DRABT

System Design

The Oryx Group

# Introduction

The purpose of this document is to describe the System Desing for DRABT with detailed sections covering the website, app, database, backend and frontend designs.

# Website System Design

The website will be built using JavaScript for the backend design and for connecting the website to the database. HTML and CSS will be used for the frontend design. While JavaScript is not strictly class base, it supports classes through ES6. To enhance code management and clarity, we will adopt a class-based approach.

## UML Class Diagram

1. **Flashcard:** This class represents each individual flashcard that contains content in front and back. It allows the user to edit the contents and manage tags for categorization supporting flexible studying options.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| id | String | Unique identifier for each flashcard. |
| question | String | The front side content of the flashcard. |
| answer | String | The back side content of the flashcard. |
| tags | Array<String> | Optional tags for categorization. |
| contentType | String | Indicates if the flashcard contains text or an image |

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Type** | **Description** |
| editQuestion(newQuestion: String) | void | Allow the user to edit the question. |
| editAnswer(newAnswer: String) | void | Allow the user to edit the answer. |
| addTag(tag: String) | void | Adds a new tag |
| removeTag(tag: String) | void | Removes existing tag. |

1. **Deck:** This class represents a set of multiple flashcards, enabling users to create custom sets for each subject. Additionally, it manages the addition, removal, and shuffling of flashcards. This class will allow the user to save the deck in the database for later review.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| id | String | Unique identifier for the deck. |
| title | String | Name of the deck |
| flashcards | Array<Flashcard> | List of flashcards in the deck |
| backgroundColor | String | Customizable background color for flashcards |

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Type** | **Description** |
| addFlashcard(flashcard: Flashcard) | void | Adds a new flashcard to the deck. |
| removeFlashcard(flashcard: Flashcard) | void | Removes a flashcard to the deck. |
| shuffleDeck() | void | Shuffles the deck. |
| saveToLibrary() | Promise | Method to save the deck to the database. |

1. **User:** This class manages user accounts, allowing secure account creation, login, and personal deck management.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| userId | String | A unique identifier for each user. |
| username | String | The user’s chosen display name. |
| passwordHash | String | Encrypted password for secure login. |
| decks | Array <Deck> | A list of decks created by the user. |
| loggedIn | Boolean | Tracks if the user is logged in. |

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Type** | **Description** |
| createAccount(username: String, password: String) | Promise | Creates a new user account securely. |
| login(username: String, password: String) | Promise | Authenticates the user and starts a session. |
| logout() | void | Ends the current user session. |
| addDeck(deck: Deck) | String | Adds a deck to user’s account. |

1. **CardManager:** This class manages study experience, handling navigation, and flipping cards.

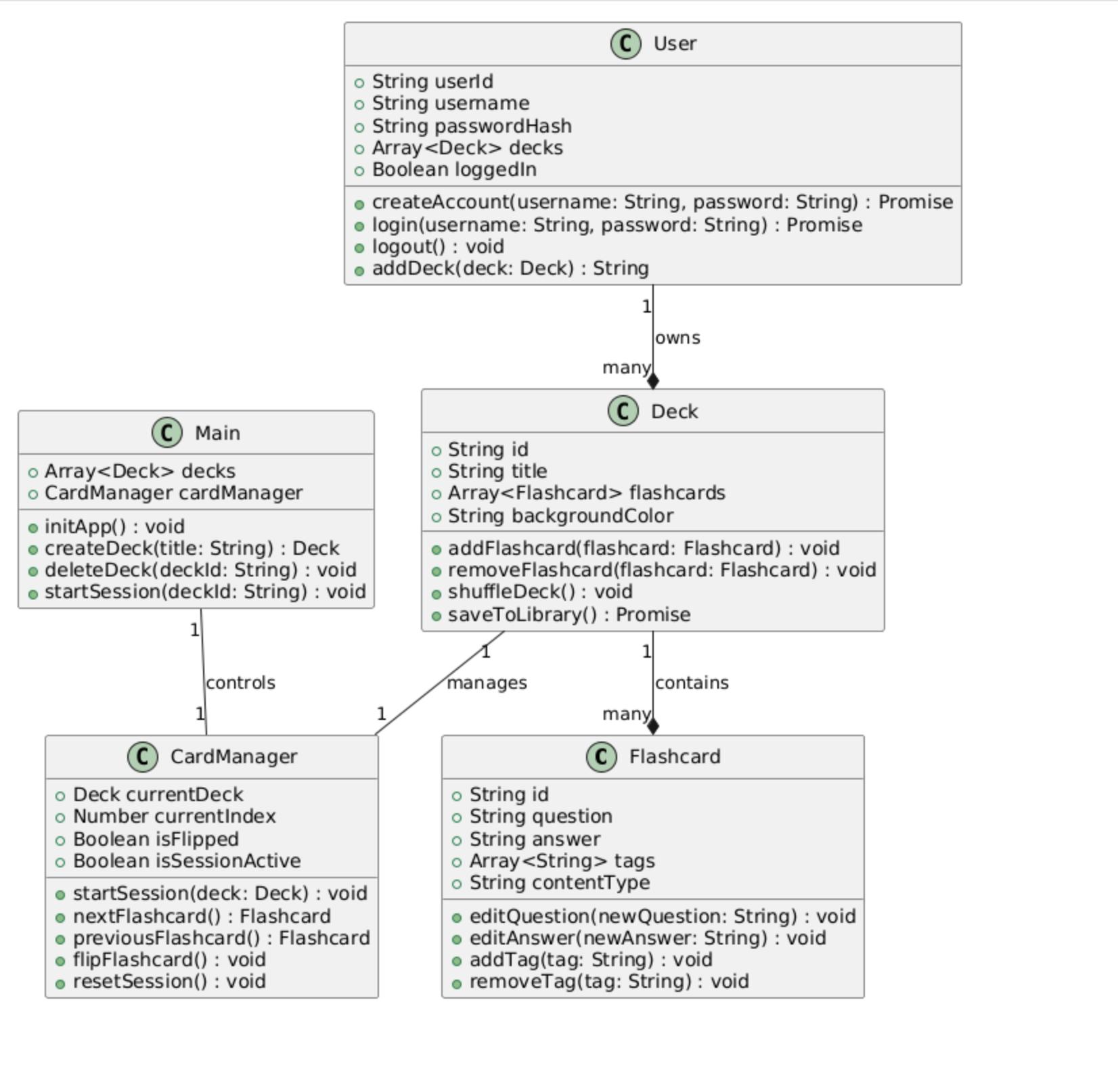
|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| currentDeck | Deck | The deck currently being used for studying. |
| currentIndex | Number | The index of the current flashcard in the deck. |
| isFlipped | Boolean | Track if the card is flipped. |
| isSessionActive | Boolean | Track if the session is active. |

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Type** | **Description** |
| startSession(deck: Deck) | void | Initializes a new study session using the specified deck. |
| nextFlashcard() | Flashcard | Retrieves the next flashcard in the deck. |
| previousFlashcard() | Flashcard | Retrieves the previous flashcard. |
| flipFlashcard() | void | Flips the current flashcard to show the other side. |
| resetSession() | void | Resets the study session. |

1. **Main:** This class serves as the central controller or “game master” of the application. It connects all other classes, initializes the app, manages decks, and coordinates study sessions.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| decks | Array | A collection of all created decks. |
| cardManager | CardManager | Manages the study sessions and card interactions. |

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Type** | **Description** |
| initApp() | void | Initializes the application, setting up necessary configuration. |
| creatDeck(title: String) | Deck | Creates a new deck with the given title. |
| deleteDeck(deckId: String) | void | Deletes a deck by its ID. |
| startSession(deckId: String) | void | Starts a study session using the deck identified by the specified ID. |



### User Interface and Styling Specifications (HTML/CSS)

#### User Interface and Navigation System for DRABT

The user interface (UI) of DRABT, a digital flashcard system, is designed to be intuitive, accessible, and efficient. Whether on a website or mobile app, users should be able to create, edit, and study flashcards with ease. To achieve this, the UI structure is built around clear navigation, seamless interactions, and user-friendly components that enhance the overall experience.

Website Pages and Navigation Structure

The DRABT website is composed of several key pages, each serving a specific purpose in the user journey. Every page is designed to be easy to navigate and visually appealing.

#### 1. Splash/Login Page

The splash page is the first thing users see when they access DRABT. Here, they can log in, sign up, or continue as a guest. The page features a clean design with fields for entering a username and password, along with buttons for logging in and signing up. Users who forget their passwords can click on a “Forgot Password?” link to recover their accounts. Additionally, branding elements like the DRABT logo and a short introduction provide context for new users.

#### 2. Home/Dashboard Page

After logging in, users arrive at the home page, which acts as the dashboard. This page showcases all the flashcard decks the user has created or imported. Users can create new decks with a “Create New Deck” button, search for specific decks using a search bar, and organize their content with sorting and filtering options. The dashboard also includes a user profile section and quick access to settings.

#### 3. Deck Editor Page

The deck editor is where users can create and manage flashcard decks. Users can name their decks, view all flashcards within them, and add new flashcards using an “Add Flashcard” button. Deck customization options, such as background color and text style, allow for personalization. To ensure flexibility, users can save, edit, or delete decks as needed.

#### 4. Flashcard Editor Page

For users who need to edit individual flashcards, the flashcard editor page provides a straightforward interface. Users can input text for both sides of a card and upload images if necessary. There is also an option to add tags for better organization. The page features clear buttons for saving or deleting flashcards, as well as an intuitive flipping mechanism to switch between the question-and-answer sides.

#### 5. Flashcard Study Page

The study page is designed for an engaging learning experience. Each flashcard appears one at a time, showing the question side first. Users can flip the card using a “Flip Card” button to reveal the answer. Navigation buttons allow users to move forward and backward through the deck, while a “Shuffle” button randomizes the order of the cards. A progress tracker shows how many cards have been reviewed, and users can exit the session with an “End Session” button.

#### 6. Settings/Profile Page

The settings page offers customization options that let users tailor their experience. Users can change their password, adjust accessibility features like font size and color contrast, and enable audio dictation. A logout button is also present for convenience.

## UI Components and Styling

The DRABT interface incorporates well-defined UI components to ensure clarity and ease of use:

Navigation Bar: A top navigation bar remains visible across all pages, providing quick access to the Home, Deck Editor, Profile, and Study Mode sections. A logout button is also included.

Buttons: Clearly labeled buttons with hover effects and click animations make interactions smooth. Important actions like “Save” and “Study” are highlighted to draw attention.

Forms and Inputs: Simple and user-friendly text fields make it easy to enter data. Buttons are large and easy to tap, ensuring accessibility for both desktop and mobile users.

Flashcard Display: Flashcards feature high-contrast text and customizable background colors. Users can choose to enlarge text or activate audio reading for better accessibility.

User Interactions: Users can flip flashcards with a tap or click. Keyboard shortcuts (e.g., arrow keys) allow for quick navigation, and drag-and-drop functionality enables easy reordering of decks and flashcards.

## Navigation Flow and User Experience

The UI of DRABT follows a structured, intuitive navigation flow:

Users start at the Splash/Login Page, where they sign in or continue as a guest.

They proceed to the Home/Dashboard Page, where they can view and manage decks.

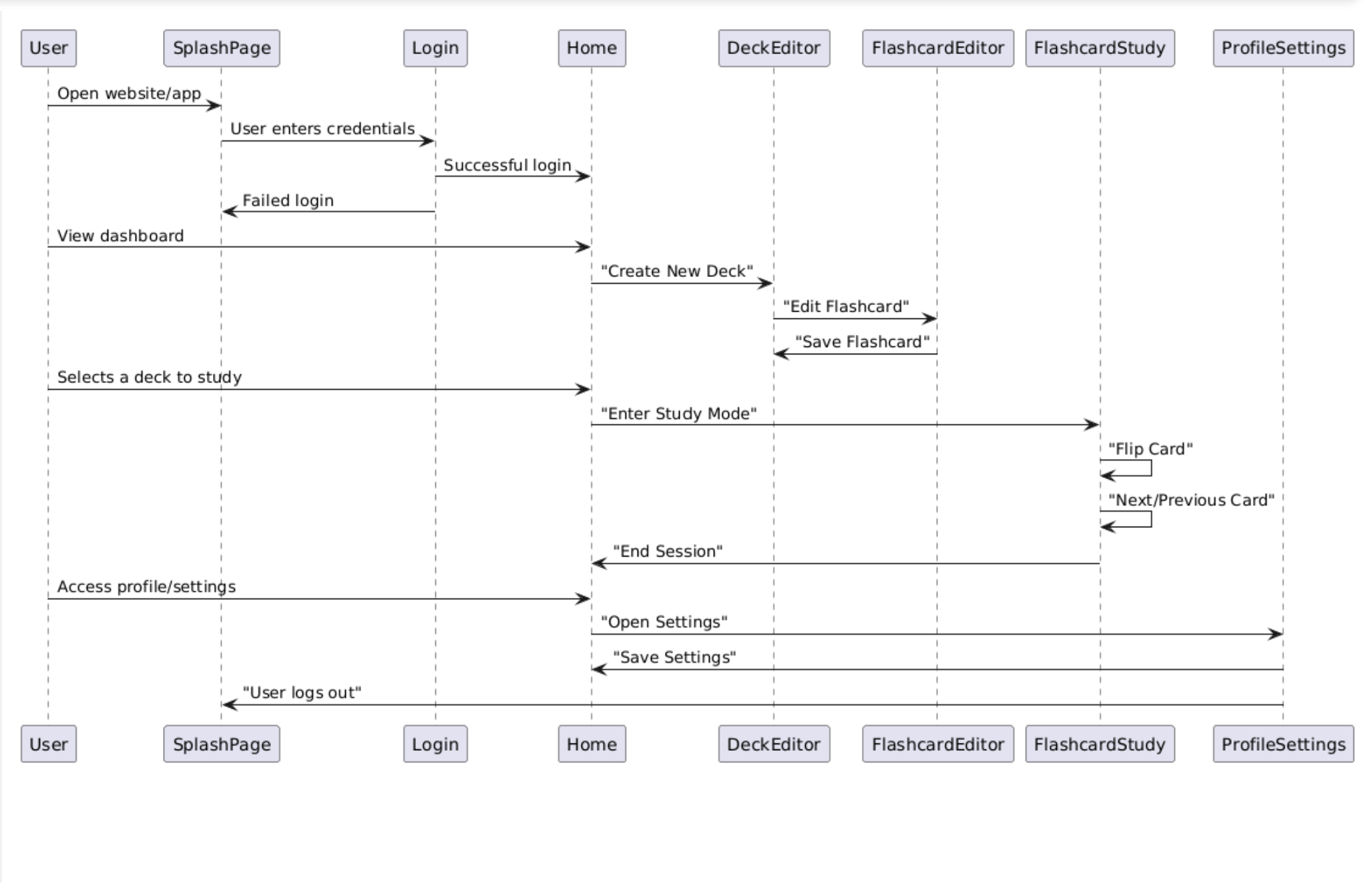
Selecting a deck takes them to the Deck Editor Page, where they can add and organize flashcards.

Clicking on a flashcard brings them to the Flashcard Editor Page for detailed editing.

When ready to study, users navigate to the Flashcard Study Page to review their flashcards.

The Settings/Profile Page allows users to customize their experience.

Each transition is designed to be smooth and responsive, ensuring minimal loading time and an enjoyable user experience.



# Mobile App System Design

# The mobile app will be built using Unity for cross-platform compatibility. For the backend, we will use Node.js with Express.js to handle API requests and manage database interactions efficiently. The backend will be connected to a cloud-based database, such as Firebase Firestore or MongoDB Atlas, to ensure scalability and real-time data synchronization. For frontend development, while Unity primarily handles UI, we may integrate React Native (JavaScript/TypeScript) or Flutter (Dart) for hybrid development where needed. The frontend will follow a component-based structure using React Native’s component system to ensure modularity and reusability. Although JavaScript is not strictly class-based, we will adopt ES6+ classes and TypeScript (if applicable) to enhance code structure, enforce type safety, and improve maintainability.

# To optimize performance, we will utilize:

# RESTful APIs or GraphQL for efficient data fetching.

# Caching strategies (e.g., local storage, IndexedDB, or Redux Persist) for improved user experience.

# Unity’s built-in optimization techniques, such as object pooling and asset bundling, to ensure smooth performance.

# Security considerations include:

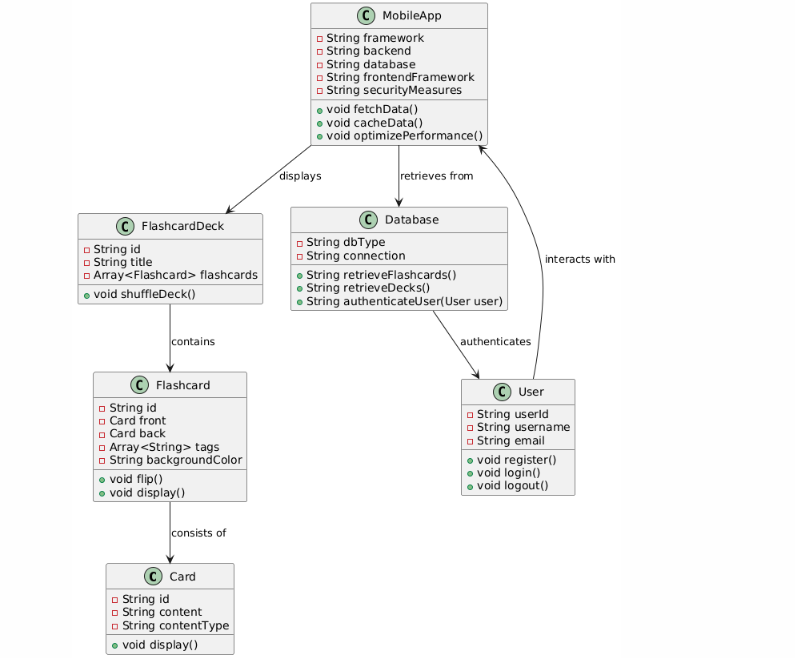
# Authentication and Authorization (OAuth, Firebase Auth, or JWT-based authentication).

# Data encryption for sensitive user information.

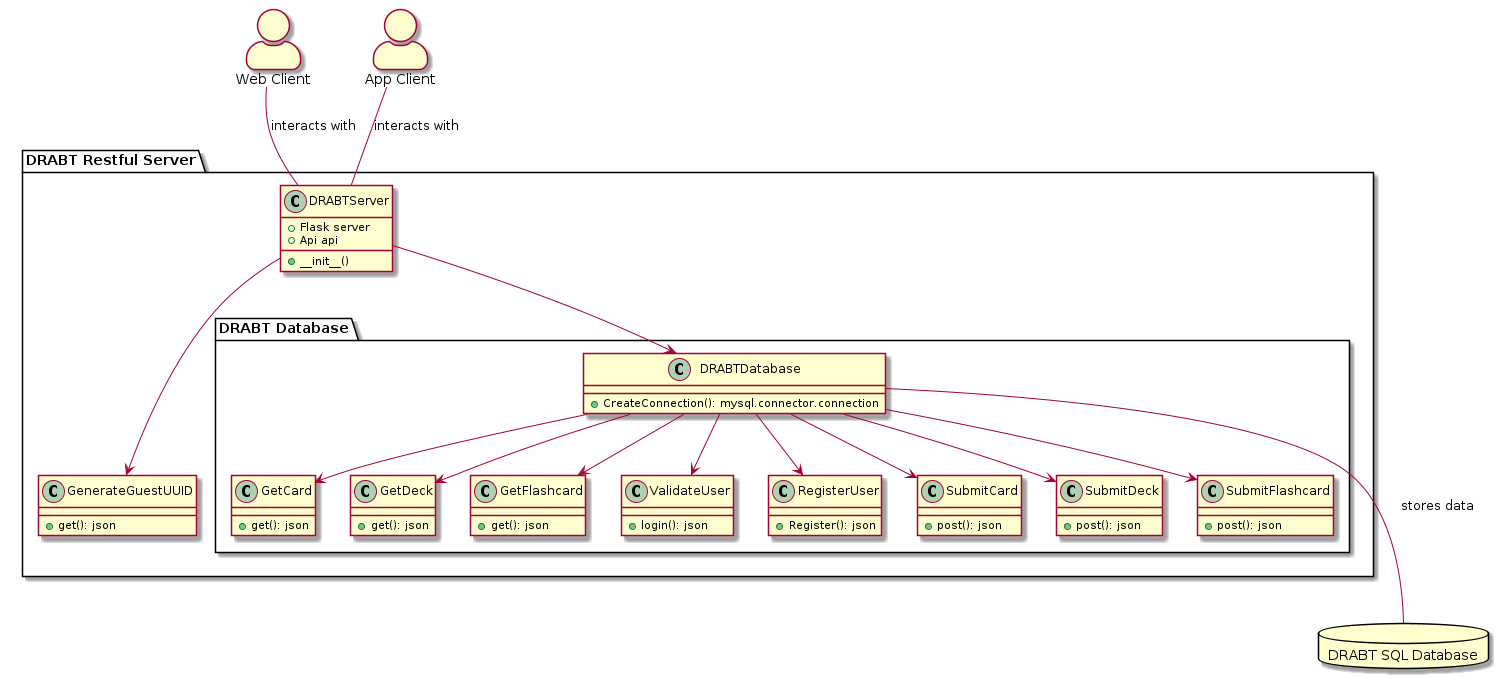
# Secure API communication using HTTPS and token-based authentication.

# By structuring the app this way, we ensure a scalable, maintainable, and high-performance mobile experience that integrates well with backend services and database management.

UML Diagram



# Server System Design

  
 Running on the host server is a python script using the REST library that is used to query a local SQL database for user, deck, and flashcard information. The web and app clients send GET and PUT requests to this script, and the script validates those requests. Functionality includes the ability to generate unique IDs for guest users, write updates to the database, and return query results in JSON format.

The DRABTServer class serves to spawn a REST server, monitor incoming traffic, and includes a class to generate unique IDs.

Then there is a DRABTDatabase class which encapsulates the SQL queries to the database. It includes several classes that can be invoked to either return data from or submit data to the database.

# Database Design

The SQL database for DRABT consists of five tables. The ‘decks’, ‘cards’, and ‘flashcards’ tables contain the id of the user who created them. Entries in the ‘users’ table have unique names and email address.  
  
 Each entry in the ‘cards’ table represents a single side of a flash card, and can contain up to 50 text characters, or an image (in which case the ‘text’ field becomes alt-text on the website). The ‘flashcard’ table uses two references to the front and the back of the card.  
  
 Entries in ‘decks’ have a name, but in order to represent the many-many relationship that can exist between flashcards and decks, a separate table ‘deck\_flashcards’ is used which has a list of deck-flashcard associations.