Author

G Vagadeeshwar
21f1005494
21f1005494@ds.study.iitm.ac.in

I am Vagadeeshwar g currently doing my online BS degree from IIT Madras as well as pursuing an offline degree in B. Tech Computer Science from VIT Chennai.
My passions include Music, Web Development and Data Science.

Description

The project requires creating a web application for managing venues, shows, and bookings. This involves building a user interface, implementing CRUD operations for the data models, and setting up authentication and authorization for secure access.

Technologies used

Flask & Werkzeug: To create the web application, handle routes, and render templates.
Flask_Login: To manage user sessions, handle authentication, and provide secure access to resources.
Flask_RESTful: To create RESTful APIs for CRUD operations with ease
flask_sqlalchemy: To integrate SQLAlchemy ORM (Object Relational Mapper) with the Flask web application
Flask_WTF & WTForms: To create, validate, and handle form data in the web application.
SQLAlchemy: To handle database operations, create data models, and perform database queries using Python objects and methods.
black: A Code Formatter for python as a dev-dependency

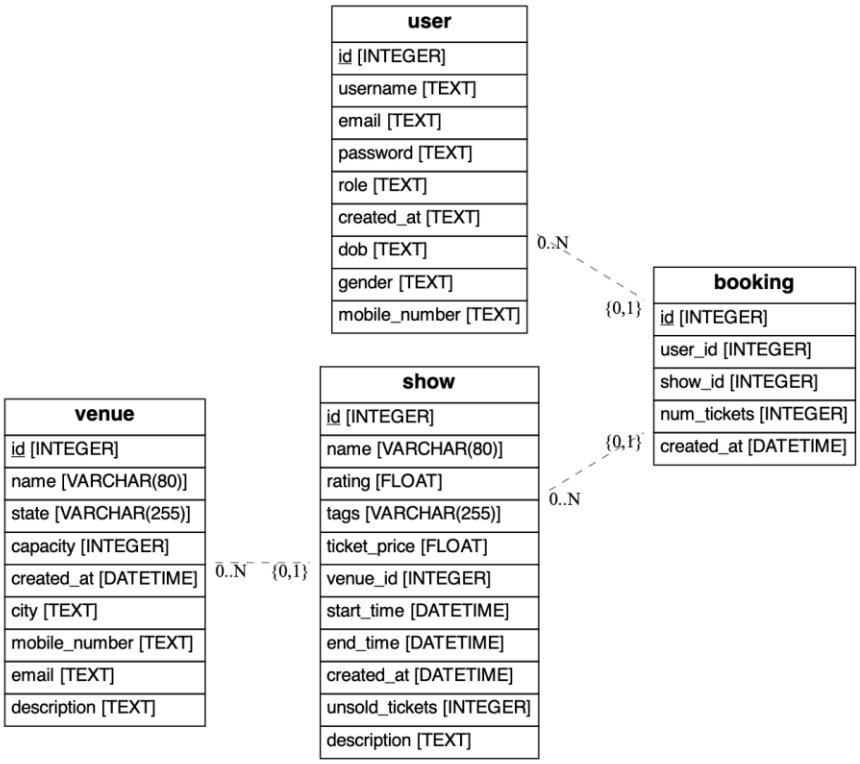
Others: HTML, CSS-Bootstrap, Javascript, Python, Jinja Templating
IDE: VSCode

API Design

I have created an API for the four models: Venue, Show, Booking, and User.
These elements are managed using Flask-RESTful. The API resources are defined as separate classes (VenueResource, ShowResource, BookingResource, and UserResource), which inherit from the Resource class provided by Flask-RESTful. Each class has a get method to retrieve the data for a specific resource based on its ID. The API routes are registered using api.add_resource() and associated with specific URL patterns.

The API functionality has been kept minimal as the CRUD Operations on the models have been implemented in a full fledged manner.

DB Schema Design



A User can have 0 or many Bookings, and a Booking must be associated with 1 User.
A Show can have 0 or many Bookings, and a Booking must be associated with 1 Show.
A Venue can have 1 or many Shows, and a Show must be associated with 1 Venue.

Architecture and Features

The project is organized using a modular structure, which helps maintain the separation of concerns and enhance the maintainability and readability of the code. The controllers are located in separate files under the 'application' directory, such as 'venue_views.py' for handling venue-related views and 'api.py' for defining the API endpoints. The templates are stored in the 'templates' folder, which is further divided into subdirectories for organizing views related to different aspects of the application (e.g., 'venue', 'user'). Similar structure is followed for forms.
The features implemented in this project include user authentication, CRUD operations for venues, shows, and bookings, as well as filtering venues based on specific criteria.
Default features in this project include user registration, login, and dashboard views. Additional features include a Minimalist User Friendly UI for quick navigation between the pages, dynamic rating allocation to shows based on several factors, Strict Validation rules,etc.

Video

https://drive.google.com/file/d/1nuF5S4UXda5VzkNfYMOnrMOh1KRdTO37/view?usp=share_link

Completed Checklist

- User/Admin Login
- Venue Management

- Show Management
- Search for Shows/Venues
- Book Show Tickets
- Validation for all CUD Operations
- CRUD on Venues/Shows/Bookings/Users
- API interaction with GET requests on Venues/Shows/Bookings/Users
- Styling and Aesthetics
- Proper Login System
- Dynamic Rating