Ticket Show V2

Problem definition

Modern Application Development - II

Last Updated: 23-04-28

Frameworks to be used

- Flask for API
- VueJS for UI
- VueJS Advanced with CLI (only if required, not necessary)
- Jinja2 templates if required
    - Not to be used for UI
- Bootstrap, if required (No other CSS framework is allowed)
- SQLite for database (No other database is allowed)
- Redis for caching
- Redis and Celery for batch jobs
- It should be possible to run all the demos on the student's computer, which should either be a Linux based system or should be able to simulate the same. You can use WSL for Windows OS.

Ticket Show

- It is a multi-user app (one required admin and other users)
- Used for booking show tickets
- User can book many tickets for many movies
- Admin can create theatres and shows
- Each theatre will have
  - ID
  - Name
  - Place
  - Capacity etc.
- Each show will have
  - ID
  - Name
  - Rating
  - Tags
  - TicketPrice etc.
- Every theatre can run a number of shows
- System will automatically show the latest added shows

Terminology

- Ticket Booking Platform
- Theatre - List of shows, capacity etc.
- Show - Name, Rating, Price etc.
- Dynamic Pricing (optional) - Show prices can go up/down depending upon the popularity

Wireframe: Ticketshow

Similar Products in the Market:

1. BookMyShow
    - Web, IOS and Android
2. TicketNew
    - Web, IOS and Android

- These are meant for exploring the idea and inspiration
- Don’t copy, get inspired

Core Functionality

- This will be graded
- Base requirements:
  - User signup and login
  - Mandatory Admin Login (using RBAC)
  - Theatre and Show Management
  - Booking show tickets
  - Search for shows/theatres
- Backend Jobs
  - Export Jobs
  - Reporting Jobs
  - Alert Jobs
- Backend Performance

Core - User Signup and Login

- Form for username and password (both login and signup)
- Use Flask Security or JWT based Token Based Authentication only
- Suitable model for user

Core - Admin Login (Using RBAC)

- Form for username and password (can be same as normal users)
- Add an admin user whenever a new database is created
- The app should differentiate between an admin and a normal user

Core - Theatre Management (Only for Admin)

- Create a new theatre
  - Storage should handle multiple languages - usually UTF-8 encoding is sufficient for this
- Edit a theatre
  - Change title/caption or image
- Remove a theatre
  - With a confirmation from the admin

Core - Show management (Only for Admin)

- Create a new show
  - Storage should handle multiple languages - usually UTF-8 encoding is sufficient for this
- Edit a show
  - Change title/caption or image
- Remove a show
  - With a confirmation from the admin
- Allocate theatres while creating shows
- Varied pricing for each theatre (optional)

Core - Search for Shows/Theatres

- Ability to search theatres based on location preference
- Ability to search movies based on tags, rating etc.
- Basic home view for a theatre

Core - Book Show Tickets

- List the shows available for a given timeframe to the users
- Ability to book multiple tickets for a show at a given theatre
- Ability to stop taking bookings in case of a houseful.

Core - Daily Reminder Jobs

- Scheduled Job - Daily reminders on Google Chat using webhook or SMS or Email
  - In the evening, every day (you can choose time of your choice)
  - Check if the user has not visited/booked anything
  - If yes, then send the alert asking them to visit/book

Core - Scheduled Job - Monthly Entertainment Report

- Scheduled Job - Monthly Entertainment Report
  - Come Up with a monthly progress report in HTML (email)
  - The entertainment review report can consist of bookings made by a user in a given month, shows seen, ratings for the shows etc.
  - On the first day of the month
    - Start a job
    - Create a report
    - Send it as email

Core - User Triggered Async Job - Export as CSV (Only for Admin)

- User Triggered Async Job - Export as CSV
  - Come up with an export CSV format for theatres
  - The export is meant for a single theatre (at a time) to export details like number of shows, bookings, rating etc.
  - Have a dashboard where the user can export
  - Trigger a batch job, send an alert once done

Core - Performance and Caching

- Add caching where required to increase the performance
- Add cache expiry
- API Performance

Recommended (graded)

- Well designed PDF reports (User can choose between HTML and PDF reports)
- Single Responsive UI for both Mobile and Desktop
  - Unified UI that works across devices
  - Add to desktop feature

Optional

- Styling and Aesthetics
- Predict popularity of a show/venue based on the previous trends

Evaluation

- Report (not more than 2 pages) describing models and overall system design
  - Include as PDF inside submission folder
- All code to be submitted on portal
- A brief (2-3 minute) video explaining how you approached the problem, what you have implemented, and any extra features
  - This will be viewed during or before the viva, so should be a clear explanation of your work
- Viva: after the video explanation, you are required to give a demo of your work, and answer any questions
  - This includes making changes as requested and running the code for a live demo
  - Other questions that may be unrelated to the project itself but are relevant for the course

Instructions

- This is a live document and will be updated with more details and FAQs (possibly including suggested wireframes, but not specific implementation details) as we proceed.
- We will freeze the problem statement on or before 14th May, beyond which any modifications to the statement will be communicated via proper announcements.
- The project has to be submitted as a single zip file.
- Please refer to the guidelines document available on the portal.