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. Vue JS has been used to provided interactivity and reactivity to the front end.

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. This API has been extensively used by the front-end to provide the requisite interactivity. Furthermore, this has also allowed me to create a near Single Page Application.

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

Architecture and Features

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 VueJS for frontend reactivity, and CSS (from bootstrap). By implementing VueJS into the project, the app has been converted into a pseudo-Single page Application, in that it uses three routes (viz. For home page, dashboard page and review page), while all other functionality such as adding, manipulating and deleting decks and cards, have been integrated into the Dashboard route using appropriate routers and issuing fetch calls whenever data needed to be received or sent.

- Register and Login Facility: Flask-Security-too was used to provide this facility, specifically it was implemented using JWT. Once the user provides email ID and password and clicks on login, a fetch (POST) request is sent with the credentials to '/login?include_auth_token'. An API authentication token is thereby received which is later stored in a cookie and used in subsequent fetch queries wherever authentication is required. Further, logging in using this method automatically satisfies '@login_required' tags in other routes of the flask application.
- User Dashboard: Once the user logs in, they are directed to the dashboard where the user can see their user profile details, i.e. their user ID and API auth token. The navigation bar provides a router link (implemented using Vue Router) to the user's Deck list (which is a separate Vue Component), where they can view their list of decks, the date the deck was last reviewed etc. The Deck List router then allows users to review cards (next feature), add decks, manipulate the deck (i.e. add cards), and delete decks. Here, when the user clicks on the 'Edit Deck' button, the Deck ID is stored in Vuex and then the user is redirected to the Card List using the router link. When the Card List component is initialized, the Deck ID from the Vuex is retrieved to perform the Fetch request to get card data for the deck. Here, the user can perform CRUD operations on the cards, much like for decks.
- Review Page: Although on a separate route, the review page is essentially one Vue Component. The review page returns all the cards associated with the Deck ID, and displays the Front of one card while keeping the Back hidden. The user can click the 'Show Answer" button to view the hidden answer. Subsequently, the user can select the difficulty for remembering the answer and clicks on next. When the next button is clicked, a Fetch Post request is sent to update the 'Interval' attribute of the card in the database. Subsequently, the next card is displayed with the Back hidden.
- 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.

Further, a very rudimentary form of Caching is also implemented. Here, the home page has been cached using Flask-Caching as it is the only page in the app that remains relatively static.

Finally, using Redis, Celery and SMTPlib, I have designed a scheduled job that sends an email to the client once every week at a specified time.