apps that utilizes the same services. An Android app, an iOS app, and a web app can be registered to this project, and they share the same authentication and storage services. By sharing these resources, users can register their accounts on the web app and log in to them on their phones, accessing the same data across various platforms. Additionally, using Cloud Firestore enables the data to be synchronized in real-time, handles concurrent transactions appropriately, and ensures data consistency and integrity.

Applications can be added by following the guided steps on the Firebase console at any point in time. For this project, two applications are added, one Android app and one iOS app. If a web application is developed, it can be added to the project in the future.

4.3.2 Authentication

Firebase provides an excellent authentication service that supports multiple methods of authentication. It offers solid solutions for authenticating users using email/password, phone, or through popular third-party accounts such as Google, Facebook, and Apple. They can be enabled and disabled accordingly, as shown in the image below. Furthermore, Firebase Authentication also includes ready-made templates for email address verification, password reset, or SMS verification. These templates can be customized extensively to fit specific needs.

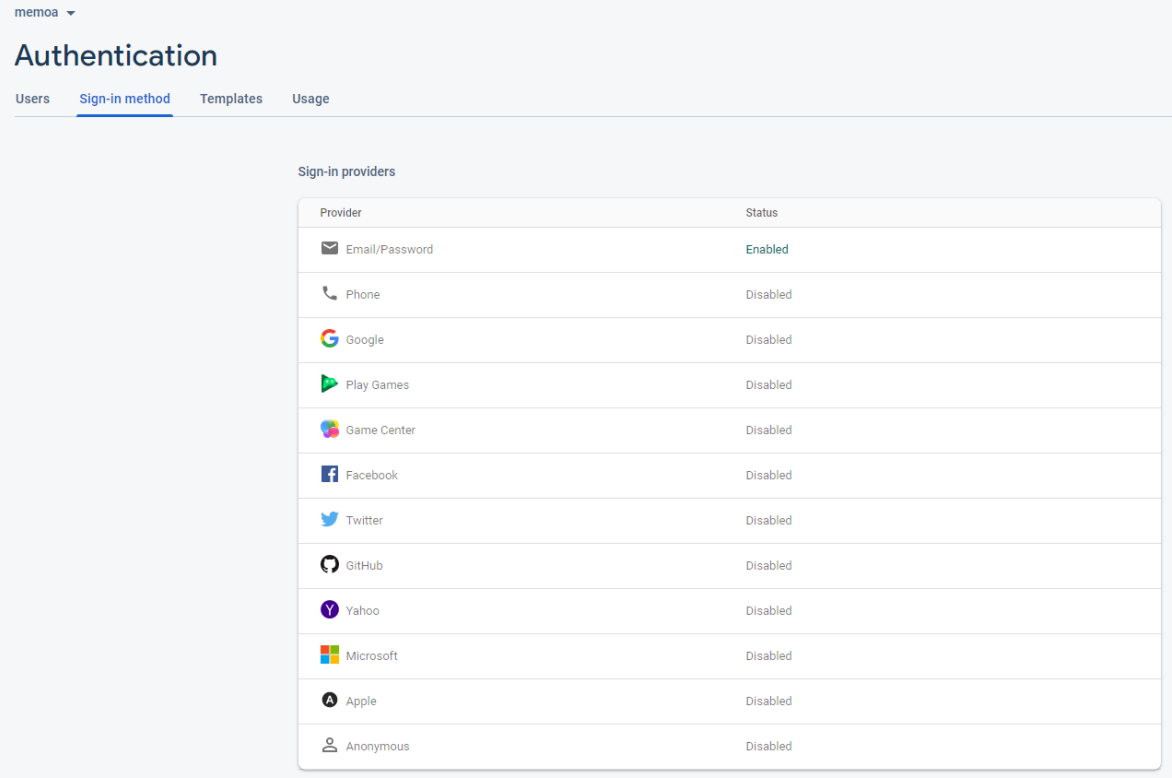

Figure 10. Firebase Authentication sign-in methods

4.3.3 Cloud Firestore

The app takes advantage of Google Firebase's Cloud Firestore for storing and syncing users' data instead of utilizing the device's local storage. By doing so, the data will be ready for future features such as sharing and competing between users.

As mentioned before, Cloud Firestore stores data in collections and documents. These entities can be created on the fly without any prior declaration. When a collection or a document is queried, if it cannot be found among the existing data, then a new one is created. Hence, there is no command to create a document or a collection; the typical commands are GET, SET, UPDATE and DELETE.

4.3.4 Pricings

Firebase offers two pricing plans Spark and Blaze (Firebase 2021). The Spark plan is highly beneficial to students who would like to do personal or school projects as it is free and offers a sufficient amount of resources. Suppose a particular project scales larger over time with a higher number of users or a more considerable amount of data. In that case, the Blaze plan is reasonably priced as the cost is calculated by the number of resources used.

4.4 Flutter development

4.4.1 Working with FlutterFire packages

Firebase provides Flutter packages that can be added to the Flutter application as dependencies. They are called FlutterFire packages. These packages provide ready-made methods for communication between the Flutter application and Firebase's services.

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^1.0.3
  firebase_auth: ^1.0.0
  cloud_firestore: ^1.0.0
  firebase_storage: ^8.0.3
```

Figure 11. FlutterFire packages

The firebase_auth package will handle user authentications: registering and signing in. It handles everything in the background. The app only needs to tell what data to be sent to the API by calling different methods with email and password as parameters. Firebase_auth package provides createUserWithEmailAndPassword(email, password), signInWithEmailAndPassword(email, password) methods to create new user and sign in to an existing account.

```
41  @override
42  Future<MemoaUser?> createUserWithEmailAndPassword(
43      String email, String password) async {
44    final UserCredential userCredential = await _firebaseAuth
45        .createUserWithEmailAndPassword(email: email, password: password);
46    if (userCredential.user != null) {
47      await FirebaseFirestore.instance
48          .collection('users')
49          .doc('${userCredential.user!.uid}')
50          .set(
51        {
52          'email': userCredential.user!.email,
53        },
54      );
55    }
56    return _userFromFirebase(userCredential.user);
57  }
```

Figure 12. Create user with email and password method

As demonstrated in line 44 in Figure 12, the FirebaseAuth instance, constructed when the FirebaseAuth service is initiated, connects to Firebase API and calls the method createUserWithEmailAndPassword(). It tells Firebase's Authentication service to create a

new user with the included email and password. If the email is already in used, an error will be sent back and handled by a try-catch block. If the call succeeds, the newly created user's credential will be assigned to the variable named userCredential.

Next, the newly created user's id needs to be saved to Firebase's Cloud Firestore as a new entry to the Users collection. This entry will store all the user's information, such as decks, flashcards, and the virtual pet. This is where the FirebaseFirestore package comes into action. A FirebaseFirestore instance makes a request to Firebase's Cloud Firestore API to query the "users" collection and find a document by the given user ID. In this case, there is not yet a document with that ID since it is a new one. Thus, Firebase's Cloud Firestore creates a new document automatically and store the user email inside that document. As mentioned in section 4.3, Cloud Firestore provides high flexibility with the fields inside a document. They do not need to be declared in advance and can be added or removed on the fly. Here, a document with only email as a field is created, but more fields will be added later as the user interacts with the app, such as creating a new deck of flashcards and studying it.

The following code snippet demonstrates the process when the user adds a new deck of flashcards. The FirebaseFirestore instance accesses the document mentioned above with the user ID and query for the "decks" collection, which in this case does not exist. Hence, a "decks" collection is created automatically, and a new document is added to it with the given information, as can be seen from the code snippet line 247 to line 253. This function is asynchronously called as indicated in the function declaration. The async keyword indicated that this function takes time to run, and the process called it needs to wait for this function to complete before moving on to the next task.

```
240
241   Future<void> addDeck(Deck deck) async {
242     final DocumentReference newDoc = await FirebaseFirestore.instance
243         .collection('users')
244         .doc(uid)
245         .collection('decks')
246         .add(
247       {
248         'name': deck.name,
249         'topic': deck.topic,
250         'color': deck.color.value,
251         'createdAt': deck.createdAt.toIso8601String(),
252         'lastReviewed': deck.lastReviewed.toIso8601String(),
253       },
254     );
```

Figure 13. Method to add a deck to Firebase Cloud Firestore

At this point, a deck document does not contain any flashcards. The process of adding any other kind of data to Firebase Firestore is similar to what was discussed above. For instance, when the user starts adding flashcards to the deck, they are then added to the database by accessing the deck with the deck ID, creating a "flashcards" collection, and pushing new documents into the collections.

4.4.2 The widget tree

As mentioned in the theory part, a widget tree depicts the widget hierarchy of an application. It provides a better grasp of the flow of the application state for the developers to make decisions about business logic and optimization. By understanding the structure of the application, developers can optimize the code and increase the performance of the app by reducing unnecessary rebuilds of widgets.

A deep widget tree that spans many levels can reduce code readability and maintainability. Thus in this application, the author aims to maintain a shallow widget tree. This is archived by regularly reviewing the codebase, separating pieces of code into widgets classes, and marking unchanging UI elements constants. As a result, the code remains maintainable as it grows larger over time, bugs are easier to be detected, and any considerable changes to the code structure can be made modularly.

The widget tree of the Memoa application is designed as followed:

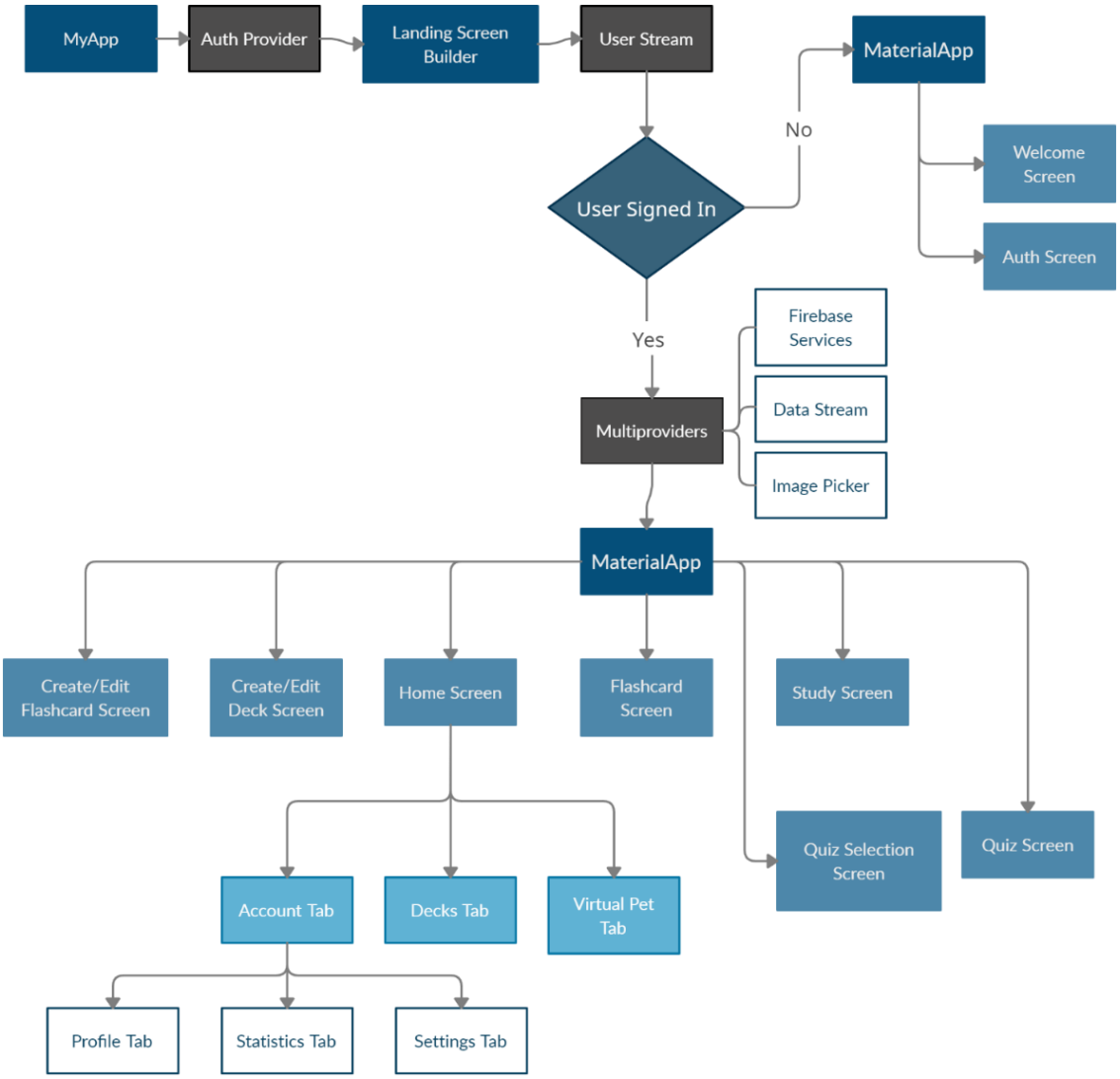

Figure 14. Memoa application's simplified widget tree

This structure of the widget tree complements the following sections where the app theme and app state management are discussed.

4.4.3 App theme

Every application needs a UI theme to offer the user a feeling of consistency when navigating through the app. With Flutter applications' hierarchical structure, theming can be set up quickly as the theme data is passed down along the widget tree inside the context. As can be seen in the widget tree in the previous section, when a ThemeData object is assigned to the MaterialApp widget, all descendant screen widgets can inherit the theme and build the UI elements with the inherited data. A ThemeData object can contain many attributes such as texts, colors, and button styles. The following code snippet demonstrates a part of a ThemeData object declaration.

```
ThemeData appTheme = ThemeData(
  backgroundColor: MEMOA_LIGHT_GREY,
  primaryColor: MEMOA_BLUE,
  accentColor: MEMOA_YELLOW,
  fontFamily: 'PoppinsVN',
  textTheme: ThemeData.light().textTheme.copyWith(
      headline1: TextStyle(
        fontSize: 24,
        fontWeight: FontWeight.w700,
        color: MEMOA_WHITE,
      ), // TextStyle
      headline2: TextStyle(
        fontSize: 44,
        fontWeight: FontWeight.w700,
        color: MEMOA_EGGPLANT,
      ), // TextStyle
      headline3: TextStyle(
        fontWeight: FontWeight.w700,
        fontSize: 16,
        color: MEMOA_WHITE,
      ), // TextStyle
```

Figure 15. Setting ThemeData

After the declaration and assigning the object to the MaterialApp widget, the ThemeData can be accessed as followed:

```
style: Theme.of(context).textTheme.bodyText2,
```

Figure 16. Accessing ThemeData

As the ThemeData object stores all the settings for styling widgets, when there is a need to update the UI, it can be done quickly by changing the ThemeData object's attributes for the widgets to be updated instantly.

4.4.4 In-app data management

For any application, there always be a need for data management during runtime. When the user clicks on a product and adds it to the shopping cart or browses through their list of ordered products, their actions need to be processed and saved to a shared location since this information will be needed for other components of the app to use. For example, products added to a shopping cart need to be saved so that when the user navigates to the shopping cart interface, the app knows what products to show and their quantity. Spe-

cifically, in this flashcard application, the user interacts with decks and flashcards in various ways. The app needs to react to changes instantly and displays the correct information upon requests.

The simplest way to manage the app state in Flutter is using a package called Provider. It interacts with low-level widgets and provides the mechanisms for passing data and services from parent widgets to their descendants. Flutter also provides a datatype called stream, which returns a stream of data to the calling function. This stream returns a new value whenever there are changes in the data source. Additionally, the Provider package also supports the Stream datatype through a class named StreamProvider. (Flutter Documentation 2021.)

For this project, a stream of user authentication data is set up to listen for changes from the Firebase authentication server. This ensures that whenever a user's session expires, that user will be signed out of the application. This stream is stored in the Auth Provider above the LandingScreenBuilder, as shown in the widget tree. The LandingScreenBuilder widget receives the User Stream, checks if a user is currently signed in, and renders the corresponding widgets. If there is no signed-in user, the Welcome Screen will be rendered. Otherwise, the signed-in user will be taken to the Home screen.

Putting providers too high in the widget tree would expose the data to unwanted components while putting them too low would make the app unfunctional. Since Firebase services such as Cloud Firestore and Storage as well as the data-stream need to be available to multiple screens, the suitable location, in this case, is the MaterialApp widget. This MaterialApp widget wraps around the Home Screen and provides the necessary services for the app to function correctly. These services are only provided after the user has signed in and will not be accessible to unauthenticated sources. Therefore, the user can navigate through the app and use the provided services securely.

Figure 17 demonstrates the use of MultiProvider to provide different services to descendant widgets. The services are only provided if the user variable is not equal to "null" or, in other words, the user is authenticated.

```dart
if (user != null) {
  return MultiProvider(
    providers: [
      Provider<MemoaUser>.value(value: user),
      Provider<FirestoreService>(
        create: (_) => FirestoreService(uid: user.uid),
      ), // Provider
      Provider<FirebaseStorageService>(
        create: (_) => FirebaseStorageService(uid: user.uid),
      ), // Provider
      Provider<ImagePickerService>(
        create: (_) => ImagePickerService(),
      ), // Provider
      StreamProvider<List<Deck>>(
        create: (_) =>
            FirestoreService(uid: user.uid).getUserDecksStream(),
        initialData: [],
      ) // StreamProvider
      // NOTE: Any other user-bound providers here can be added here
    ],
    child: builder(context, snapshot),
  ); // MultiProvider
```

Figure 17. MultiProvider usage

Finally, for the children widgets in the widget tree to access the data or services provided by the Providers, they can access them by reading from the application's context, which is passed down from the root of the widget tree. In the following code snippet, a child widget can access Cloud Firestore's services by calling the Provider's method "of(context)".

```dart
final _streamProvider =
    Provider.of<FirestoreService>(context, listen: false);
```

Figure 18. Accessing provider's data

4.4.5 Leitner System algorithm

As discussed in the theoretical part, applying the Leitner system to physical flashcards would become overly complicated over time. With the assistance of computational power, the Leitner system's problem with physical flashcards can be solved effortlessly.

Digital flashcards can carry much more information with them comparing to physical ones. For the Memoa app, each flashcard is assigned a box value from zero to eight, representing the user's progression with each flashcard and a date value, indicating the flashcard's time to be reviewed. These kinds of data that change frequently could be hard to keep track of for physical flashcards. When a newly created flashcard is added to the deck, it will be assigned to box zero and need to be reviewed instantly.

Every day, the app will check for flashcards that need to be reviewed on that day and notify the user to study at the preset time. Each flashcard's box value is used to calculate the user's progression with each deck of flashcards. The higher the box value is, the more contribution the flashcard makes to the overall progression. Calculating the flashcards' box values to show the user's progression as a percentage helps the user know how well they are learning and stimulates them to put in the effort to close the percentage circle.

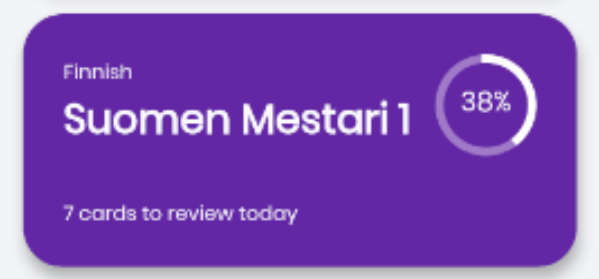

Figure 19. UI element to present deck's progress

When the user reviews a flashcard, they can choose whether they remember the term and the definition written on the card or not. If they remember, the flashcard will be moved to the next box, and the date to be reviewed is updated accordingly. On the contrary, the flashcard is moved back to box number one and reviewed the next day.

4.4.6 Quiz generator

The app will offer a few test types for the user to evaluate their progress with a particular deck. Since the test needs to be generated according to the user's choice, the test generator function needs to return a list of Map<String, dynamic>. It is a flexible data structure since the value part can store a single value, an array, or an object. The map data structure is suitable for this situation since the selected type of test (between multiple-choice, matching, or mixed) is not known in advance.

Multiple-choice: A multiple-choice question is set up by showing a flashcard's term and four possible choices. The user will determine among the choices which is the correct definition for the showing term. In some cases, the choices are made to be similar to each other so that the student cannot simply choose the right choice by guessing. Since the app is designed to fit multiple purposes, it is challenging to auto-generate difficult questions from a particular deck of flashcards. Therefore, for each flashcard in a deck, the approach is to pick three random definitions from the deck and shuffle them with the correct definition for the given term. If the deck has less than four cards, a multiple-choice test cannot be generated.

Matching: All the flashcards of a deck will be separated into two objects; each one will have the flashcard ID as the key. The list of objects will be shuffled and rendered on the screen. When the user chooses two cards with the same ID, their choice is evaluated to be a correct answer, and a certain number of points are added to their total score. Otherwise, the chosen cards will be marked with a red cross icon to indicate an incorrect choice. Then the user will have to choose again until all the pairs have been chosen correctly.

4.4.7 Customized widget implementation

Since some ready-made Flutter widgets appear differently on iOS and Android devices, it is necessary to implement a few customized widgets to ensure the same user experience on the two platforms. Doing so also increases the reusability of UI widgets and the maintainability of the codebase. Flutter's ready-made widgets are highly customizable by nature; thus, they can be customized easily by wrapping them in another widget and passing the necessary parameters. Therefore, instead of rewriting all the parameters for every button that appears throughout the app like this:

```
212  ElevatedButton(
213    onPressed: _review,
214    child: Text(
215      'Review',
216      style: TextStyle(
217        fontFamily: 'Roboto',
218        fontWeight: FontWeight.w500,
219        fontSize: 16,
220        color: MEMOA_WHITE,
221      ), // TextStyle
222    ), // Text
223    style: ButtonStyle(
224      backgroundColor:
225          MaterialStateProperty.resolveWith<Color>(
226              (Set<MaterialState> states) {
227        if (states.contains(MaterialState.disabled))
228          return MEMOA_GREY;
229        return MEMOA_BLUE;
230      }),
231      elevation:
232          MaterialStateProperty.resolveWith<double>(
233              (_) => ELEVATION_6),
234    ), // ButtonStyle
235  ) // ElevatedButton
```

Figure 20. ElevatedButton customization

The amount of code can be significantly reduced like the following code snippet.

```
259  MemoaElevatedButton(
260    label: 'Review',
261    onPressed: _review,
262  ), // MemoaElevatedButton
```

Figure 21. Customized widget usage

A widget named MemoaElevatedButton is created as a wrapper around the ready-made button that accepts only the necessary parameters like the button label and the function to run when pressed. Other characteristics of an elevated button are kept inside one file named "MemoaElevatedBut-ton.dart". This way of conduct ensures the consistency of UI elements throughout the application. Changes can be made to only one file and applied to all the appearances of a particular customized widget in the whole project instantly.

4.5 Release

After completing the app's main features, the development team decided to deploy an MVP version to Google Play Store for internal testing. Selected testers will test this version and create feedbacks for the app improvement.

To publish apps to Google Play Store, a developer must register a Google Play Console developer account and pay a one-time registration fee of $25. The process of publishing an application to Google Play Store is described carefully by the Flutter documentation. This includes signing the app with the developer's digital signature, bundling the app to an Android App Bundle file (.aab), and then upload it to Google Play Console. With detail guides, Flutter has made it extremely easy to publish apps to Google Play Store. Additionally, the user interface of Google Play Console also makes it easy to manage the test tracks, from internal to beta testing. The console provides a friendly user interface to design the main store listing and view the store performance.

4.6 Result

In the end, an MVP was achieved as the first milestone for the project team to evaluate the project's potentials. The foundation of the app is constructed, and improvement will be made based on it. With the MVP, the developers can convey the app's idea to a broader audience to collect feedback and suggestions. There are apparently many rooms for improvement within the project regarding resource management and product quality. Thus the developers will keep working on this application and bring out its best potentials.

5 Discussion

As the first impression, Flutter is a powerful tool to develop cross-platform applications. The learning curve is quite steep at first, but the framework is pretty robust and full of potentials. With the ability to produce different platform applications from a single codebase, Flutter surely will attract more and more attention in the future.

This very first phase of the project aims to create a minimum viable product from the first ideas of the flashcard app Memoa. The developing team had to go through requirement analysis, designing, and developing the app from scratch. With the current stage of the application, this phase is considered a success. An MVP is produced within the time scope, and the quality of the application is at a reasonable level. The development process has been challenging due to the team being inexperienced in project management and having to work with a new framework. Carrying out the project in an agile fashion is a correct decision as it helped the team work more efficiently toward the preset goals. From the collected test users' feedback, improvements will be made to the app, and a stable version should be released soon.

The author had the chance to develop his skills in application development and project management during this project. By working in iterations, the author had to constantly review his performance and decide which features or tasks to be prioritized to achieve the project goals in time. The use of the Kanban board and other project management tools immensely helped prioritize tasks and keep track of the project progress. It is also an excellent opportunity to try new technologies and explore new concepts.

References

Burke, B. 2014. Gamify: How Gamification Motivates People to Do Extraordinary Things. Bibliomotion. Brookline.

Brainscape. 2021. Are flashcards effective? The top 3 ways they can boost your grades. URL: https://www.brainscape.com/academy/are-flashcards-effective/ . Accessed: 20 April 2021.

Churcher, C. & Faroult, S. 2012. Beginning Database Design. Apress.

Dooley, J. F. 2017. Software Development, Design and Coding: With Patterns, Debugging, Unit Testing, and Refactoring. Apress. New York.

eLearning Industry. 2018. What is the forgetting curve (and how do you combat it)? URL: https://elearningindustry.com/forgetting-curve-combat . Accessed: 20 April 2021.

Firebase Documentation 2021. Cloud Firestore. URL: https://firebase.google.com/docs/firestore/data-model. Accessed: 2 March 2021.

Flutter Documentation 2021. Simple app state management. URL: https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple. Accessed: 1 April 2021.

Flutter Team. 2021. Announcing Flutter 2. URL: https://developers.googleblog.com/2021/03/announcing-flutter-2.html . Accessed 5 April 2021.

KCG College. 2021. Leitner System for your better learning. URL: https://kcgcollege.ac.in/leitner-system-for-your-better-learning/ . Accessed: 21 April 2021.

Mainkar, P. & Giordano, S. 2019. Google Flutter Mobile Development Quick Start Guide. Packt Publishing. Mumbai.

Stevenson, D. 2018. What is Firebase? The complete story, abridged. URL: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0. Accessed: 5 April 2021.

Thomas, G. 2019. What is Flutter and Why You Should Learn it in 2020. URL: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/. Accessed: 5 April 2021.

University of Southern Maine. 2021. Academic gains through improved learning effectiveness (Agile). URL: https://usm.maine.edu/agile/using-flashcards. Accessed: 20 April 2021.