BookIt - A ticket Booking Application

FINAL PROJECT FOR APP DEV-2

CLICK TO VIEW THE SCREENCAST

PREPARED BY

RISHAV JAIN

ROLL NO. : 21f1005352

Email : 21f1005352@ds.study.iitm.ac.in

Introduction

As part of the Final Project, I have made the BookIt Web Application which is used for creating and booking tickets for shows and events . The frontend is made with Vue Framework setup by Vue-CLI and backend by Flask.

Package Dependency

Following Packages have been used for making the Application:

1. **1. Vue** - To render components and routes. Other dependencies which are used in vue are:
   - **a. Routes:** to route to different pages
2. **2. Flask-** To create APIs and controllers providing and posting data in the database and also validations to frontend. Other Dependencies used with Flask are
   - **a. Flask-SQLAlchemy** : handling of a database connection across the app.
   - **b. flask-restful** : Extension of flask to create API Resource
   - **c. werkzeug-security** : Extension of flask to create hashing and verifying the login-password
   - **d. flask-cors:** To accept cross-origin requests from frontend fetch requests.By default not accepted.
   - **e. flask-jwt-extended:** very easy library providing token authorisation
   - **f. Celery[redis] and redis:** To facilitate synchronous jobs like daily reminder and monthly progress.
3. **3. Bootstrap:** It is used for styling and aesthetics.

Models

1. User Model

- This model represents users of the application.
- It has attributes: `user_id` (primary key), `email`, `name`, and `password`.
- It is associated with bookings made by users through a one-to-many relationship.
- The `bookings` relationship allows users to be associated with multiple bookings.
- The `lazy` parameter is set to "True" for the `bookings` relationship, indicating that the related data should be loaded lazily using subqueries.

2. Admin Model

- This model represents administrators of the application.

● It has attributes: `admin_id` (primary key), `name`, `email`, and `password`. ● It is associated with venues managed by administrators through a one-to-many relationship.
● The `venues` relationship indicates that each admin can be associated with multiple venues.
● The `lazy` parameter is set to "True" for the `venues` relationship, indicating that the related data should be loaded lazily using subqueries.

3. Venue Model

● This model represents event venues.
● It has attributes: `venue_id` (primary key), `name`, `venue_location`, and `admin_id` (foreign key referencing the Admin model).
● It is associated with shows held at the venue through a one-to-many relationship.
● The `shows` relationship signifies that each venue can host multiple shows.
● The `lazy` parameter is set to "True" for the `shows` relationship, indicating that the related data should be loaded lazily using subqueries.

4. Show Model

● This model represents shows or events hosted at venues.
● It has attributes: `show_id` (primary key), `name`, `date_time`, `seats_available`, `seats_booked`, `price`, `show_screen`, `image`, and `tags`. ● It is associated with a venue through a foreign key `venue_id` (referencing the Venue model).
● It is associated with bookings made for the show through a one-to-many relationship.
● The `bookings` relationship represents that each show can have multiple bookings.
● The `lazy` parameter is set to "True" for the `bookings` relationship, indicating that the related data should be loaded lazily using subqueries.

5. Booking Model

● This model represents bookings made by users for specific shows.
● It has attributes: `booking_id` (primary key), `user_id` (foreign key referencing the User model), `show_id` (foreign key referencing the Show model), and `tickets`.
● It stores the number of tickets booked by a user for a particular show.

Front-end

Project was setup with Vue-cli through its UI.App instance is create with main.js importing all dependencies.Home view is created with conditional rendering of Home component page with Login/Register option and Dashboard Option with NavBar. The Dashboard Component acts as base component. Action for different components are as follows:

- **Admin Dashboard:** can create, edit and Delete venues and add as well as edit and delete shows in the said venues. Exporting is available for each venue. Admin has to book
- **Admin Summary:** Displays count of how many venues and shows are currently being managed by the admin. Shows a table of expired shows that can no longer be edited or deleted.
- **User Dashboard:** User can see all the different shows that are available for booking. Expired shows are not displayed to the user.
- **User Bookings:** Displays all the bookings made by a user in the present and in the past. The user can cancel their booking if they wish to.
- **Search Component:** Displays the results of search made by the user. Can book the tickets to the searched shows directly from the search page.

Backend

Based on Flask, it provides an API endpoint for the fetch request to the frontend actions. It also validates the input data received from frontend form before committing to the database. It also performs performance improvement jobs through caching and synchronous jobs through celery workers. It runs on localhost-port 8081. To run synchronous tasks celery workers and celery beat in parallel need to be run on the terminal. Hogmail is used as a fake SMTP email receiver. Also for these tasks redis server needs to be run in background.