1.INTRODUCTION

1.1 PROJECT-TITLE: Movie Ticket Booking System

The **Movie Ticket Booking System** is a full-stack web application built using the **MERN stack** (MongoDB, Express.js, React, and Node.js). This project aims to provide users with a seamless and efficient platform to browse movies, check show timings, book tickets, and manage their reservations online.

OBJECTIVE

The **Movie Ticket Booking System** aims to provide a user-friendly**,** efficient, and secure platform for booking movie tickets online. The primary objectives of this project include:

1. **Automating the Ticket Booking Process**
    - Eliminating the need for physical ticket counters by allowing users to book tickets online.
    - Reducing waiting times and enhancing the overall user experience.
2. **Enhancing User Convenience**
    - Providing a seamless interface to browse movies, check show timings, and select seats.
    - Allowing users to book tickets from anywhere, anytime.
3. **Ensuring Secure Transactions**
    - Implementing secure payment gateways to handle online payments safely.

- Using JWT-based authentication to protect user data.

4. **Real-Time Seat Availability & Booking**
    - Displaying available seats dynamically to avoid overbooking.
    - Preventing duplicate bookings through real-time updates.
5. **Providing an Efficient Admin Panel**
    - Enabling admins to manage movies, showtimes, and bookings effortlessly.
    - Offering analytics & reports for better decision-making.
6. **Scalability & Performance Optimization**
    - Ensuring the system can handle multiple users simultaneously without performance issues.
    - Using MongoDB for efficient data storage and retrieval.

1.2 TEAM MEMBERS AND THEIR DETAILS:

NAME	ROLL NO	ROLE
A.Vijay Durga Reddy	218X1A4203	FRONTEND
J.Krutthivasa Murthy	218X1A4221	BACKEND
Ch.Bhanu Prakash Guptha	218X1A4254	TESTING
Ch.Viswanadh	218X1A4210	DATABASE

2.PROJECT OVERVIEW

2.1 PURPOSE:

2. Project Overview

Purpose

The Movie Ticket Booking System is designed to simplify the process of booking movie tickets online. It allows users to browse available movies, select showtimes, choose seats, and make secure payments. The system enhances user convenience while ensuring efficient management for theatre administrators.

Features

User Features:

- Browse movies, showtimes, and theatres
- Select seats and book tickets
- Secure online payments
- View booking history and download tickets

Admin Features:

- Add, update, and delete movies and showtimes
- Manage seat availability and bookings
- Monitor user transactions and generate reports

.

3.ARCHITECTURE

3.1 FRONTEND:

The Movie Ticket Booking System is developed using the MERN (MongoDB, Express.js, React.js, Node.js) stack, providing a seamless and user-friendly platform for booking movie tickets online. The system ensures efficient movie listings, real-time seat availability, secure transactions, and an intuitive user experience**.**

---

1. Frontend (React.js)

The frontend is built using React.js, ensuring a dynamic and responsive user experience across different devices.

Key Features of the Frontend

- **Component-Based Architecture**: The application is structured into reusable components like:
    - **Movie List:** Displays available movies with details such as name, showtime, and ratings.
    - **Movie Details:** Provides an overview of selected movies, trailers, and cast information**.**
    - **Seat Selection:** Allows users to choose seats interactively in a graphical format**.**
    - **Payment Form:** Handles the checkout process for ticket booking.
    - **User Dashboard**: Shows booking history, profile management, and transaction records.

- **React Router:** Enables seamless navigation between pages without reloading**.**
- **State Management:** Uses React Context API or Redux for handling user authentication, selected seats, and transaction details.
- **UI Framework:** The interface is styled using Material-UI, Tailwind CSS, or Bootstrap for a modern and appealing look.
- **API Communication:** Utilizes Axios or Fetch API to interact with the backend for movie listings, seat availability, and payment processing.

---

2. Backend (Node.js & Express.js)

The backend is developed using Node.js and Express.js, responsible for managing business logic, user authentication, and API endpoints.

Key Features of the Backend

- RESTful API Development for handling movie listings, bookings, authentication, and payments**.**
- Authentication & Security using JWT (JSON Web Tokens) for secure user login, bcrypt.js for password hashing, and CORS & rate limiting for API protection.
- Middleware Implementations to manage error handling, logging, and request tracking.
- Integration with Payment Gateways like Stripe, Razorpay, or PayPal for secure transactions**.**

---

3. Database (MongoDB)

MongoDB serves as the database, ensuring fast and efficient storage of user data, movie details, and bookings.

Database Structure & Optimization

- **Users Collection:** Stores user details and booking history**.**
- **Movies Collection:** Maintains movie information, including showtimes and available seats.
- **Bookings Collection:** Tracks user reservations and payment statuses.
- **Indexes for Fast Queries:** Indexed movie names and booking history for quick retrieval.
- **Data Replication & Backup:** Ensures database security and high availability.

Conclusion

This MERN-based movie ticket booking system delivers an efficient and scalable solution for online movie reservations. By integrating React.js, Node.js, Express.js, and MongoDB, the system ensures a smooth user experience, secure transactions, and real-time booking updates, making it ideal for theatres and online ticketing platforms.

4.SETUP INSTRUCTIONS

4.1 PREREQUISITES:

Before running this project on your local machine, make sure you have the following software installed and properly set up:

1. Node.js and npm:

- **Purpose:**
    - Node.js is required to run the backend server (Express.js).
    - npm (Node Package Manager) helps manage and install required project dependencies for both backend and frontend.
- **Version:**
    - Node.js (v16 or above recommended)
    - npm (comes bundled with Node.js)
- **Download & Install:**
  https://nodejs.org/

2. MongoDB:

- **Purpose:**
    - MongoDB is the database where all the application data (users, products, orders, etc.) is stored.
- **Options:**
    - **Local MongoDB:** Install MongoDB Community Server on your system.

    - **Cloud MongoDB (MongoDB Atlas):** You can also use a free cloud database.

- **Download & Install (Local):** https://www.mongodb.com/try/download/community

3. Git:

- **Purpose:**
    - To clone the project repository from version control (GitHub or any Git server).
- **Download & Install:** https://git-scm.com/downloads

4. Visual Studio Code (Recommended):

- **Purpose:**
    - A code editor to open, edit, and manage the project files with integrated terminal support.
- **Download & Install:** https://code.visualstudio.com/

5. Postman (Optional):

- **Purpose:**
    - To test the backend APIs easily during development.
- **Download & Install:** https://www.postman.com/downloads/

6. Browser:

- **Purpose:**
    - To run and test the frontend React application.
    - Recommended: **Google Chrome** or **Microsoft Edge**.

4.2 INSTALLATION:

Follow these step-by-step instructions to set up and run the project on your local machine:

1. Clone the Project Repository:

First, download the complete project from the repository using Git.

git clone <repository-link>

Replace <repository-link> with your actual project repository URL.

2. Open the Project in VS Code:

After cloning, open the project folder in **Visual Studio Code** or any code editor.

cd code

code .

3. Install Backend Dependencies:

The backend is located inside the **server/** folder.
Follow these steps:

cd server

npm install

This will install all required backend packages like express, mongoose, dotenv, etc., as listed in package.json.

4. Install Frontend Dependencies:

The frontend is inside the **client/** folder.
Open a new terminal and run:

cd client

npm install

This will install all frontend packages like react, axios, react-router-dom, etc.

5. Configure Environment Variables:

Inside the **server/** folder, create a .env file for sensitive configurations:

PORT=3001

MONGO_URI=mongodb://localhost:27017/yourDatabaseName

JWT_SECRET=yourSecretKey

Make sure MongoDB is running locally or provide your MongoDB Atlas connection string.

6. Start MongoDB:

- If you’re using local MongoDB:
    - Open a new terminal and start MongoDB.
    - mongod

7. Run the Backend Server:

In the **server/** folder, start the backend:

npm start

By default, this will run the backend on **http://localhost:3001**.

8. Run the Frontend Application:

In the **client/** folder, start the React frontend:

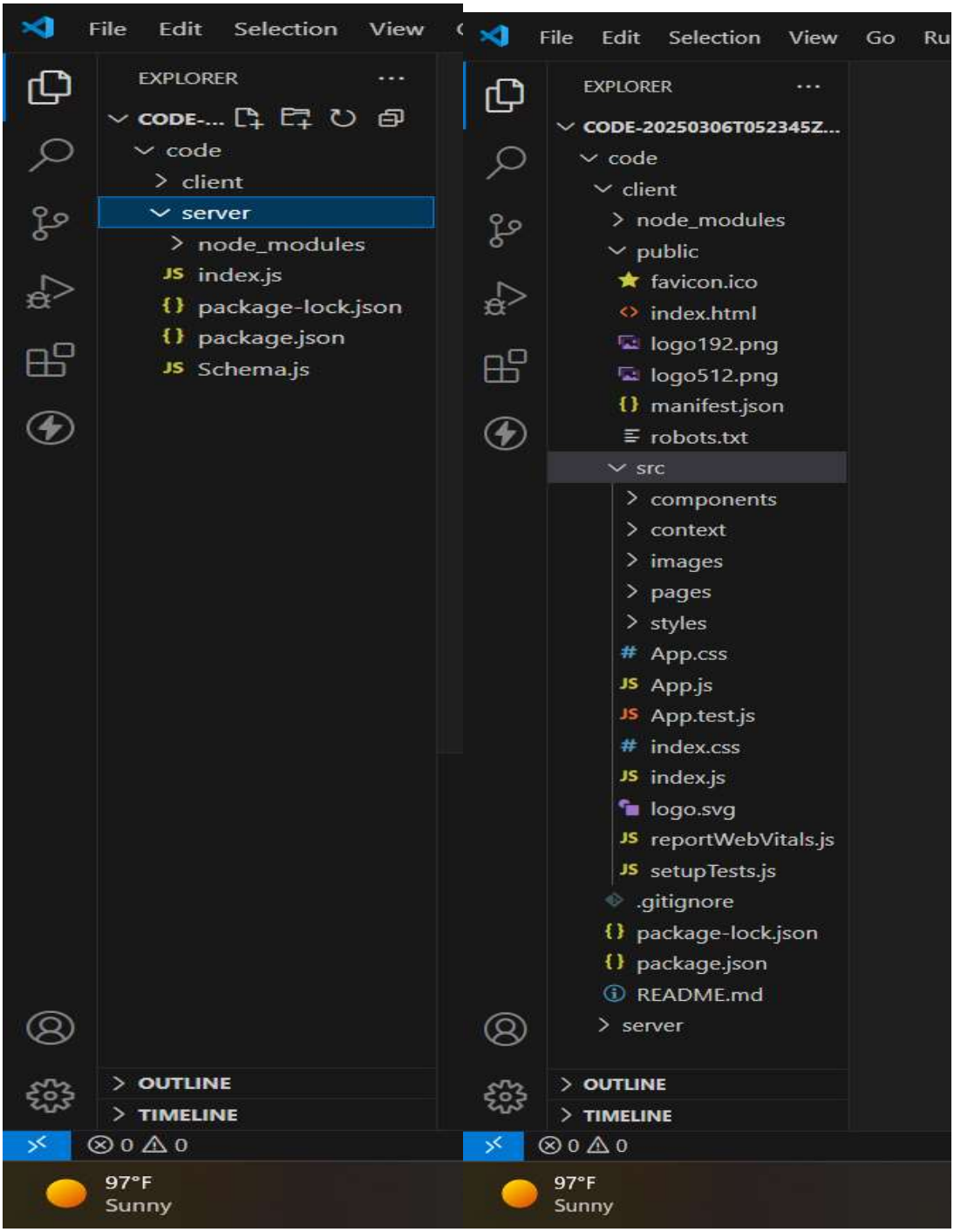
npm start

By default, this will run the frontend on **http://localhost:3000**.

9. Access the Application:

- **Frontend UI:** http://localhost:5173
- **Backend API:** http://localhost:5173

5.FOLDER STRUCTURE



6.RUNNING THE APPLICATION

Commands to start frontend and backend server locally:

6.1 Frontend Server: (React.js)

1. Navigate to the frontend directory:

cd client

2. Install dependencies (if not installed):

npm install

3. Start the frontend server:

npm start

4. The React app will run on http://localhost:3000/

6.2 Backend Server: (Node.js + Express.js)

1. Navigate to the backend directory:

cd server

2. Install dependencies (if not installed):

npm install

3. Start the backend server:

node index.js4.

The backend API will be available at http://localhost:3000/

7.API DOCUMENTATION

7.1 BACKEND API ENDPOINTS:

1. Authentication

APIs to handle User Registration and Login:

- **POST /register** – Register a new user with name, email, password, and user type.
- **POST /login** – Authenticate the user and return user details.

2. Products

APIs to manage products:

- **GET /fetch-products** – Retrieve all products.
- **GET /fetch-product-details/:id** – Get details of a single product by ID.
- **POST /add-new-product** – Add a new product (Admin only).
- **PUT /update-product/:id** – Update an existing product (Admin only).

3. Banner & Categories

APIs to handle homepage banner and categories:

- **GET /fetch-banner** – Get the homepage banner.
- **POST /update-banner** – Update the homepage banner (Admin only).
- **GET /fetch-categories** – Get all product categories.

4. Users
API to manage users:

- **GET /fetch-users** – Retrieve all registered users (Admin only).

5. Orders
APIs to handle orders:

- **GET /fetch-orders** – Get all placed orders (Admin only).
- **POST /buy-product** – Place an order for a single product.
- **POST /place-cart-order** – Place an order for all items in the cart.
- **PUT /cancel-order** – Cancel an order by order ID.
- **PUT /update-order-status** – Change the status of an order (Admin only).

6. Cart
APIs to manage cart items:

- **GET /fetch-cart** – Get all items in the cart.
- **POST /add-to-cart** – Add a product to the cart.
- **PUT /increase-cart-quantity** – Increase quantity of a cart item.
- **PUT /decrease-cart-quantity** – Decrease quantity of a cart item.
- **DELETE /remove-item** – Remove an item from the cart.

8.AUTHENTICATION

Authentication Handling:

- The project uses JWT (JSON Web Tokens) for authentication.
- On successful login (POST /login), the backend generates a JWT with user details (userId, email, role).
- The token is signed with a secret key and sent to the frontend, stored securely in localStorage or sessionStorage.

Authorization Handling:

- For protected APIs, the token is sent in the Authorization header: Authorization: Bearer <JWT_TOKEN>
- Backend middleware verifies the token.
    - If valid → request proceeds.
    - If invalid/expired → returns 401 Unauthorized.

Session & Security:

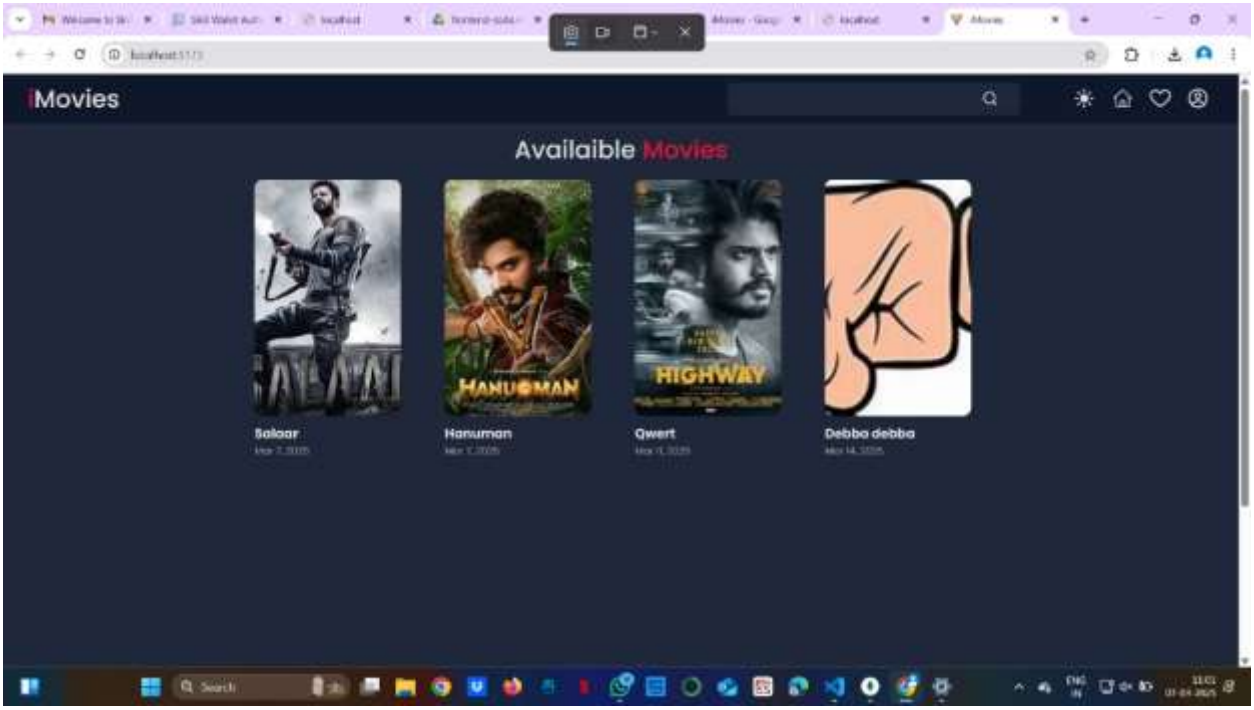
- No traditional sessions; uses stateless JWT authentication.
- Tokens have an expiry time (e.g., 1 hour).
- After expiry, users must log in again to get a new token.

User Roles & Access:

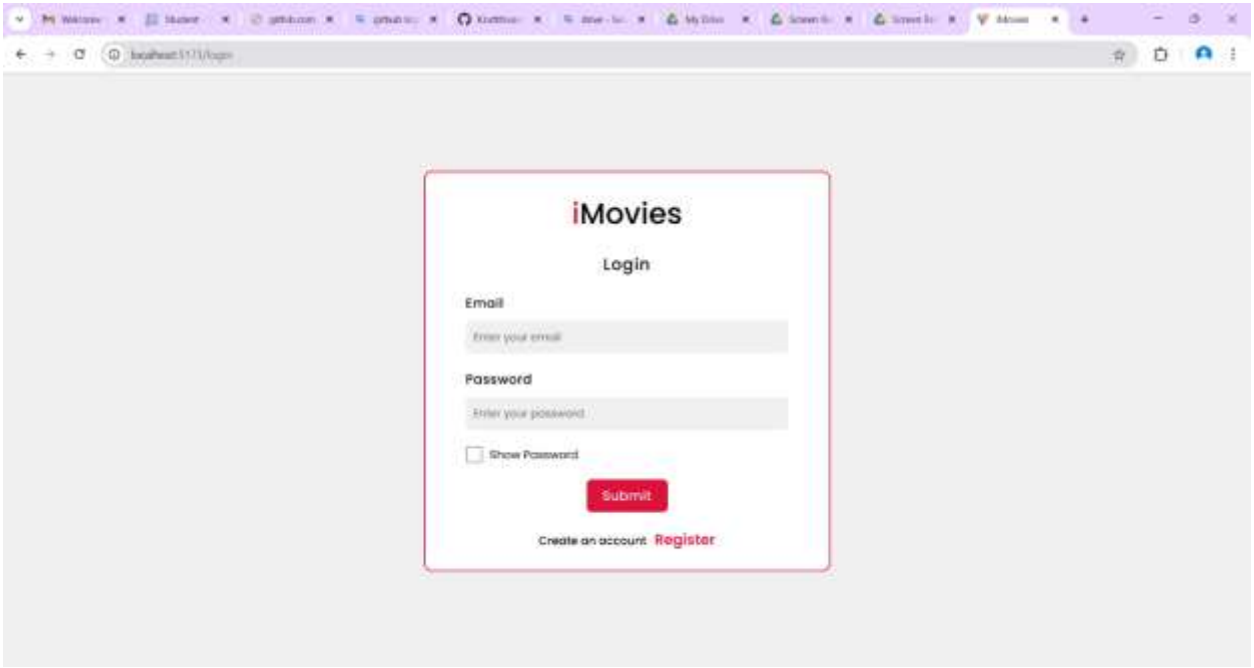
- JWT also contains the user's role ("admin" or "user").
- APIs check roles to allow or restrict actions.

9.USER INTERFACE

HOME PAGE:



USER INTERFACE:



10.TESTING

Testing Strategy for Movie Ticket Booking System

A well-defined testing strategy ensures the Movie Ticket Booking System functions correctly, is free of bugs, and provides a smooth user experience. The testing process includes unit testing, integration testing, and end-to-end (E2E) testing, using various tools and frameworks.

---

1. Testing Strategy

1.1 Unit Testing

- **Focuses on individual components and functions (e.g., API endpoints, database queries, UI components).**
- **Ensures that each module works correctly in isolation.**
- **Applied to backend routes, authentication logic, and frontend components.**

1.2 Integration Testing

- **Verifies that different modules work together as expected.**
- **Tests frontend-backend interactions, ensuring proper communication via API calls.**

- **Ensures database operations, such as user authentication, movie listings, and booking transactions, function correctly.**

1.3 End-to-End (E2E) Testing

- **Simulates real-world user interactions from start to finish.**
- **Covers sign-up, login, movie selection, seat booking, and payment processing.**
- **Ensures the system handles edge cases (e.g., invalid logins, unavailable seats, payment failures).**

1.4 Performance Testing

- **Measures system response time and scalability.**
- **Ensures the system can handle high traffic loads (e.g., during movie ticket releases).**

1.5 Security Testing

- **Validates user authentication & authorization mechanisms (e.g., JWT verification).**
- **Checks for SQL injection, XSS, and CSRF attacks.**
- **Ensures secure storage of sensitive data (passwords, payment details).**

---

2. Tools Used

For Unit & Integration Testing

- **Jest – JavaScript testing framework for both frontend and backend.**
- **Mocha & Chai – Used for backend API testing with Node.js & Express.js.**

For API Testing

- **Postman** – Manually tests REST API endpoints.
- **Supertest** – Automates API request testing.

For End-to-End Testing

- **Cypress** – Simulates real user behavior in a browser.
- **Selenium** – Automates UI testing for web applications**.**

For Performance Testing

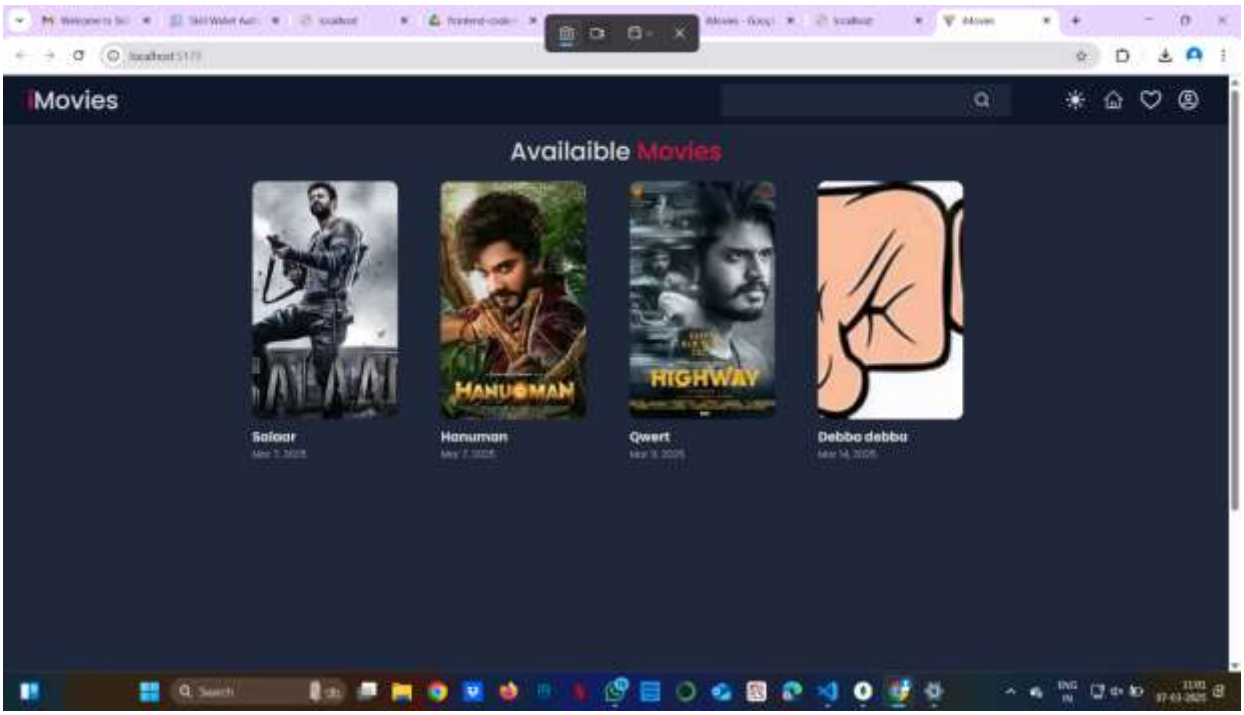
- JMeter – Measures system load handling.
- Lighthouse – Tests frontend performance and optimization.

For Security Testing

- OWASP ZAP – Identifies security vulnerabilities.
- Helmet.js – Helps secure Express apps by setting HTTP headers.

11.SCREENSHOTS OR DEMO

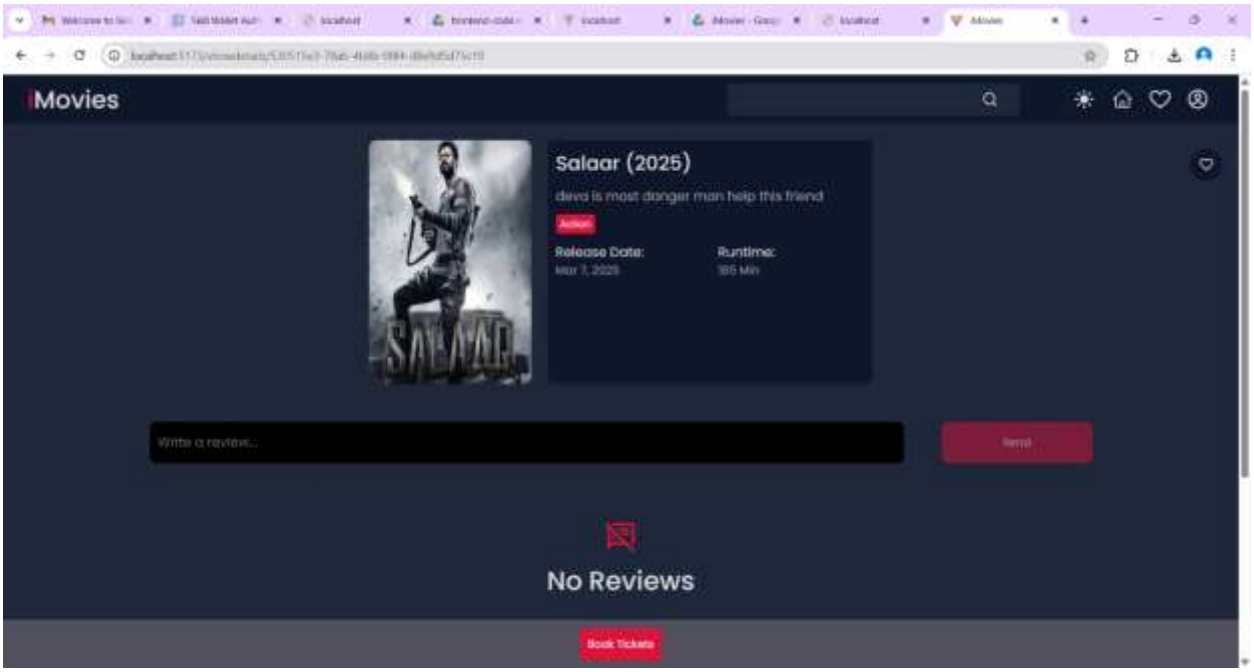
STEP:1



***Here we can select the movie**

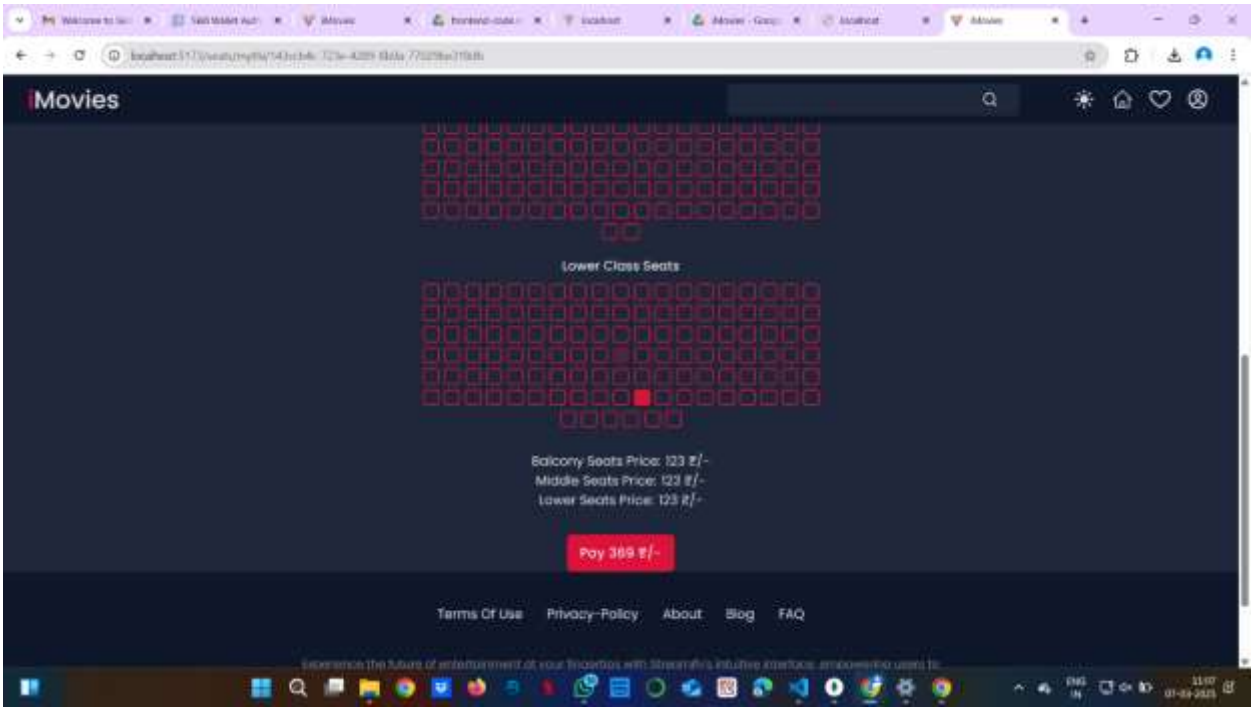
***We will open it**

STEP:2



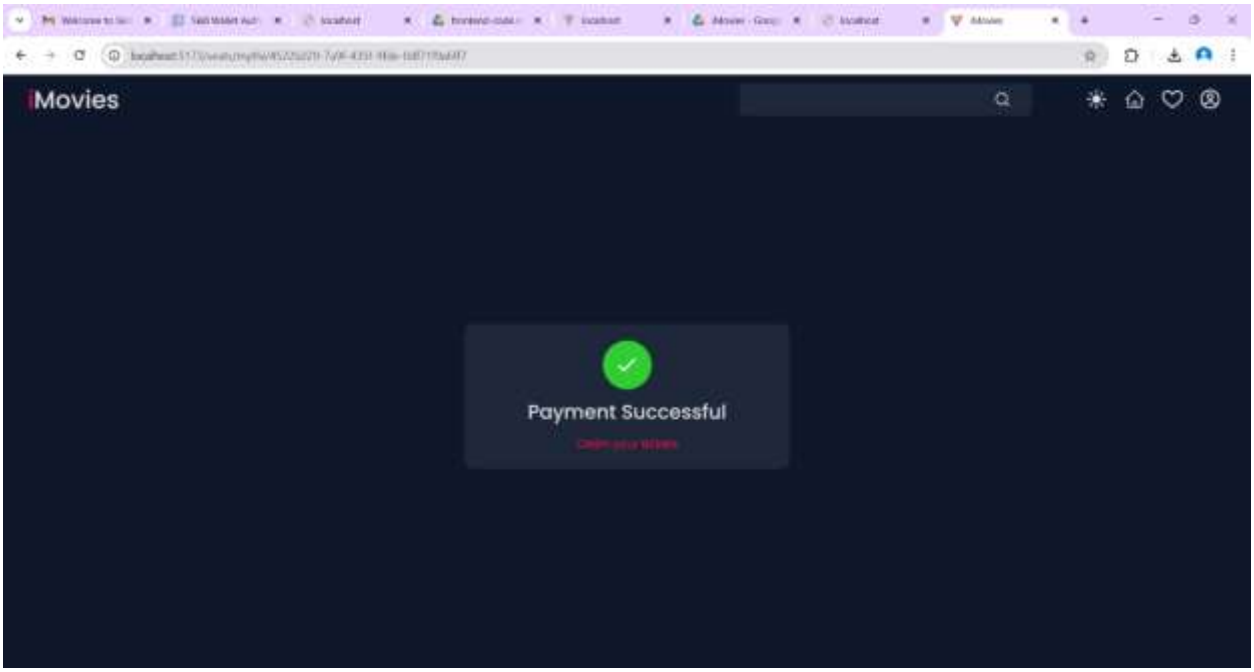
***Here we can start book the tickets**
***We can give review**

STEP:3



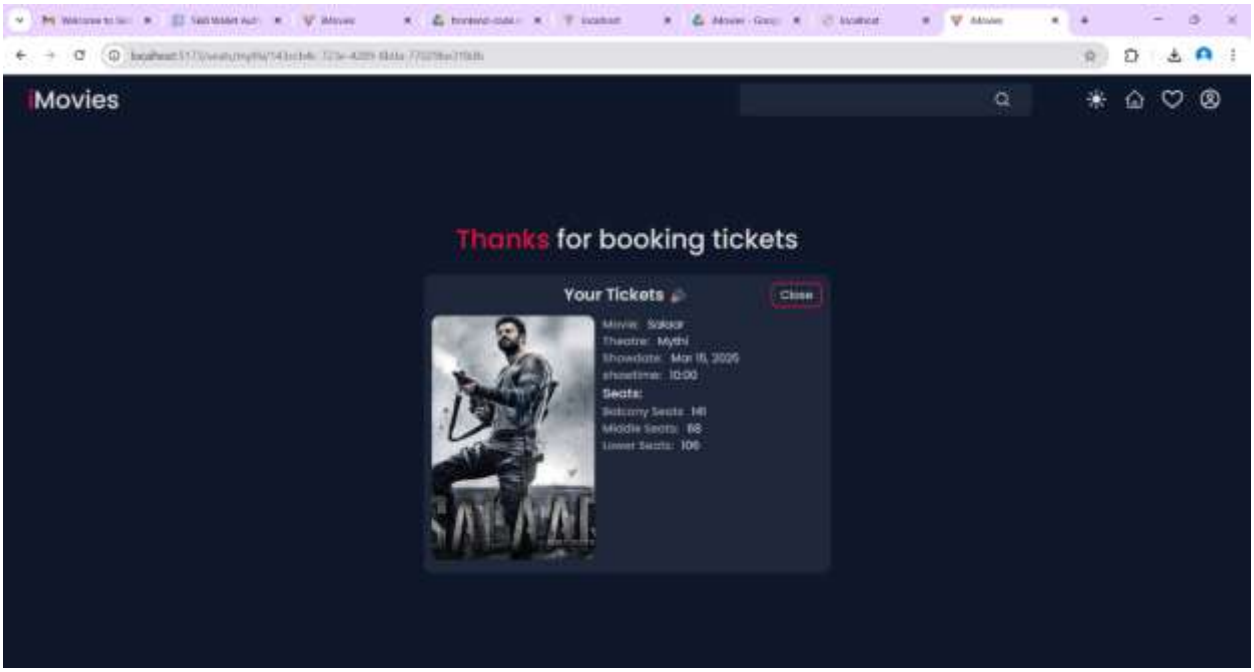
- **Here we can select the seats**

STEP:4



*Here we can get conformation of tickets

STEP:5



*Here we can see the tickets booked

12.KNOWN ISSUES

Despite thorough testing, there are a few known bugs and limitations in the current version of ShopEZ that developers and users should be aware of:

1. Token Expiry Handling

- When the JWT token expires, users are not automatically logged out or redirected to the login page.
- Users may continue interacting with the site until they refresh the page or make a new request, which then fails due to an expired token.
- Fix Suggestion: Implement automatic token expiry checks on the frontend and redirect users to login on expiry.

2. Duplicate Add to Cart

- Sometimes, if the "Add to Cart" button is clicked multiple times quickly, the same product is added more than once unexpectedly.
- Fix Suggestion: Disable the button while the request is processing to avoid duplicate submissions.

3. No Email Notifications

- Currently, ShopEZ does not send email confirmations for successful orders, registrations, or password resets.
- This might cause confusion for users expecting order confirmations.
- Fix Suggestion: Integrate email services like Nodemailer or SendGrid for email notifications.

4. No Password Reset Feature

- There is no "Forgot Password" or password reset flow.
- Users who forget their password need admin help to regain access.
- Fix Suggestion: Add a password reset system with email-based OTP or reset links.

5. Slow Loading on Large Product Lists

- When there are many products in the database, the homepage and product pages may load slowly since pagination is not yet implemented.
- Fix Suggestion: Add backend pagination and lazy loading on the frontend to improve performance.

6. Admin Error Handling

- On the admin dashboard, if invalid product data is entered while adding/updating products, the error messages are not always clear.
- Fix Suggestion: Improve backend validation and display clear, user-friendly error messages on the frontend.

7. Mobile Layout Glitches

- On small screen devices, some elements in the cart and admin pages overlap or get cut off.
- Fix Suggestion: Fine-tune the responsive design using CSS fixes and thorough mobile testing.

These are the current known issues in ShopEZ.
Future updates will focus on resolving these to enhance the overall user experience and system stability.

13.FUTURE ENHANCEMENTS

To improve ShopEZ and provide a better user experience, the following features and improvements can be planned for future updates:

1. Password Reset Feature

- Add a "Forgot Password" functionality where users can reset their password via email verification (OTP or reset link).
- Improves user account recovery without admin support.

2. Email Notifications

- Integrate email services like Nodemailer or SendGrid to send automatic emails for:
    - Order confirmations.
    - Shipping updates.
    - Password resets.
    - Registration welcome emails.

3. Product Reviews & Ratings

- Allow users to rate and review products after purchase.
- Display average product ratings and reviews on product detail pages to help buyers.

4. Wishlist Feature

- Provide users the option to add products to a wishlist for future purchases.
- Helps users save items without adding them to the cart.

5. Advanced Search and Filters

- Add search functionality with keyword matching.
- Include filters (like price range, categories, ratings) on product listing pages to improve browsing.

6. Pagination and Infinite Scroll

- Implement pagination or infinite scroll to improve performance when loading large product lists.
- Prevents long loading times on the homepage or category pages.

7. Payment Gateway Integration

- Integrate popular payment gateways like Razorpay, Stripe, or PayPal for secure online payments.
- Currently, orders are placed without real payment processing.

8. Delivery Tracking

- Add order tracking where users can see live delivery status updates after placing an order.

9. Mobile App Version

- Develop a mobile app (using React Native or Flutter) for a better shopping experience on Android and iOS devices.
    - Most sold products.
    - User activity.

11. Multi-language Support

- Add language translation options to support different regional languages and reach a wider audience.