MAD-I Final Project Report

Author

Tushar Shrikrishna Supe
21F1003637
21f1003637@student.onlinedegree.iitm.ac.in
I hold a BTech degree in Mechanical Engineering from VNIT Nagpur. I have worked in RIL for a year but as I am interested in AI & ML, I joined this BSc Degree in Programming and Data Science offered by IIT Madras.

Description

In this **Flashcard Web App** project we need to create a web based app using Flask and Python which follows basic MVC design and Client–server model. Client can create deck and store on server and can review cards anytime anywhere using web browser and internet connection. While reviewing flashcard, front side will be displayed and user will flip the card to check answer and choose where the card was Easy, Medium, Difficult.

Technologies used

Technology	**Purpose**
Python	Core programming language of this app
HTML	Creating web pages
CSS	Styling web pages
JavaScript	For basic functionalities in web pages like display/hide content, change background colour of specific element
flask	Web framework library
flask-sqlalchemy	Library to create SQLite database, to connect with database
flask-login	Library for Login authentication for the app
flask-restful	Library for implementing CRUD API for deck management
Jinja2	Library for rendering html templates (comes within flask)
requests	Library to send get, post, put, delete requests to API
pandas	Library to create/read/write CSV file
*in-built: os, time, datetime, json, random	
*basic: device with web browser and internet connection	

DB Schema Design

Table name: users			
Column	**Datatype**	**Constraint**	**Description**
userid	Integer	Primary key, Autoincrement	Unique user id for every user
username	String	Unique, Not null	Unique user name (can't be edited once created)
password	String	Not null	Password specified by user

Table name: deck			
Column	**Datatype**	**Constraint**	**Description**
deckid	Integer	Primary key, Autoincrement	Unique deck id for every deck
deckname	String	Not null	Unique deck name for same user id
deckdesc	String	Not null	Description of the deck
ltime	Integer		Last reviewed time of deck (time as timestamp)
lscore	Float		Last reviewed score of the deck
oscore	Float		Overall score of the deck
userid	Integer	ForeignKey("users.userid"), Not null	To link user to deck

Table name: cards			
Column	**Datatype**	**Constraint**	**Description**
cardid	Integer	Primary key, Autoincrement	Unique card id for every card
question	String	Not null	Front side of card, must be unique per deck
answer	String	Not null	Back side of the card
ltime	Integer		Last reviewed time of card (time as timestamp)
lscore	Float		Last reviewed score of a card
deckid	Integer	ForeignKey("deck.deckid"), Not null	To link card to deck

Single database is used to store user details, decks and cards to reduce complexity while managing decks and cards. Userid from user table links to deck table and deckid links to cards table so for logged-in user other decks & cards are not accessible which provides data security. Both User-Deck and Deck-Cards follows one-many relationship (for same user two deck can have similar cards but as per this db schema two cards will have different cardid). Also one-many relationship for deck-card is defined in ‘models.py’ so given a deckid all the cards in that deck can be retrieved. Along with this, ‘on delete cascade’ constraint set on deckid so that deleting a deck will delete all its card records from cards table also.

API Design

The CRUD API is implemented on both Deck and Cards table. The ‘controllers.py’ file sends get, put, post, delete requests to api (api.py) and then api retrieves data or update/delete from database for valid requests else sends error message with status code and then ‘controllers.py’ file renders template and flashes appropriate messages if required.

Resource	**Request**	**Endpoint**	**Request body**	**Purpose (for current user)**
Deck	GET	/api/deck	-	To get details of all decks
Deck	POST	/api/deck	deckname, deckdesc	To create a new deck
Deck	GET	/api/deck/<int:deckid>	-	To get deck details
Deck	PUT	/api/deck/<int:deckid>	deckdesc	To update deck description
Deck	PUT	/api/deck/<int:deckid>	ltime, lscore	To update last reviewed time, score of deck
Deck	DELETE	/api/deck/<int:deckid>	-	To delete a deck
Deck	GET	/api/deck/<string:deckname>	-	To get deckid from deckname
Cards	GET	/api/deck/<int:deckid>/card/<int:cardid>	-	To get single card details
Cards	PUT	/api/deck/<int:deckid>/card/<int:cardid>	question, answer	To update front and back details of a card
Cards	PUT	/api/deck/<int:deckid>/card/<int:cardid>	ltime, lscore	To update last reviewed time and score of a card
Cards	DELETE	/api/deck/<int:deckid>/card/<int:cardid>	-	To delete a card
Cards	POST	/api/deck/<int:deckid>/card	question, answer	To create a new card

Architecture and Features

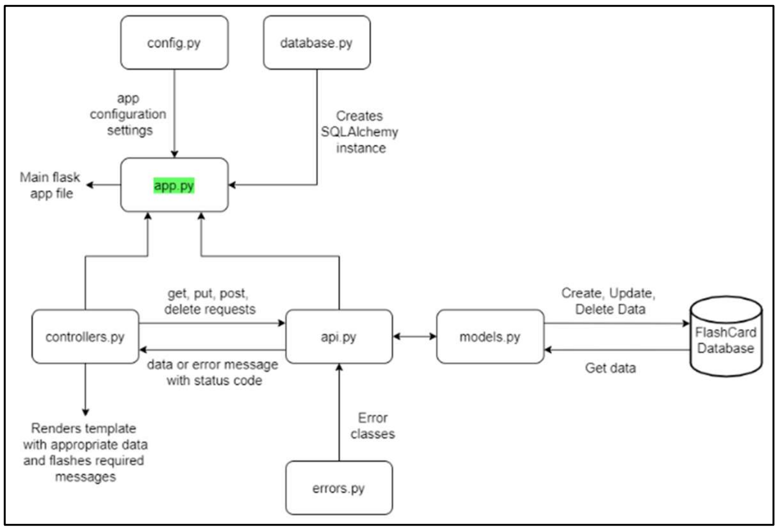
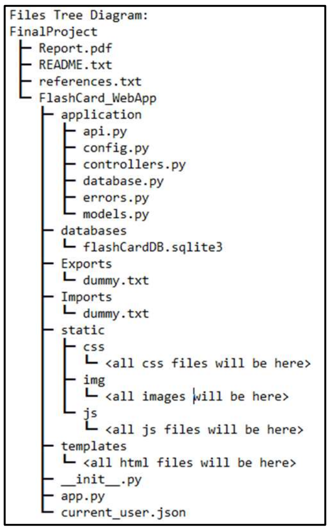

config.py
database.py
app configuration settings
Creates SQLAlchemy instance
Main flask app file
app.py
get, put, post, delete requests
Create, Update, Delete Data
controllers.py
api.py
models.py
FlashCard Database
data or error message with status code
Get data
Renders template with appropriate data and flashes required messages
Error classes
errors.py

Figure 1: Interconnection of .py files and database



```
Files Tree Diagram:
FinalProject
 ├ Report.pdf
 ├ README.txt
 ├ references.txt
 └ FlashCard_WebApp
    ├ application
    │  ├ api.py
    │  ├ config.py
    │  ├ controllers.py
    │  ├ database.py
    │  ├ errors.py
    │  └ models.py
    ├ databases
    │  └ flashCardDB.sqlite3
    ├ Exports
    │  └ dummy.txt
    ├ Imports
    │  └ dummy.txt
    ├ static
    │  ├ css
    │  │  └ <all css files will be here>
    │  ├ img
    │  │  └ <all images will be here>
    │  └ js
    │     └ <all js files will be here>
    ├ templates
    │  └ <all html files will be here>
    ├ __init__.py
    ├ app.py
    └ current_user.json
```

Figure 2: Files tree structure

Parent directory is ‘FinalProject’ and all app related code/files are inside ‘FlashCard_webApp’ dir (ref: Figure 2). Inside that, all templates are in ‘templates’ dir, then inside ‘static’ dir, css files are in ‘css’ dir, images in ‘img’ dir, JavaScript files in ‘js’ dir. Supporting .py files are in ‘application’ dir. The SQLite database file will be stored in ‘databases’ dir. While importing a deck from .csv file its temp copy will be saved in ‘Imports’ dir. While exporting a deck in .csv file it will be saved in ‘Exports’ dir.

Features:

- ✓ Decks and Cards can be added, updated as well as deleted.
- ✓ While reviewing cards, if user changes his mind, he can go to dashboard and reviewing will be aborted but scores till that card will be updated. No need to review all cards to get out of review process.
- ✓ It supports multiple languages (utf-8).

- ✓ Proper login system is created using flask-login.
- ✓ Deck can be imported from a .csv* file (*csv file must have headers: Srno, Question, Answer)
- ✓ Deck can be exported to a .csv file

Video

Link: https://drive.google.com/file/d/18Syd-yRrymwFJQ6UiXUKJGGePwOJ4usl/view?usp=sharing