Movie ticket Booking System

1. Introduction

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.

Team Members and their roll no's :

KADIYAM LEELADHAR	218X1A0419
DAMMATI NAGA NIKHITHA	228X5A0403
KAVURI DILEEP KUMAR	218X1A0429

2. Project Overview

Purpose

The purpose of this project is to develop a full-stack web application using React.js for the frontend and Node.js with Express.js for the backend. The project aims to provide an efficient and scalable solution for [briefly state the problem the project solves].

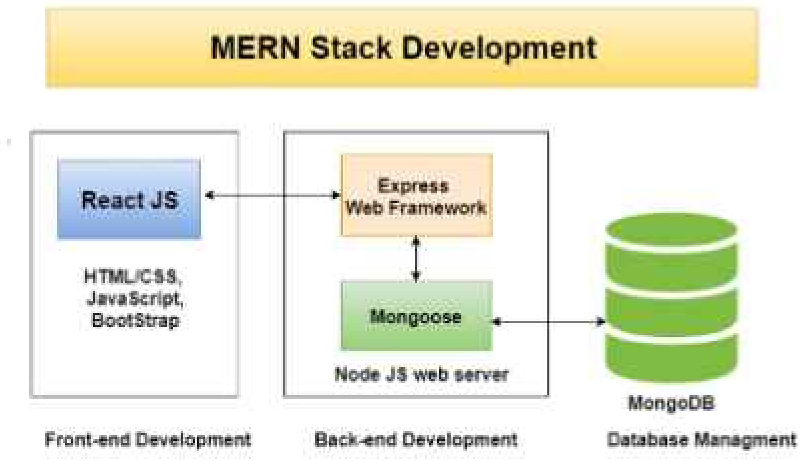
Features

- **User Authentication** - Secure login and signup with JWT authentication.
- **Database Integration** - MongoDB as the NoSQL database for storing user and application data.

- **Responsive UI** - Fully responsive design for optimal user experience on all devices.
- **RESTful API** - Well-defined API endpoints to handle CRUD operations.
- **State Management** - Efficient state management using React Context API or Redux.

---

3. Architecture



Frontend Architecture (React.js)

The frontend follows a component-based structure to ensure modularity and reusability.

- **Components** - Reusable UI components.
- **Pages** - Different pages such as Home, Dashboard, and Profile.

Backend Architecture (Node.js & Express.js)

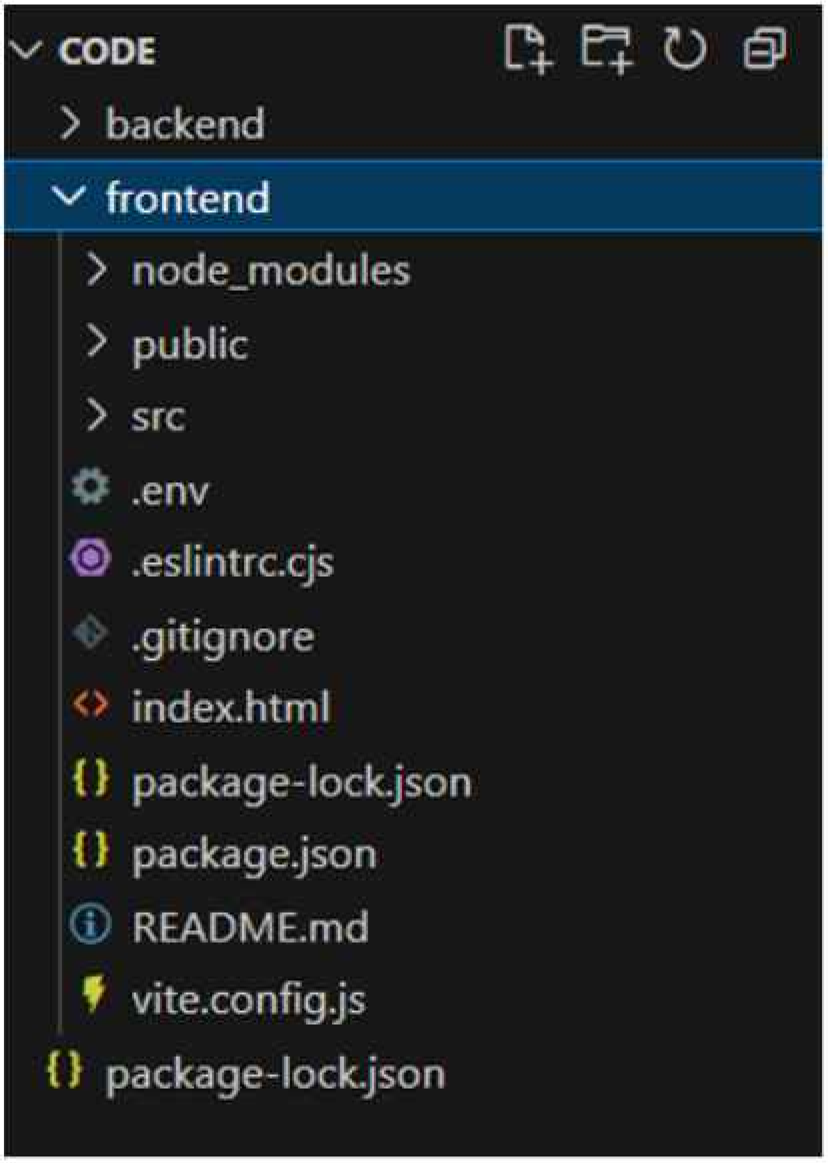
The backend is built on a RESTful API structure, with routes, controllers, and middleware layers.

- **Routes** - Organized by feature (users, posts, etc.).
- **Controllers** - Logic for handling requests.
- **Middleware** - Authentication, logging, error handling.
- **Database (MongoDB)**

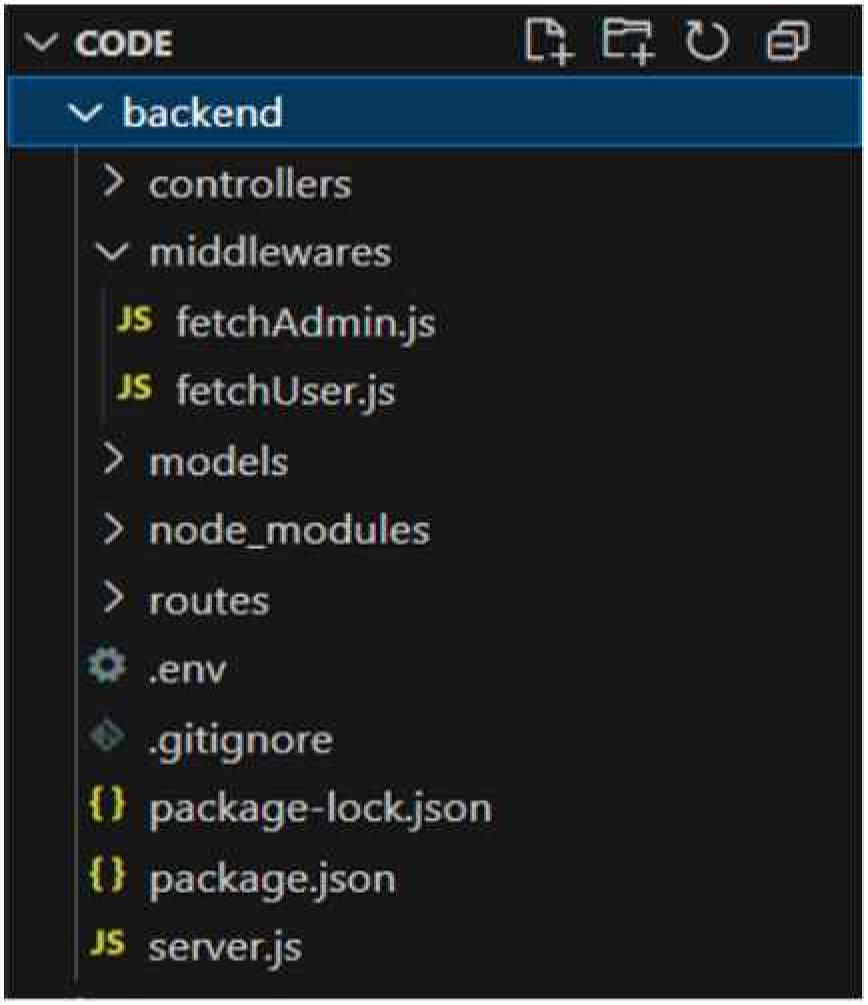
Data is stored in a NoSQL format using collections.

---

4. Folder Structure of Frontend



```
CODE
  backend
  frontend
    node_modules
    public
    src
    .env
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    README.md
    vite.config.js
  package-lock.json
```



```
backend
  controllers
  middlewares
    fetchAdmin.js
    fetchUser.js
  models
  node_modules
  routes
  .env
  .gitignore
  package-lock.json
  package.json
  server.js
```

Folder structure of Backend

5. Running the Application

Start the frontend:

NPM START

Start the backend:

NPM START

Running a full-stack application involves starting both the **backend (server-side)** and **frontend (client-side)** while ensuring they communicate properly.

1. **Backend Execution (Node.js & Express.js)**

   - Install dependencies (`npm install`)
   - Set up environment variables (e.g., database URI, API keys)
   - Start the server (`npm start` or `npm run dev`)
   - Verify API endpoints using Postman or cURL

2. **Frontend Execution (React.js)**

    - Install dependencies (`npm install`)
    - Configure API endpoints (`REACT_APP_API_URL`)
    - Start the development server (`npm start`)
    - Check UI components and API interactions

3. **Production Deployment**

    - Build the frontend (`npm run build`) and serve via backend
    - Use process managers like **PM2** for server uptime
    - Deploy on platforms like **Heroku, Vercel, AWS, or Docker**

4. **Troubleshooting**

    - Resolve **port conflicts** (e.g., change ports in `.env`)

    - Fix **CORS issues** by enabling it in Express
    - Ensure **database connections** are properly configured
5. **Testing Before Deployment**

    - Backend: Use **Jest & Supertest**
    - Frontend: Use **React Testing Library**

---

6. API Documentation

1. Importance of API Documentation

- Provides **clear instructions** on how to interact with the backend.
- Reduces **miscommunication** between frontend and backend teams.
- Simplifies **troubleshooting and debugging** for developers.
- Enhances **scalability** by making it easier for new developers to understand.

2. Key Elements of API Documentation

- **Endpoint Details** – The URL where the API can be accessed.
- **Request Methods** – The HTTP methods (`GET`, `POST`, `PUT`, `DELETE`).
- **Request Parameters** – Query parameters, headers, and body structure.
- **Response Structure** – The format of successful and error responses (usually JSON).
- **Authentication Requirements** – How the API is secured (e.g., tokens, API keys).

3. Common API Standards & Tools

- **RESTful APIs** – Follow standard HTTP methods and stateless architecture.
- **GraphQL APIs** – Provide flexible querying of data.
- **Swagger/OpenAPI** – Standardized format for documenting REST APIs.
- **Postman** – A tool for testing and documenting APIs.

4. Best Practices in API Documentation

- **Consistency** – Use uniform naming conventions and response structures.
- **Clarity** – Avoid ambiguity; provide real-world examples where possible.
- **Versioning** – Maintain different versions (v1, v2) to support updates.
- **Error Handling** – Clearly define possible error codes and their meanings.

---

7. Authentication

Authentication is handled using JWT tokens.

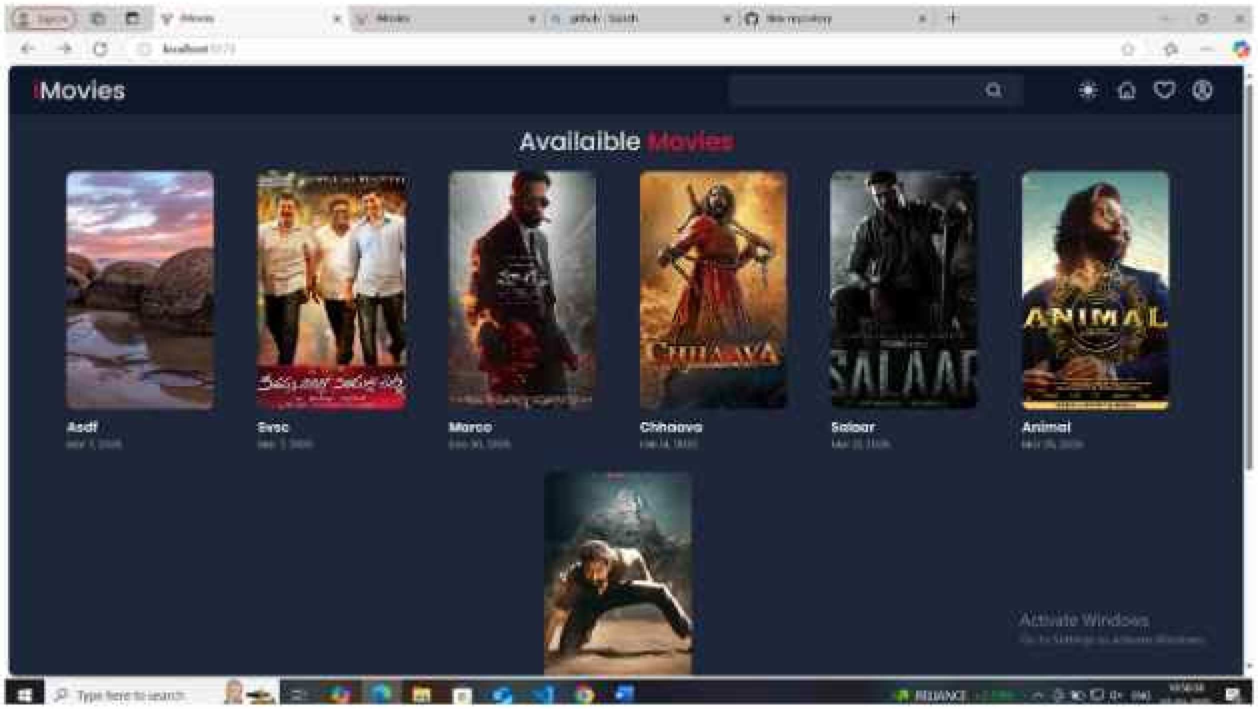
```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const token =
req.header('x-auth-token');
```

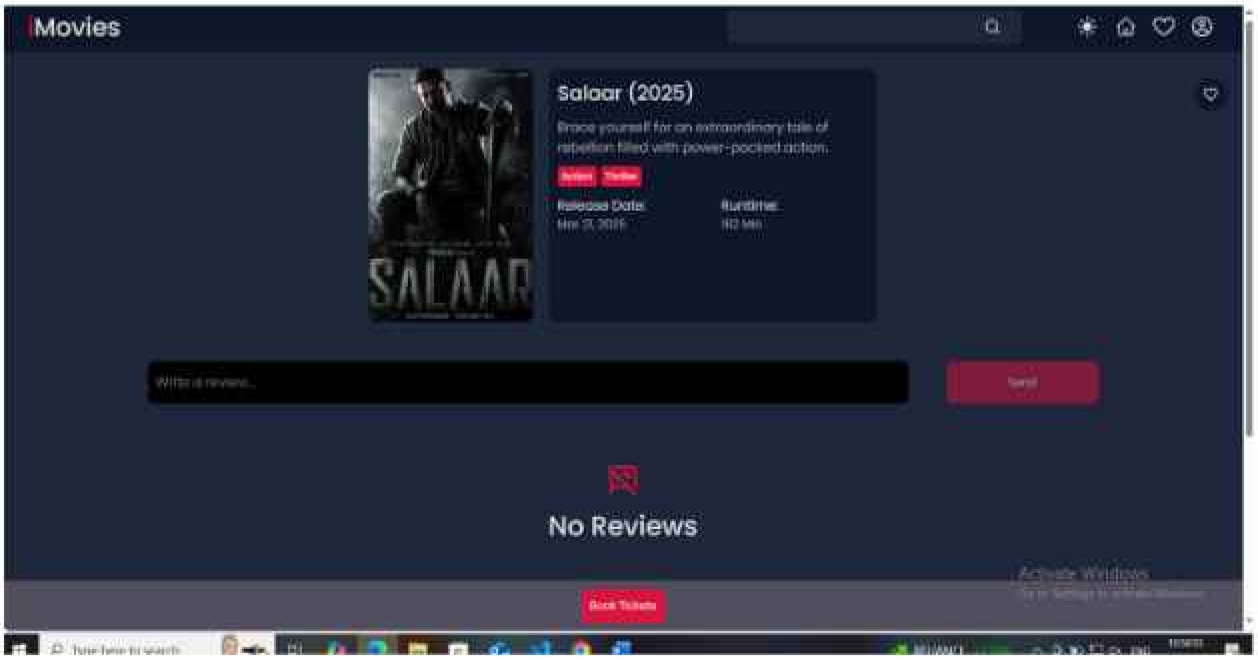
```
  if (!token) return res.status(401).json({
message: 'No token, authorization
denied' });

  try {
    const decoded = jwt.verify(token,
process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(400).json({ message:
'Invalid token' });
  }
};
```

This middleware verifies JWT tokens for authentication. It checks for a token in the request header, verifies it using **jwt.verify()** , and attaches the decoded user data to **req.user**. If the token is missing or invalid, it returns an error response.

---

8. User Interface





Salaar (2025)

Brace yourself for an extraordinary tale of rebellion filled with power-packed action.

9. 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).

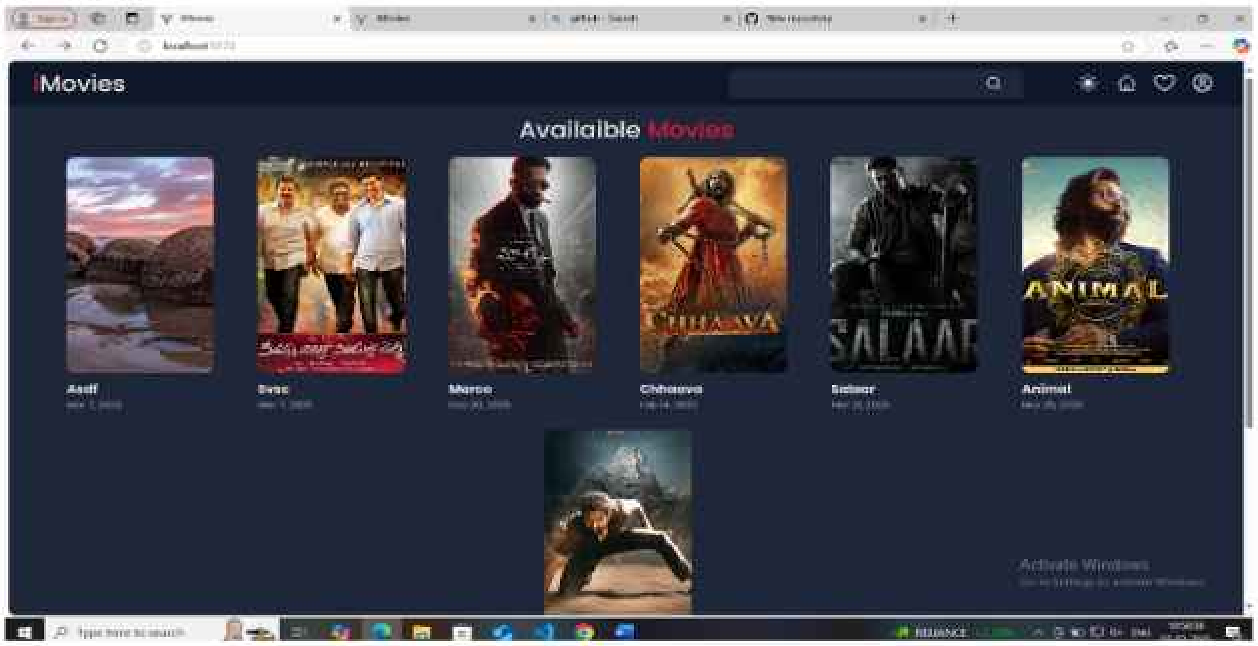
2. Tools Used For Unit & Integration Testing

• Jest – JavaScript testing framework for both frontend and backend.
• Postman – Manually tests REST API endpoints.
• Supertest – Automates API request testing. For End-to-End Testing
• Lighthouse – Tests frontend performance and optimization. For Security Testing

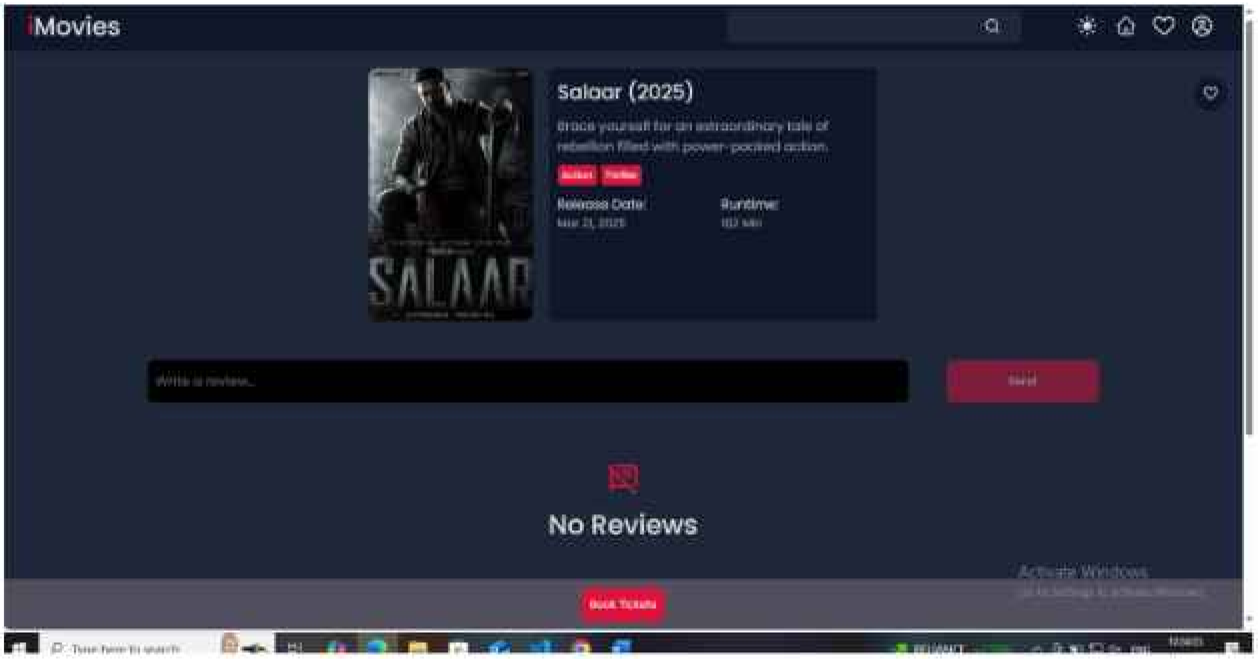
---

10. Screenshots on how the web applications works and Demo video:

This will be our web application's home page



Here we selected SALAAR movie to book ticket



iMovies

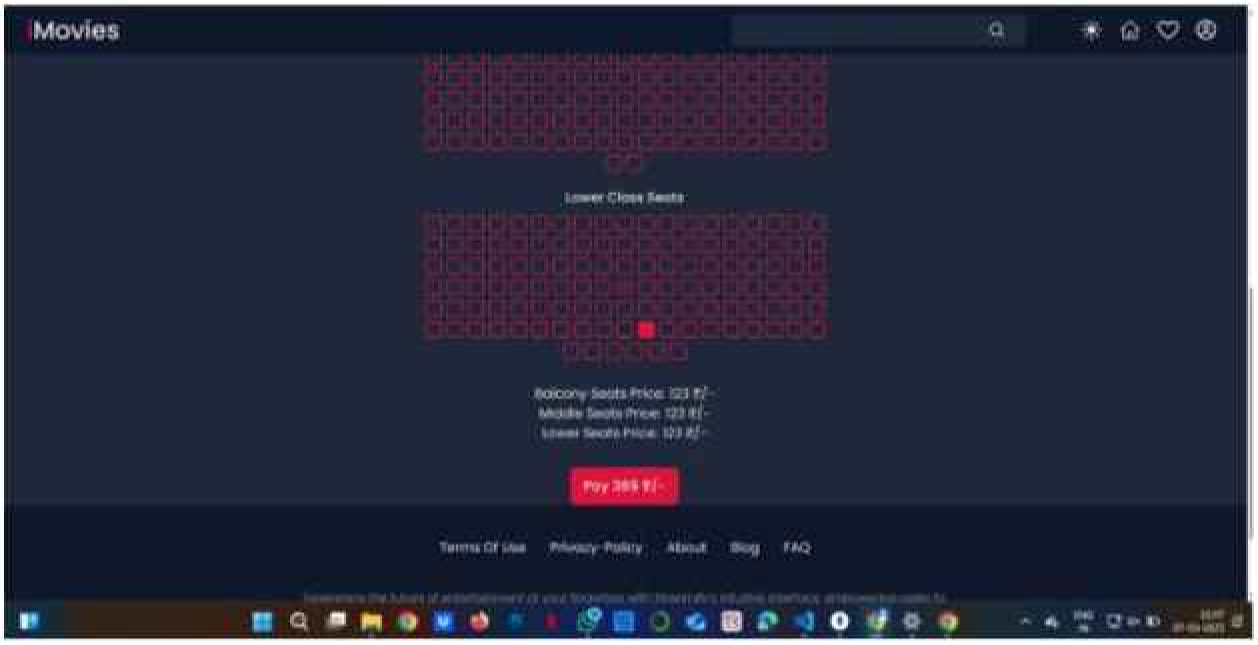
Salaar (2025)

Brace yourself for an extraordinary tale of rebellion filled with power-packed action.

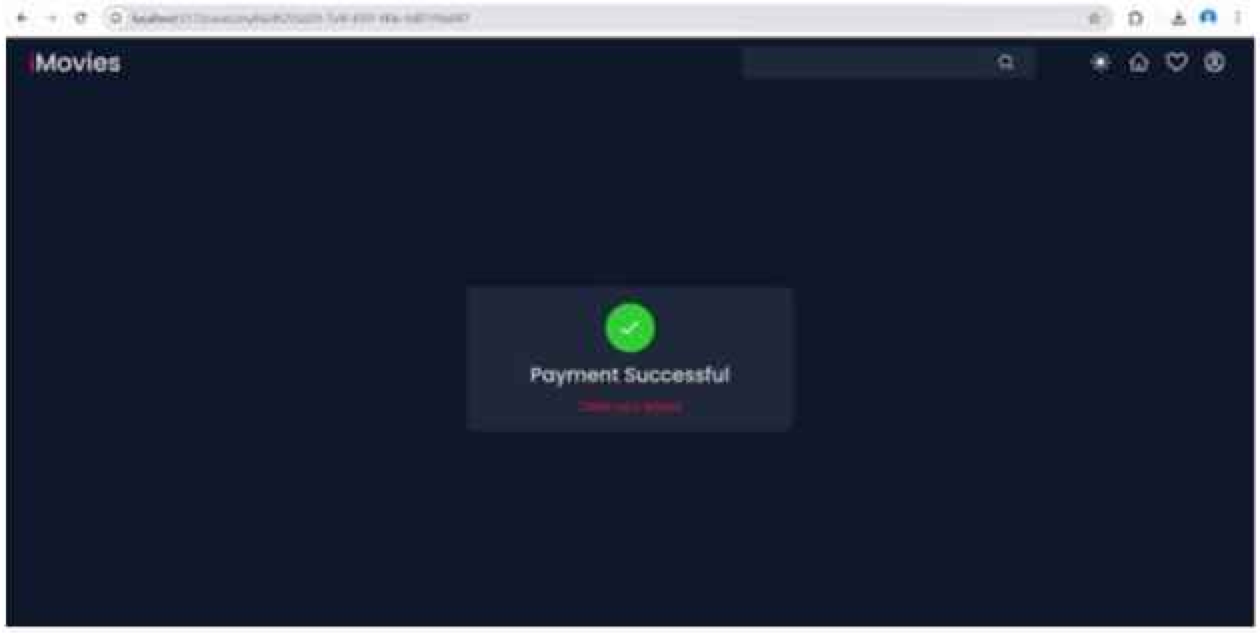
Release Date:
Mar 21, 2025

Runtime:
[illegible]

After selecting the location of theatre and show time we will see the seating arrangement


iMovies
Lower Class Seats
Balcony Seats Price: 123 ₹/-
Middle Seats Price: 123 ₹/-
Lower Seats Price: 123 ₹/-
Pay 369 ₹/-
Terms Of Use
Privacy-Policy
About
Blog
FAQ

After the seat selection we get to payment section



After booking is successful we receive this

[Click here for GITHUB link]

lick here to watch the Demo video

11. Known Issues

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.

---

12. Future Enhancements

Future Enhancements

1. **Role-Based Access Control (RBAC)** – Implement user roles with specific permissions.
2. **Dark Mode & Theme Customization** – Allow users to personalize UI themes.
3. **Real-Time Notifications** – Use WebSockets for live updates.
4. **Mobile App Integration** – Develop a React Native version for mobile users.
5. **Two-Factor Authentication (2FA)** – Add OTP-based login for extra security.
6. **Advanced Search & Filtering** – Enhance usability with better search capabilities.
7. **Multi-Language Support** – Enable localization for a global audience.
8. **AI-Powered Recommendations** – Use machine learning to suggest relevant content.

THANK YOU