SENG2021 Software Architecture and Design Report

By Negin Ghanavi (z5259418)

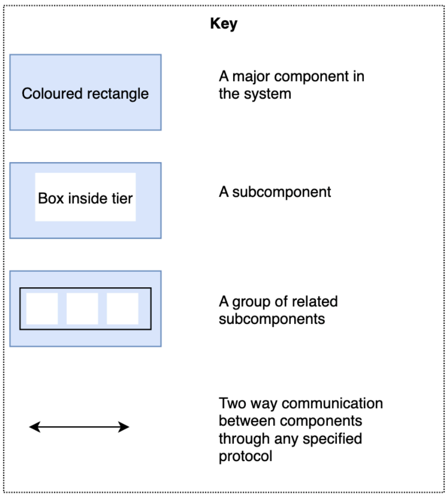
Software Architecture

Introduction

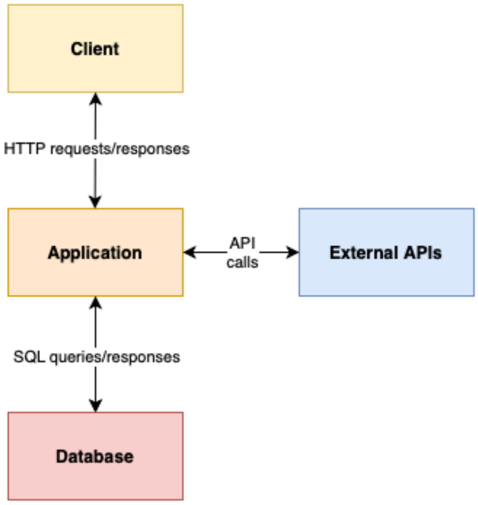
VocabCards is a web application which provides users with an easy-to-use interface for creating and revising flashcards. The application addresses the unique needs of individuals who use flashcards to learn new words by allowing for the automatic insertion of definitions, audio pronunciations and translations in flashcards using external APIs.

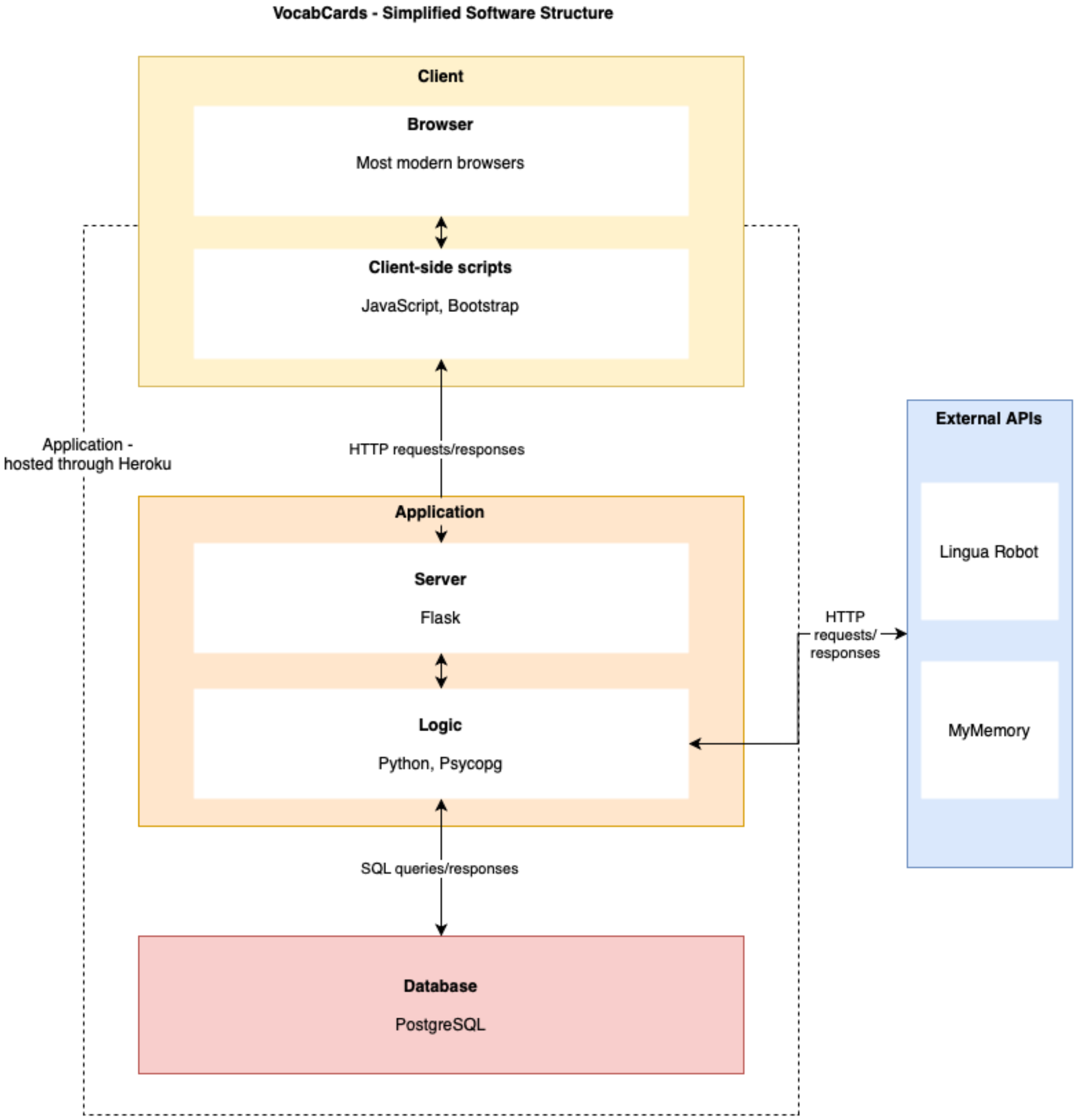
Diagrams

A tiered software architecture model was chosen, separating the application into three independent layers which can communicate with one another using specified API protocols. In addition to the three main tiers of the application, the system makes use of external APIs which form an additional major component in the architecture. A diagrammatic view of this architecture is shown below. All data exchanged between the various layers is sent in JSON format.

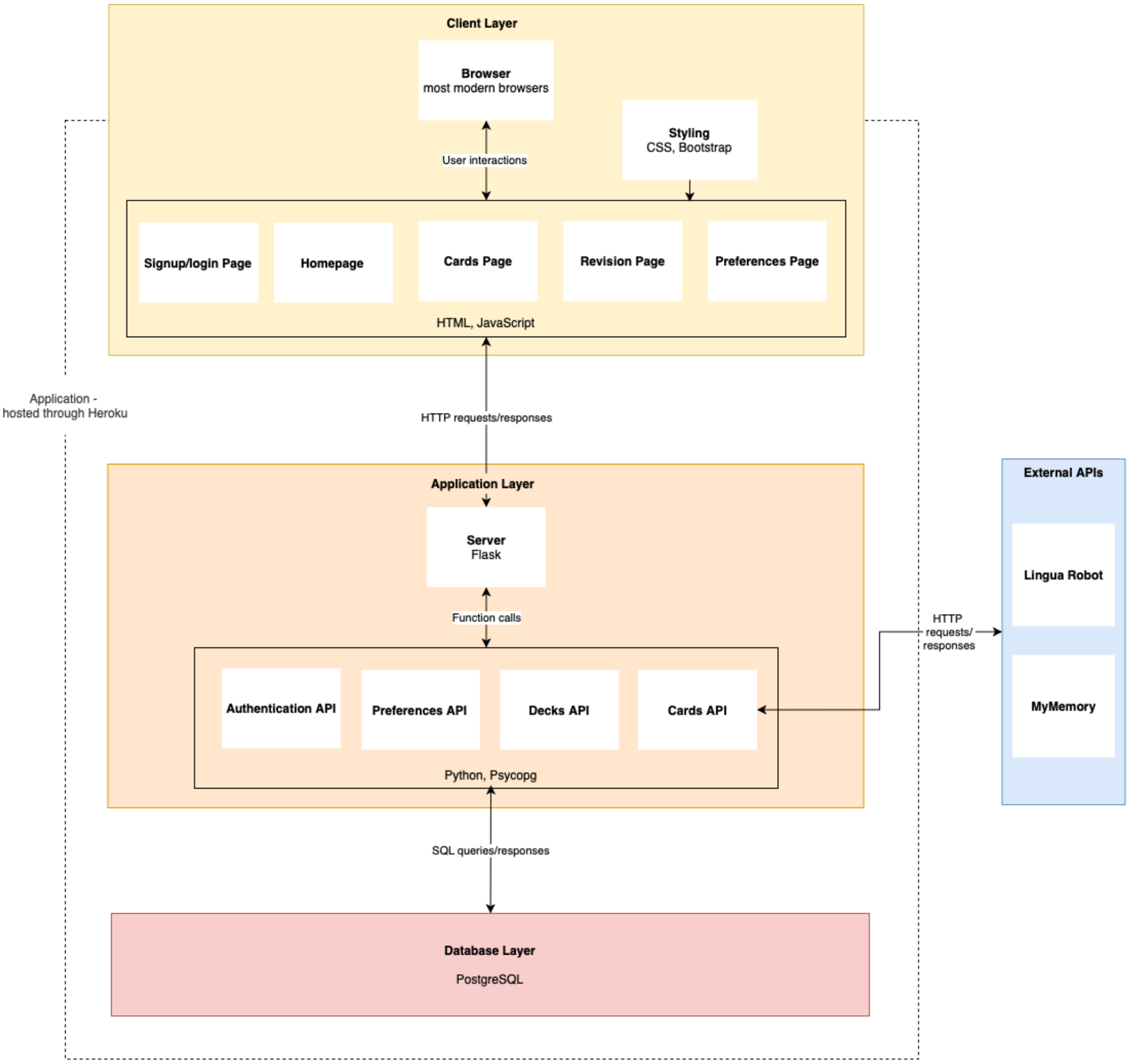

Key
Coloured rectangle
A major component in the system
Box inside tier
A subcomponent
A group of related subcomponents
Two way communication between components through any specified protocol

VocabCards - Overall Software Structure




VocabCards - Simplified Software Structure
Client
Browser
Most modern browsers
Client-side scripts
JavaScript, Bootstrap
HTTP requests/responses
Application -
hosted through Heroku
Application
Server
Flask
Logic
Python, Psycopg
HTTP
requests/
responses
External APIs
Lingua Robot
MyMemory
SQL queries/responses
Database
PostgreSQL

VocabCards - Software Architecture



Selected Technologies

Client Layer

The client layer is composed of the user's browser, a styling component and five client-side scripts for the five pages in the application. The styling component consists of a local stylesheet and a linked Bootstrap stylesheet. This component defines the visual design of DOM elements. Each of the five client-side scripts consist of an HTML page and a corresponding JS script. The scripts are responsible for rendering objects and responding to events. The user's browser provides the view.

The front-end technologies selected for this project make it possible for the application to be run on most modern web browsers. Specifically, user's browsers would need to support HTML5, CSS3, ES6 and Bootstrap 4. The following is a list of major browsers which would support the application.

Web browser	Version x and above (if applicable)
Mozilla Firefox	21
Chrome	23
Internet Explorer	10
Microsoft Edge	10
Opera	15
Safari	6

Bootstrap is a well-known HTML, CSS, and JS framework that greatly simplifies the process of building aesthetic and responsive websites. I selected this framework as it is easy to use and effective in creating responsive pages through the inclusion of row and column classes.

I decided to develop most of the client-side functionality using plain JavaScript as the project's timeframe is limited and I have no experience with any JS frameworks/libraries. Additionally, as a beginner I would benefit from strengthening my understanding of JS fundamentals prior to moving on to more advanced topics and higher levels of abstraction that are provided by a library such as React. Plain JavaScript is also sufficient to achieve the functionality of the application as specified in the requirements, which makes the use of frameworks optional.

Refactoring the code to make use of an appropriate JS library/framework in future iterations would certainly better the quality of the design and improve scalability and maintainability, but plain JS is sufficient for the scope and timeframe of this project.

Communication with the application layer will take place using AJAX. AJAX (Asynchronous JavaScript and XML) refers to the use of the XMLHttpRequest object to communicate with a web server. This allows for the exchange of data in various formats, including JSON which is the data interchange format selected for all inter-layer communications in this application. AJAX allows for asynchronous communication which is integral to the development of dynamic webpages and web applications.

Application Layer

The application layer is composed of a Flask server and four Python modules. The server acts as a wrapper around the functions defined in the modules. It defines routes which call relevant functions with input from the client as parameters and return the output of the function calls to the client. I chose Flask as it a simple framework which I have experience with and has sufficient functionality for the purposes of this application.

The logic section is composed of four Python modules that each encompass the functions required for one key aspect of the application. These modules do not communicate with one another. The logic modules make use of Psycopg2, which I chose due to some experience using it and its popularity as a PostgreSQL database adapter for Python. Additionally, the authentication module uses the PyJWT library to generate user tokens for each session. These tokens are then used to authenticate user requests that involve accessing or modifying user data (i.e. all requests except those involving external APIs) in a secure fashion.

Database Layer

The database layer is responsible for storing all user data and responding to queries from the application layer. I chose to use a relational database and selected PostgreSQL as my DBMS. Data that needs to be stored is easily organised in a simple relational database and PostgreSQL is a widely adopted relational DBMS. Although other popular DBMSs such as MySQL would also be effective, I chose PostgreSQL as I have some prior experience with it. Given that PostgreSQL closely follows SQL standards, it can be easily utilised by developers familiar with the language.

External APIs and Data

The application makes use of the two external APIs – Lingua Robot and MyMemory. Lingua Robot is an English dictionary API which, given a word or a phrase, returns its definition, audio pronunciation link and other useful data such as synonyms. It includes data for over 800 000 entries and has a latency of 281ms which is low compared to other definition APIs. Lingua Robot is also one of the few APIs I could find to provide audio pronunciations as well as definition data. The API offers a free plan with a quota of 2500 calls per day, which is sufficient for the purposes of prototype development.

MyMemory is a translation API containing billions of human translated words. The API returns a machine translation when a human translation is not available. MyMemory also supports most languages and has a latency of 421ms, making it a viable choice for this application. This API also offers a free plan with a 10 000 calls per month quota.

In addition to these APIs, I will make use of the English Vocabulary List to add a CommonWords table to the database. This table will be used to generate vocabulary decks for English learners.

Lingua Robot: https://rapidapi.com/rokish/api/lingua-robot

MyMemory: https://rapidapi.com/translated/api/mymemory-translation-memory

Hosting

The application will be deployed through Heroku using a Heroku-20 stack which is based on Ubuntu 20.04 LTS. I chose Heroku due to its ease of use and support of all required technologies for the project.

References

https://www.w3schools.com/js/js_versions.asp

https://getbootstrap.com/docs/4.0/getting-started/browsers-devices/

https://caniuse.com

https://www.hostinger.com/tutorials/what-is-ajax

https://devcenter.heroku.com/changelog-items/2005

Software Design

Requirement Updates

A requirement was added to address the management of user accounts which was overlooked in the first deliverable.

Requirement: Register as a user and login to account

Feature 1: Register an account

As a new user,
So that I can save my cards and preferences,
I want to be able to register an account.

GIVEN I am on the registration page
WHEN I input a valid username
AND enter a password
AND click on the “register” button
THEN a new user account should be created.

Feature 2: Login to account

As an existing user,
So that I can view my data and save new cards and preferences,
I want to be able to log in to access my account at the beginning of each session.

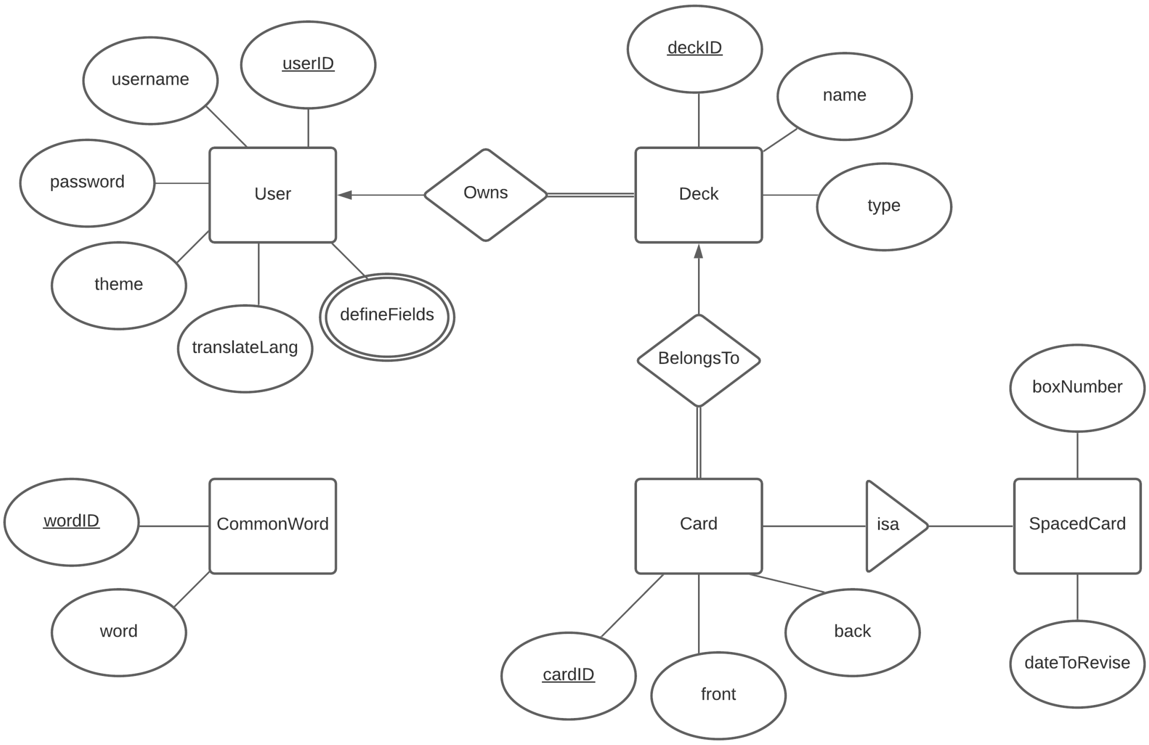
GIVEN I am on the login page
WHEN I enter my username
AND enter my password
AND click on the “login” button
THEN I should gain access to my account.

Other minor changes in the acceptance criteria of some requirements/features include:

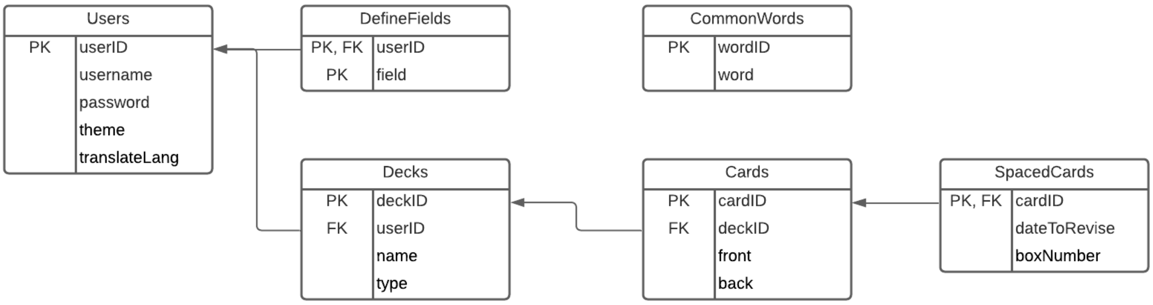
- Giving users the ability to add translations when generating decks. Users can choose from three options: translations only, definitions only, or both.
- Removing the difficulty option for generated decks. This feature is designed for English learners and will generate decks based on a set of commonly used terms.
- Adding the option of choosing default translation language in the preferences page.

Database Model

The following is an ER diagram of the database model. A double line denotes total participation.



The following is a relational model of the ER diagram. Values are atomic and arrows map foreign keys to the referenced table.



Users	
PK	userID
	username
	password
	theme
	translateLang

DefineFields	
PK, FK	userID
PK	field

CommonWords	
PK	wordID
	word

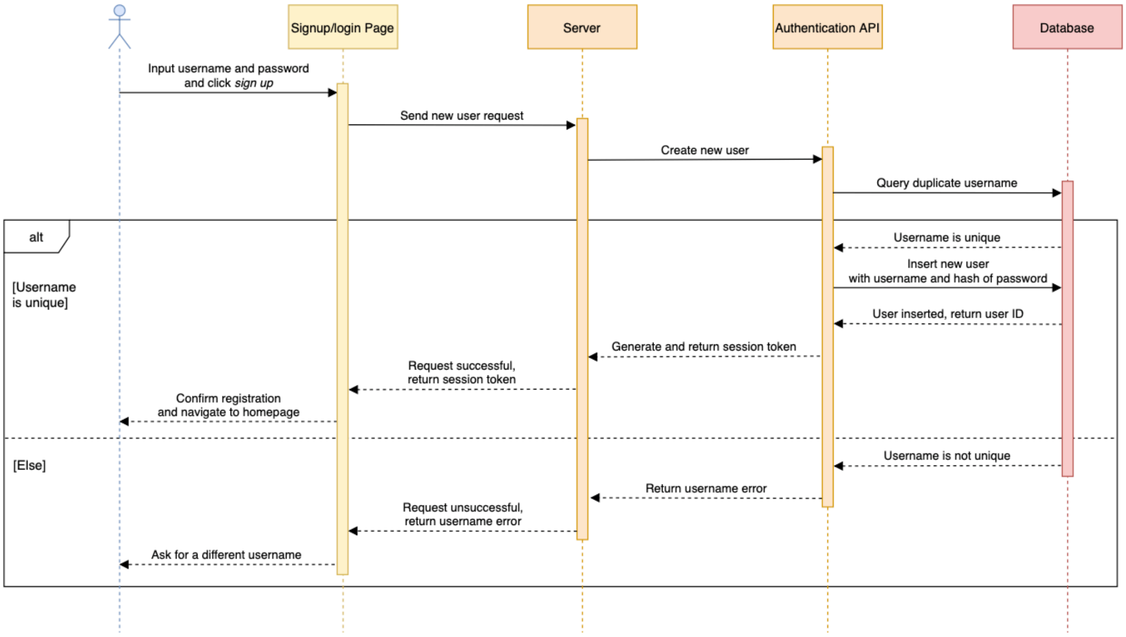
Decks	
PK	deckID
FK	userID
	name
	type

Cards	
PK	cardID
FK	deckID
	front
	back

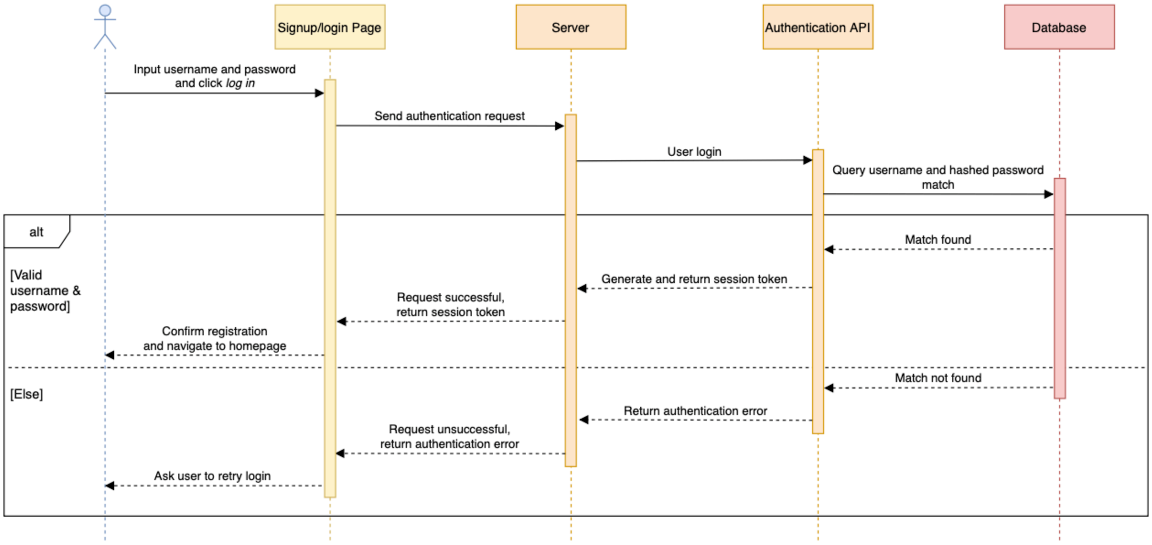
SpacedCards	
PK, FK	cardID
	dateToRevise
	boxNumber

Requirements and Sequence Diagrams

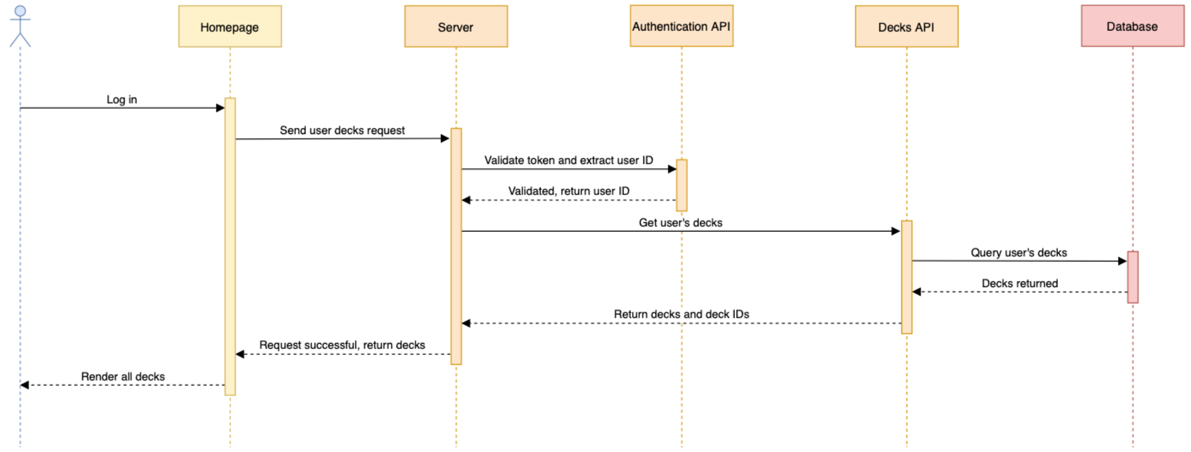
1.1 Register an account


Signup/login Page
Server
Authentication API
Database
Input username and password and click *sign up*
Send new user request
Create new user
Query duplicate username
alt
[Username is unique]
Username is unique
Insert new user with username and hash of password
User inserted, return user ID
Generate and return session token
Request successful, return session token
Confirm registration and navigate to homepage
[Else]
Username is not unique
Return username error
Request unsuccessful, return username error
Ask for a different username

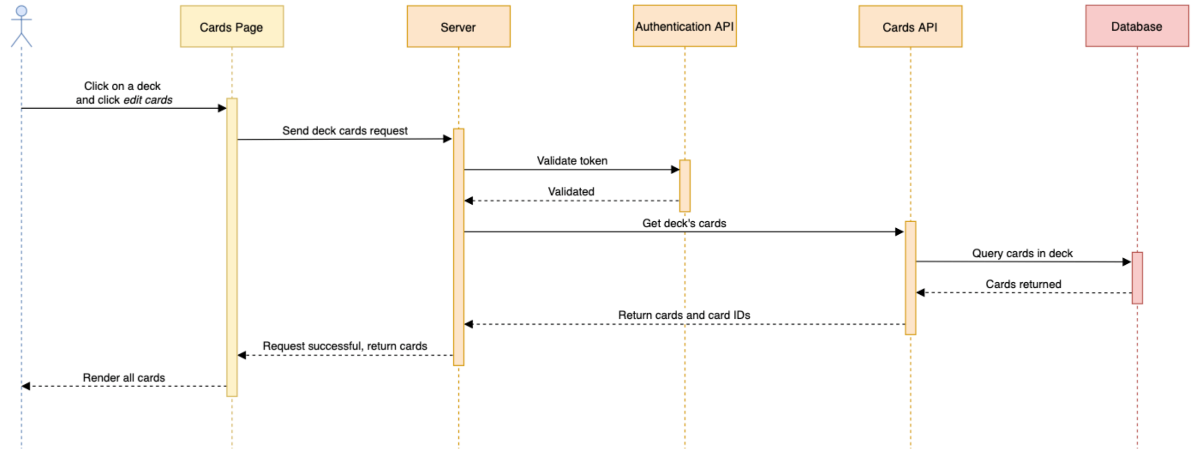
1.2 Login to account


Signup/login Page
Server
Authentication API
Database
Input username and password and click *log in*
Send authentication request
User login
Query username and hashed password match
alt
[Valid username & password]
Match found
Generate and return session token
Request successful, return session token
Confirm registration and navigate to homepage
[Else]
Match not found
Return authentication error
Request unsuccessful, return authentication error
Ask user to retry login

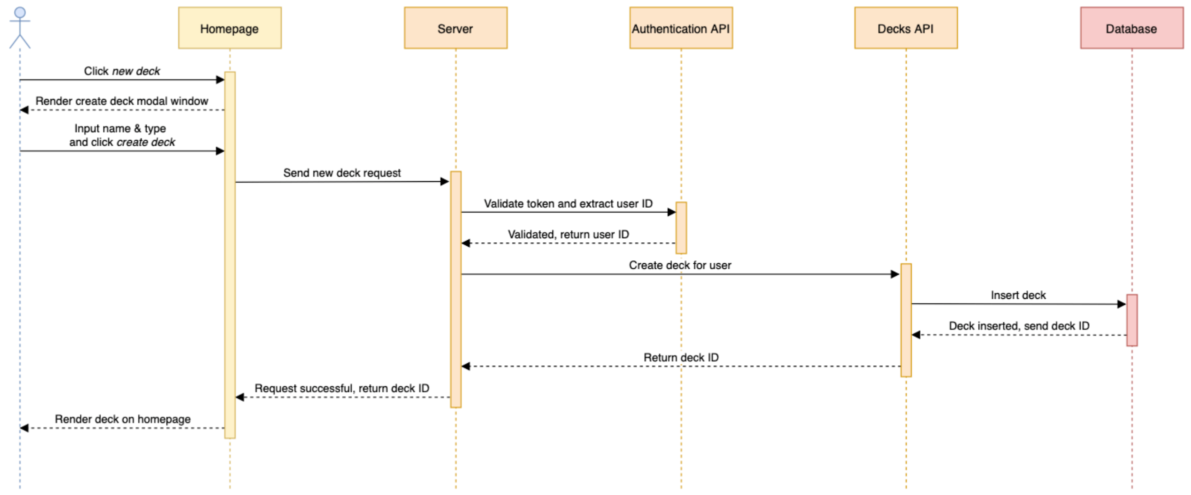
1.3 View all user’s decks



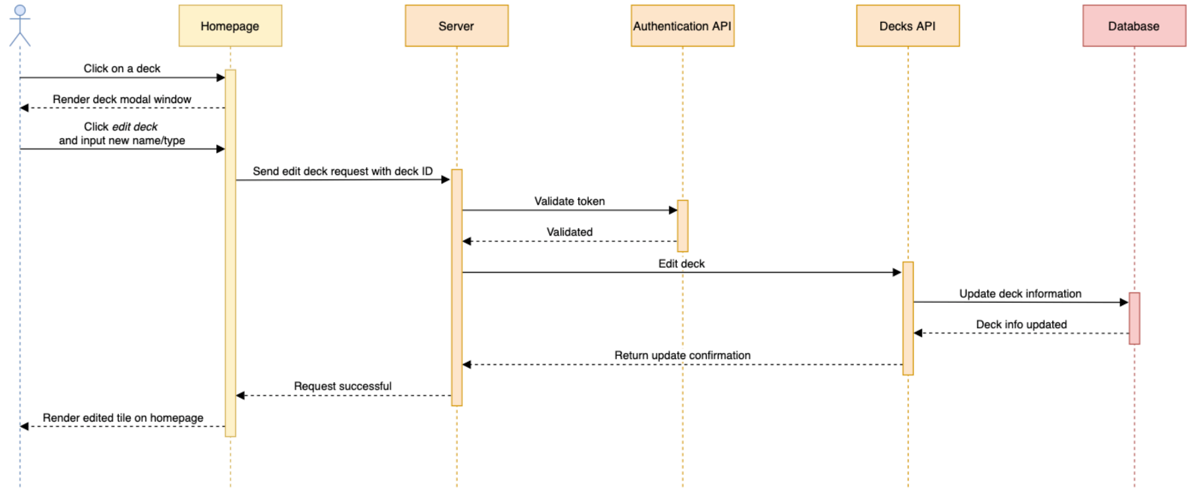
1.4 View all cards in a deck



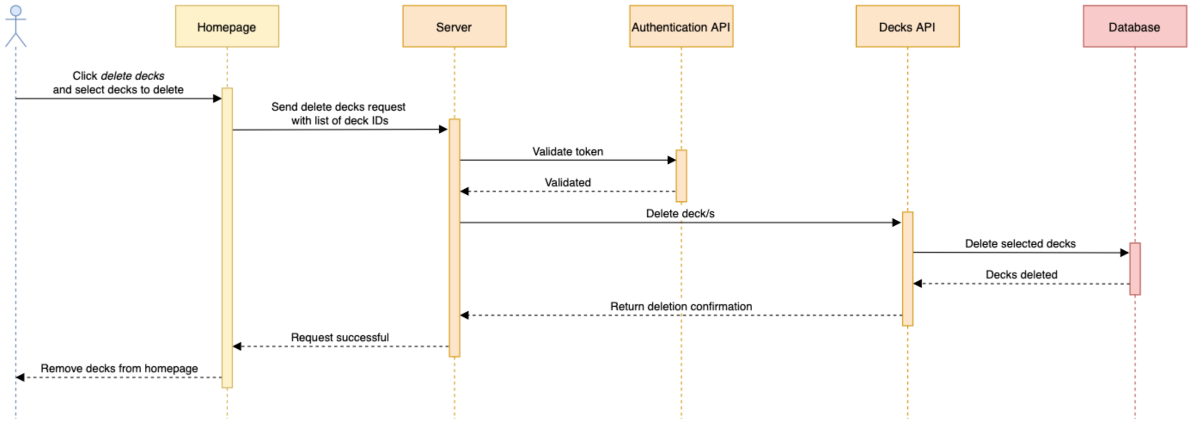
2.1 Create deck



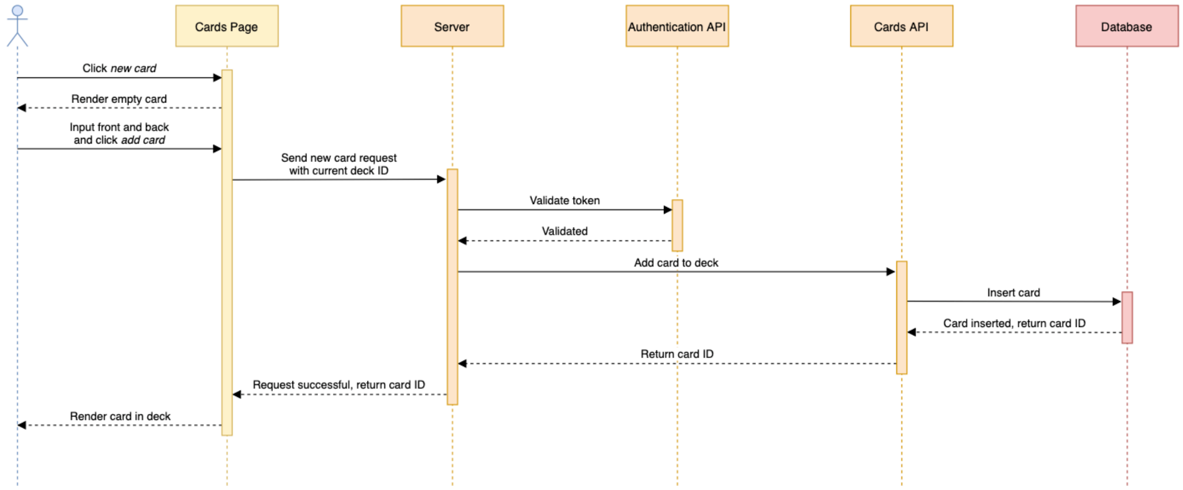
2.2 Edit deck



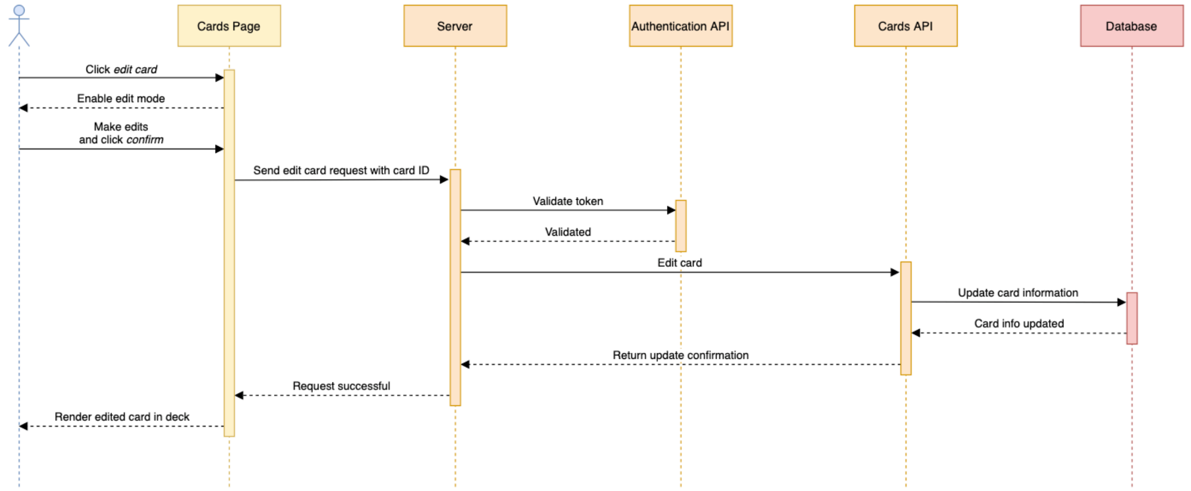
2.3 Delete decks


Homepage
Server
Authentication API
Decks API
Database
Click *delete decks* and select decks to delete
Send delete decks request with list of deck IDs
Validate token
Validated
Delete deck/s
Delete selected decks
Decks deleted
Return deletion confirmation
Request successful
Remove decks from homepage

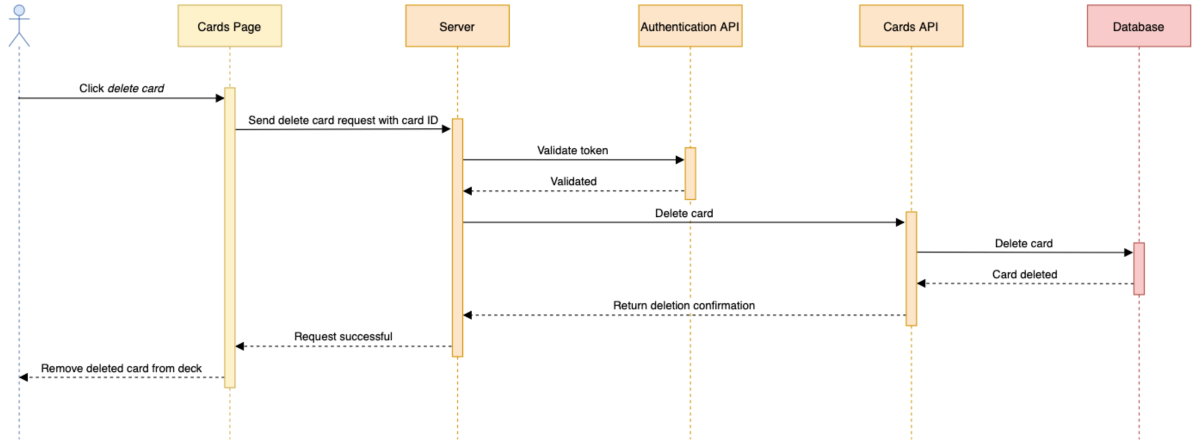
3.1 Create flashcard



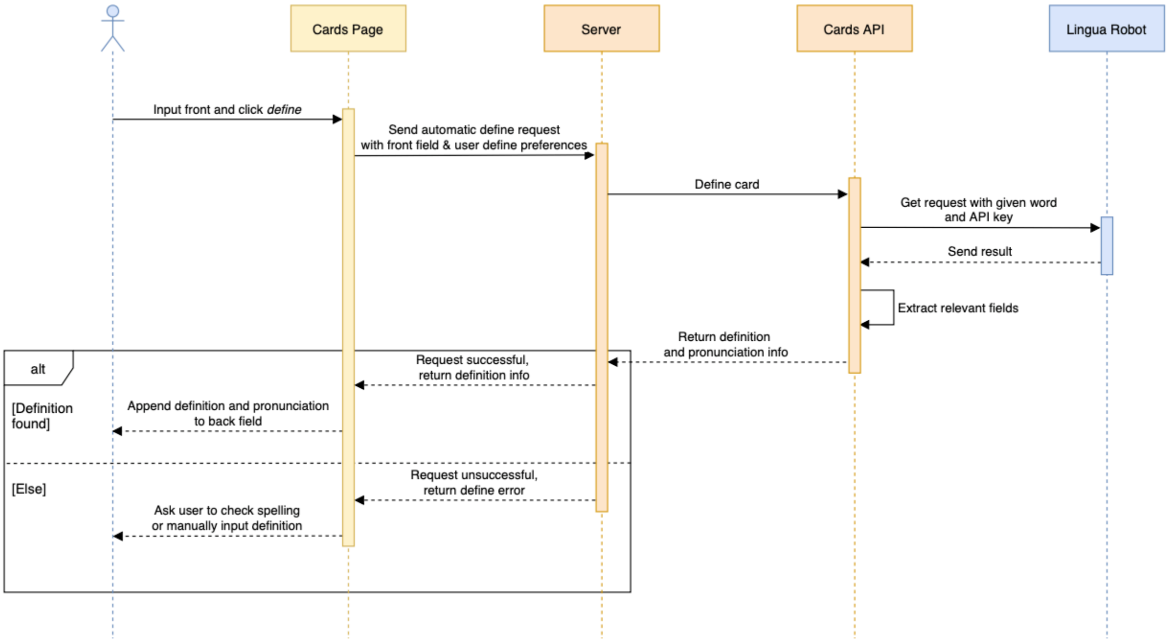
3.2 Edit flashcard



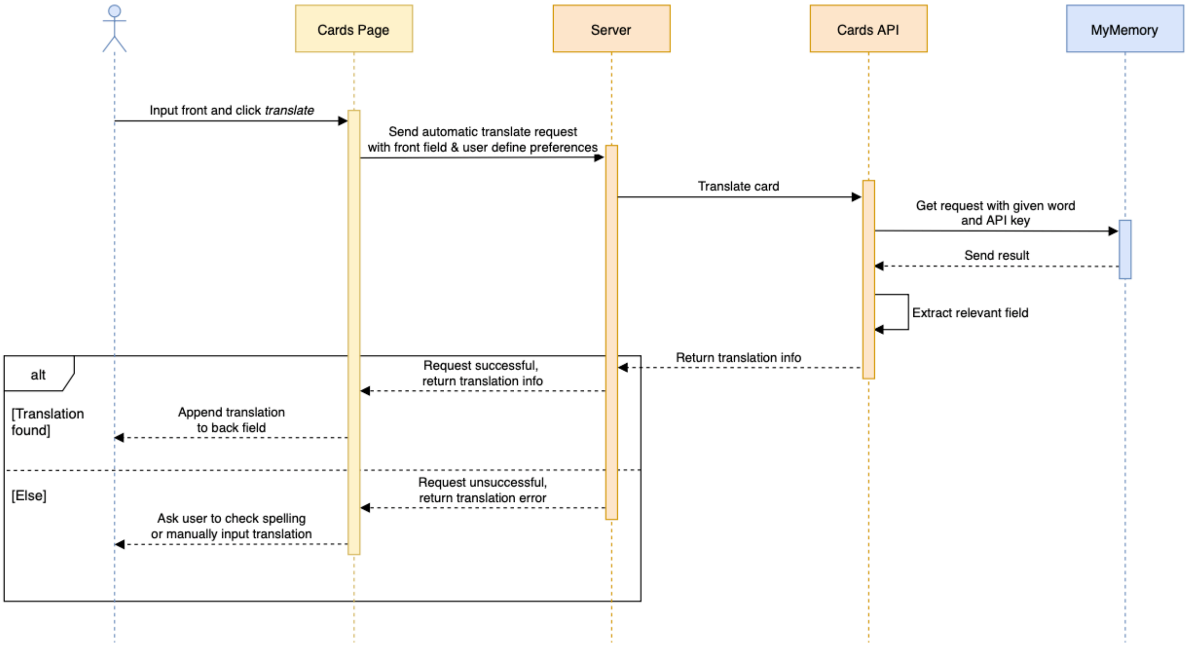
3.3 Delete flashcard



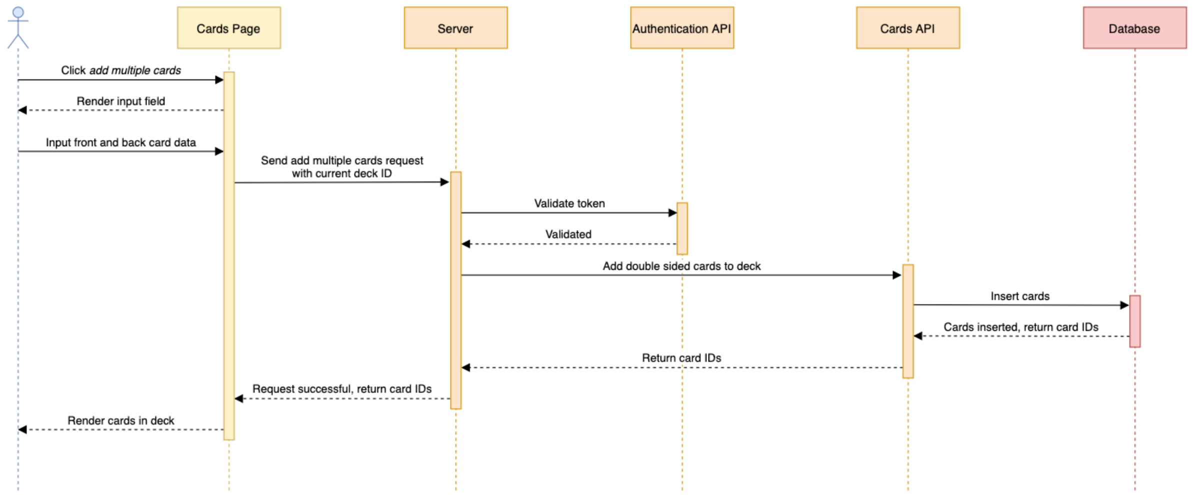
4.1 Automatically insert definition and pronunciation for vocabulary flashcard


Cards Page
Server
Cards API
Lingua Robot
Input front and click *define*
Send automatic define request with front field & user define preferences
Define card
Get request with given word and API key
Send result
Extract relevant fields
Return definition and pronunciation info
alt
[Definition found]
Request successful, return definition info
Append definition and pronunciation to back field
[Else]
Request unsuccessful, return define error
Ask user to check spelling or manually input definition

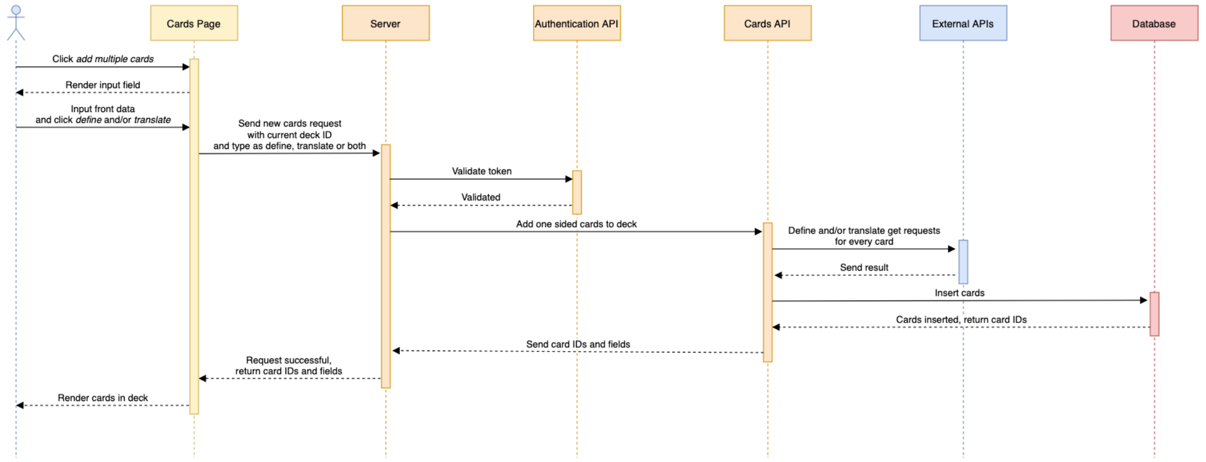
4.2 Automatically insert translation for vocabulary flashcard


Cards Page
Server
Cards API
MyMemory
Input front and click *translate*
Send automatic translate request with front field & user define preferences
Translate card
Get request with given word and API key
Send result
Extract relevant field
Return translation info
alt
[Translation found]
Request successful, return translation info
Append translation to back field
[Else]
Request unsuccessful, return translation error
Ask user to check spelling or manually input translation

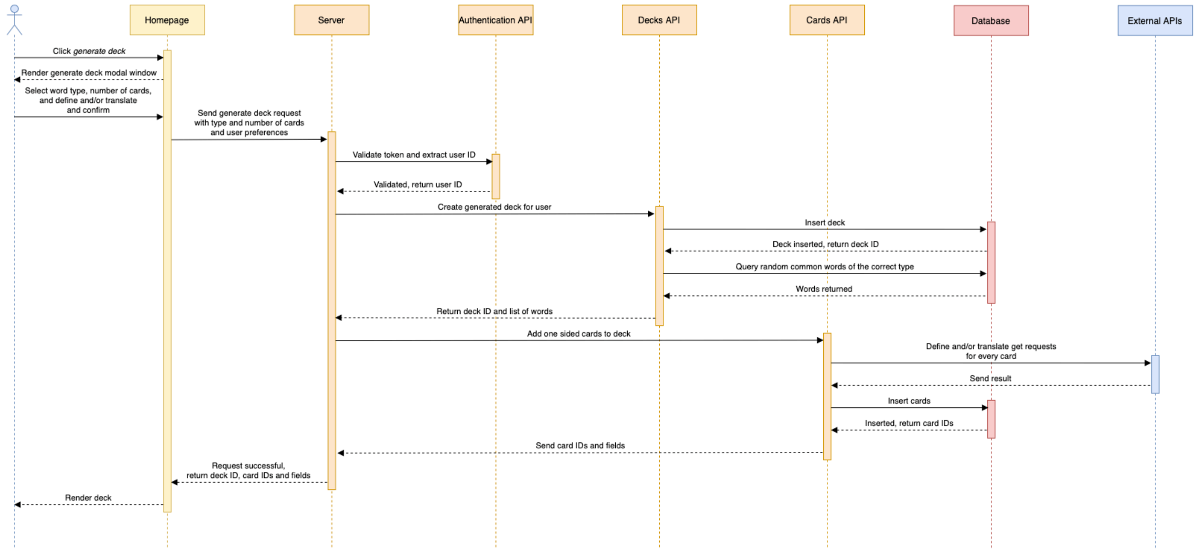
5.1 Create multiple flashcards given front and back fields


Cards Page
Server
Authentication API
Cards API
Database
Click *add multiple cards*
Render input field
Input front and back card data
Send add multiple cards request
with current deck ID
Validate token
Validated
Add double sided cards to deck
Insert cards
Cards inserted, return card IDs
Return card IDs
Request successful, return card IDs
Render cards in deck

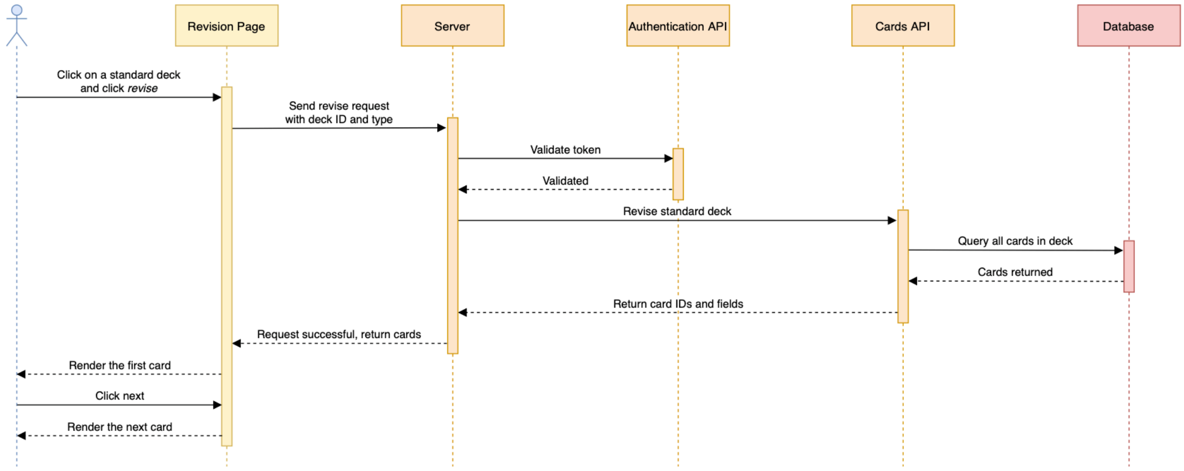
5.2 Create multiple vocabulary flashcards given front fields



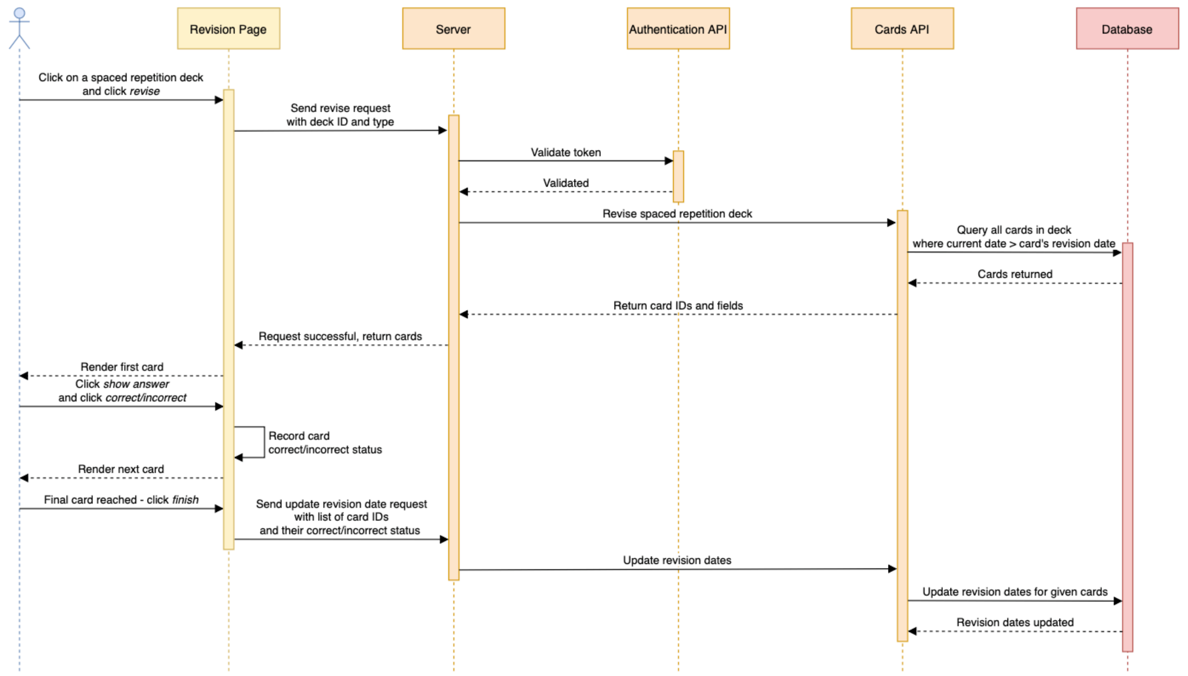
6 Generate vocabulary deck


Homepage
Server
Authentication API
Decks API
Cards API
Database
External APIs
Click *generate deck*
Render generate deck modal window
Select word type, number of cards, and define and/or translate and confirm
Send generate deck request with type and number of cards and user preferences
Validate token and extract user ID
Validated, return user ID
Create generated deck for user
Insert deck
Deck inserted, return deck ID
Query random common words of the correct type
Words returned
Return deck ID and list of words
Add one sided cards to deck
Define and/or translate get requests for every card
Send result
Insert cards
Inserted, return card IDs
Send card IDs and fields
Request successful, return deck ID, card IDs and fields
Render deck

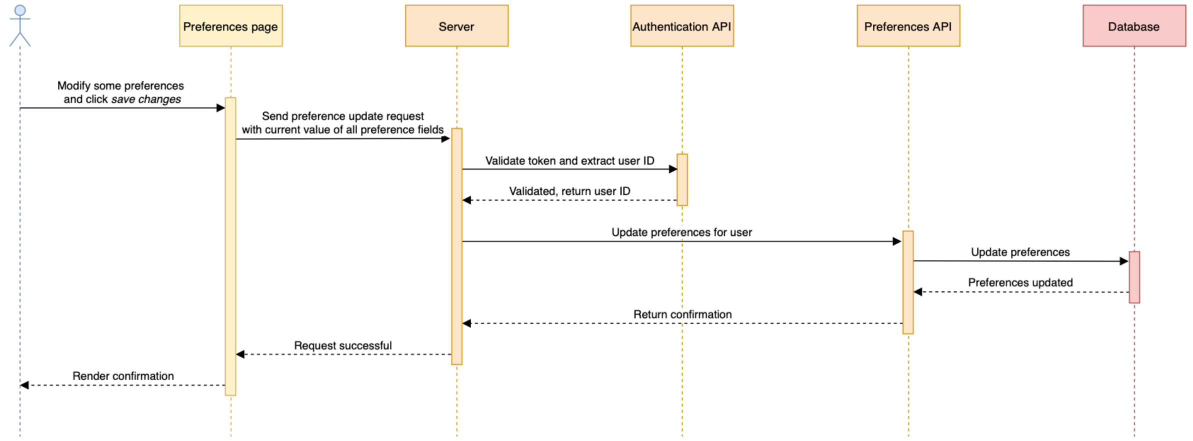
7.1 Revise flashcards in a standard deck



7.2 Revise flashcards in a spaced repetition deck


Revision Page
Server
Authentication API
Cards API
Database
Click on a spaced repetition deck and click *revise*
Send revise request with deck ID and type
Validate token
Validated
Revise spaced repetition deck
Query all cards in deck where current date > card's revision date
Cards returned
Return card IDs and fields
Request successful, return cards
Render first card
Click *show answer* and click *correct/incorrect*
Record card correct/incorrect status
Render next card
Final card reached - click *finish*
Send update revision date request with list of card IDs and their correct/incorrect status
Update revision dates
Update revision dates for given cards
Revision dates updated

8 Update user preferences


Preferences page
Server
Authentication API
Preferences API
Database
Modify some preferences and click *save changes*
Send preference update request with current value of all preference fields
Validate token and extract user ID
Validated, return user ID
Update preferences for user
Update preferences
Preferences updated
Return confirmation
Request successful
Render confirmation