Author

Arvind Sankar
21f1002061@student.onlinedegree.iitm.ac.in

I am primarily a law student with an inclination towards technology. I am interested in patents and, more recently, in smart contracts.

Description

This project requires the creation of a Flashcard application with Spaced Repetition capabilities. The problem state prescribed 4 core features, viz.: a registration and login facility for user authentication, a dashboard for each user to view each of their decks, a review mechanism with which users can test themselves, and a page for allowing user to read, insert, update and delete decks and cards.

Technologies used

The app was built using Flask and its associated libraries. For instance, Flask-SQLAlchemy was used to create and manage the database; Flask-Security-too was used to create the registration and login facilities; and Flask-RESTful was used to create an API for the app. Apart from these libraries, the datetime module and the JSON libraries were used for performing some specific operations.

DB Schema Design

The DB schema consists of 5 tables, with three tables (viz. user, role and role_user) being used for user authentication and two operational tables (viz. cardecks, and cards) being used for storing and reviewing flashcards. A detailed structure of the schema is available at this link.

The primary reason behind designing the DB schema in this manner was to allow users to seek decks and their respective cards with ease. Once the user successfully logs in, the cardecks table is queried to return all decks associated with that user. The cardecks table also contains the name (topic attribute) and the date of last review (last_r attribute). Individual cards are stored in the cards table with attributes for the front and back of the card, the interval for subsequent review, and the date for the next review of the card.

This schema retains all the functional dependencies (particularly from user to decks and decks to cards) while also removing redundancies. This schema ensures that each deck and each card is appropriately attributed to its creator user and parent deck respectively; without repeating data that may be redundant for the purpose in hand. The schema also reflects the manner in which the app is structured, thereby making it easy to make queries.

API Design

Here is the link to the plain YAML file and the Insomnia YAML Export file.

The API essentially allows the users to utilize all features available in the app. Using the API, the user can obtain their list of decks, and add and update decks using the user_id. The user can additionally obtain the list of cards in a deck using the deck_id and the user_id. This allows the user to view, add and delete cards in a deck. Finally, by also using the card_id, the user can update and delete an existing card.

The API was implemented by using the Flask-RESTful library and adding appropriate ‘resources’ from the library.

Architecture and Features

The organization of the files and folders can be viewed in detail here. Essentially, the root directory contains all the python files along with a template folder which contains the html templates, and the static folder which further consists of a CSS folder and a JS folder containing CSS and JS files respectively.

Currently, each page is rendered using flask for handling requests, jinja2 for rendering HTML files, JS for validation and other functionalities, and CSS (from bootstraps). I am working on implementing VueJS into the project and essentially trying to convert the current set of features into a Single page Application using routers, components, etc.

- Register and Login Facility: Flask Security was used to provide this facility. After creating models in accordance with its documentation, the Security module (used in conjunction with SQLAlchemySessionUserDatastore) was used to implement the login feature. In the app,
- User Dashboard: Once the user logs in, they are directed to the dashboard where they can view their list of decks, the date the deck was last reviewed, and a score, which is essentially a percentage of the average of each card's interval over the largest interval in the deck. The user Dashboard then allows users to review cards (next feature), add decks, manipulate the deck by redirecting to the deck page, and delete decks.
- Review Page: The review page returns all the cards associated with the deck_id, and shows a list of cards with the back (answers) hidden, using JS. The user can click the 'Show Answer" button to view the hidden answer. Subsequently, the user can select the difficulty and press the submit button to end review.
- Deck Management: By accessing the deck page, the user can create, update and delete cards. These functions are validated with client-side and server-side validations.

Apart from the above core functionalities, the app also has an API where users can perform CRUD operations on decks and cards. Further, the app also performs validations on the data provided by the user. For example, the validator ensures that users only submit alphabets as inputs and cards or decks are not duplicated.