MAD-2 Ticket-Show Project: (Note: Functionality Checklist is included.)

Author:

Name: Priyadarsh Singh

Roll No: 21f1005104 ; Student Email: 21f1005104@ds.study.iitm.ac.in

About Me: Myself Priyadarsh Singh, apart from being a Diploma Level student at IIT Madras BS Degree, I'm a graduate in Statistics and currently working as a Data Science Intern at Culinda Ai. My keen interest lies in the domain of ML, AI and Computer Vision & I'm looking forward to working in the respective domain.

Description

This Ticket-Show project is a movie ticket booking platform where the users can register, login and then book movie tickets for the movies which are currently running in various theatres at various locations as well as rate the movies watched via my-profile section on the platform.

Technologies Used

1. Flask – for API.
2. VueJS CLI – for UI.
3. Bootstrap.
4. SQLite and SQLAlchemy for data storage.
5. Redis and Celery for Batch Jobs.
6. Jinja2 templates for Sending emails.

DB Schema Design

Four tables: **user, theatre, show, bookin**g was created –

TABLE "user" ("id" INTEGER, "username" TEXT NOT NULL UNIQUE, "password" TEXT NOT NULL, "is_admin" INTEGER NOT NULL, "email" TEXT NOT NULL, PRIMARY KEY ("id" AUTOINCREMENT))

Purpose: - Stores the data related to each and every user who has created account on Ticket-Show application. It also contains admin data.

TABLE "theatre" ("id" INTEGER, "name" TEXT NOT NULL UNIQUE, "location" TEXT NOT NULL, "place" TEXT NOT NULL, "capacity" INTEGER NOT NULL, PRIMARY KEY ("id" AUTOINCREMENT))

Purpose: - Stores the data related to the theatres which are in various cities.

TABLE "show" ("id" INTEGER, "showname" TEXT NOT NULL, "ratings" NUMERIC, "tags" TEXT NOT NULL, "price" INTEGER NOT NULL, "theatre_id" INTEGER, "time" TEXT NOT NULL, "counts" INTEGER NOT NULL DEFAULT 0, "capacity" INTEGER NOT NULL, FOREIGN KEY("theatre_id") REFERENCES "theatre"("id"), PRIMARY KEY ("id" AUTOINCREMENT))

Purpose: - Stores the data related to the shows which are currently running in various theatres.

TABLE "booking" ("id" INTEGER, "theatre_id" INTEGER NOT NULL, "show_id" INTEGER NOT NULL, "count" INTEGER NOT NULL, "user_id" INTEGER NOT NULL, PRIMARY KEY ("id" AUTOINCREMENT), FOREIGN KEY("show_id") REFERENCES "show"("id"), FOREIGN KEY("user_id") REFERENCES "user"("id"), FOREIGN KEY("theatre_id") REFERENCES "theatre"("id"))

Purpose: - Stores the data related to the movie tickets being booked by various users of various shows at respective theatres.

API Design:

The API is implemented using the Flask framework, and it provides endpoints to interact with these entities. The API includes features such as user signup, login, theater and show creation, booking shows, searching for theaters and shows, and generating summary reports.

Endpoints & Functionalities:

1. **Default Route for Testing**
    - Endpoint: **/**
    - Method: **GET**
    - Description: A default route to check if the API is running.
2. **User Signup**
    - Endpoint: **/api/usersignup**
    - Method: **POST**
    - Description: Allows users to sign up by providing a username, password, and email. Validates email format and stores user details in the database.
3. **User Login**
    - Endpoint: **/login**
    - Method: **POST**
    - Description: Authenticates users and generates an access token for authorized access. Requires a username and password.
4. **Admin Dashboard**
    - Endpoint: **/admin/dashboard**
    - Method: **GET**
    - Description: Provides an admin dashboard view, accessible only to admin users.
5. **User Dashboard**
    - Endpoint: **/user/dashboard**
    - Method: **GET**
    - Description: Provides a user dashboard view, accessible only to normal users.
6. **Create Theater**
    - Endpoint: **/createtheatre**
    - Method: **POST**
    - Description: Allows admin users to create new theaters by providing details like name, place, location, and capacity.
7. **Edit Theater**
    - Endpoint: **/edittheatre**
    - Method: **GET**, **PUT**, **DELETE**
    - Description: Allows admin users to retrieve, update, or delete theater details based on the theater's ID.
8. **Get All Theaters**
    - Endpoint: **/getalltheatre**
    - Method: **GET**
    - Description: Retrieves details of all theaters created, including theater name, place, location, and capacity.
9. **Create Show**
    - Endpoint: **/createshow**
    - Method: **POST**
    - Description: Allows admin users to create new shows by providing details like show name, ratings, tags, price, and time.
10. **Edit Show**
    - Endpoint: **/editshow**
    - Method: **GET**, **PUT**, **DELETE**
    - Description: Allows admin users to retrieve, update, or delete show details based on the show's ID.
11. **Get Shows**
    - Endpoint: **/getshows**
    - Method: **GET**
    - Description: Retrieves details of all shows running in a specific theater. Shows that haven't started yet on the current day are included.
12. **Book Show**
    - Endpoint: **/bookshow**
    - Method: **POST**
    - Description: Allows normal users to book shows by providing theater ID, show ID, and the number of tickets to book.
13. **Get Bookings**
    - Endpoint: **/getbookings**
    - Method: **GET**
    - Description: Retrieves booking details for the current user, including the show name, theater name, and timing.
14. **Rate Show**
    - Endpoint: **/rateshow**

- Method: **PUT**
- Description: Allows normal users to rate shows they have attended.

15. **Search Theaters**
    - Endpoint: **/search**
    - Method: **GET**
    - Description: Allows users to search for theaters based on a query string (e.g., theater name, location, place).
16. **Search Shows**
    - Endpoint: **/searchshows**
    - Method: **GET**
    - Description: Allows users to search for shows based on a query string (e.g., show name, tags, ratings).
17. **Generate Summary**
    - Endpoint: **/summary**
    - Method: **GET**
    - Description: Generates summary reports for theaters and shows, including the number of bookings made.
18. **Export Theater Report (CSV)**
    - Endpoint: **/export**
    - Method: **POST**
    - Description: Allows admin users to export a CSV report containing information about a specific theater's shows and bookings.

Architecture & Features: (Features Implemented Checklist)

1. **User Signup & Login:**
   - ✓ Form for username & password (both login & signup) & used JWT Token based authentication.
   - ✓ Suitable model for user.
2. **Admin Login (Using RBAC):**
   - ✓ Form for username & password.
   - ✓ Admin user created whenever new database is created & App differentiates between a normal user & admin.
3. **Theatre Management (Only for Admin):**
   - ✓ Create a new theatre, Edit a theatre, Remove a theatre.
4. **Show Management (Only for Admin):**
   - ✓ Create a new show, Edit a show, Remove a show, Allocate theatres while creating shows.
   - o Varied pricing for each theatre(optional).
5. **Search for Shows/Theatres:**
   - ✓ Search theatres based on location preference, Search movies based on tags, ratings etc., Basic home view for a theatre.
6. **Book Show tickets:**
   - ✓ List shows within available timeframe to users, Ability to book multiple tickets at a given theatre.
   - ✓ Ability to stop booking in case of Housefull.
7. **Daily Reminder Jobs:**
   - ✓ Scheduled Job – Daily reminders using Email on Mailhog.
8. **Monthly Entertainment Report:**
   - ✓ Entertainment report (HTML) via Email on Mailhog.
9. **Export CSV (Admin Triggered Job):**
   - ✓ A csv file is generated for respective theatre information.
10. **Performance & Caching:**
    - ✓ Added Caching, Added Cache Expiry.
11. **Recommended (Graded):**
    - ✓ PDF Reports attached in Mails (Entertainment Report).
12. **Styling and Aesthetics(done).**

Video Link: Click Here to watch the Summary video of MAD2- TicketShow Project