Author

Name: Utkarsh Sahu
Student roll number: 21f1001107
Student mail id: 21f1001107@student.onlinedgree.iitm.ac.in

Hi, I am Utkarsh Sahu, a second-year engineering student at Delhi Technological University. I like coding and development. I also enjoy watching web series and cooking. I hope you will like my project.

Description

My understanding of the problem statement was that the app in itself is straightforward. We need to create a database to store users, their decks and the cards within multiple decks. In short, we had to implement an app that allows users to register/login and create decks, and within decks, create cards and track the progress, i.e., **there will be multiple users, one user might have multiple decks and one deck have numerous decks.**

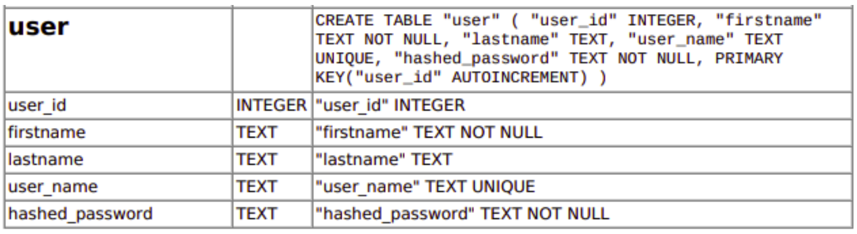
Technologies used

These are the technologies/dependencies that I have used:

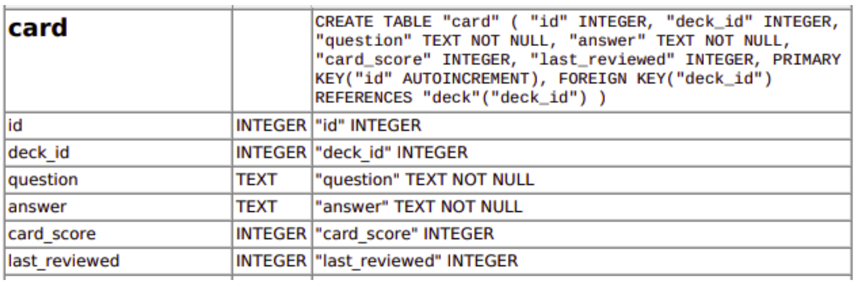
- **flask** - for main flask app
- **flask_restful** - for REST API implementation
- **flask_sqlalchemy** - add support to image upload, code syntax highlighting and more
- **JWT** - for token based authentication
- **Vue** - for frontend UI
- **json** - for returning json response from APIs
- **sqlalchemy** - Object Relational Mapper in python for SQL
- **Redis** - for caching
- **Celery** - as message broker

DB Schema Design

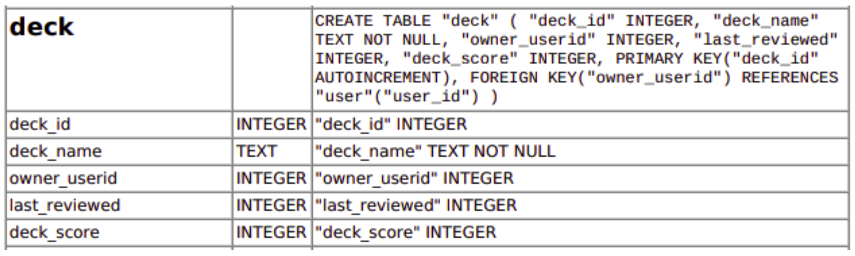
The Schema Design has been kept very simple, one table for Users, one for Decks and one for Cards. User Table has **user_id** as primary key and **user_name** field has been kept as unique because first-name and last-name might be similar for two different users and **user_id** is a foreign key in **Deck** table so as to know the owner of that deck and **deck_id** is a foreign key in **Card** table so as to know which deck does the card belongs to. Each deck and card has a last_reviewwed column and score column so as to store the latest time and score for tracking the progress of the user. In simple words, **there can be multiple users, one user can create multiple decks, one deck might have many cards.** The following images describe tables in the database, their columns and their column type with constraints(if any). **"user_name"** has been kept unique because that identifies the user too and is a common way.



user		CREATE TABLE "user" ("user_id" INTEGER, "firstname" TEXT NOT NULL, "lastname" TEXT, "user_name" TEXT UNIQUE, "hashed_password" TEXT NOT NULL, PRIMARY KEY("user_id" AUTOINCREMENT))
user_id	INTEGER	"user_id" INTEGER
firstname	TEXT	"firstname" TEXT NOT NULL
lastname	TEXT	"lastname" TEXT
user_name	TEXT	"user_name" TEXT UNIQUE
hashed_password	TEXT	"hashed_password" TEXT NOT NULL



card		CREATE TABLE "card" ("id" INTEGER, "deck_id" INTEGER, "question" TEXT NOT NULL, "answer" TEXT NOT NULL, "card_score" INTEGER, "last_reviewed" INTEGER, PRIMARY KEY("id" AUTOINCREMENT), FOREIGN KEY("deck_id") REFERENCES "deck"("deck_id"))
id	INTEGER	"id" INTEGER
deck_id	INTEGER	"deck_id" INTEGER
question	TEXT	"question" TEXT NOT NULL
answer	TEXT	"answer" TEXT NOT NULL
card_score	INTEGER	"card_score" INTEGER
last_reviewed	INTEGER	"last_reviewed" INTEGER



deck		CREATE TABLE "deck" ("deck_id" INTEGER, "deck_name" TEXT NOT NULL, "owner_userid" INTEGER, "last_reviewed" INTEGER, "deck_score" INTEGER, PRIMARY KEY("deck_id" AUTOINCREMENT), FOREIGN KEY("owner_userid") REFERENCES "user"("user_id"))
deck_id	INTEGER	"deck_id" INTEGER
deck_name	TEXT	"deck_name" TEXT NOT NULL
owner_userid	INTEGER	"owner_userid" INTEGER
last_reviewed	INTEGER	"last_reviewed" INTEGER
deck_score	INTEGER	"deck_score" INTEGER

API Design

The REST API has been created for **GET, PUT, DELETE, POST** methods which allow us to perform **CRUD operations** on the User, Deck, Card entities.

UserApi:

METHOD	**API endpoints**	**Request Parameters**	**Response Parameters**
GET	"/api/user/<string:username>"	-	userid,username,firstname,lastname
PUT	"/api/user/<string:username>"	-	new_fname,new_lname
DELETE	"/api/user/<string:username>"	-	deleteduser_id,deleteduser_fname,deleted user_lname,deleteduser_username
POST	"/api/user"	firstname, lastname, username, password	newuserid,newuser_fname,newuser_lname, newuser_username,message

DeckApi:

METHOD	**API endpoints**	**Request Parameters**	**Response Parameters**
GET	"/api/deck/<int:deckid>"	-	deck_id, deck_name, owner_userid, last_reviewed, deck_score
PUT	"/api/deck/<int:deckid>"	-	deck_id, updateddeck_name, owner_userid, last_reviewed, deck_score
DELETE	"/api/deck/<int:deckid>"	-	deck_id, deck_name, owner_userid, last_reviewed, deck_score
POST	"/api/deck/"	deckid,deckname,ownerid	deck_id, deck_name, owner_userid, last_reviewed, deck_score

CardApi:

METHOD	**API endpoints**	**Request Parameters**	**Response Parameters**
GET	"/api/card/<int:cardid>"	-	cardid, question, answer, cardscore, parent_deckid, last_reviewed
PUT	"/api/card/<int:cardid>"	-	cardid, updated question, updated answer, cardscore, parent_deckid, last_reviewed
DELETE	"/api/card/<int:cardid>"	-	deleted cardid, question, answer, deleted cardscore, parent_deckid, last_reviewed
POST	"/api/card/"	question,answer,deckid	created cardid, question, answer, cardscore, parent_deckid, last_reviewed

Architecture and Features

The Architecture has been kept fairly simple , backend is in backend folder and frontend is in frontend folder.
Vue is used for User Interface purpose.
Features implemented are **Dashboard management, Secure login framework, Deck/Card management, Reviewing card for progress, Backend Validation, Styling and Aesthetics and last but not least a rich text editor** using standard ways and the technologies/dependencies listed above.

Video

Please watch this video for learning more about the project.
https://drive.google.com/drive/folders/1baa3tgnj-_f053wgdAKq_l40Ud0_oQfd?usp=sharing