Author

Name: C. Glenn Varshan

Roll Number:22F2000961

Student Email: 22f2000961@ds.study.iitm.ac.in

Description: I am 19 years old and I am doing BS in data science and programming as a full time course. This is my first time coding a full-fledged application.

Description

This project puts to work all the basics concepts of app development from frontend to backend database management using python and flask as the base.
It is a Show booking app which has multiple users and at least one Admin. Users can book shows and later access their bookings in their particular profile page.

Technologies used

Python, flask
Flask-extensions
-Flask-Login: used to Login user and keep track of Logged in users
-Flask-Admin: Creates an admin panel which allows admin to use CRUD operations on shows, venues, and other Database tables
-Flask-SQLalchemy: Used as the database model for this project
Other extensions
-werkzeug.security: Used for hashing the user and admin password in sha256 format so that the app never directly stores the passwords.
-datetime: used for ordering the shows by time added
-os: os is imported to check if the database exists or not in the folder
-json: Json is used for the summary bar chart as it uses javascript and json is used to convert the jinja2 passed through the script into the entered data list

DB Schema Design

TABLE user (id INTEGER NOT NULL, username VARCHAR(30) NOT NULL, password VARCHAR(80) NOT NULL, is_admin BOOLEAN, PRIMARY KEY (id), UNIQUE (username))

TABLE venue (id INTEGER NOT NULL, "Name" VARCHAR(100), "Place" VARCHAR(100),"Capacity" INTEGER, PRIMARY KEY (id))

TABLE show (id INTEGER NOT NULL, "Name" VARCHAR(100) NOT NULL, "Rating" INTEGER, "Tags" VARCHAR(100), "Ticket_price" INTEGER NOT NULL, "Booked" INTEGER, starttime INTEGER, endtime INTEGER, "Venue_id" INTEGER NOT NULL, date_added DATETIME, PRIMARY KEY (id), FOREIGN KEY("Venue_id") REFERENCES venue (id))

TABLE bookings (id INTEGER NOT NULL, "Tickets" INTEGER NOT NULL, "Name" VARCHAR(100) NOT NULL, "Rating" INTEGER NOT NULL, "Ticket_price" INTEGER NOT NULL, "Venue_id" INTEGER NOT NULL, "User_id" INTEGER NOT NULL, PRIMARY KEY (id), FOREIGN KEY("User_id") REFERENCES user (id))

User: ‘is_admin’ column is used to check if the user is an admin after the user is authenticated.

Venue: is linked to the show table as a back reference to the Show.Venue_id as it is a foreign key.

Show: The foreign key ‘Venue_id ‘ is used to link a show with a particular venue

Booking: each booking has to be linked to a user and to do that the foreign key ‘User_id’ is used

Venue to show is a one to many relation ship.

API Design

The elements of the api for the shows are api/listhshow,api/createshow, api/updateshow,api/deleteshow
And for the venue it is api/listvenue, api/updatevenue, api/deletevenue

The api/listvenue and api/listshows take GET methods and display the shows and venues

The api/createvenue and api/createshow take POST requests and adds a new entry to the database if the respective fields required are all filled

The api/updatevenue and api/updateshow take PUT requests and update each and every value of the shows and venues

The api/deletevenue and api/deleteshow take DELETE requests and delete the show or venue with the respective ‘id’ that is entered.

Architecture and Features

The project is saved in one base PROJECT folder which is divided into three sub parts: Files, instances and main.py. The Files subfolder contains the necessary flask code and html templates for the app.
The Instances sub-folder holds the database instance
To run the app simply run the main.py file and it calls the python files in the Files folder and the app will run.

All the core requirements are satisfied in this project such as User and admin login, venue management, show management, booking of shows and searching for a particular venue or show.

Basic bootstrap styling is used in the html files.

A proper login framework is also used and the admin can view a bar chart summary of the number of tickets sold for each show in the admin panel.
The shows Ratings are decided only after the user books tickets for the show and rates the show in his/her profile. This Rating is then used to change the price of the show using a simple formula (ticket price = ticket price + (rating *10)) this is done so that the tickets are priced dynamically.

Video

https://drive.google.com/file/d/16lVYne9WMvO08LLotLvdaziK0ltpP5Yr/view?usp=sharing