Full Stack Development with MERN

Project Documentation

1. Introduction

- **Project Title: iMovies - Movie Ticket Booking System**
- **Team Members:** This project was developed by a team of four Computer Science students, each taking on roles relevant to full stack development:

1. **Sanskriti Tyagi** – *Team Leader & Frontend Developer:* Oversaw team coordination, UI design, and implementation of user-facing features using React.

2. **Khushi Thakur** – *UI/UX Designer & Tester:* Crafted the user experience and interface flow, ensuring usability. Also managed testing and error handling for both frontend and backend.

3. **Eishani Bhattacharya** – *Backend Developer:* Responsible for creating RESTful APIs using Express, implementing business logic, and managing user authentication and seat booking logic.
4. **Anushka Singh** – *Database Engineer:* Designed the MongoDB schemas and handled all CRUD operations related to bookings, shows, and user data.

2. Project Overview

- **Purpose:** In the modern digital era, users prefer online solutions over manual processes. The **eMovie Ticket Booking System** addresses this need by providing a complete online solution for booking movie tickets. It allows users to browse movie listings, view showtimes, choose seats from a visual layout, and complete their booking process online — eliminating the need to physically stand in lines or call cinemas. The system also supports administrative controls for managing movies and schedules.

- **Goals of the Project:**

1. Create an intuitive user interface that simplifies ticket booking.

2. Allow real-time seat availability updates.
3. Provide authentication and secure data handling.
4. Enable an admin interface to manage backend movie data dynamically.
5. Offer a responsive application that works on both desktop and mobile devices.

- **Key Features:**

1. **User Authentication**: Secure login and registration using encrypted credentials and JWT tokens.
2. **Movie Listings**: Dynamic movie listings with filters such as language, genre, and ratings.
3. **Showtime Selection**: Users can view showtimes per movie and choose the preferred slot.
4. **Interactive Seat Layout**: A visual layout shows available and booked seats. Users can select multiple seats and proceed to book.
5. **Ticket Booking**: Confirmed seat bookings are stored in the database and displayed in the user's history.
6. **Payment Integration**: Simulated payment interface to mimic real-world flow.
7. **Booking History**: Logged-in users can view, download, and cancel past bookings.
8. **Admin Dashboard**: Role-based admin control to add/edit/delete movies and show timings.

3. Architecture

- **Frontend (React.js):** The frontend was built using React.js for its component-based architecture, fast rendering with a virtual DOM, and excellent developer experience. The UI was designed using Material UI and Bootstrap for responsiveness. Navigation across pages (Home, Login, Register, Movie Details, Seat Booking) is handled using React Router DOM. Axios was used for interacting with the backend API.

Components like MovieCard, Navbar, SeatLayout, BookingSummary, and AdminPanel were created for modular development. Form validations and loading indicators were added for better UX.

- **Backend (Node.js & Express.js):** The backend is a RESTful API built with Node.js and Express.js. It handles all business logic including user registration/login, seat reservation, booking confirmation, payment simulation, and admin-level controls.

Key backend modules include:

- **Auth Controller**: Manages JWT-based login and registration
- **Booking Controller**: Handles seat selection, validation, and booking logic
- **Admin Controller**: Manages admin privileges and movie/show management
- **Middleware**: Verifies tokens, checks roles, and handles errors globally

All APIs are protected by token-based authentication and cross-verified with middleware functions.

- **Database (MongoDB + Mongoose):** MongoDB, a NoSQL document database, stores all persistent data. Collections include:

  Users: Stores user credentials, roles, and metadata

  Movies: Each movie document includes title, description, poster URL, and genre

  Shows: Tied to a movie ID and includes screen, showtime, and availability

  Bookings: Contains user ID, selected seats, payment status, and booking time

Mongoose schemas were used to define models and enforce data validation. Relationships were managed using ObjectId references.

4. Setup Instructions

- **Prerequisites:**
  - Node.js (version 18+)
  - MongoDB (local installation or MongoDB Atlas cloud)

- Git (for cloning the repository)
- npm (Node Package Manager)

- **Installation Steps:**
  1. Clone the project repository:
     bash:
     git clone https://github.com/your-username/eMovieBookingSystem.git

     cd eMovieBookingSystem

  2. Install dependencies for both client and server:

     bash:
     cd client

     npm install

     cd ../server

     npm install

  3. Configure environment variables:

     server/.env:

     ini:
     PORT=5000
     MONGO_URI=mongodb+srv://your_credentials
     JWT_SECRET=supersecret

     client/.env:

     bash:
     REACT_APP_API_URL=http://localhost:5000/api

  4. Start the application:
     Server:

     bash:
     cd server

     npm start

Client:

bash:
cd client
npm start

5. Folder Structure

- **Client Folder (React Frontend):**

  bash:

  ```
  client/
  ├── public/
  ├── src/
  │   ├── components/
  │   ├── pages/
  │   ├── context/
  │   ├── services/
  │   └── App.js
  └── .env
  ```

- **Server Folder (Node.js Backend):**

  bash:

  ```
  server/
  ├── controllers/
  ├── models/
  ├── routes/
  ├── middleware/
  ├── config/
  ├── server.js
  └── .env
  ```

6. Running the Application

To successfully run the application in your local environment, two terminal instances are required — one for the frontend (React) and another for the backend (Express + Node.js). MongoDB must also be running in the background (either locally or via MongoDB Atlas).

- **Start the Backend Server:**

  bash:
  cd server
  npm start

  The backend will start on port **5000** (or as defined in .env). It serves the API endpoints that power the functionality of user login, movie listings, seat selection, booking, etc.

- **Start the Frontend React App:**

  bash:
  cd client
  npm start

  The React frontend starts on **http://localhost:3000**, providing users access to the booking interface.

  Once both servers are running:

  - Users can navigate the site, browse movies, login/register, and make bookings.
  - Admins can access their dashboard using proper credentials and manage the database.

7. API Documentation

The backend exposes a set of RESTful API endpoints, segregated based on functionality (auth, movies, shows, bookings, admin). All user-specific and admin-specific routes are protected using JWT authentication middleware.

1. **Auth Routes**

Method	Endpoint	Description	Auth Required
POST	/api/auth/register	Registers a new user	No
POST	/api/auth/login	Logs in and returns JWT token	No

2. **Movie & Show Routes**

Method	Endpoint	Description	Auth Required
GET	/api/movies	Retrieves list of all movies	No
GET	/api/shows/:id	Retrieves shows for a selected movie	No

3. **Booking Routes**

Method	Endpoint	Description	Auth Required
POST	/api/bookings	Books seats for a show	Yes
GET	/api/bookings/user	Retrieves user's booking history	Yes
DELETE	/api/bookings/:id	Cancels a booking	Yes

4. **Payment Route**

Method	Endpoint	Description	Auth Required
POST	/api/payment	Simulates a payment	Yes

5. **Admin Route**

Method	Endpoint	Description	Auth Required
GET	/api/admin/dashboard	Fetches admin dashboard statistics	Yes (Admin only)

8. Authentication

Authentication in this project is handled using **JWT (JSON Web Token)**, a secure method for transmitting information between parties. It includes:

- **Login Flow:**
    - User logs in with email and password.

    - Server verifies credentials, generates a token.
    - Token is sent to the client and stored in localStorage.
- **Token Usage:**
    - On future requests, the token is added to the HTTP Authorization header.
    - Backend middleware checks the validity of the token and grants/denies access.
- **Role-Based Access:**
    - Users have a role field (either user or admin).
    - Admin routes validate this role before granting access.
- **Password Security:**
    - Passwords are hashed using **bcrypt** before storage.
    - Passwords are never stored or sent in plain text.

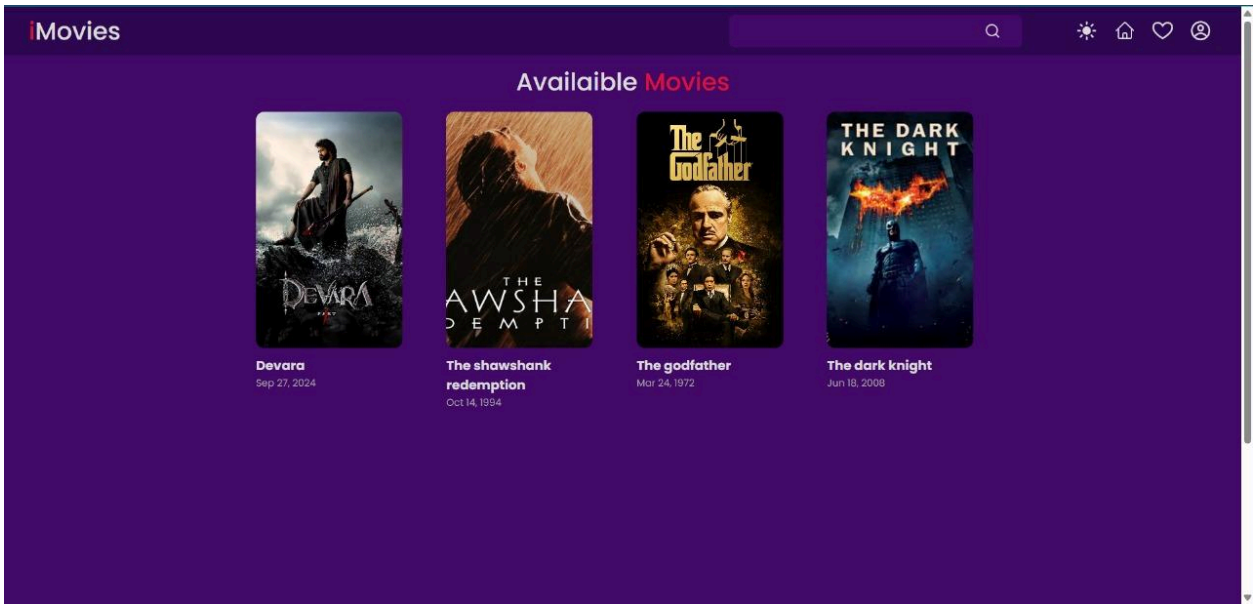
This structure ensures robust protection of data and enforces access rules without compromising user experience.

9. User Interface

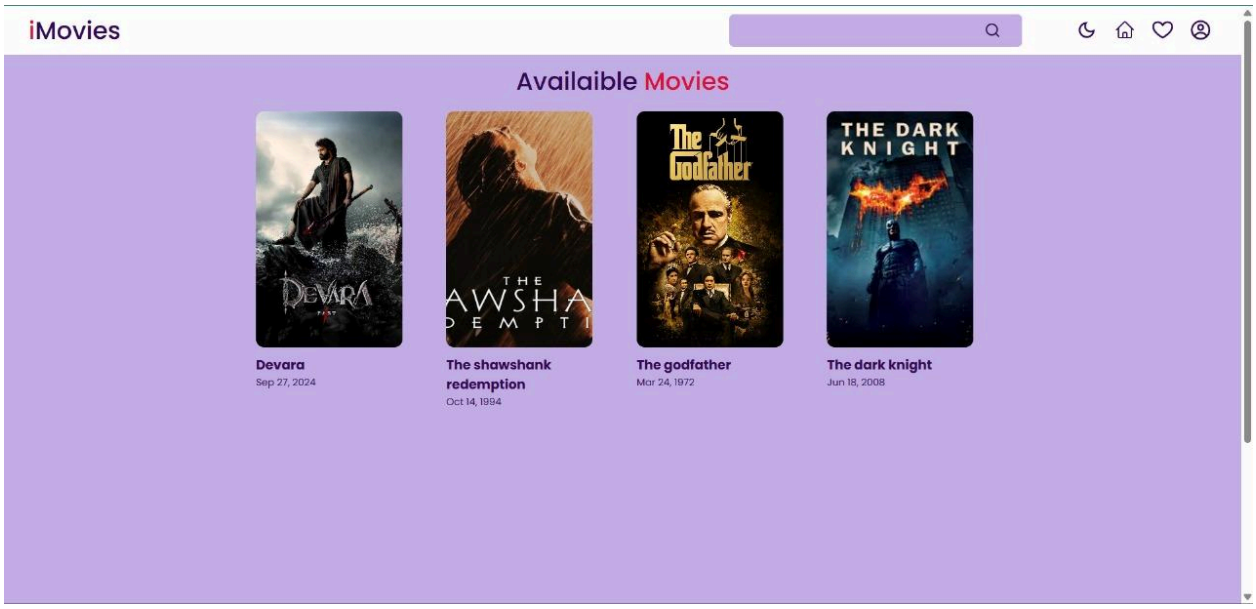
The UI is designed for simplicity, responsiveness, and clarity — making it suitable for both desktop and mobile users. The interface focuses on:

- **Modern Aesthetics:** Styled using **Material UI**, offering consistent spacing, buttons, and color schemes.
- **User Flow:** Clear navigation from browsing → selecting → booking → confirming.
- **Forms:** All forms (login, register, seat booking) are validated in real-time.
- **Feedback Mechanisms:** Toast notifications alert users to actions like "Booking Confirmed", "Login Successful", or "Error Occurred".

- **Interactive Seat Layout:** Clicking a seat toggles its selection; already booked seats are grayed out.



Dark Mode



Light Mode

Screens include:

- Home Page (movie listings)
- Movie Detail Page (showtimes)
- Booking Page (seat selection and confirmation)
- Admin Dashboard (for movie/show management)
- User Dashboard (booking history and cancelation)

10. Testing

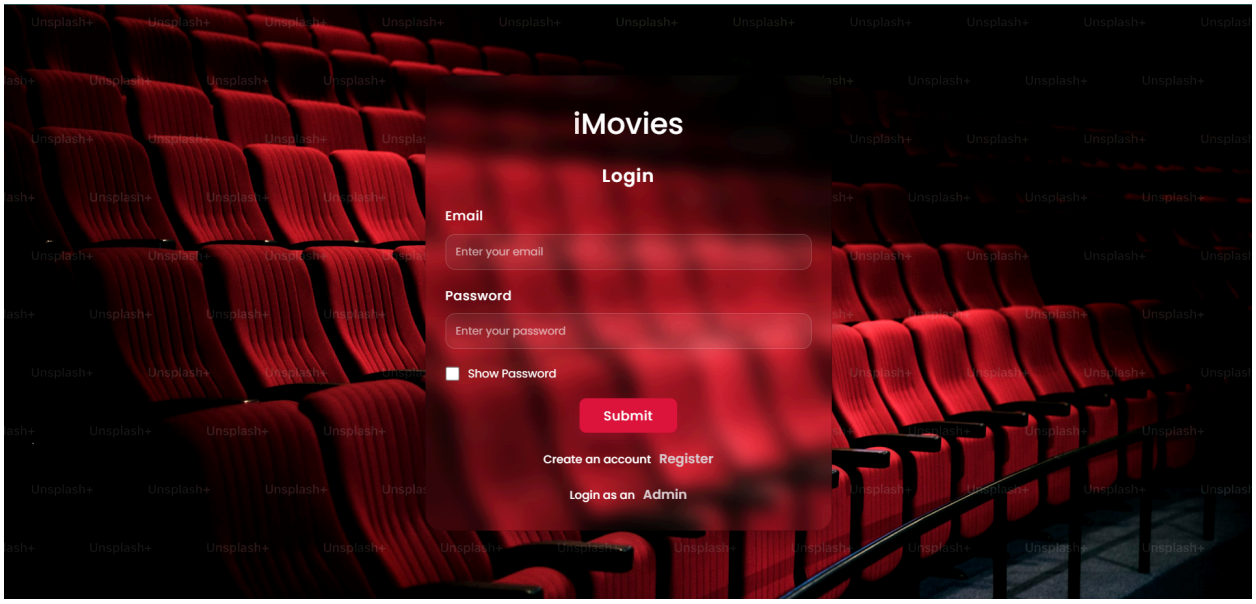
Testing was conducted in two stages: **manual testing** and **API endpoint testing**.

1. **Manual Testing:**
   - Each screen was manually tested for responsiveness, field validation, and performance.
   - Tested on different screen sizes to ensure mobile-friendliness.

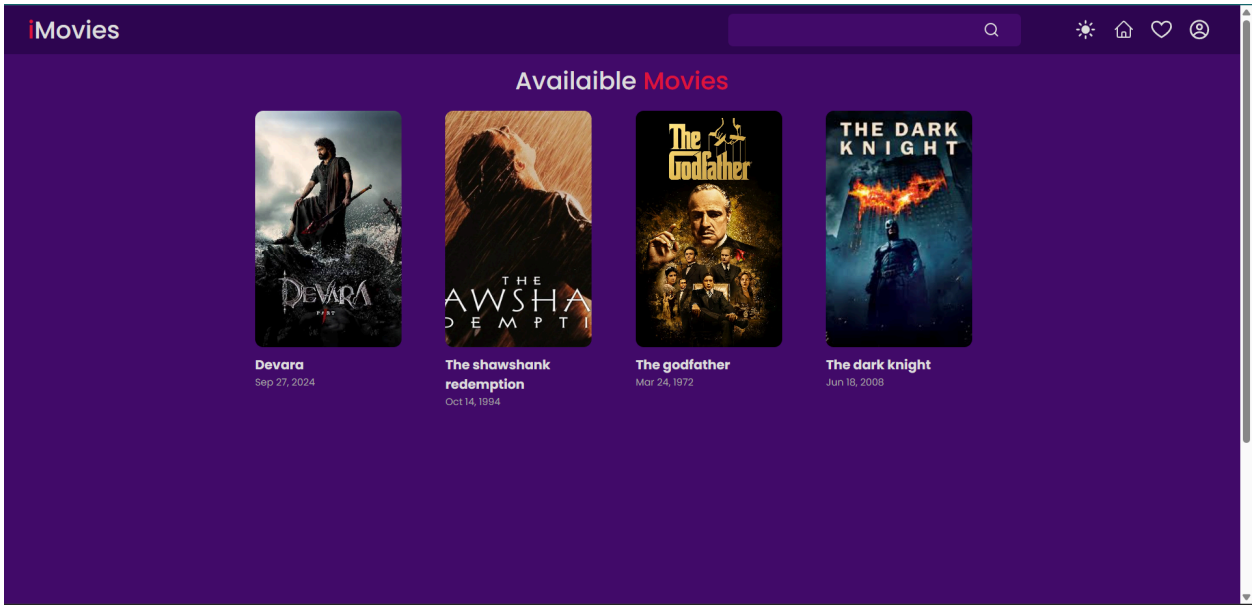
2. **API Testing:**
   - **Postman** was used to test API routes individually.
   - Routes were tested with and without tokens to verify authentication logic.
   - Edge cases (like double booking the same seat) were tested to ensure logic correctness.

3. **Future Plans:**
   - Implement **unit testing** using Jest and React Testing Library.
   - Add **integration tests** for user journeys like booking → confirming → viewing history.

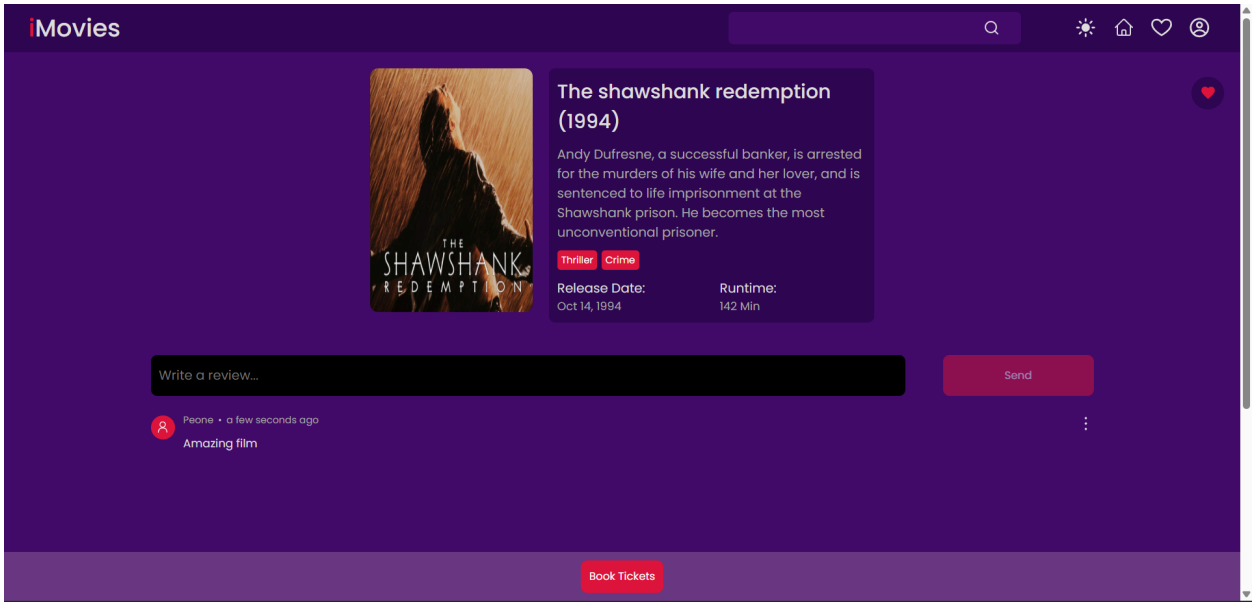
11. Screenshots or Demo



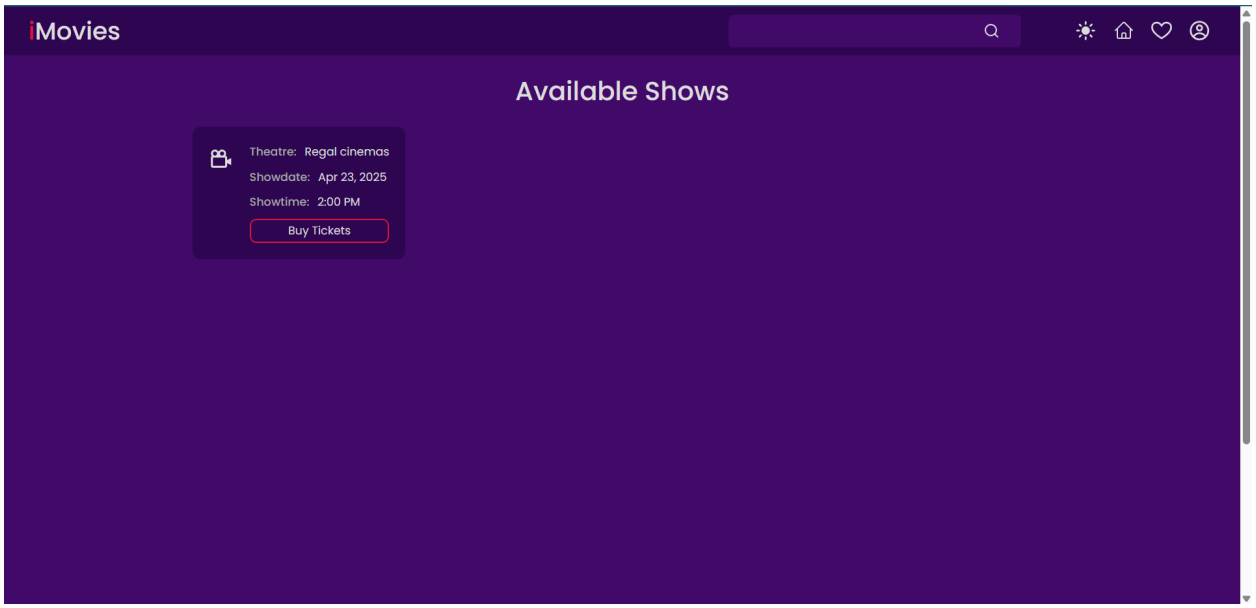
Login as a User



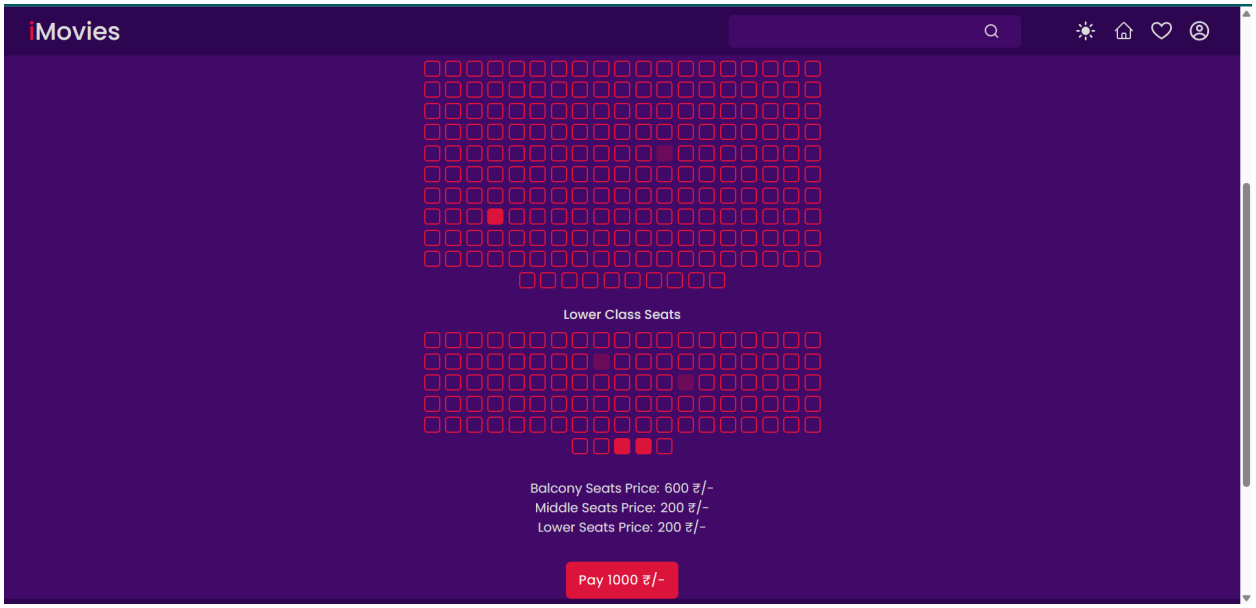
Available Movies List



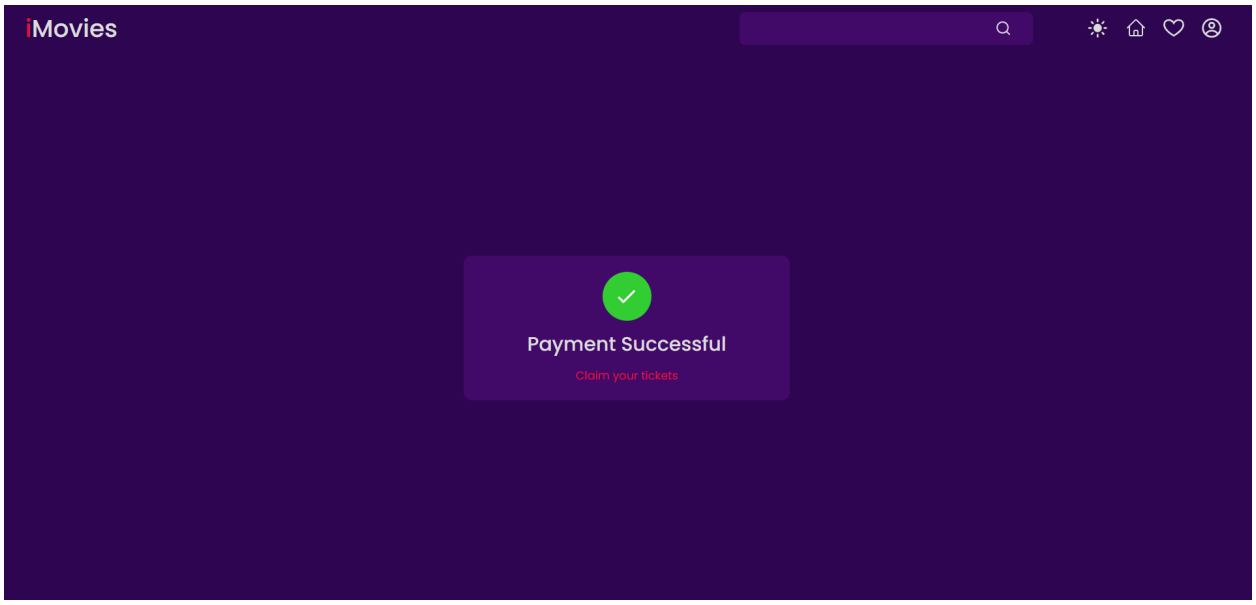
Movie Description and Review



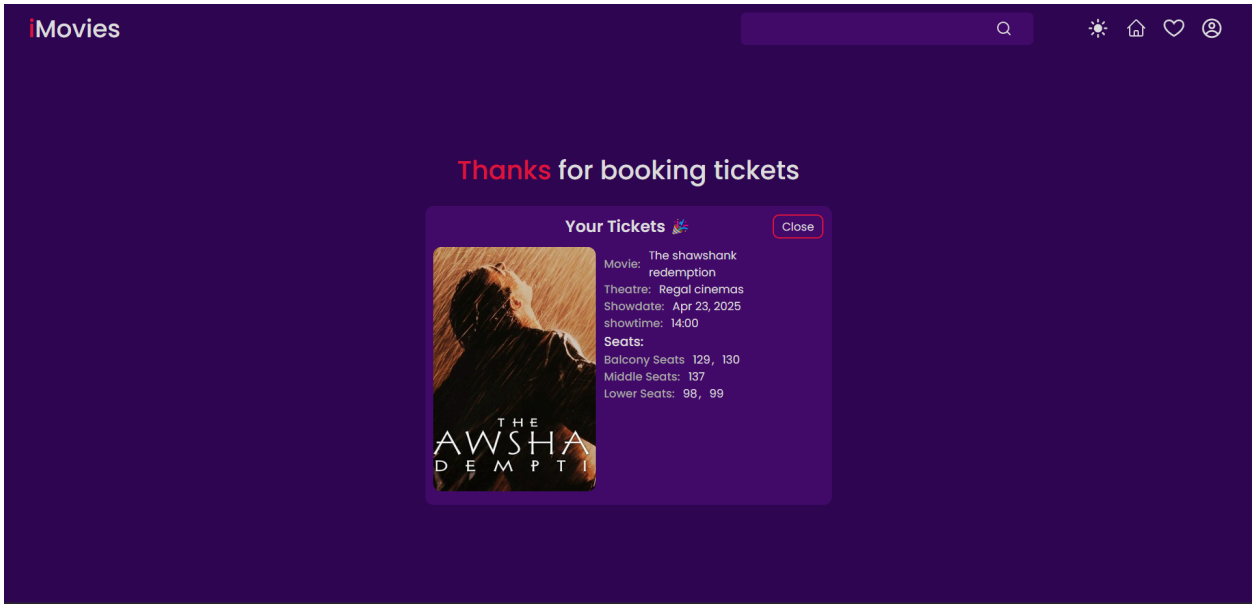
Available Shows and Details



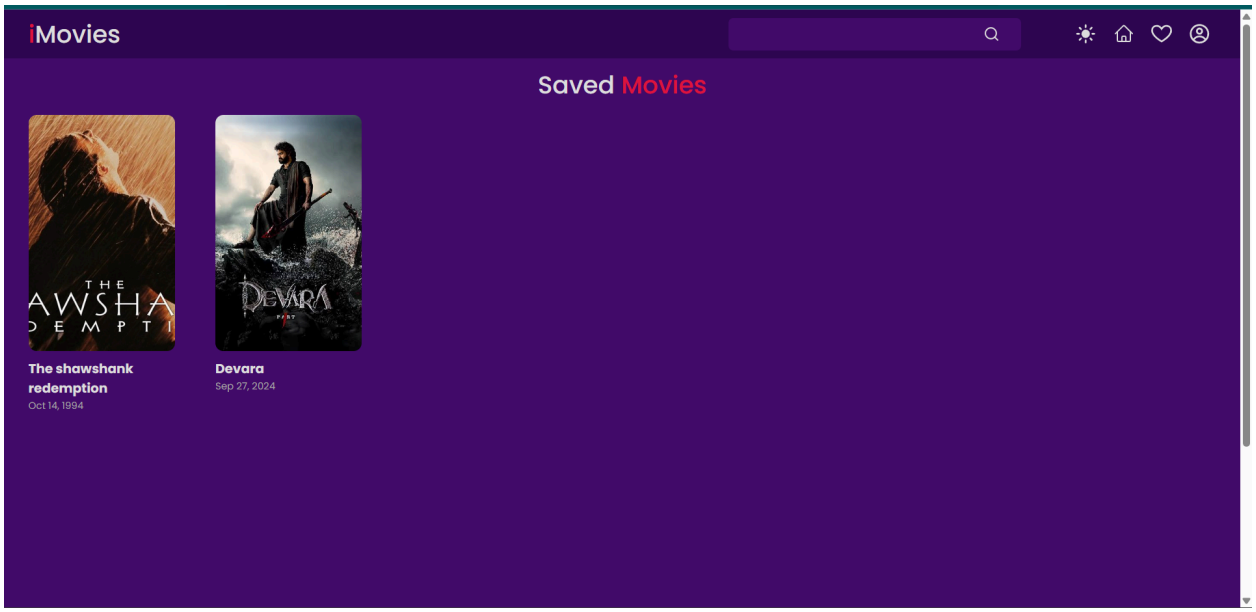
Seat Selection and Payment Option



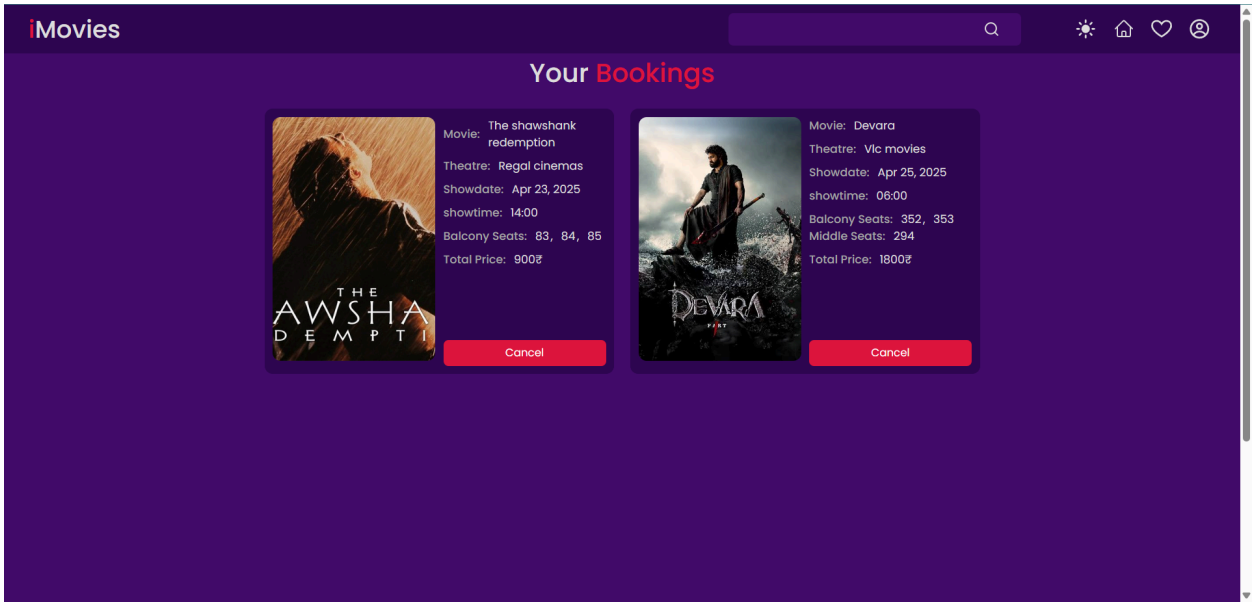
Payment Confirmation



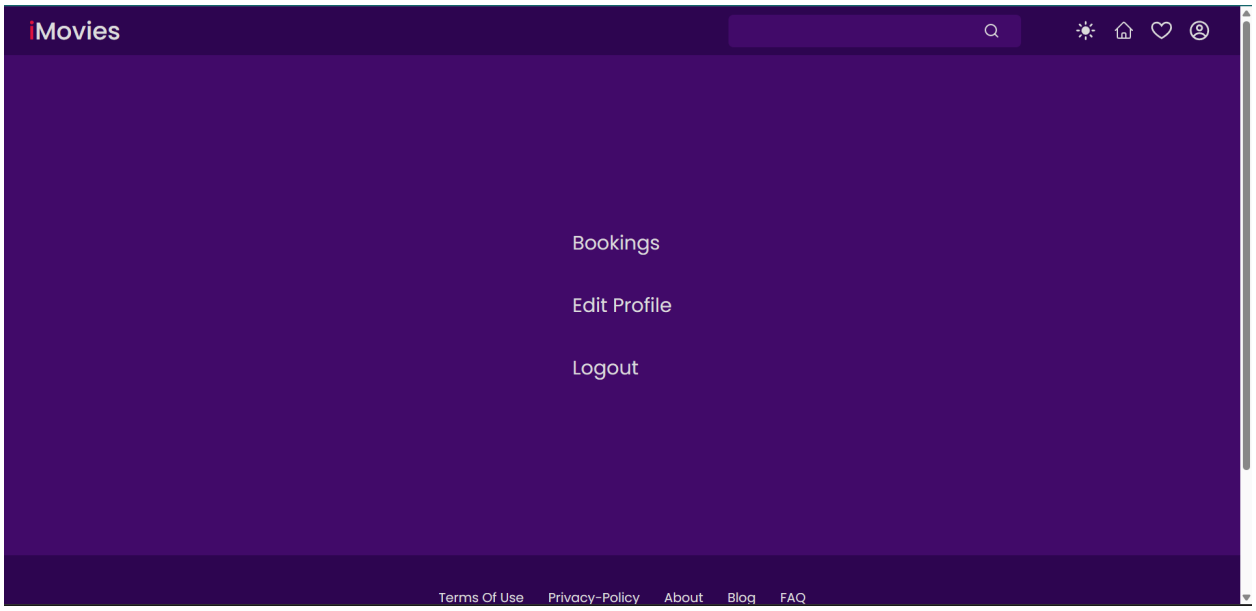
Booking Confirmation and Ticket Details



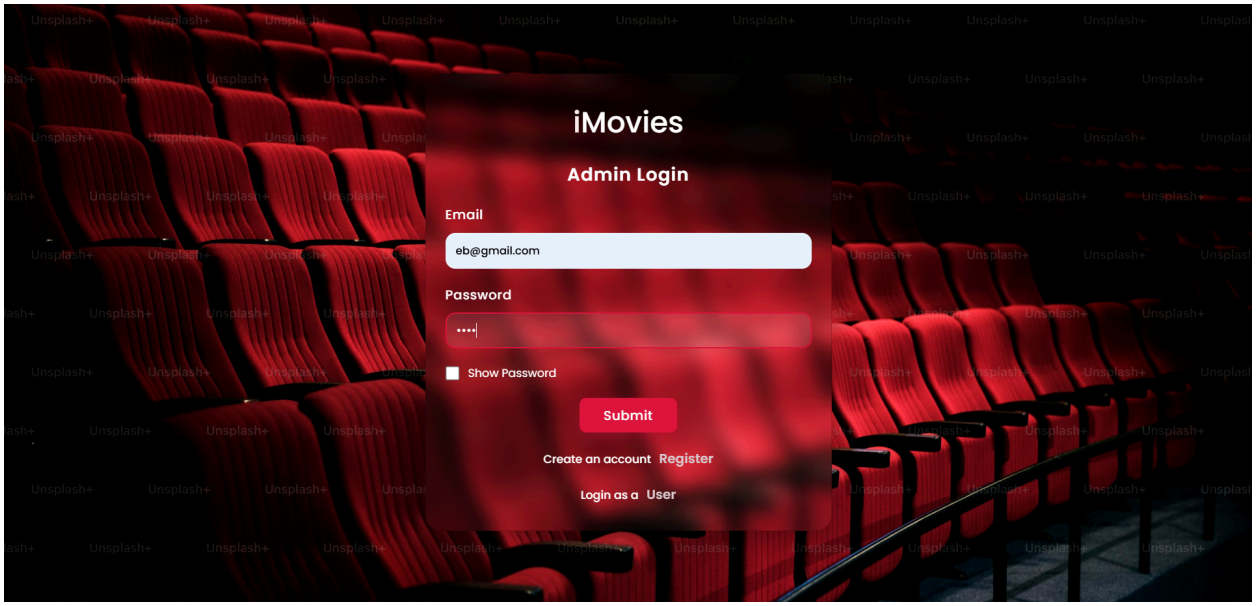
Movies Saved in Favourites



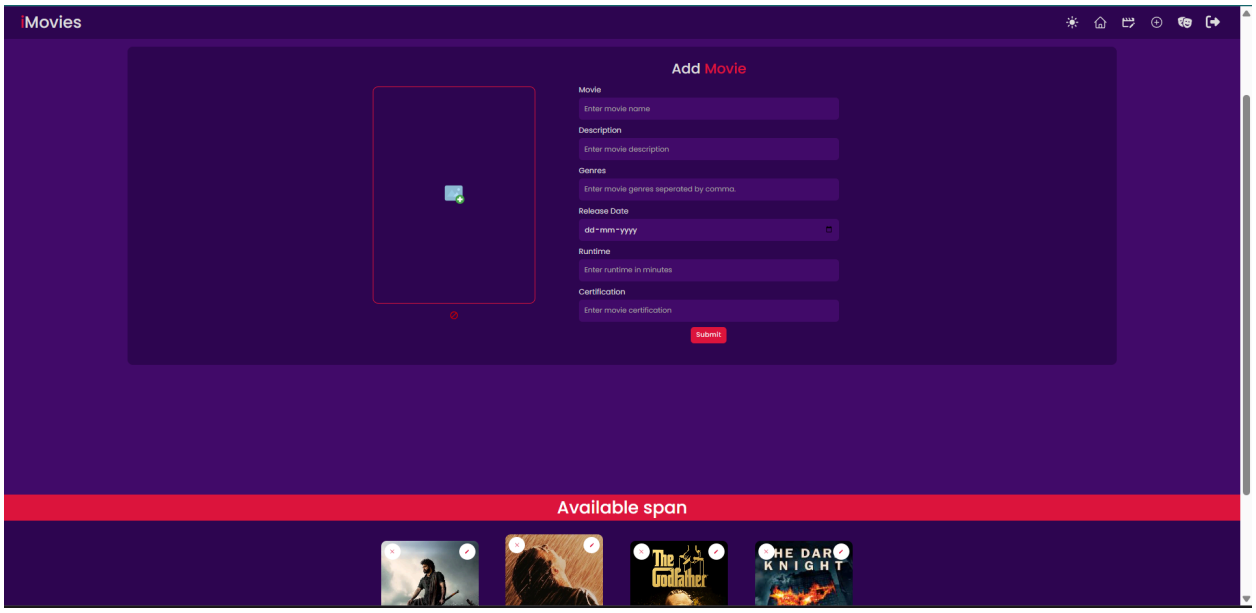
Booking already made by the User



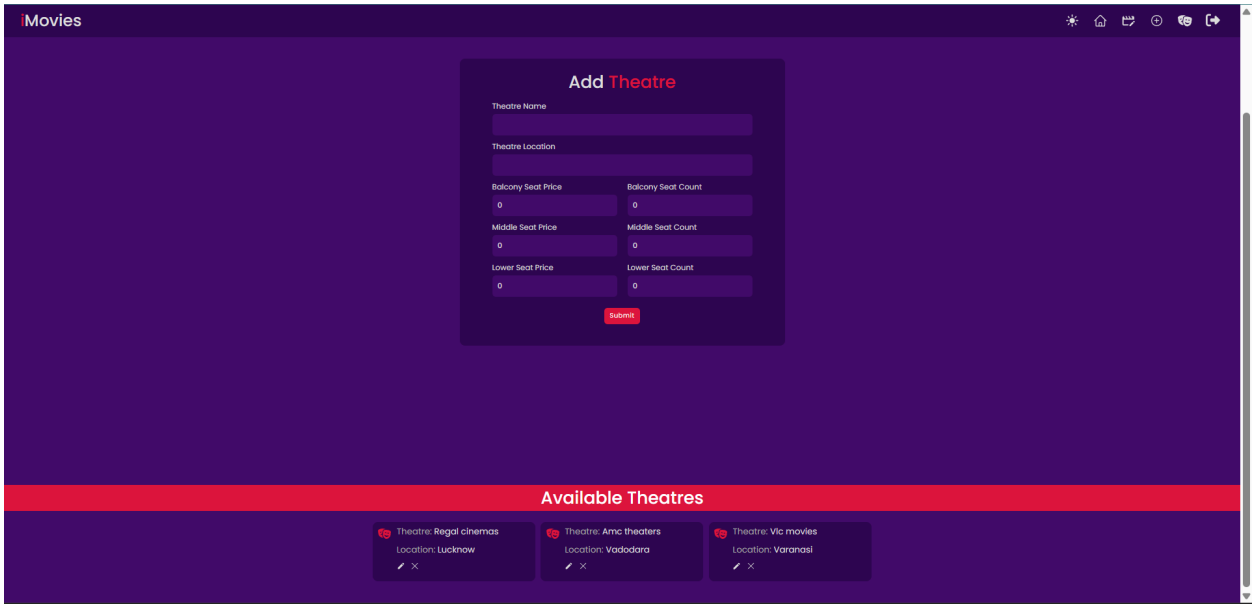
Account section



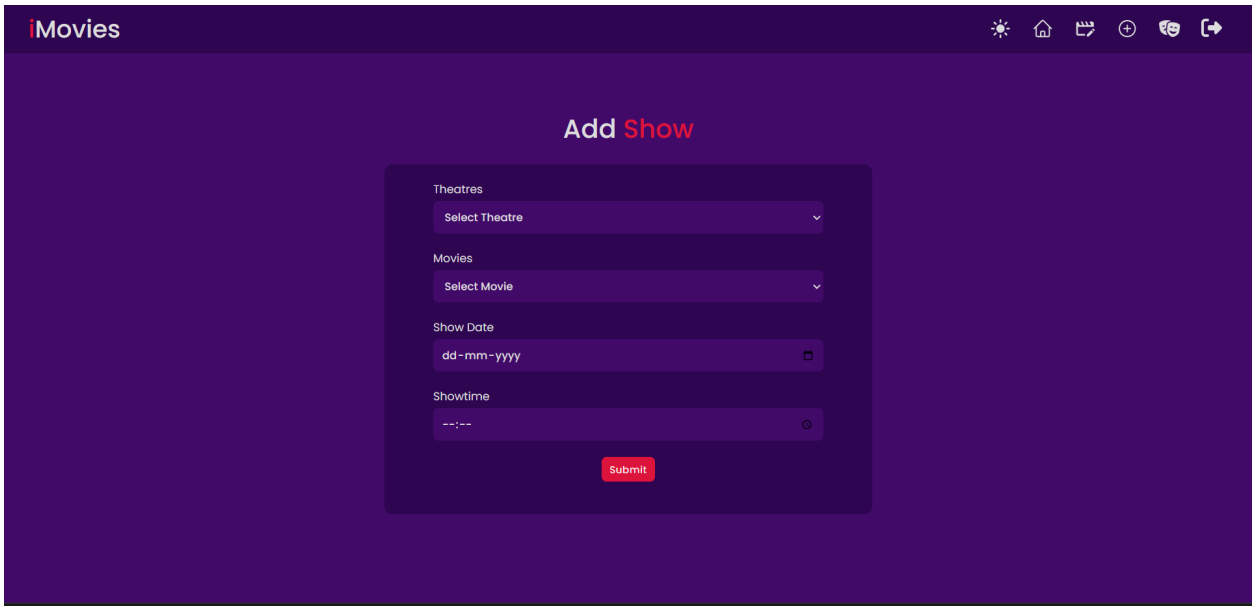
Login as an Admin



Add Movie and Movie Details



Add new Theater details or edit previous theaters



Add Shows

Demo Link:
https://drive.google.com/file/d/1eltmtYi-42D06kYSyD4nfQZt32fjrnRD/view?usp=sharing

12. Known Issues

Although the application is functional, there are a few areas identified for improvement:

- **No Real Payment Gateway:** The payment is currently simulated for demonstration purposes.
- **No Email Notifications:** Booking confirmations and cancellation alerts are shown only in-browser.
- **Static Seat Syncing:** Seat availability does not auto-update in real-time if two users book simultaneously.
- **Limited Admin Analytics:** No charts or dashboards for visual insights into bookings or revenue.

13. Future Enhancements

To make the project production-ready and more robust, we plan the following enhancements:

- **Integrate OTP & 2-Factor Authentication** for enhanced security.
- **Mobile App Version** using React Native to allow ticket booking on the go.
- **Real Payment Gateway Integration** like Razorpay or Stripe.
- **Email Notifications** for ticket confirmation, cancellation, and reminders.
- **Admin Analytics Dashboard** using Chart.js for bookings, income trends, etc.
- **QR Code Ticketing System** for scanning tickets at entry points.
- **Coupons & Discounts** system for frequent users or first-time bookings.