## Chapter 3: Project Specification

This chapter outlines the all of the specifics of the design of the application. All of the following sections have been created with a target userbase of those already interested in language learning.

### 3.1. System Overview

For the initial release, Dokkai will be a web application. Users will be able to register accounts using an email and username, track their flashcard and overall language progress, study through other tools within the app, and view other users through gamification systems such as a leaderboard. They will be able to create custom flashcards, import cards from lessons, import decks from Dokkai's official repository, and review them all through the most efficient SRS system (Su et al., 2023) currently in existence. At launch, the platform will support English for the interface, and Japanese for the language content. Further languages are planned but exceed the scope of the MSc project.

### 3.2. Intended Users

As discussed earlier, serious language learners who ideally have goals of fluency within a reasonable timeframe are the target audience for Dokkai. This includes both beginners who want to make language study a consistent habit and intermediate learners that want to push themselves further. All flashcard and lesson content will be designed to start from zero knowledge, with the option to skip over lessons and add only advanced words to the user flashcard deck if one wishes. The platform therefore supports three different types of users. These are a guests (non logged-in user, likely first time visitors), learners (logged-in users), and an admin, who can manage user accounts and database information when necessary.

### 3.3 Functional Requirements

The requirements listed here are all presented with the MoSCoW method. This is to easily distinguish between "must haves" for essential features, "should haves" for features that should be included in an ideal state, but aren't essential for the app to function, and "could haves" for anything that may be able to be added (Ahmad et al., 2017) if time allows for it to be developed. These will be separated into three different categories for each of the types of user, with the most significant list being for the learners.

## Guests

1. The system **must** display a “Home” page with an overview of the functionality of the app and a sign-up button.
2. This home page **must** be the landing page for guests.
3. The system **must** include clear navigation bar links on the home page to the “How to Use” and “About” pages so that users are able to have all key information about the app before registering.
4. The system **must** allow users to register for an account by entering their email, a unique username, and a password.
5. The system **must** enforce password strength rules that require criteria such as capital letters and symbols to ensure that accounts are secure.
6. The system **must** allow users to log in securely using their username and password.
7. The system **must** use the provided email for password recovery and optional promotional communications.
8. The system **must** restrict access to the flashcard, profile dashboard, and other feature pages when a user is not logged in, prompting a sign in page if they try to access them.
9. The system **must** restrict access and prompt users to login if they click on the pages on the navigation bar when signed out.
10. The system **should** provide clear password strength recommendations during account creation.
11. The system **could** notify guests of new features through an announcement section on the “Home” page.
12. The system **could** provide a demo mode allowing guests to try a short flashcard review session or grammar concept lesson before signing up.

## Learners

1. The system **must** display the profile dashboard upon logging in, showing the user’s study streak, activity feed, total flashcards, and other overall progress information.
2. The system **must** allow users to create flashcards that includes fields for the vocabulary word or concept, word reading, meaning, and an example sentence.
3. The system **must** use the FSRS algorithm to schedule these flashcards for review based on the user’s performance history and answer ratings.
4. The system **must** allow users to review their flashcards using “again” or “good” buttons in order to reschedule them.
5. The system **must** include a navigation bar at the top of the screen for accessing all core features of the application, with clear labels for each.
6. The system **must** provide recommendations for time to engage with native content engagement through the time spent on flashcards.
7. The system **must** allow users to update basic account settings, including their email address, password, and profile picture.
8. The system **must** allow users to export their flashcard data as a downloadable file.

9. The system **must** include gamification features, such as an XP and leveling system and a leaderboard based on these elements.
10. The system **must** allow users to have the option to make their account activity private if they do not want other users to be able to view their activity through leaderboard pages or a direct link to their profile.
11. The system **should** allow users to import pre-made flashcard decks into their main deck, and be able to use them like they would with any other card.
12. The system **should** have a basic grammar course that is integrated with the flashcard system through “add to deck” buttons.
13. The system **should** have a hiragana and katakana guessing game where streaks are maintained within their account or current browser session.
14. The system **could** support the creation and sharing of custom flashcard decks between users, being moderated by administrators and Dokkai staff.
15. The system **could** recommend native reading or listening materials dynamically based on the user’s flashcard data, though this may require additional data privacy agreements.
16. The system **could** integrate with third-party open-source dictionaries or AI APIs to auto-generate example sentences based on learnt grammar content and vocabulary. For clarity, Appendix 3 displays a sequence diagram for how this chatbot would operate within Dokkai clearly distinguishing itself from other AI assistants in current language learning applications.

### Admins

1. The system **must** include an admin account role with permissions that allow them to manage user accounts and associated data, and moderate and information.
2. The system **must** allow admins to review flagged or reported flashcards for inappropriate or potentially system-damaging information and code.
3. The system **should** notify admins of suspicious activities, such as repeated failed login attempts or abnormal account activity such as adding many flashcards in a short time period (e.g. 100 cards within 5 minutes).
4. The system **could** allow admins to create and manage new pre-made flashcard decks for learners to import and use in their own decks.
5. The system **could** enable admins to manage interface translations as new languages are added.

## 3.4. Non-functional Requirements

The functional requirements in this next section make use of the ISO/IEC model that splits the aspects of the system into distinct categories (Raharja & Siahaan, 2019) for clarity. This provides a standardized model to evaluate what the system must do through categories such as performance, security, and usability (Raharja & Siahaan, 2019). The MoSCoW system from the previous section will also be used in combination with it.

### Performance Efficiency and Compatibility

1. The system **must** load all key pages such as the home page, the dashboard, and the flashcard page within 1-2 seconds under normal network conditions. For the flashcard page specifically, all elements must also be rendered and be interactable during this same time frame.
2. The system **must** respond to user actions, such as rating a flashcard “good” or “again”, with minimal delay to ensure a smooth study experience.
3. The system **must** be compatible with all major modern browsers such as Chrome, Firefox, and Edge.
4. The system **should** display correctly and remain fully functional on mobile browsers such as Safari. The key test here is that flashcards should be usable from a mobile browser for portable study.

### Usability and Reliability

5. The system **must** be intuitive, have instructions for each of the features, and be easy to navigate, even for users who may be unfamiliar with SRS tools or language learning methods in general.
6. The system **should** use clear buttons and feedback messages to guide the user through key actions and pages.
7. The system **should** follow a clean and minimal design, reminiscent of those identified during the literature review, in order to reduce distractions during study.
8. The system **must** be stable when under normal usage conditions with disruptions being limited to planned periods of maintenance only.
9. The system **must** store user data persistently in the database without unexpected losses or corruptions.

### Security, Maintainability and Portability

10. The system **must** store user passwords securely using hashing so they are not visible in plain text to any threat actors or malicious attacks.
11. The system **must** only give access to personal user data and flashcards to the correct user that is currently logged in, apart from admins for moderation purposes.
12. The system **could** provide an option to log out of inactive sessions. This will be after a period of time has passed to determine that the user is idle.
13. The system **must** follow an architecture design that cleanly separates the frontend, backend, and database layers for easier maintenance and development in the future.
14. The system **should** include clear code comments throughout all of the system files that help support the addition of new features in the future.
15. The system **should** provide a layout that is responsive and adjusts page elements to mobile browsers seamlessly allowing users to study content portably.

### 3.5. System Tools

Dokkai will be built using Flask and Python as the framework for the backend, being chosen as it is simple and flexible, as well as being the framework that I currently have the most experience developing with. The database will initially use SQLite due to low user counts being expected for testing and on the first release. This will be managed through SQLAlchemy. For the frontend, Jinja2 is used with Bootstrap to provide a portion of the layout, being also used in combination with custom CSS and Javascript. Javascript will be responsible for the interactive features of the flashcard system such as keyboard bindings to make studying a smooth and accessible experience for learners. Flashcards are scheduled using the aforementioned FSRS algorithm, to optimise review timing for reasons previously outlined. This will be used in combination with a custom “steps” system which will be outlined in a later section.

Development will be carried out in PyCharm with GitLab for version control, using a university-managed repository, being regularly committed to. This ensures secure archival of development as well as protecting against data loss or corruption. Commits will be conducted at least once per week. Sprint testing will be done locally by running the Flask app through Pycharm’s built in run configuration. This launches the development server at http://127.0.0.1:5000 on the local machine, allowing features to be quickly tested and debugged which is ideal for Dokkai. Appendix 1.3 details the instructions for starting the application locally. Deployment for the testing phase will be initially done through PythonAnywhere. The Flask app will run using WSGI and Gunicorn, and static files will be served to the platform directly. This may not mirror the final design of the project when it’s deployed. Design diagrams and wireframes will be created using the free version of Lucidchart and Adobe Photoshop, occasionally combining both platforms for consistency of table sizes and text fonts.

### 3.6. Data & Sourcing

The flashcard review scheduler for Dokkai is the aforementioned FSRS, a scheduling algorithm designed to improve memory retention and make study time more efficient (Su et al., 2023; Ye, 2025). Implementation be sourced from the official Python repository fsrs4anki (Ye, 2025) and imported to the Dokkai code through the python library supported by PyCharm. No third-party user data will be used during development, with initial tests being run only by myself. The HTML and CSS of the flashcard template were designed before the development of the app began, and this will be imported and integrated into Dokkai’s flashcard system during development. Appendix 4 displays the card template in its original implementation and design in both monolingual and bilingual card formats, along with a link to the original GitHub repository.

### 3.7. Resource Constraints

The project will be developed entirely on a personal laptop with no additional paid services being used unless it was provided by the university. All software and frameworks used will

be either open source or available through university resources. Research material was accessed through online databases using institutional login when required.

### 3.8 Features not considered for the application

In the real world, conversations with sympathetic native speakers who are willing to help the acquirer understand are very helpful (Krashen, 1982) in developing output ability. Though this is the case, and native communication features were judged to be a positive (Tommerdahl et al., 2022) element during the research section, real-time communication and chat features will not be implemented in the initial release.

The decision here is grounded in the idea that a more productive use of development time would be polishing the other learning tools and keeping in line with the input-first philosophy. Creating the groundwork for communication features is still considered to be important for advanced users, but remains secondary to the other outlined features as it was also not identified as a gamification element of any of the existing popular apps. An alternative approach to the feature for the future could also incorporate elements of the proposed AI grammar helper in section 3.3.1, allowing users to interact with the AI to also simulate dialogue with a native, perhaps after reaching a certain account level to ensure they aren't outputting too early in line with Krashen's (1982, 2008) theories.

## Chapter 4: Legal, Ethical, & Social Considerations

### 4.1. Legal

Dokkai will use publicly available knowledge, resources, and software during development. All libraries and tools are open-source or were accessed under licenses provided through university portals. FSRS is sourced from the official FSRS repository. It was released under an open license and is free to use and modify. No other proprietary systems or third-party APIs are planned to be used within any of the "must have" features. A "could have" functional requirement detailed the possibility of the implementation of an AI assistant, in which case OpenAI's most recent GPT model may be accessed via the API to assist with the generation of example sentences for users during grammar study. No third-party user data will be accessed during development. Any data used during testing will be created manually in order to evaluate and confirm the core functionalities of the app are working as intended through regular sprint tests.

### 4.2. Ethical

Ethical concerns are minimal but some may arise should any of the requirements about user-generated flashcard content make it into the initial build. These features are not listed

under “must have” and therefore may not arise in the intial build, though it is worth noting that all submitted content would not be visible to other users unless moderated and pushed to the custom decks page by the previously defined admin user accounts and Dokkai staff. It would be checked for being harmful, offensive, or potentially containing copyrighted material.

All participants in testing will be over age of 18 and will be asked to give clear consent before taking part through Google Forms. They will be instructed that results would only be used to assess how the platform supports language learning and in achieving their personal goals in the short-term. Testing cannot be conducted in the long-term due to time constraints. Further testing guidelines and information will appear in a later section. All third-party tools and libraries have been reviewed to ensure they do not pose ethical concerns such as there being invasive scripts or hidden downloads within the code.

### 4.3. Social

While Dokkai will not be intended to be primarily a social platform, it incorporates gameification elements such as leaderboards to encourage engagement and a sense of community among users of the platform. These elements will be ideally designed to nurture motivation through friendly competition, such as being able to see other user’s flashcards in their account and their overall profile level, but it has also been taken into account that some may prefer to avoid these features entirely in favour of staying focused on their own language learning journey. For this reason there will be an option to hide all profile activity from other users in the profile settings.

#### 4.3.2 Contribution To Language Learning Communities

Beyond the leaderboard systems within the app, Dokkai may contribute to the broader language learning community online by further promoting awareness of goal driven self-study and encouraging users to use language learning literature to achieve their goals.

#### 4.3.3 User Testing

All user testing will be conducted with adults over the age of 18 and informed consent will have to be given for them to participate. This testing will use a close to complete version of the initial release of the app, ideally with all planned features implemented. A questionnaire will be used to collect results about satisfaction with the key domains of the application. These questionnaire results will not collect any personal data of the participants, giving them test user accounts to work with, and focus solely on the reactions and experiences with the app. These test accounts along with their progress will be wiped from all local backups upon the completion of the testing period.

### 4.3.4 Accessibility

With assistance from the WCAG 2.1 guidelines to ensure usability for users with a wide range of disabilities (Chatziemmanouil & Katsanos, 2024), text-to-speech elements will be developed for flashcard content and will be able to be turned on and off by the user.

These also double as being effective for users who may not be in need of accessibility seetings, as they can check their answers to flashcards audibly rather than just reading the vocabulary word on the screen. Font sizes will be readable on both desktop and mobile browsers, and no animations or visual distractions are expected to be included that could negatively affect users with sensory sensitivities. A site-wide dark mode will also be included as an option when the user is logged in through a flip in the CSS template. One final accessibility addition is keyboard bindings will also be provided for flashcard interaction should a user wish to not use their mouse.

The overall theme here is that Dokkai aims to provide as many considerations for users to have a comfortable study experience to ensure that there is no friction in getting users to come back daily and make long-term language gains. The site will be available in English only for the initial release, and the language content will only be available in Japanese. Theoretically, a user could create flashcards for other languages within Dokkai making use of the card template and FSRS scheduler, but this will not be officially supported or advertised to test users.

### 4.3.4 Inclusivity

Although Dokkai is focused on those already with an interest in pursuing language learning, it is designed to be able to be intuitive and accessible to anyone regardless of their background in language learning. The app will not assume prior knowledge of language content, starting vocabulary from zero using frequency lists and grammar content starting with basic concepts, with the option to skip to later sections or more advanced flashcard decks should the user wish to. Explanations of vocabulary words and grammar will be written to be easily understandable to those who may also not be native speakers of English. Usage of complex English grammatical terms that a non-native speaker may not have familiarity with will be avoided where possible in order to achieve this inclusivity goal.

## Chapter 5: Design

This chapter will outline the specifics of the system’s design with the components of the application covered in detail. It covers the core structure of the entire system, dividing responsibilities across the front-end, back-end, and database layers. This will establish the entire foundation for the application’s development.

## 5.1. Architecture

The application uses a three-layer structure. At the top of the structure is the front-end, which contains the parts of the site that are visible to the end-user such as the homepage, login and register forms, and profile dashboard. These are rendered using standard HTML templates with Flask, Jinja2, and Javascript for managing dynamic content. Bootstrap is also used for styling and layout along with custom CSS. The middle layer is the application logic, handled by the Flask and Python back-end. This component handles user requests, validates input, and includes the most complex functionalities of the app such as the scheduling flashcard logic and AI assistant code. It also handles user authentication, ensuring that actions that should only be handled by authorized users are correctly performed.

The final layer is the database, which uses SQLite to store and manage all persistent data, such as user accounts and their associated flashcards. The database also stores elements such as activity feed items and the users account level. Data management and queries are handled with SQLAlchemy. Each user's data is separated and linked through the relationships that will be defined in the models.py file, which will be illustrated in detail in a later diagram. As Dokkai is currently a project created and maintained by a single person, a complex tech stack is not necessary and this three layer approach is sufficient. This stack is also illustrated in figure 5.1.

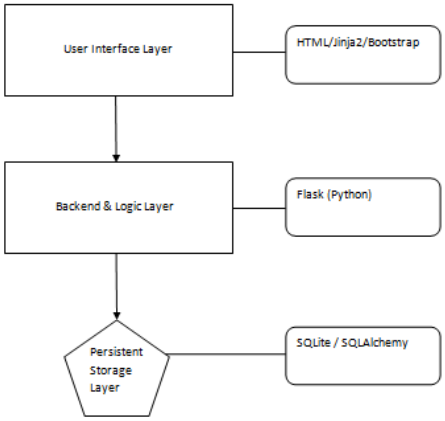



Figure 5.1: Tech stack diagram to what will be used at each level of Dokkai.

## 5.2. System Components

Each of the three system layers is divided into smaller components to maintain simplicity and improve manageability during new feature development or maintenance. Pages such as the homepage and grammar pages are constructed by using reusable elements such as the navigation bar and UI elements. This ensures consistency across the platform and reduces redundant code.

When the user interacts with the application, such as rating a flashcard, the front-end sends a request to the back-end where Flask processes the route logic and validates any requests should it need to. Before any action is carried out the system verifies all of the key conditions such as if the user needs to be logged in and whether they have the authentication to carry out the action. If the request requires any data to be retrieved or updated in the database, SQLAlchemy queries or modifies the database information and returns anything necessary to the front-end. This structure is illustrated in figure 5.2, with dotted lines to clearly show the three layers of Dokkai.

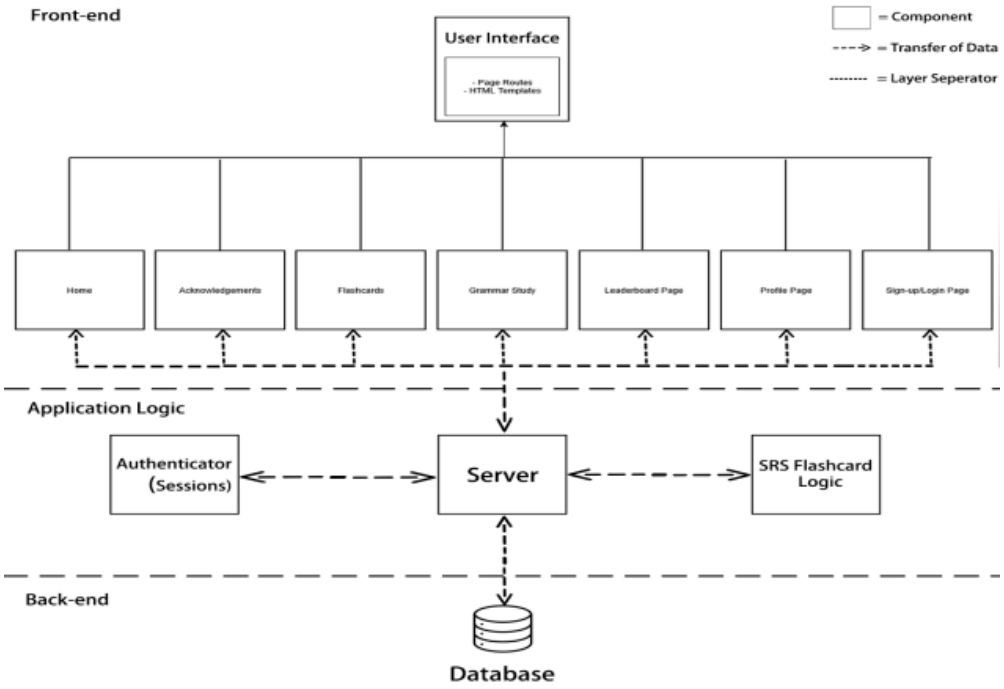



Figure 5.2: Diagram displaying how the different layers of the system interact with each other.

## 5.3. Database Design

As mentioned previously, Dokkai will use a relational database that is managed via SQLite and SQLAlchemy to store and query persistent data. The schema is designed for clarity and security, being commented appropriately within the code to make the addition of new columns seamless. The main tables will include:

1. Users – stores account information, profile settings, XP levels, and activity preferences for their visibility to other users through their profile.
2. Flashcards – stores vocabulary words and other card information such as example sentences, associated user IDs, and spaced repetition data such as intervals and due dates.

3. Decks – stores organized groups of flashcards. Users will have their own personal account deck, as well as Dokkai having custom made decks that can be imported to user accounts.
4. Activities – records of user actions such as daily streaks of flashcard usage, which is then given to features such as the activity feed for the user's profile.

Figure 5.3 illustrates these relationships between the tables. There are foreign keys linking flashcards, decks, and activities to individual users. This structure is also appropriately commented in the code. It ensures that queries are efficient and that the relationships are clear and flexible for future features such as user made decks or an expansion on the activity feed. The classes here only cover functionality defined within the "must haves" of the functional requirements section, as additions may have to be made if additional features are added during development. These would likely be only an additional line in these pre-existing classes, and will not constitute an entire new class.

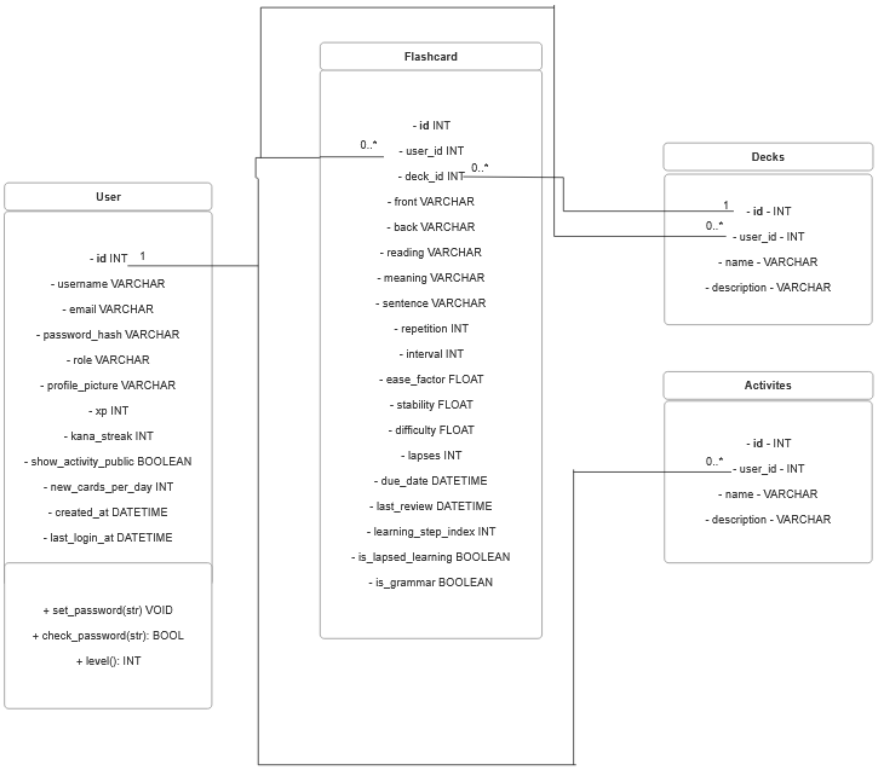



Figure 5.3: UML class diagram displaying the database items and their relationships within the application. Includes all necessary information for users, flashcards, flashcard decks, and activity feed items. The lines between components show the foreign key links.

### 5.4. User Interface

It is essential to present content in a palatable way, similar to existing platforms, in order to not deter new users. It operates on the principles of being minimalistic while containing all key information necessary to users for effective study, as well as feeling familiar and easy to use to both individuals with experience using language learning applications and those who are trying it for the first time. As such, wireframes have been produced in line with this philosophy.

#### 5.4.1. Minimal Design

Figure 5.4.1 illustrates the wireframe design for Dokkai's homepage with placeholder text for some of the elements. This will be the landing page for users when they visit the site without a link to a specific page. A consistent colour scheme for the UI had not yet been decided upon at the time of these diagrams creation, hence their usage of black, white, and grey, but will appear in a later section.

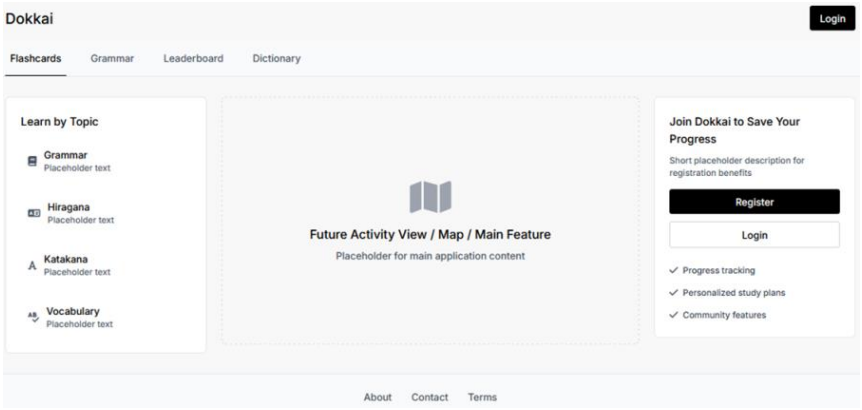



Figure 5.4.1: Wireframe diagram for the home page when a guest visits the site, or when a user is not logged in.

#### 5.4.2. Intention

Dokkai's role in the user's language learning journey will be clear from the landing page with short slogans to communicate the philosophy of the application at first glance. The intention here is to clearly communicate to serious language learners and those interested in the endeavour that this is a platform they can trust to help achieve their goals of basic fluency and beyond in their target language. The about page will also be available to guest users where a more detailed mission statement and language learning philosophy explanation will be provided in a palatable way to new users, so that it is easy to understand to those who may not be familiar with the process. The about page will also outline how user data is used and stored, creating a bond of trust and transparency between Dokkai and users.

### 5.4.3 Intentional Learning

To further illustrate Dokkai's commitment to purposeful and goal-driven learning, UML diagrams have been produced to show how a typical learner would interact with the system. It illustrates the FSRS scheduler as an actor in the system, and how the system mediates in this process. This is shown in the use-case diagram in figure 5.4.3, and in the activity diagram in Appendix 5.

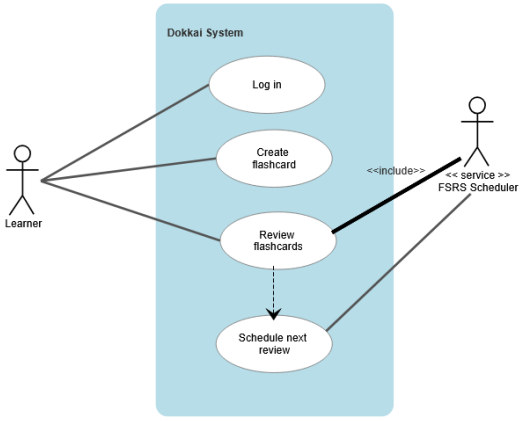



Figure 5.4.3: Use-case diagram showing the interaction between the user, the system, and the FSRS scheduler for their flashcards.

### 5.4.3. Clarity & Simplicity

As the diagrams should now have made clear, the application prioritizes a streamlined and distraction-free experience for learners. While gamification elements will be a large component in boosting user motivation should they desire it, they are intentionally placed away from the flashcard experience to not be distracting. This simplistic design also allows for an easier experience on mobile browsers as there are fewer elements other than essential features to consider for the screen orientation and size. The navigation will also be kept simple, clear, and intuitive with the currently active page being highlighted on the bar so that learners always know what page they are using. Buttons and labels will be consistent through the reusage of Bootstrap and custom CSS elements, reducing any confusion that may arise when new users try additional features outside of the flashcards.

### 5.4.4. Profile Page & Gamification

The profile page will provide learners a simple space to view their learning stats through simplistic, clear figures, as well as manage their account settings in a single area. It will display key numbers such as total number of flashcards studied and review streaks. These values will also be the same as they appear on the leaderboard. The profile page also displays these values next to their username and custom profile picture, fostering a sense of attachment to their progress and flashcard database, promoting continued engagement.

This page will also be where users can export their data for offline backup. The wireframe is displayed in figure 5.4.4.

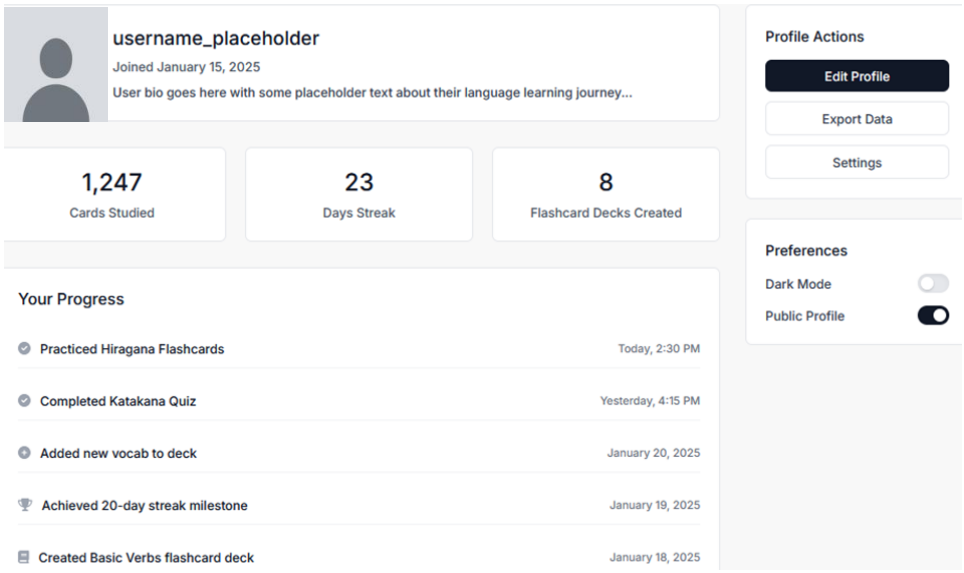



Figure 5.4.4: Profile Dashboard Wireframe

### 5.4.5. Labelling

The application will use consistent labels and buttons for all elements. It is also essential that bootstrap and custom CSS elements are consistent with each other. They will clearly communicate their purpose to users, with alternate text being implemented in all relevant HTML lines to ensure accessibility for impaired users.

### 5.4.6. XP System & Profile Levelling

The XP system is tied to accounts and is displayed on the profile page. During a flashcard review session each card will increment a counter for the current session, with XP being rewarded once per day upon full completion of all the cards in the deck. This will also add an activity feed post. The session counter will be multiplied by 10 to create the XP value, adding this to the account's current value and updating the values on the profile and leaderboard pages as necessary. This combines the gameification, light social leaderboard elements, and flashcards all into an encouraging workflow to ideally keep users coming back daily to achieve their goals.

Profile level will come from the total XP value by using a square root formula of the XP value divided by 100. This makes earlier levels easy to reach through a few days of study, setting apart new users from intermediates. Higher levels are therefore representative of a

user that has spent a significant amount of time with the app and, ideally, has a large vocabulary through using their flashcard deck for so long.

### 5.5 UI Accessibility Considerations

Accessibility will be built into the UI design through toggles and profile settings, as well as standard practices such as alt text, to allow as many language learners to interact with the platform as possible. Not all of the aforementioned WCAG 2.1 guidelines apply to a language learning platform, but all of the relevant principles have been applied to the design philosophy. These primarily focus on readability through optional dark mode, minimalistic study pages, keyboard navigability, and clear focus indicators, creating a comfortable study environment.

## Chapter 6: Project Management and Development Strategy

The project follows an agile development approach as this allows for flexibility to add the previously outlined “could have” features should time allow for it. Development will be broken into short sprints with each being dedicated to the development of a specific feature. The application will remain functional locally throughout this process and will be incrementally tested locally through the Flask development server in PyCharm. Regular commits will be made the aforementioned GitLab repository. Early commits will be made roughly every week during the early stages, changing to being more frequent as development accelerates later into the project.

### 6.1. Timeline

A timeline for development has been created that covers the duration of the MSc project. The 15-week schedule covers all of the core features while leaving room for UI refinement and additional “could have” features. As the project uses an agile approach, the timeline serves as a flexible roadmap that allows priorities to shift, though the key features such as FSRS integration and the activity feed should be completed by the deadlines in the chart to ensure smooth progress.

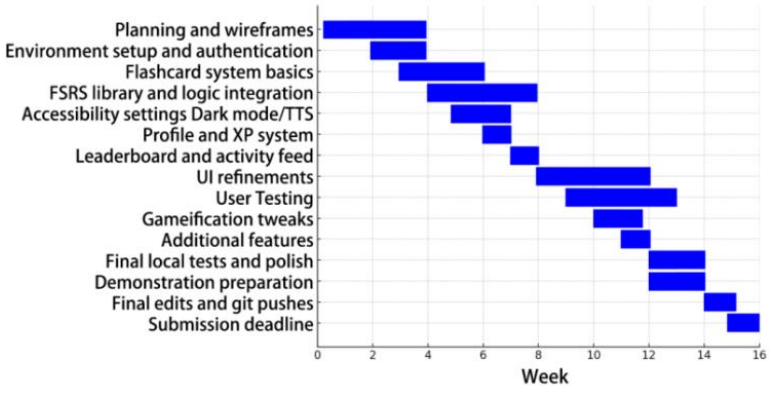


Figure 6.1: Gantt chart that displays project management through the timeline of when tasks were completed.

### 6.2. Management Practices and Risk Management

Regular meetings will be held with the project supervisor to review progress and get feedback on development. These will take place fortnightly, alternating between online over Microsoft Teams and on campus, as well as emails being exchanged whenever there is a question that doesn't warrant a full meeting. In addition to the figure in section 6.1, an informal .txt file was maintained with daily notes to maintain focus when working on multiple parts of the code and project report at once. Code comments will also occasionally be dated and will be documented throughout the version control history on the GitLab repository. The primary risk that will be experienced during the project is data loss as development is done fully on a single personal laptop. To mitigate this, as well as files being pushed to the GitLab repository, they will also be locally backed up daily to an external drive.

## Chapter 7: Implementation

This chapter explains how Dokkai was built, outlining the core technologies and their roles across the aforementioned system layers. The section was written after the development of features were completed, hence the change to past and present tense when describing the system and development process. Examples within the main body of text will mainly focus on the flashcard portion of the app, due to it being representative of the core of the project, as well as being the most complex section to explain. All screenshots of code have been taken inside of PyCharm IDE using the default colour scheme. Line numbers have been intentionally left within the screenshots to ensure easy cross-referencing between code screenshots and the GitLab repository.

### 7.1. Front-end Overview

The implementation of the front-end tries to adhere to the wireframe designs from chapter 5 as closely as possible, though there may have been adjustments in the placement of some elements due to a change of preference, but this is not expected to have compromised the outlined design philosophy in any way.

### 7.2 Structure

The app is loaded from the run.py file which initializes the Flask application and links to the app folder. Within the app folder, the routes defined in views.py render the templates from the templates directory, which is exclusively filled with HTML files. There are shared elements for these render templates using the line: % extends "base.html" %, which is always placed on the first line of each file for clarity that it is reusing elements. This

ensures a consistent layout and UI. Appendix 1.2 displays an annotated graph of the file structure. It should be viewed from left to right, following the blue arrows as a representation of going deeper into the file structure.

Models.py handles all database interactions and defines the structure of tables and relationships through the foreign keys. Each table was represented in this file as a class. The User model stores authentication data, profile privacy and accessibility preferences, profile XP, and functions for password hashing and verification. The flashcard model stores the card content in each of the fields to render on the front-end, as well as information used by the back-end and SRS logic in views.py, such as the interval data, ease factor, stability, and due date of the card. The deck model groups cards into collections that can be imported to the main deck. The activity model logs user actions as they are completed and then feeds this to the features that need it such as the activity feed on the profile page. The foreign key relationships are clear and commented within the file when necessary, making it easy to understandable for potential future development. Appendix 6 displays the structure of the Flashcard model to give an example of this implemented in the code.

### 7.3 Views, Models, and Template Logic

Each route in views.py is decorated with an @app.route, and is protected with login_required whenever authentication is necessary. These routes handle the user interactions, pass data between the database via models.py, and render the HTML templates on the front-end. This will be shown throughout code snippets in Appendix 7. Appendix 7.1 displays how the route deals with loading the page, and the interactions at the front end during a review session. How these are handled in more detail will be discussed in the back-end section. For POST requests from learners, it reads the form data to determine how the user rated the card and then updates the session state, re-rendering accordingly.

### 7.4 Example Page

Following from the previous section, the template in 7.2 shows the usage of Jinja with Bootstrap for a portion of the UI in combination with the CSS. The previous section's view injects flashcard, cards in the queue, and a flag for show_answer. The template renders the front of the card unconditionally and shows the back through conditional "if" statements. Interactions are HTML forms that POST the card_id along with action=show or a rating value back to the same route, allowing the card to be rescheduled.

For the accessibility settings, three toggles are given to the user. There is word text-to-speech (TTS), sentence TTS, and the option to show flash messages with additional flashcard scheduling information. These are persistent preferences for the logged-in user by using localStorage, and are automatically preserved when refreshing. The code for this

process is displayed in Appendix 7.3 for clarity. Playback of TTS elements is only triggered when the back of the card is revealed by the user.

### 7.5 Interaction of Main Features, Backend, and Database

The home page adapts depending on whether the user was authenticated or not. Learners will have a different view when logged in with a query of the flashcard table that takes the amount of reviews over each day for the previous 7 days and then gives this to the template as card_data. This is then rendered using Chart.js to give users an easy to understand view of their recent progress.

#### 7.5.1. Explaining Card Types (Due, Learning, New)

The flashcard system was implemented using three different states for each flashcard that first have to be clearly defined for the next section to make sense. The card types are:

1. New Cards – Cards that have never been studied before with a repetition value of 0. These are introduced to the user's deck gradually based on their new cards per day value.
2. Learning Cards – Cards currently passing through the learning or relearning process. They are active within the deck and have timed intervals.
3. Review Cards – Cards that have reached their scheduled day for review and are shown to the user first when they open their deck for the day.

#### 7.5.2 Flashcards – Review Session and FSRS scheduling

Now that the card types have been defined, the logic behind the flashcards can be explained. The flashcard route in views.py mediates between FSRS scheduling and the state of the fields on each flashcard through the model. New cards advance through a "steps" feature. These are fixed values stored in new_steps that currently cannot be updated by users, with the values of 1, 5, and 30. These values are representative of minutes. On a GET, the route computes the review, learning, and new counts and selects the next card in order of priority and passes them to the template to render. The order priority is sorted by review cards, then learning cards due, and then new cards. The default amount of new cards is 10 but this can be adjusted by the user depending on how many words they would like to learn per day, increasing or decreasing their workload. Appendix 7.4 shows where these values are held in the code.

When a learner rates a card, a POST is sent with the rating of "again" or "good", which is then converted into the FSRS rating value and updates the card's state. A "good" rating advances the card to the next step or reschedules it based on the FSRS scheduler if it has passed through all of the steps, putting it into "graduation". An "again" rating resets it to the start of these steps, taking note of this for the FSRS scheduler that this card should be shown more often to the user.

```
            #review card (mature card in review stage)
            else:
                if rating == Rating.Again:
                    flashcard.lapses = (flashcard.lapses or 0) + 1
                    flashcard.repetition = 1
                    flashcard.is_lapsed_learning = True
                    flashcard.learning_step_index = 0
                    flashcard.due_date = due_in_minutes(RELEARN_STEPS[0])
                else:
                    flashcard.learning_step_index = None
            db.session.commit()
            flash( message: f"Card updated! Next due: {flashcard.due_date}", category: 'success')
            return redirect(url_for('flashcards'))
```

Mature cards are those that have been in the user's memory previously as they have already studied them to graduation. If these cards are marked "again" when they appear again in the queue via their scheduling date, they are treated as "lapsed" and follow the shorter step cycle of relearning_steps with only two values of 2 and 5 minutes. As these cards have been previously studied by the user, they have less need to be studied as often as the more recent cards in the queue do. After every interaction these changes are reflected in the database through the db.session.commit() line, with the cards themselves being tied to the user's profile through the relationships that were discussed previously. Appendix 7.5 shows how the flashcard page is displayed to the learner during an active study session. The buttons are clearly labeled, with "again" being red and "good" being chosen as brown to match the colour scheme that was eventually decided upon for Dokkai's UI. All interactions point to the same route so that the studying session remains seamless while the user goes through their due cards.

### 7.5.3. Different Card Adding Methods

Dokkai has multiple avenues for adding flashcards to the deck. Users are able to add cards manually, edit existing cards through a "browse cards" page, and import pre-made decks focused on specific topics, currently provided by Dokkai. Forms here are handled similar to other pages with FlaskWTF for validation and submissions updating the database record with SQL alchemy. Appendix 7.6 shows these features as they would be seen by users rendered in the browser.

### 7.5.4 Grammar Page and Adding Grammar Cards

Following from the previous section, cards can also be added to the main flashcard deck during grammar study. The grammar page is a simple GET that renders the HTML as long as the user is authenticated. All lessons are rendered at once with markers in the HTML to detect which lesson the user is on, along with buttons at the top to jump to a lesson.

Each Add button next to the example sentences for grammar concepts adds a new flashcard to the user's current deck. This is through a payload of front, back, reading, meaning, and example sentence. These flashcards are also tagged with is_grammar = True to distinguish them from other cards in their deck. This process is handled by a different route to the one that renders the grammar HTML page and commits the card to

the database. It also logs an activity to the model every time a grammar flashcard is added and displays it on the user's activity feed.

### 7.5.5 Additional Feature – AI Grammar Sentence Helper

This feature was not originally in the Gantt chart that is displayed in figure 6.2 and was implemented due to additional time found towards the end of development around week 12-13. Therefore, it should be treated as a proof of concept unlike the other features of the application which are more complete in their implementation. This was outlined in section 3.3.1's FR16 for learners about developing an AI assistant for grammar lessons. Due to its proof of concept state, it was not used in the build for user testing which will be discussed later.

The assistant detects the current lesson by scanning for lesson ID and picking the nearest one to where the browser currently is on the page. It then sends the message, lesson_title, and lesson_context to the endpoint. The server then returns the reply and flashcard information with the client filling the hidden "add to deck" form, which allows the user to add the generated example sentence and grammar concept to their main flashcard deck. These cards are created similarly to the previous section with the user id and flag for is_grammar being true. The OpenAI API key has been omitted from all commits to the GitLab repository for obvious privacy reasons, but can be tested by placing the key in the .env file present in the main project folder.

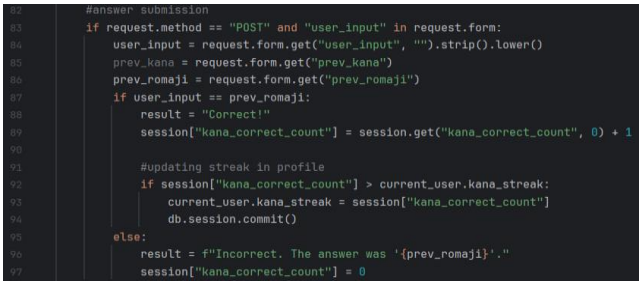
The prompt for the API call includes the lesson, a smaller version of the lesson content to not break token limit, and the users request inputted to the chat box. The code strictly enforces a specific way of outputting so that the flashcard is consistent with the current model in Dokkai, with all of the usual fields. Appendix 7.7 shows the implementation through how the prompt is submitted. The assistant originally used GPT-4 as its model during the local testing sprints, but this has since been deprecated by OpenAI in favour of GPT-5. Therefore if the assistant is tested now, this should be kept in mind that no guarantees can be made for output quality when using this model from the API.

### 7.5.6 Hiragana and Katakana Guesser

In line with Dokkai's philosophy of being an "all-in-one" language learning application, a guessing game was implemented for the basic Japanese writing scripts of hiragana and katakana. These scripts make up all of the phonetic sounds in the language, so this would make for a good starting point for absolute beginners. It is featured on the navigation bar with all of the other key components of the app.

The tool allows users to drill the Japanese alphabet by selecting the groups of five letters sorted at the bottom of the screen, and inputting the English equivalent for the Japanese

letter on their keyboard. These letter groups can be customised as the user wishes should they want to study specific groups of letters first and others later.



```
    #answer submission
    if request.method == "POST" and "user_input" in request.form:
        user_input = request.form.get("user_input", "").strip().lower()
        prev_kana = request.form.get("prev_kana")
        prev_romaji = request.form.get("prev_romaji")
        if user_input == prev_romaji:
            result = "Correct!"
            session["kana_correct_count"] = session.get("kana_correct_count", 0) + 1

            #updating streak in profile
            if session["kana_correct_count"] > current_user.kana_streak:
                current_user.kana_streak = session["kana_correct_count"]
                db.session.commit()
        else:
            result = f"Incorrect. The answer was '{prev_romaji}'."
            session["kana_correct_count"] = 0
```

It is also worth mentioning here that the tool was adapted from a previous GitHub repository, though the code was significantly changed and updated to the point that it is almost unrecognisable from the original implementation.

For the streak feature, the server state is kept in session for the current streak of correct answers and selected groups of letters. The value for the user's longest streak is held in their account through the user model. The buttons at the bottom of the page act as toggles that allow users to select groups, and builds the pool of guesser characters from the user's chosen groups. The buttons are clearly labelled, displaying the characters for each group, and their correct answers are held and pulled from the file kana_dict.py. When a group button is pressed by the user, the page posts back the value of the group and adds it to selected_groups, rebuilding the pool of characters that will appear to the user for them to guess at random. If the pool is empty or it's the user's first time visiting the page for the current session, the value is set to the default first five Japanese characters of あ, え, い, お, う (a, e, i, o, u) in hiragana. In the previously mentioned deck import feature, katakana and hiragana decks were created and were available to test users. This was to give the user the option to learn the characters through the core flashcard system rather than this drilling feature should they wish to.

### 7.5.7 Profile, Dashboard, and Gamification Elements

As planned in the wireframe shown in figure 5.4.4, the dashboard is the first page shown after a learner logs in or registers. This page presents the user's level, XP to next level, their amount of days active, and their personal activity feed, as well as toggles for profile settings and updating their profile picture. The feed displays the five most recent activity messages in the account from the database, and is always visible to the logged in user even if they have set their activity feed to private from other users. Appendix 7.9 shows the how this is implemented within the backend layer for reference.

The dashboard reads the values to display for the account directly from the database. Flashcard.query.filter_by() counts how many unique flashcards are in the account of the signed-in user. Days active is calculated by counting the different calendar dates in the

activity item rows when the message starts with “completed”, which will enumerate all of the days they have completed their flashcard deck, encouraging daily usage.

XP values are stored in the User table. The level isn’t stored directly, rather is calculated through views.py through square number growth so that each next level takes more points to reach than the last one. This ensures that long time users are rewarded with a high, hard to reach level, distinguishing them from beginners on the leaderboard and further encouraging long term engagement. The activity feed items are displayed according to their timestamp in descending order. Appendix 7.10 shows how this is displayed to the user on the frontend, and this therefore us what was shown to users during testing when they first logged in.

### 7.5.8 Leaderboard

The leaderboard is rendered in a single pass. It queries each user for their amount of flashcards, using an outer join so that new users without any flashcards still appear on the leaderboard too. This then sorts the list in descending order by the level values, which is derived from the XP value in their account and calculated in the same way it was previously disccused. Ranks are given to users through enumerate and are passed to the template to show on the page. A fallback is also given for the user’s profile picture to set it to a default image in case the default set in the app isn’t working after an update. Figure 7.2.5 shows this leaderboard in its backend and frontend implementation.


Figure 7.2.5: Leaderboard page rendered in Firefox.

### 7.5.9 Database Implementation

While the database interacts with the system features has been clearly outlined, it is also worth clearly stating the structure in its own section. In the first release implementation, Dokkai uses a lightweight data.sqlite file in the app/data directory for the database. This is sufficient for a small application and userbase, but posed problems which will be explored in the next section. The database tables and scheme was created through models.py where SQLAlchemy manages the relationships between tables and any queries. Uploaded

files such as profile pictures are stored in the uploads directory. These are restricted by file type with only .png, .jpeg, .gif, and .jpg being allowed which is enforced by the allowed_file() line in views.py whenever a user uploads a new image.

## Chapter 8: Testing

To best illustrate the agile testing process that was conducted locally, the flashcards portion of the app will serve as the example as it is the most complex to test. Testing first focused on single functions and smaller parts of code as they were developed, for example for the flashcards page, the Flask shell was used in a separate testing file to test the values of new_steps and relearn_steps with the due minutes function. This was to verify that each value within new and relearning steps produced the expected offset from the current time (UTC) and that the boundary conditions were working as intended for moving cards out of learning and into relearn, and vice-versa, as well as resetting cards. An example of this testing workflow can be seen in figure 8.1.

```
NEW_STEPS = [1, 5, 30]
RELEARN_STEPS = [2, 5]
now = datetime.now(timezone.utc)
def mins_until(m):  2 usages
    return round(((now + timedelta(minutes=m)) - now).total_seconds() / 60)

print("new steps:", NEW_STEPS, "~", [mins_until(m) for m in NEW_STEPS])
print("relearn:", RELEARN_STEPS, "~", [mins_until(m) for m in RELEARN_STEPS])

last_idx = len(NEW_STEPS) - 1
print("Last learning index num:", last_idx)
print("Graduates:", last_idx == 2)
```

```
C:\ProgramData\miniconda3\envs\SW2\python.exe C:\dokkai\
new steps: [1, 5, 30] ~ [1, 5, 30]
relearn: [2, 5] ~ [2, 5]
Last learning index num: 2
Graduates: True
```

Figure 8.1: An example of a test file that was not committed to the repository.

This was then integrated back into the main views.py file, and tested locally through Flask's test client. For each POST to the flashcards route, such as rating a flashcard "again" or "good", it was checked that the correct input was mapped to the card, the step index moved logically and as expected, the repetition value incremented correctly, is_lapsed_learning updated if a card was lapsed, updated last_review, and has a due date from FSRS that logically matches the card and it's repetition history. This method of rigorous testing was repeated across features.

### 8.1.1 Challenges, Debugging, and Accidental Features

Early challenges faced when testing features became apparent through the development of the flashcards system and how it integrated with the rest of the app. On the Grammar page, difficulty was faced with how cards were displayed on the frontend due to them having a slightly different layout to normal cards, despite using the same model and template. Grammar cards sometimes rely on tags such as <br> to directly move the content on the front of the card to a new line to better explain concepts, due to grammar being more complex than simple vocabulary cards. This then presented an issue of the <br> appearing on the card directly to the learner during a review session, and by extension the TTS elements not being read properly for these cards.

The fix after testing through trial and error similar to what was described in section 8 was to store the raw HTML for cards tagged with true on the is_grammar flag, but then render them differently on the page to not show the <br> tags to the user. This is done by using Jinja2's |safe for the front field so that the tags aren't visually shown, with the text being read by TTS being taken from stripped text, so that the <br> tags aren't spoken in the user's browser. This is also why when a user browses their personal card database, the grammar cards show the <br> on this page. This may be useful to the more technically savvy users should they wish to edit the styling on their card beyond the just the content, providing them with more customisation options and control of their data. While this was unintended, when it was discovered, the further control it gives to users over how their cards are displayed was determined to be a positive in-line with Dokkai's philosophy, and therefore was kept as an additional feature. This difference in how these cards are displayed throughout the different pages is shown in Appendix 8.

## 8.2. User Testing

As well as the local software tests, formative user tests were also conducted with 10 participants using the most recent build of Dokkai, with the grammar assistant helper removed. This was removed so as to not skew questionnaire results when the feature was not considered fully complete yet. Participants all described themselves as not having been a serious language learner within the past 6 months when provided with the definition in the terminology section earlier in the report. Essentially, the results may represent an extremely motivated serious language learner with multiple hours to dedicate in and outside of the application, rather than one with a moderate amount of motivation and more limited time.

Each session participants were asked to follow the process of signing in, completing a single lesson, studying their flashcards due for the day, use the kana guesser for at least two minutes, and visit the leaderboard and profile dashboard pages at least once. No strict time usages were recorded, but all participants confirmed to follow this format over the five consecutive days that testing was carried out. All participants gave informed consent

through Google Forms and completed the questionnaire seen in through the format in Appendix 9. There were no risks to notify participants of. All participants were over the age of 18, and no personal data was collected, with the user accounts being wiped upon testing completion.

The questionnaire consisted of a 5-point Likert scale in order to assess the applications' ease of navigation, usefulness of flashcards, motivation through the gamification elements, and overall satisfaction with the product. They were also asked to provide any desires for missing features, and any other additional comments at the end of the form in order to inform future development considerations. Appendix 12 displays visualisations of all of the following questions in the next section through bar charts by using the raw data gathered from the form.

### 8.2.1 Ease of navigation

For the first question, the 10 participants rated ease of navigation at an average of 4.6 (out of 5). Throughout the task workflow that was instructed to participants, everyone reached the correct pages without assistance indicating that the navigation bar was correctly placed, clear, and properly labeled. There was a comment from one participant about what the "again/good" buttons meant for the flashcard and how it's scheduled to the next day, indicating that either the "how to use" page could perhaps be rewritten to be more clear to users new to flashcards, or the buttons could have an on hover hint for new accounts.

### 8.2.2 Usefulness of Flashcards

A specific question was created just for the flashcards portion due to it being the core of the application. Due to time constraints of the project, user satisfaction with their long-term retention of vocabulary could not be tested, so the usefulness factor here can only be judged based on short-term retention and the general ease of use of the flashcard page.

Participants rated this question at an average of 4.8, indicating that the implementation of this feature was successful. Suggestions in the text boxes indicated that participants wished for features such as "undo last answer" and more in dept statistics in the account section or after a review session. It should be noted that participants may not have been as harsh as external users would be as they are aware this is an MSc project developed entirely in roughly three months. Though this is the case, this still does not take away that the proof of concept for the app as a first version is successful in its main feature shown by the score here in the questionnaire.

### 8.2.3 Motivation from Gamification

Participants rated their motivation from the gamification features at an average of 3.7. This was the most distributed response of all the questions asked. Six participants agreed,

three were neutral, and one disagreed. This suggests that the gamification elements helped for some users, but for this group of participants their motivation was primarily self-driven. Comments related to this question indicated that the “leveling system was logical”, but the leaderboard would “only feel motivating when it was specific to the user’s friend list” rather than having a list of every Dokkai learner on a single page.

### 8.2.4 Overall Satisfaction and Additional Comments

Participants rated overall satisfaction to be an average of 4.7, with nine participants selecting either agree or strongly agree. One participant was neutral and there were no negative responses. Comments were positive about the flashcard workflow, with it also being informally mentioned during discussion afterwards by one user that the application features are integrated well with each other, such as the “add to deck” button on the grammar page. There were no comments made on the kana guesser page by any participants. This was not judged to be a negative.

### 8.2.5 Summary and User Requests

To summarise, user testing suggests that Dokkai is well aligned with its goals for the intended userbase, and where improvements could be made, the roadmap is well aligned. No critical errors, crashes, or issues were encountered, though it would be useful to conduct further testing with a random sampling method and with more participants and a larger database model. As for the user requests, while participants consistently praised the flashcard workflow for its clarity and short-term effectiveness on memory, two comments specifically mentioned the lack of an “undo last answer button” being a negative on the flashcard workflow experience, so this has been added to the roadmap for the future.

As touched upon in 8.2.3, there was also a request for gamification features specific to user friend lists rather than a single leaderboard for every user on the platform. As Dokkai was launched to this testing phase with no prior advertising or announcements, no larger userbase outside of the participant pool was gathered, meaning that the leaderboard effectively functioned this way because most participants were acquainted with each other already. This was short sighted as it had not been considered how the leaderboard would look with a larger, more disconnected userbase, so this has also been noted for future development as a necessity. Multiple test users also informally requested a native mobile app during discussions after the completion of testing and the form, and though this is a large undertaking, it is currently the number one major feature on the roadmap, as much of the language learning literature discussed in section 2 also outlined how important this was for users that enjoy studying portably.

Overall, what’s indicated here by the user testing phase is that the core of Dokkai and its philosophy works effectively and is motivating to users at least in the short-term, but

requires polishing and additional functionalities, along with further long-term testing, to be a marketable product on the same playing field as those identified in the literature review.

### 8.3 Stress Testing

As a consequence of the user testing, concurrent usage of Dokkai was also tested. Five learners were the most recorded users being logged in at one time on the first day of testing, with them completing their instructed workflow through the features of the application. There were no observations or reports of any downtime during this period and all features and database updates functioned as intended. Though this is the case, feedback after the project demonstration by the external supervisor advised more robust testing of the system for clarity and security.

In response, further tests were conducted against the system through scripted parallel requests targeted at the main feature pages and routes for sixty seconds. These were requests to the flashcard, dashboard, and kana routes in views.py through GETs, as well as POSTs that added dummy flashcards using the normal template from models.py. This has been organised into an image in Appendix 13, along with the terminal output results for the first values of each time the test was carried out with the same build of Dokkai used in the user testing phase. The first test used the value of 10 users to simulate the user testing participant number had they all been using the system simultanously, increasing by 10 each test until errors and high latencies were reported in the terminal output.

Additionally, during the test that used 40 users as the input value, there were occasionally errors relating to the database being locked from requests of adding flashcards. This was likely due to using SQLite as it has a single-writer limit and submitting too many flashcards at once disrupted this. This could be simply fixed through a migration to PostgreSQL, but due to time constraints and being after the project demonstration period had already ended this was not carried out and was instead placed on the future development roadmap as an urgent development necessity.

## Chapter 9: Evaluation Against Functional and Non-functional Requirements

Using the functional requirements detailed in section 3.1.1, the majority of the “must haves” were all implemented to a satisfactory level for users and functioned as intended during the testing period, with this shown throughout the section 8 testing. One feature of the guest requirements that remains unfulfilled is the password reset feature through the users email address. While this does not directly affect the functionality of the system, it is a glaring issue that must be fixed before wider deployment to ensure that user data is safe and can be accessed when a password is forgotten or mistyped. Dokkai staff are able to manage the database and can assist users, but this should not be marked as a permanent solution meaning this requirement was not fulfilled as intended. As well as this, a demo mode was not created for this build, however, this feature as a “could have” may be

reconsidered as a result of the user testing, in favour of a more comprehensive “how to use” page instead.

The features outlined for learners aligns with the features in section 3.1.1, with some of the “could haves” being implemented too. Learners land on the dashboard after logging in, they can create, review, and export their cards to csv, giving them full control of their data. All navigation bar links are logical as shown through the user testing, and account settings are all editable besides the username. “Should have” features that were implemented include the pre-made importing of custom decks (currently only made by Dokkai staff and administrators), and the kana guesser with streak functionality. The main “could have” that was implemented was the grammar assistant, and is effective as a proof of concept through local testing.

For administrators, models.py includes a basic admin role for the account that is set manually on the server through SQLite. It is still basic in its implementation besides the ability to remove content, though this still satisfies the requirement. Assessing the pie charts in figure 9 shows a good balance, though upon reflection it may have been more efficient for the deployment to first focus on ensuring the “must haves” before the “could haves” like the AI assistant for the grammar page, informing how future development should proceed.

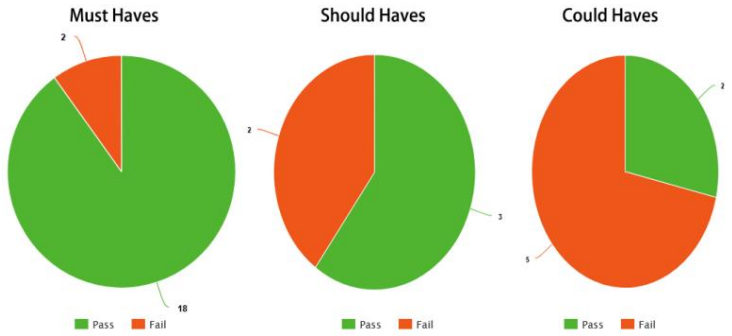



Figure 9: Functional requirement pie charts, separated by must, should, and could haves.

For the non functional requirements, 13 passed and 2 failed. The main feature pages and actions were functional within the target of 1-2 seconds being evident through the testing phase, with a manageable latency with 30 users active. The pie chart for the non-functional requirements can be seen in Appendix 14. Usability passed as can be seen in the section 8.2 results for having a clear and easy to navigate system. Compatibility passed on current versions of Chrome and Firefox. Reliability and data stability was functional during normal use, though the testing problems experienced when using 40 users having an identified fix that is yet to be implemented. This requirement will be marked as passed for the sake of the report, but it should be kept in mind that this is only under a small userbase and would fail with a larger one. Security passed with hashed passwords. Maintainability was passed through the clear distinction between the files, all

being clearly named and commented for the final GitLab push. Portability failed as the layout is usable on mobile, but does not display the flashcard page in an easy-to-use manner like it does on desktop browsers, and could do with tweaking in later versions.

### 9.1. Accessibility Evaluation

Section 4.3.4 detailed accessibility concerns for Dokkai's design with them being evaluated against the implementation here. Text is readable and interacts correctly in both desktop and mobile browsers, though the flashcard workflow could be improved on mobile as the page refresh on card rating occasionally makes usage difficult over long sessions.

The proposed dark mode was implemented successfully through the toggle which changes the CSS through a switch in the model for the User. Interactive elements follow a consistent structure and colour scheme, and assistive features such as text to speech all functioned as intended across different flashcard types. The flashcard page is fully operational through keyboard bindings, with the only shortcoming being that these keys are currently fixed, and for better accessibility a setting could be added for users to change these keys as they wish. The usage instructions were on the "how to use" page and were clear to follow, with a potential improvement being to implement instructions on the flashcard page itself for first time users. During testing, a participant who was partially impaired visually confirmed that the keyboard bindings and screen reader allowed them to easily navigate the site without assistance. Boostrap and the custom CSS being reused consistently across all pages aligned with WCAG 2.1's recommendations for navigation and compatibility.

As there were no chat features implemented inclusivity concerns outlined earlier are not present, and the profile visibility setting allows users to hide their activity from others. An improvement here could be to hide the full profile from the leaderboard, as the setting does not completely remove the entire profile from view.

### 9.2 Critical Evaluation Against Goals and Existing Systems

Through the user testing and findings, it can be clearly seen that Dokkai functions well in its first release as a language learning platform with the unique theoretical underpinning of optimised algorithms to help users achieve fluency, positioning itself against other language learning platforms defined in section 2.3 well. With FSRS already being identified as potentially the most effective algorithm available and saving users time through a reduced 17% cost in predicting recall rates (Su et al., 2023), due to its successful implementation within the app, it can be considered a success on this front. Along with the implementation of gamification elements, though perhaps slightly primitive in areas, the core ecosystem functions extremely effectively for the identified userbase. Gathering public interest in Dokkai's philosophy toward language learning is easier said than done, but even in the worst case scenario where wider users cannot be attracted to the platform

and end up preferring the kind of “feel good language learning” applications, a niche platform has been established where hardcore learners can gather for all of their language learning needs.

The app successfully met the majority of requirements outlined for the system, with all of the outlined core features functioning as intended during user and stress tests. The features that were not implemented due to time constraints or accidental negligence, such as the password reset through email, while they should be implemented immediately in the next version, they do not make the app unusable or negatively impact study sessions. Due to this, it is apt to say that Dokkai has delivered on its core mission of being a language app for serious learners and has laid the foundation for differentiating itself in an over-saturated market.

Specifically, gaps have been directly addressed regarding current platforms using outdated flashcard algorithms (Su et al., 2023), and has placed an emphasis on input based learning (Krashen, 1982; 2008) by avoiding features that encourage early output to encourage comprehension of content first (Tommerdahl et al., 2022; Levina et al., 2024). Gamification elements remain limited in the current build, though the leaderboard and XP systems provide a good basis for further development in not creating a distracting environment (Shortt et al., 2021), prioritising the user making real language progress and forming positive habits rather than short term motivation spikes (Muckenhumer et al., 2023). Secondary tools such as the kana guesser and AI grammar sentence assistant add further depth to the app, with further development looking like it will bring Dokkai closer and closer to the “all-in-one” feel (Karasimos, 2022) that ultimately remains the ideal for the application to strive towards.

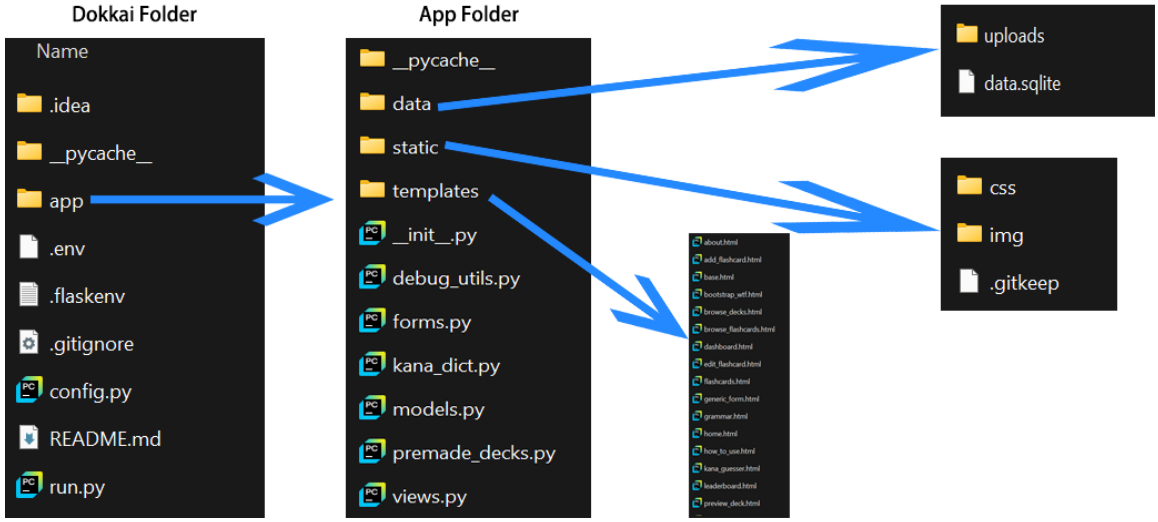
### Chapter 10: Conclusion

To conclude, Dokkai’s initial build validates the vision for the project in that FSRS combined with an input-first based learning philosophy works well with the identified type of learner. The balance of efficiency, gamification, and useful tools establishes a very strong foundation for Dokkai to be a worthwhile addition to the plethora of language learning applications, despite the market being very saturated already. By addressing the outlined shortcomings and continuing the development roadmap toward the vision of the “all-in-one” platform for serious learners, Dokkai can achieve its own aim, and, by extension, help the userbase achieve their language learning goals too.

## A1.2 – Structure of Files

The apps structure was separated logically for ease of understanding and future development. The app folder contains all of the templates for the HTML pages, the static folder holds CSS and images, and the data folder holds the SQLite database and any uploaded files. Views.py is the main file for the routes and logical operations of the

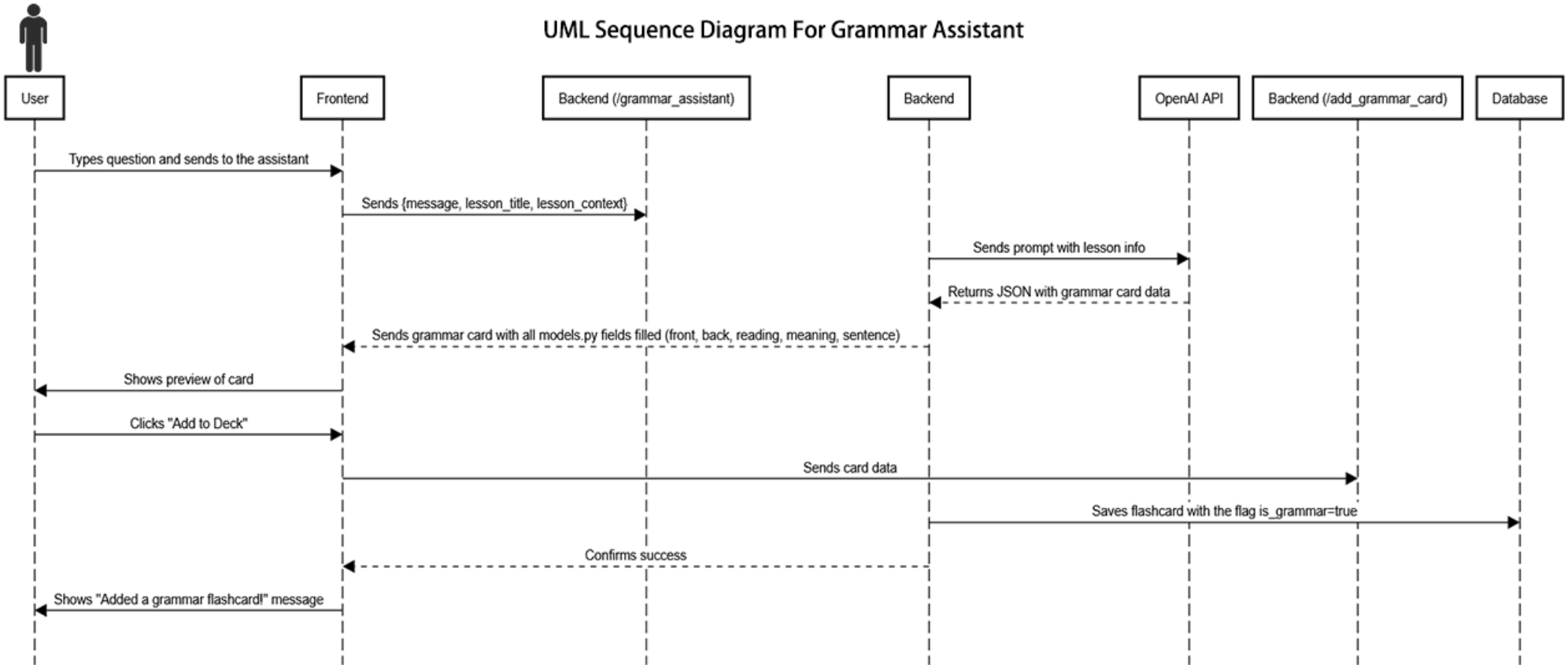
backend, models.py holds the database models, and forms.py has templates for forms and validation.



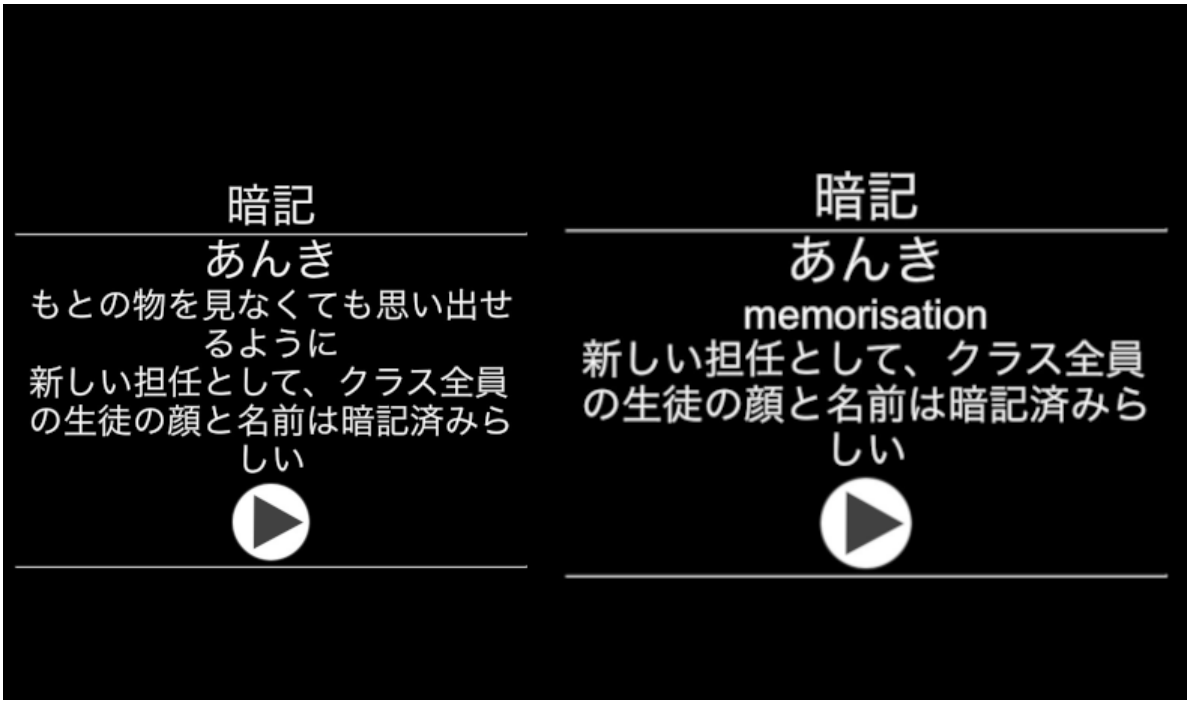

### A1.3 – Application Instructions (Local)

Assuming the user has the project open in PyCharm with Python already installed, configure the run settings by entering flask into the python module option with run as the parameter. After applying these settings, run the app with them as normal, and then click on the address in the terminal to open the app locally in the browser.

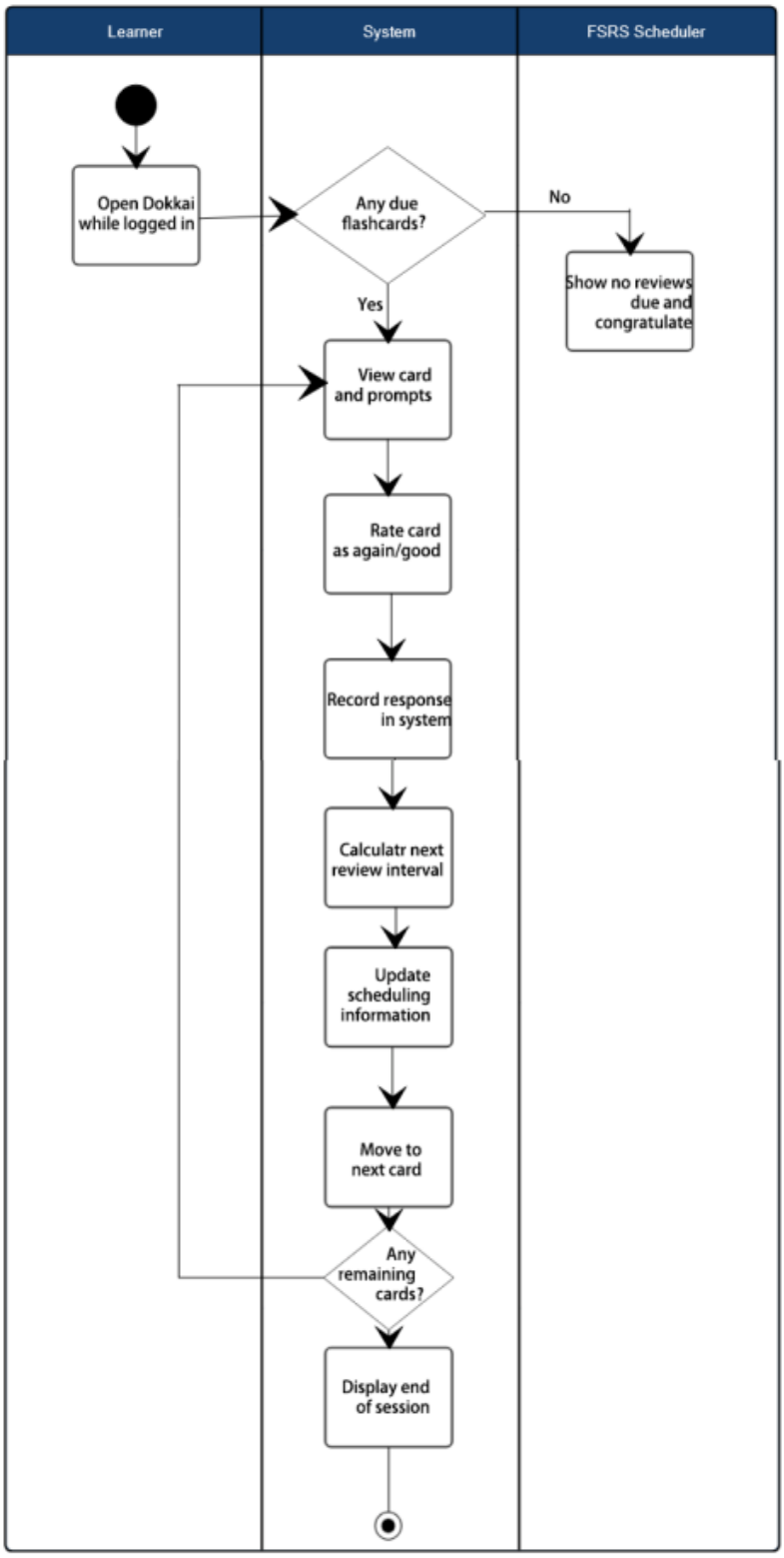
## Appendix 3 – AI Example Sentence Generator Sequence Diagram



## Appendix 4 – Original Card Template



## Appendix 5 – Activity Diagram For Learners



## Appendix 6 – Models.py Flashcard Model Example

```python
class Flashcard(db.Model):  76 usages
    __tablename__ = 'flashcards'

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    front = db.Column(db.String(255), nullable=False)
    back = db.Column(db.String(255), nullable=False)
    reading = db.Column(db.String(255))
    meaning = db.Column(db.String(255))
    sentence = db.Column(db.String(500))

    #spaced repetition metadata
    repetition = db.Column(db.Integer, default=0)
    interval = db.Column(db.Integer, default=1)
    ease_factor = db.Column(db.Float, default=2.5)
    stability = db.Column(db.Float, default=4.0)
    difficulty = db.Column(db.Float, default=3.0)
    lapses = db.Column(db.Integer, default=0)
```

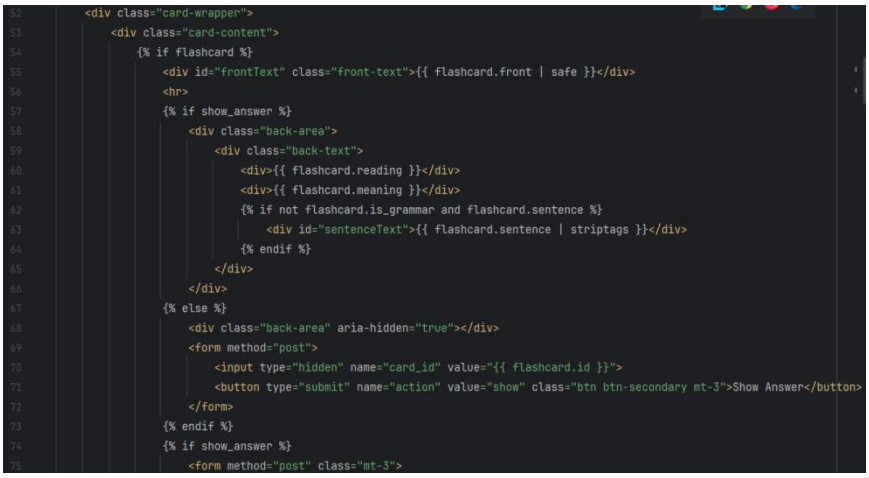
## Appendix 7 – Code Snippets

### A7.1 – Route and Template Interaction

```python
@app.route('/flashcards', methods=['GET', 'POST'])
@login_required
def flashcards():
    now = datetime.now(timezone.utc)
    if request.method == 'POST':
        card_id = request.form.get('card_id', type=int)
        action = request.form.get('action')
        rating_str = request.form.get('rating')
```

```python
        if action == 'show':
            return render_template(
                template_name_or_list: 'flashcards.html',
                flashcard=flashcard,
                total_due=total_due,
                new_due=new_due_today,
                review_due=review_due,
                learning_due=learning_total,
                combined_due=combined_due,
                new_available=new_available,
                show_answer=True
            )
```

## A7.2 – Flashcard HTML



```
<div class="card-wrapper">
    <div class="card-content">
        {% if flashcard %}
            <div id="frontText" class="front-text">{{ flashcard.front | safe }}</div>
            <hr>
            {% if show_answer %}
                <div class="back-area">
                    <div class="back-text">
                        <div>{{ flashcard.reading }}</div>
                        <div>{{ flashcard.meaning }}</div>
                        {% if not flashcard.is_grammar and flashcard.sentence %}
                            <div id="sentenceText">{{ flashcard.sentence | striptags }}</div>
                        {% endif %}
                    </div>
                </div>
            {% else %}
                <div class="back-area" aria-hidden="true"></div>
                <form method="post">
                    <input type="hidden" name="card_id" value="{{ flashcard.id }}">
                    <button type="submit" name="action" value="show" class="btn btn-secondary mt-3">Show Answer</button>
                </form>
            {% endif %}
            {% if show_answer %}
                <form method="post" class="mt-3">
```

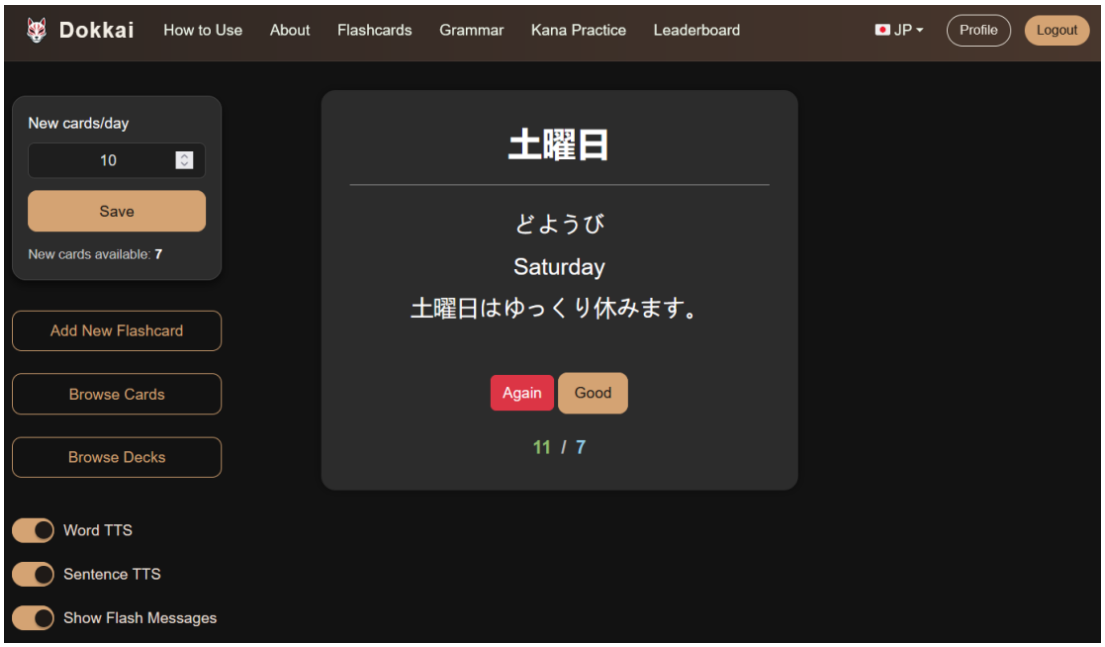
## A7.3 – TTS Integration

```
// ---------- tts (word/sentence) ----------
function speakText(text) {
    if (!window.speechSynthesis || !text) return;
    const u = new SpeechSynthesisUtterance(text);
    u.lang = "ja-JP";
    u.rate = 0.9;

    const setVoice = () => {
        const voices = window.speechSynthesis.getVoices();
        const ja = voices.find(v => v.lang && v.lang.startsWith("ja"));
        if (ja) u.voice = ja;
        window.speechSynthesis.speak(u);
    };

    if (speechSynthesis.getVoices().length) {
        setVoice();
    } else {
        speechSynthesis.onvoiceschanged = setVoice;
    }
}
```

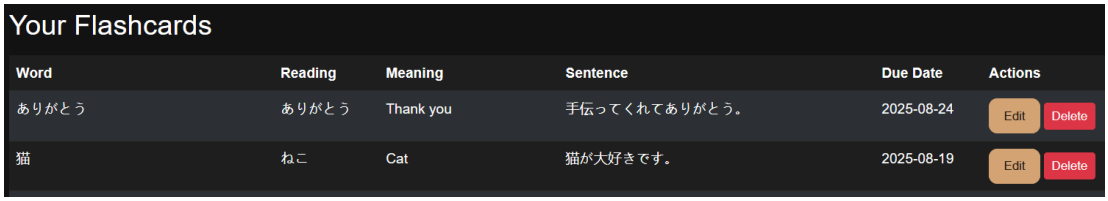
## A7.4 – Step Values

```
            NEW_STEPS = [1, 5, 30]
            RELEARN_STEPS = [2, 5]
            def due_in_minutes(m): return now + timedelta(minutes=m)

            step = flashcard.learning_step_index
```

## A7.5 – Flashcard Review Page Rendered in Firefox




## A7.6 – Pre-Made Deck and Card Browsing


## A7.7 – Grammar Assistant Implementation

```
@app.route('/grammar_assistant', methods=['POST'])
@login_required
def grammar_assistant():
    data = request.get_json(silent=True) or {}
    user_msg = (data.get('message') or '').strip()
    lesson_title = (data.get('lesson_title') or '').strip()
    lesson_context = (data.get('lesson_context') or '').strip()
```

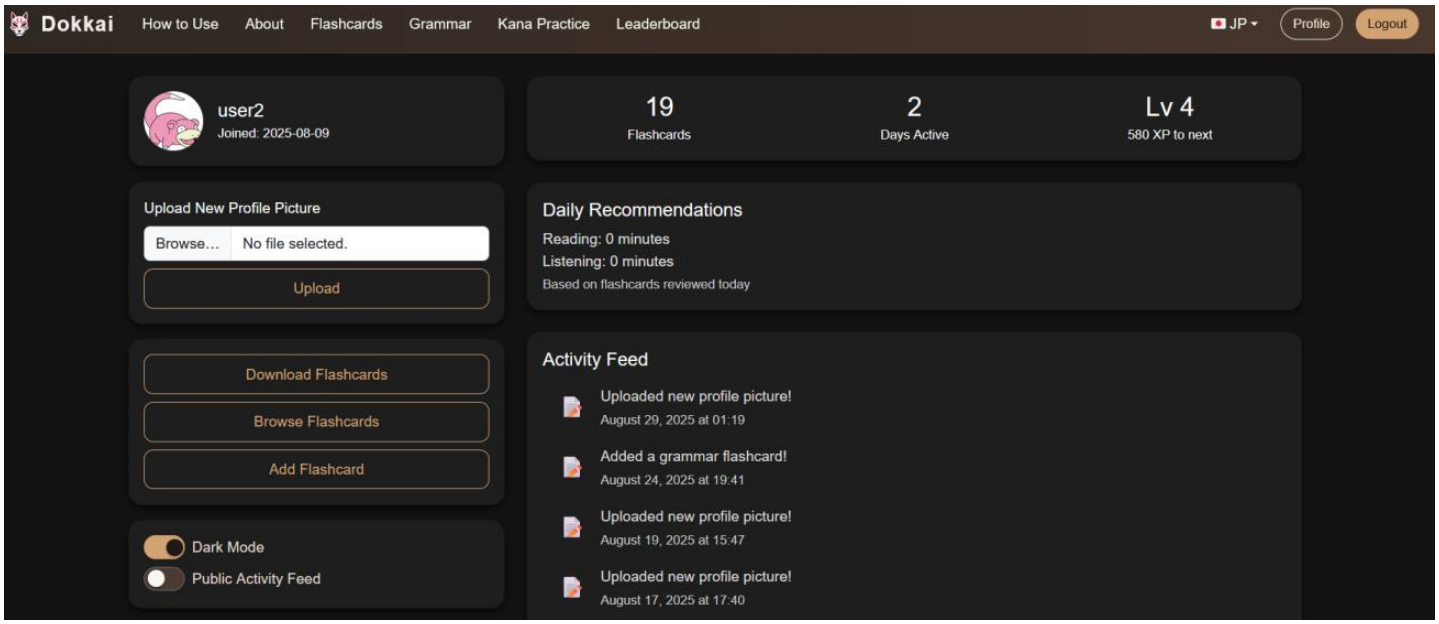
```
    system = (
        "You are a Japanese grammar assistant inside a language learning platform. "
        "Given a current lesson title, short lesson text, and a user request, "
        "return exactly one concise example in strict JSON (no prose) with the schema of:\n"
        "{"
        '  \"focus_token\": \"string (e.g., だ, です, へ, に, ます, でした)\",'
        '  \"jp_sentence\": \"string (one natural sentence, ending with punctuation)\",'
        '  \"en_sentence\": \"string (short one-line english sentence)\",'
        '  \"meaning_label\": \"string (e.g. Copula (polite)/Particle (direction) /Verb (irregular polite))\"'
        '  \"short_explanation\": \"string (a short line explaining the grammar usage)\"'
        "}\n"
        "Rules: Use vocabulary/grammar consistent with the lesson."
        "Prefer forms highlighted in the lesson (e.g., でした for polite past)."
    )

    user = {
        "lesson_title": lesson_title,
```

```
        try:
            j = json.loads(raw)
            focus = (j.get("focus_token") or "").strip()
            jp = (j.get("jp_sentence") or "").strip()
            en = (j.get("en_sentence") or "").strip()
            meaning_label = (j.get("meaning_label") or "").strip()
            short_expl = (j.get("short_explanation") or "").strip()
```

## A7.9 – Dashboard Implementation

```
    #total flashcards created by the user
    flashcard_count = Flashcard.query.filter_by(user_id=user.id).count()
    #track unique days user completed all due flashcards
    unique_days = db.session.query(func.date(Activity.timestamp)).filter(
        Activity.user_id == user.id,
        Activity.message.like("Completed all due flashcards%")
    ).distinct().count()
    #get 5 most recent activities for the activity feed
    recent_activities = Activity.query.filter_by(user_id=user.id).order_by(
        Activity.timestamp.desc()).limit(5).all()
    #calculate xp level (level grows with square root of xp/100)
    xp = user.xp
    level = int((xp / 100) ** 0.5)
    xp_for_next = ((level + 1) ** 2) * 100
    xp_to_next = xp_for_next - xp

    #how many cards reviewed today
    today = date.today()
    reviewed_today = Flashcard.query.filter(
        Flashcard.user_id == user.id,
        Flashcard.last_review != None,
        func.date(Flashcard.last_review) == today
    ).count()
```

### A7.10 – Dashboard

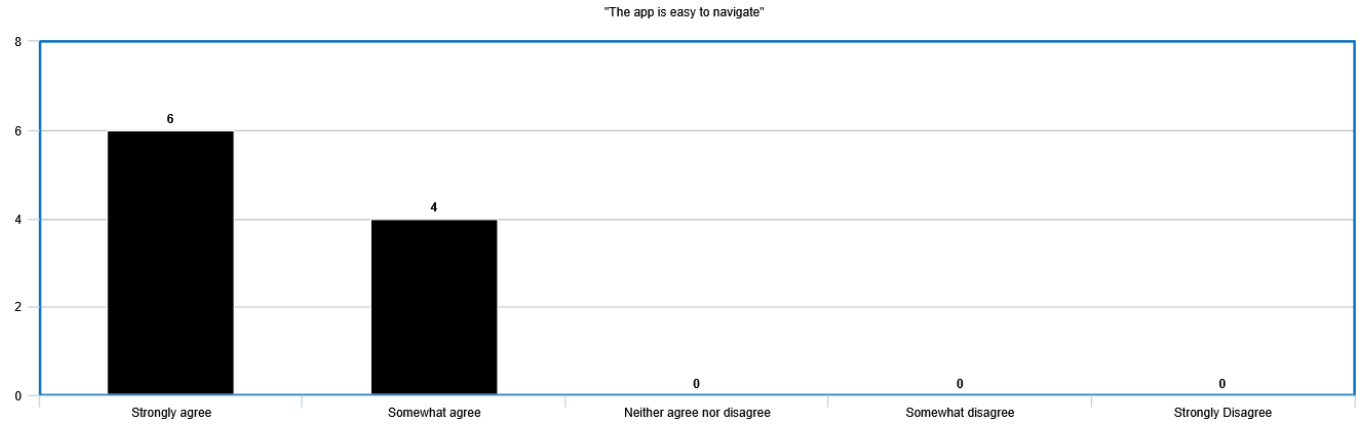


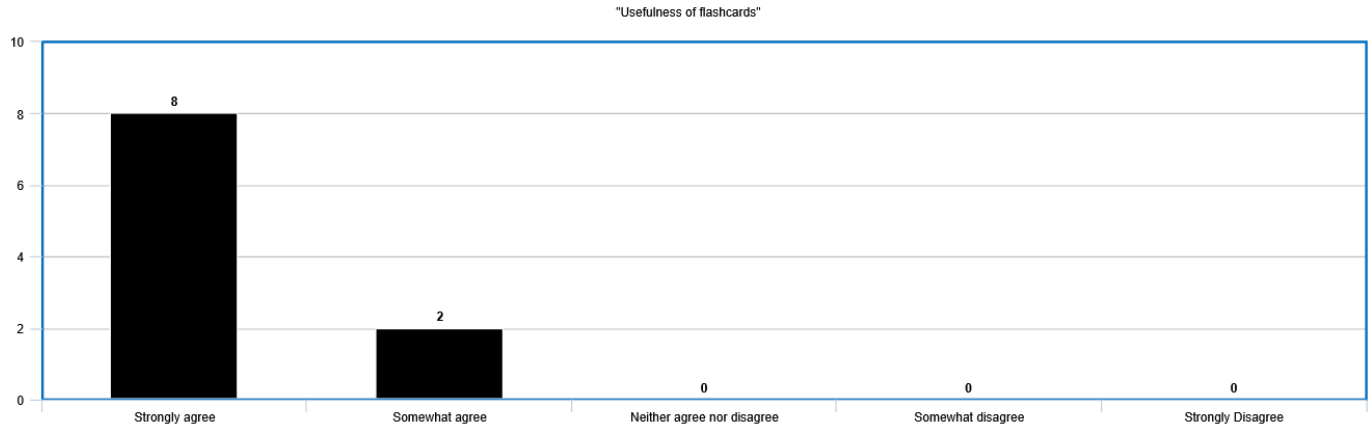

## Appendix 8 – Display Differences Between Flashcard Page and Browsing Cards Page

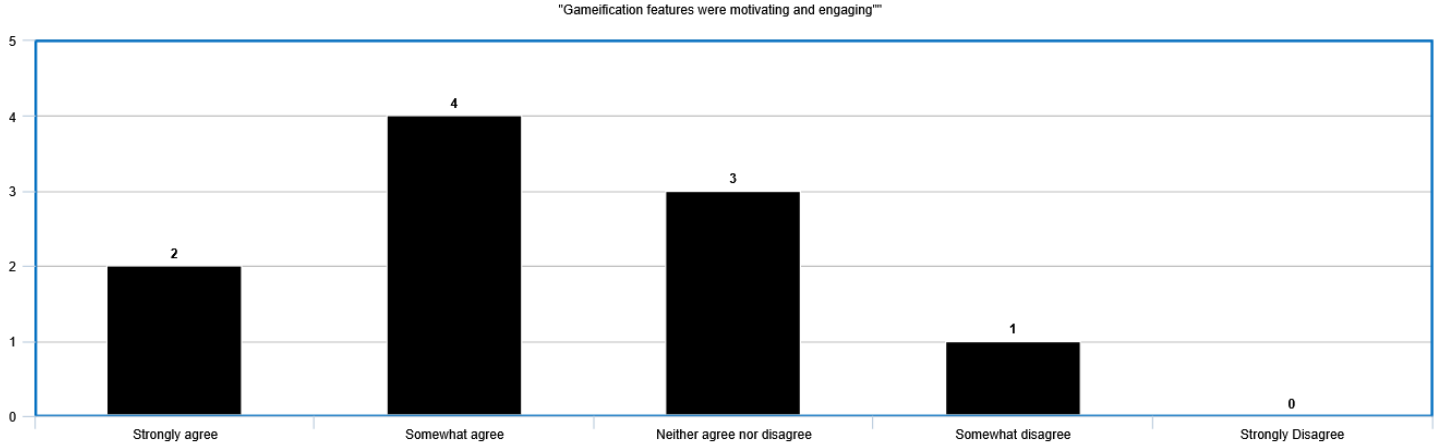
flashcards.html:

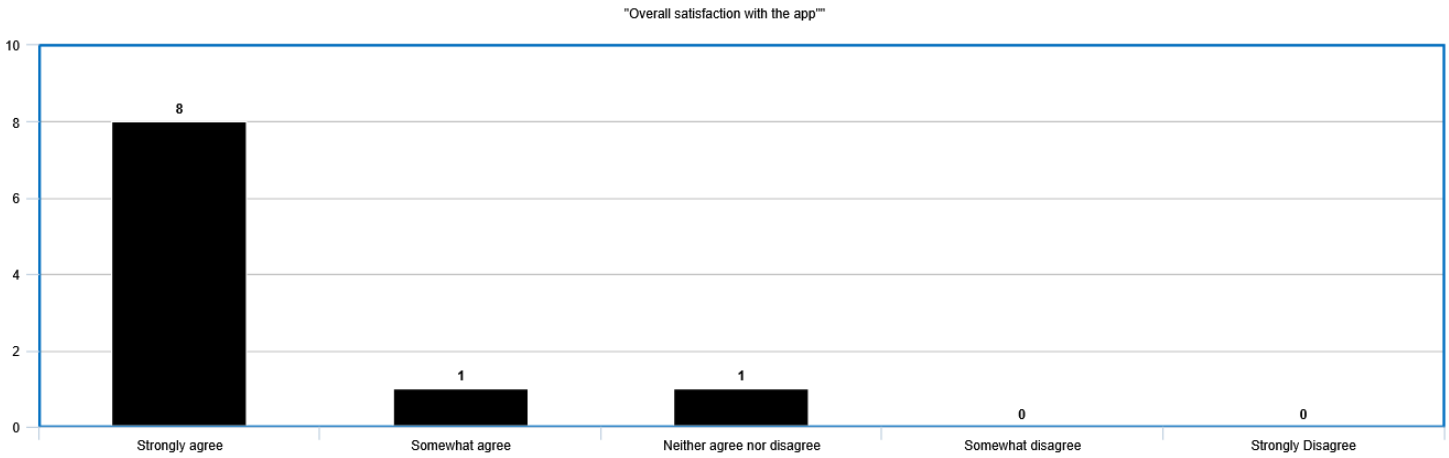

browse.html:

## Appendix 12 – Bar Charts Created From Raw Data Responses (In Order as They Appeared In the Text)






## Appendix 13 – Mini_stress.py Stress Test File Results

```python
def worker():  1 usage
    global errors
    s = requests.Session()
    if auth_cookie: s.cookies.update(auth_cookie)
    while time.time() < end_time:
        method, path, data = random.choice(routes)
        t0 = time.time()
        ok = True
        try:
            r = s.request(method, base + path, data=data, timeout=5)
            ok = 200 <= r.status_code < 400
        except Exception:
            ok = False
        dt = time.time() - t0
        with lock:
            latencies.append(dt)
            if not ok: errors += 1

threads = [threading.Thread(target=worker) for _ in range(users)]
for t in threads: t.start()
for t in threads: t.join()
total = len(latencies)
avg_ms = (sum(latencies)/total*1000) if total else 0
print(f"requests={total}  errors={errors}  avg_ms={avg_ms}")
```

| Users Value | Results |
|---|---|
| 10 | `requests=5475  errors=0  avg_ms=110` |
| 20 | `requests=6429  errors=0  avg_ms=187` |
| 30 | `requests=6557  errors=0  avg_ms=275` |
| 40 | `requests=5288  errors=737  avg_ms=510` |

## Appendix 14 – Non-Functional Requirements Pie Chart