I-Movies : Movie Ticket Booking System

ProjectDescription:

Users can effortlessly browse movies, select showtime, and choose seats from the comfort of their homes. purchase a robust client-server architecture with Express.js and MongoDB, the app ensures efficient data storage and retrieval. With features like user hallmark, booking management, and real-time seat handiness tracking, it provides a personalized and convenient movie-going experience. This support provides a comprehensive guide for setup, development, and understanding of the application's technical architecture and functionalities.

Scenario

Imagine Sarah, a movie enthusiast, excitedly expecting the release of an anticipated film. With a busy schedule, she prefers the convenience of booking her movie tickets online. Logging into the iMovies app, Sarah navigates effortlessly through the user-friendly interface, browsing the list of usable movies and show times.

Upon finding the movie she has been eagerly awaiting, Sarah selects her preferred cinema location and showtime. The app displays a seating layout, allowing her to choose the perfect seats for an optimal viewing see. With a few clicks, Sarah confirms her booking and proceeds to the payment section.

Using the secure payment gateway integrated into the iMovies app, Sarah completes her transaction smoothly, receiving a booking confirmation with all the details she needs for her upcoming movie night. As the date approaches, Sarah can easily access her booking history through the app, ensuring a hassle-free experience from start to finish.

Thanks to the iMovies app's intuitive design and robust features, Sarah enjoys a seamless moviegoing experience, making her next cinema outing memorable and stress-free.

Technical Architecture

In This Architecture Diagram:

1. The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Watch list, Movies page, Movie shows page, profile, Seat allotment page, Bookings page etc.,
2. The backend is represented by the "Backend" section, consisting of API endpoints for Users, Favorites, Shows, movies, bookings etc

The Database section represents the database that stores collections for Users, bookings, Favorites of the users, and movies, shows, theater, etc.

The technical architecture of the iMovies app follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses not only the user interface and presentation but also incorporates the axiom library to connect with backend easily by using RESTful Api.

On the backend side, we employ Express.js frameworks to handle the server-side logic and communication.

For data storage and retrieval, our backend relies on MongoDB. MongoDB allows for efficient and scalable storage of user data, adding docs, etc. It ensures reliable and quick access to the necessary information.

Together, the frontend and backend components, along with Express.js, and MongoDB, form a comprehensive technical architecture for our iMovies app. This architecture
enables real-time communication, efficient data exchange, and seamless integration, ensuring a

smooth and immersive experience for all users.

ER-Diagram


passuond
usernome
email
aser
password
usornome
email
Adrain
movieId
shows
movieName
media
genres
certification
Movie
releaseDate
runtime
description
Relationship
showId
adminEmail
movieId
bookingId
Booking
Relationship
Show
theatreName
userEwnil
favorite
ticketnData
userEmail
tickets
showtime
showdate
movieId
theatroNama
adminName
location
lowerSeats
theatreId
theatre
middleSeats
location
balconySeats
baiconySeatPrice
lowerSeatPrice
middleSeatPrice

PREREQUISITES

To develop the iMovies app, essential prerequisites include Node.js and npm for server-side JavaScript, MongoDB for data storage, Express.js for backend API development, React.js for dynamic UI creation, and Mongoose for database connectivity.

1. Node.js and npm

To execute server-side JavaScript, install Node.js. You can download it from the official website. Refer to the installation guide for detailed instructions.

1. **Download**: https://nodejs.org/en/download/
2. **Installation Guide**: https://nodejs.org/en/download/package-manager/

2. MongoDB

Set up a MongoDB database for storing application data (e.g., hotel/bookings).

1. **Download (Community Edition)**: https://www.mongodb.com/try/download/community
2. **Installation Guide**: https://docs.mongodb.com/manual/installation/

3. Express.js

A Node.js framework for backend API development.

- Install via npm:
  bash
  'npm install express'

4. React.js

JavaScript library for building dynamic UI's.

- **Installation Guide**: https://reactjs.org/docs/create-a-new-react-app.html

5. HTML, CSS, and JavaScript

Front-end developers need fundamental knowledge.

6. Database Connectivity

Use **Mongoose** (ODM for MongoDB) to connect Node.js with MongoDB.

- **Guide**:https://www.section.io/engineering-education/nodejs-mongoosejsmongodb

7. Firebase Storage (for Images)

To upload/store images:

1. **Create a Firebase Project**: https://console.firebase.google.com/
2. **Install Firebase SDK**: 3. bash
   'npm install Firebase'
   Enable Storage in Firebase Console and configure:
   javascript
   'import { getStorage, ref, uploadBytesResumable, getDownloadURL } from "Firebase/storage";'

8. Version Control (Git):

- **Download Git**: https://git-scm.com/downloads

9. Development Environment

Choose an IDE:

1. **VS Code**: https://code.visualstudio.com/download
2. **Sublime Text**: https://www.sublimetext.com/download
3. **WebStorm**: https://www.jetbrains.com/webstorm/download

1. Project setup and configuration

The project setup involves installing Node.js and Git, creating a structured folder system for the frontend and backend, initializing the backend with Express, and implementing version control using Git.

- **Install Node.js and Git**

  Run these commands one by one in your terminal:

  bash

  # Verify installations

  'node --version'

  npm --version

  git --version

  If any command fails, download Node.js from nodejs.org and Git from git-scm.com.

- **Create Project Structure**

  Execute these commands to set up the basic folder structure:

  bash

  mkdir imovie-app

  cd imovie-app

- **Set Up React Frontend**

  Create your frontend with this command:

  bash

  npx create-react-app frontend

  This will generate:

1. frontend/src/ for React components
2. frontend/public/ for static assets
3. All necessary configuration files

- **Initialize Backend**

  Run these commands to set up your Node.js backend: bash mkdir Backend cd Backend

  npm init -y

- **Create Backend Structure**

  While still in the Backend folder, run:

  bash mkdir

  models routes

  touch server.js

  This creates:

  1. models/ for database schemas
  2. routes/ for API endpoints
  3. server.js as your main server file

■ **Set Up Version Control**

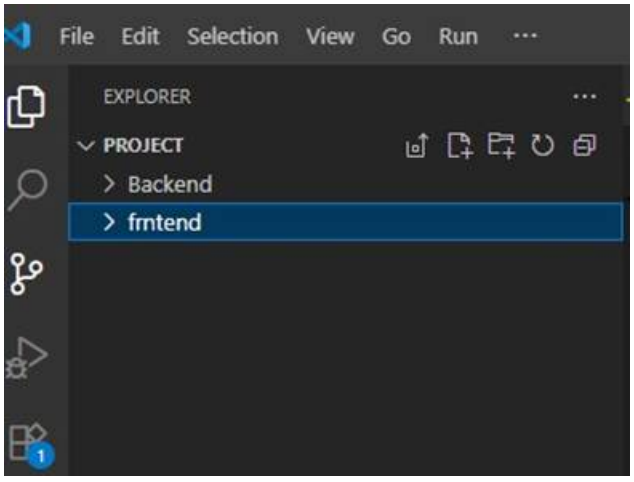
Navigate back to your project root and initialize Git:
bash cd ..
git init
touch
.gitignore

Add these lines to your .gitignore file:
text
node_modules/
.env

■ **Install Backend Dependencies**

Run these commands to install required packages:
bash
cd Backend
npm install express mongoose cors
dotenv ■ **Start Development Servers**
Open two terminal windows and run: bash

Terminal 1
(frontend) cd
frontend npm start
Terminal 2
(backend) cd
Backend
node server.js



2.Backend Development

-

1. **Initialize Express Server**
2. Navigate to your Backend folder and install required dependencies:
   bash
   cd Backend
   npm install express cors body-parser mongoose jsonwebtoken dotenv bcryptjs
3. Create a new index.js file as your main server file:
   bash
   touch index.js
4. **Configure Environment Variables**
5. Create a .env file in your Backend folder:
   bash
   touch .env
6. Add these configurations to your .env file: env

   PORT=5000
   MONGODB_URI=mongodb://localhost:27017/imovie_db
   JWT_SECRET=your_strong_secret_key_here

- **Basic Server Setup**

1. Add this code to your indexserver.js file:

Authentication Setup

2. Create a models/User.js file for user schema:

backend > models > JS User.js > ...

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true
  },
  email: {
    type: String,
    required: true,
    unique: true,
    trim: true
  },
  phone: {
    type: String,
    required: true
  },
  isAdmin: {
    type: Boolean,
    default: false
  },
```

3. Create a routes/auth.js file for authentication routes:

Backend > routes > JS authRoutes.js

```
const express = require('express');
const authController = require('../controllers/authController');

const router = express.Router();

// Public routes
router.post('/register', authController.register);
router.post('/login', authController.login);

// Protected routes
router.use(authController.protect);
router.get('/me', authController.getMe);

module.exports = router;
```

- Error Handling Middleware

1. Add this to your server.js after all routes:

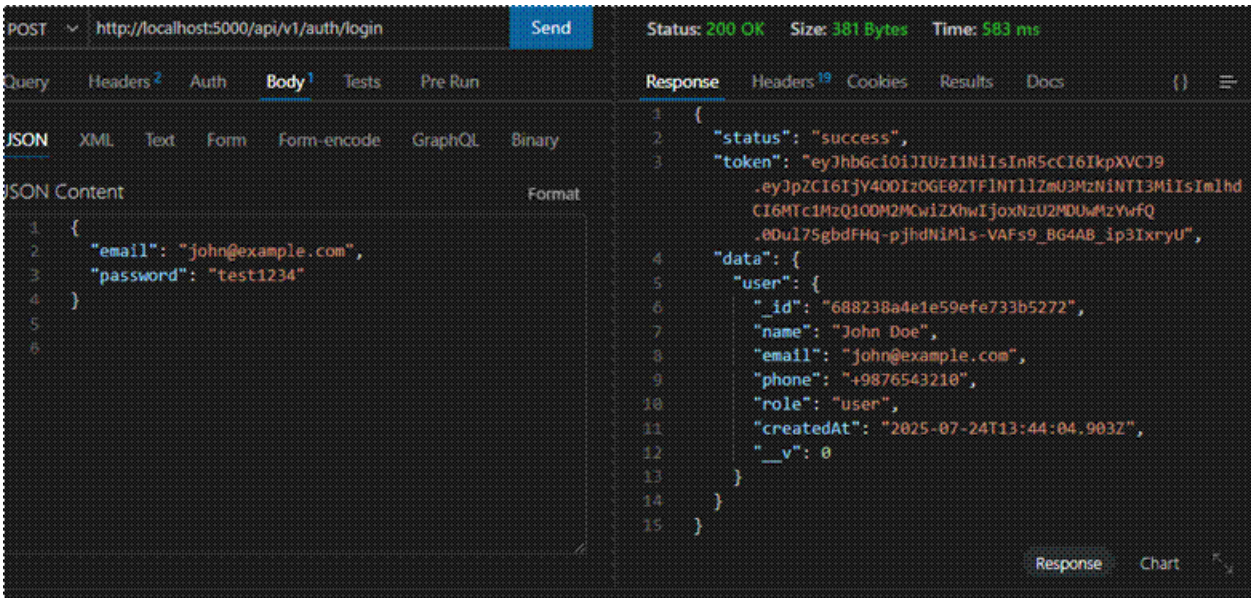
Backend > middleware > JS errorHandler.js

```
const AppError = require('../utils/appError');

const handleCastErrorDB = (err) => {
  const message = `Invalid ${err.path}: ${err.value}.`;
  return new AppError(message, 400);
};

const handleDuplicateFieldsDB = (err) => {
  const value = err.errmsg?.match(/([""])(\\?.)*?\1/)?.[0] || 'duplicate value';
  const message = `Duplicate field value: ${value}. Please use another value!`;
  return new AppError(message, 400);
};

const handleValidationErrorDB = (err) => {
  const errors = Object.values(err.errors).map((el) => el.message);
  const message = `Invalid input data. ${errors.join('. ')}`;
  return new AppError(message, 400);
};

const handleJWTError = () =>
  new AppError('Invalid token. Please log in again!', 401);

const handleJWTExpiredError = () =>
  new AppError('Your token has expired! Please log in again.', 401);
```

Start Your Server

2. Run the backend server with:
   bash
   node server.js

■ Testing The Backend by using the thunder client:



3.Database Development

Creating Data Schemas

User Schema:

1. Schema: userSchema
2. Model: 'User'
3. The User schema defines fields like username, email, and password with specified constraints such as minimum and maximum lengths and uniqueness.
   - It represents user data in the application, ensuring data integrity and security for user accounts.

```
backend > JS server.js > ...
1   const express = require('express');
2   const cors = require('cors');
3   const connectDB = require('./config/database');
4   const dotenv = require('dotenv');
5
6   // Load environment variables
7   dotenv.config();
8
9   // Connect to MongoDB
10  connectDB();
11
12  const app = express();
13
14  // Middleware
15  app.use(cors());
16  app.use(express.json());
17
18  // Routes
19  app.use('/api/auth', require('./routes/authRoutes'));
20  app.use('/api/movies', require('./routes/movieRoutes'));
21  app.use('/api/cinemas', require('./routes/cinemaRoutes'));
22  app.use('/api/showtimes', require('./routes/showtimeRoutes'));
23  app.use('/api/bookings', require('./routes/bookingRoutes'));
```

Theater Schema:

4. Schema: theaterSchema
5. Model: 'Theater'
6. The theater schema captures details about theaters including names, IDs, locations, seat prices, and seat layouts.
7. It facilitates the management of theater information such as seating arrangements and pricing for movie screenings.

```
const mongoose = require("mongoose");

const theatreSchema = new mongoose.Schema({
  theatreName: {
    type: String,
    required: true,
  },
  adminName: {
    type: String,
    required: true,
  },
  theatreId: {
    type: String,
    required: true,
  },
  location: {
    type: String,
    required: true,
  },
  balconySeatPrice: {
    type: Number,
    required: true,
  },
  middleSeatPrice: {
    type: Number,
    required: true,
  },
  lowerSeatPrice: {
    type: Number,
    required: true,
  },
  balconySeats: {
    type: Object,
    required: true,
  },
  middleSeats: {
    type: Object,
    required: true,
  },
  lowerSeats: {
    type: Object,
    required: true,
  },
  adminEmail: {
    type: String,
    required: true,
```

Show Schema:

8. Schema: showSchema
9. Model: 'Show'
10. The Show schema represents individual movie showings with attributes such as admin email, movie ID, theater name, date, time, and ticket details.
11. It organizes and stores data related to movie screenings, enabling users to

browse and book show-times effectively.

```
const mongoose = require("mongoose");

const showSchema = new mongoose.Schema({
  //show created by admin
  adminEmail: {
    type: String,
    required: true,
  },
  showId: {
    type: String,
    required: true,
  },
  movieId: {
    type: String,
    required: true,
  },

  theatreName: {
    type: String,
    required: true,
  },
  showdate: {
    type: String,
    required: true,
  },
  showtime: {
    type: String,
    required: true,
  },
  tickets: {
    type: Object,
  },
});

module.exports = mongoose.model("show", showSchema);
```

Movie Schema:

12. Schema: movieSchema
13. Model: 'Movie'
14. The Movie schema defines properties for movies including name, description, genres, release date, runtime, certification, media format, and associated show

IDs.

15. It encapsulates movie data, facilitating organization and retrieval of information about available films for users to explore and book.

```
const mongoose = require("mongoose");

const movieSchema = new mongoose.Schema({
  movieName: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  genres: {
    type: String,
    required: true,
  },
  releaseDate: {
    type: String,
    required: true,
  },
  runtime: {
    type: Number,
    required: true,
  },
  certification: {
    type: String,
    required: true,
  },
  media: {
    type: String,
    required: true,
  },
  movieId: {
    type: String,
    required: true,
  },
  //contains showId's
  shows: [{ type: String }],
});

module.exports = mongoose.model("Movie", movieSchema);
```

Favorite Schema:

16. Schema: favoriteSchema

17. Model: 'Favorite'

18. The Favorite schema records user preferences by storing user email and movie ID pairs for favorite movies.

19. It enables users to bookmark and access their preferred movies easily, enhancing their experience on the platform.

```
const mongoose = require("mongoose");

const favoriteSchema = new mongoose.Schema({
  userEmail: {
    type: String,
    required: true,
  },
  movieId: {
    type: String,
    required: true,
  },
});

module.exports = mongoose.model("Favorite",
favoriteSchema);
```

Booking Schema:

20. Schema: bookingModel
21. Model: 'Booking'
22. The Booking schema captures details of user bookings including booking ID, user email, show ID, and ticket data.
23. It facilitates the management and tracking of user reservations, ensuring a

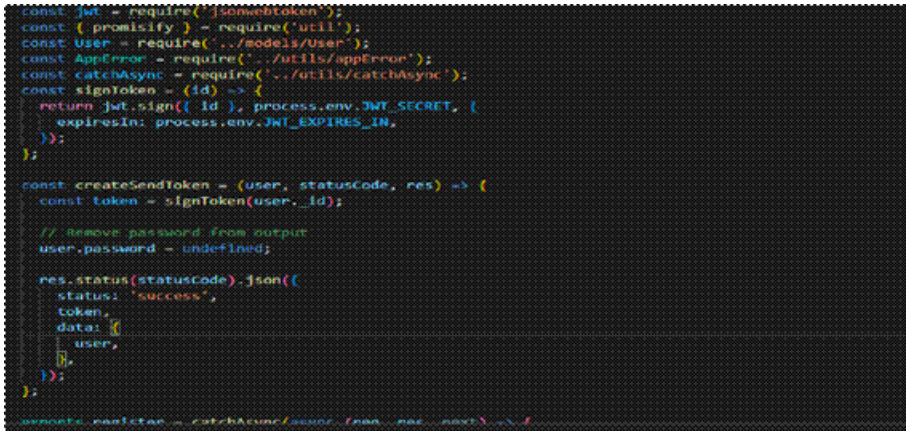
smooth booking process and accurate record-keeping.

```
const mongoose = require("mongoose");

const bookingModel = new mongoose.Schema({
  bookingId: {
    type: String,
    required: true,
  },
  userEmail: {
    type: String,
    required: true,
  },
  showId: {
    type: String,
    required: true,
  },
  ticketsData: {
    type: Object,
    requires: true,
  },
});

module.exports = mongoose.model("Booking",
bookingModel);
```

Admin Schema:

24. Schema: adminSchema
25. Model: 'Admin'
26. The Admin schema defines fields for admin accounts such as username, email, and password with specified constraints.
27. It represents administrative users in the system, providing access to privileged

functionalities for managing the application.



4.Frontend development

1. Setup React Application

Step 1: Create React App

Run this command in your project root:

bash

npm create vite@latest

cd frontend

Step 2: Install Essential Libraries

bash

npm install axios react-router-dom react-icons react-toastify @mui/material @mui/iconsmaterial
npm install -D tailwindcss Postcss autoprefixer
npx tailwindcss init -p

1. axios: For API calls
2. react-router-dom: For routing
3. @mui/material: UI components

Step 3: Folder Structure

.next
app
app\admin
app\bookings
app\cinemas
app\login
app\movies
app\profile
app\register
app\globals.css
app\layout.jsx
app\loading.jsx
app\page.jsx
backend
backend\config
backend\middleware
backend\models
backend\models\Booking.js
backend\models\Cinema.js
backend\models\Movie.js
backend\models\Showtime.js
backend\models\User.js
backend\node_modules
backend\routes
backend\scripts
backend\.env
backend\.gitignore
backend\package-lock.json
backend\package.json
backend\README.md
backend\server.js
components
components\ui
components\ui\accordion.tsx
components\ui\alert-dialog.tsx

components\ui\alert.tsx
components\ui\aspect-ratio.tsx
components\ui\avatar.tsx
components\ui\badge.tsx
components\ui\breadcrumb.tsx
components\ui\button.tsx
components\ui\calendar.tsx
components\ui\card.tsx
components\ui\carousel.tsx
components\ui\chart.tsx
components\ui\checkbox.tsx
components\ui\collapsible.tsx
components\ui\command.tsx
components\ui\context-menu.tsx
components\ui\dialog.tsx
components\ui\drawer.tsx
components\ui\dropdown-menu.tsx
components\ui\form.tsx
components\ui\hover-card.tsx
components\ui\input-otp.tsx
components\ui\input.tsx
components\ui\label.tsx
components\ui\menubar.tsx
components\ui\navigation-menu.tsx
components\ui\pagination.tsx
components\ui\popover.tsx
components\ui\progress.tsx
components\ui\radio-group.tsx
components\ui\resizable.tsx
components\ui\scroll-area.tsx
components\ui\select.tsx
components\ui\separator.tsx
components\ui\sheet.tsx
components\ui\sidebar.tsx
components\ui\skeleton.tsx
components\ui\slider.tsx
components\ui\sonner.tsx
components\ui\switch.tsx
components\ui\table.tsx
components\ui\tabs.tsx
components\ui\textarea.tsx
components\ui\toast.tsx
components\ui\toaster.tsx

components\ui\toggle-group.tsx
components\ui\toggle.tsx
components\ui\tooltip.tsx
components\ui\use-mobile.tsx
components\ui\use-toast.ts
components\MovieCard.jsx
components\Navbar.jsx
components\PrivateRoute.jsx
components\theme-provider.tsx
contexts
contexts\AuthContext.jsx
contexts\ThemeContext.jsx
hooks
lib
lib\mockData.js
lib\utils.ts
node_modules
public
styles
.gitignore
components.json
next-env.d.ts
next.config.mjs
package-lock.json
package.json
pnpm-lock.yaml
postcss.config.mjs
tsconfig.json

2. Design UI Components

Step 1: Create Core

Components ● Header.jsx

Navigation bar with logo
Menu items (Home, Movies, Theaters)
User auth section

```
frntend > src > components > Header.jsx > Header
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { Search, MapPin, User, Menu, X } from 'lucide-react';

const locations = [
  'Visakhapatnam (Vizag)',
  'Vijayawada',
  'Guntur',
  'Nellore',
  'Tirupati',
  'Kurnool',
  'Rajahmundry',
  'Eluru',
  'Anantapur',
  'Kadapa',
  'Chittoor',
  'Srikakulam',
  'Narasaraopet',
  'Mangalagiri',
  'Kakinada',
  'Tadepalligudem',
  'Bhimavaram'
];
```

- **Footer.jsx**

  Copyright information

  Contact links

  Social media icons

```
frntend > src > components > Footer.jsx > Footer
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import { Facebook, Twitter, Instagram, Youtube, Mail, Phone, MapPin, Star } from 'lucide-react';
import FeedbackForm from './FeedbackForm';

const Footer = () => {
  const [showFeedback, setShowFeedback] = useState(false);

  return (
    <>
      <footer className="bg-gradient-to-r from-gray-900 via-gray-800 to-gray-900 text-white">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-12">
          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8">
            {/* Company Info */}
            <div className="space-y-4">
              <div className="flex items-center space-x-2">
                <div className="bg-gradient-to-r from-red-600 to-red-700 text-white px-4 py-3 roun
                  IMovie
                </div>
              </div>
              <p className="text-gray-300 text-sm leading-relaxed">
                Your premier destination for movie ticket booking in Andhra Pradesh.
                Experience the magic of cinema with the best theaters and latest movies.
```

FeedbackForm.jsx

Rating input

Comment textarea

Submission button

frntend > src > components > FeedbackForm.jsx > ...

```jsx
import React, { useState } from 'react';
import { X, Star, Send } from 'lucide-react';

const FeedbackForm = ({ onClose }) => {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    phone: '',
    rating: 0,
    category: 'general',
    subject: '',
    message: ''
  });
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleRatingClick = (rating) => {
```

Step2: Implement Styling

1. Configure Tailwind in tailwind.config.js
2. Add base styles in index.css
3. Use utility classes for components

frntend > src > # index.css > ...

```css
@tailwind base;
@tailwind components;
@tailwind utilities;

/* Custom scrollbar */
::-webkit-scrollbar {
  width: 8px;
}

::-webkit-scrollbar-track {
  background: #f1f1f1;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb {
  background: #dc2626;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
  background: #b91c1c;
}

/* Smooth scrolling */
```

Step3: Set Up Routing

Define routes in App.jsx

1. Home (/)
2. Movies (/movies)
3. Movie Details (/movies/:id)

4. Cinema Selection (/cinemas/:movieId)
5. Seat Selection (/book/:showtimeId)
6. Payment (/payment)
7. Booking Confirmation (/confirmation)
8. Login (/login)
9. Register (/register)
10. User Dashboard (/dashboard)
11. Admin Dashboard (/admin)

frntend > src > App.jsx > ...

```jsx
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import Header from './components/Header';
import Footer from './components/Footer';
import Home from './pages/Home';
import MovieDetails from './pages/MovieDetails';
import CinemaSelection from './pages/CinemaSelection';
import SeatSelection from './pages/SeatSelection';
import Payment from './pages/Payment';
import BookingConfirmation from './pages/BookingConfirmation';
import UserDashboard from './pages/UserDashboard';
import AdminDashboard from './pages/AdminDashboard';
import Login from './pages/Login';
import Register from './pages/Register';

function App() {
  const [user, setUser] = useState(null);
  const [selectedLocation, setSelectedLocation] = useState('Visakhapatnam (Vizag)');

  useEffect(() => {
    // Check for logged in user
    const userData = localStorage.getItem('userData');
    if (userData) {
      setUser(JSON.parse(userData));
```

3. Implement Frontend Logic

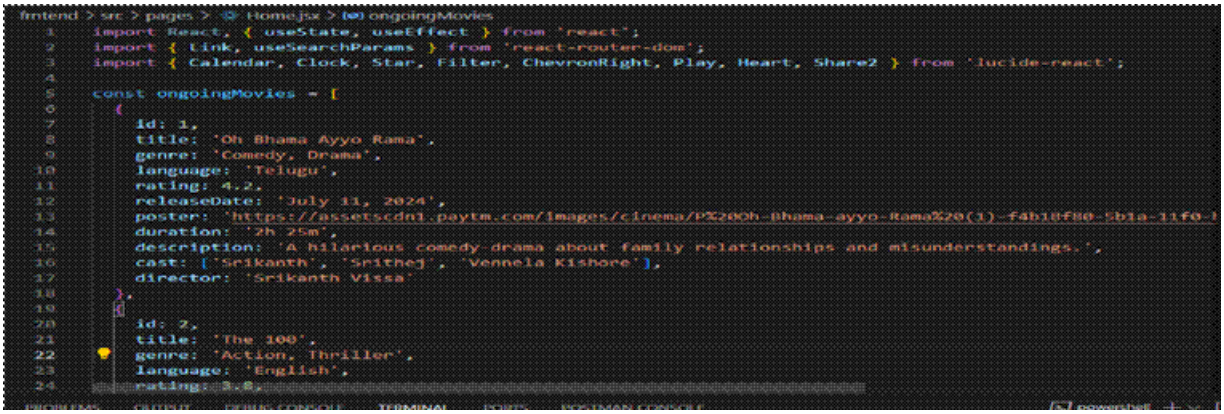
Step1: API Services

1. Create services/movieService.js: ● fetchMovies()
    a. getMovieDetails(id)
    b. getShowtimes(movieId)
2. Create services/bookingService.js:
    a. bookSeats(showtimeId, seats)
    b. getBookingHistory()

Step 2: Data Binding

1. Home Page:

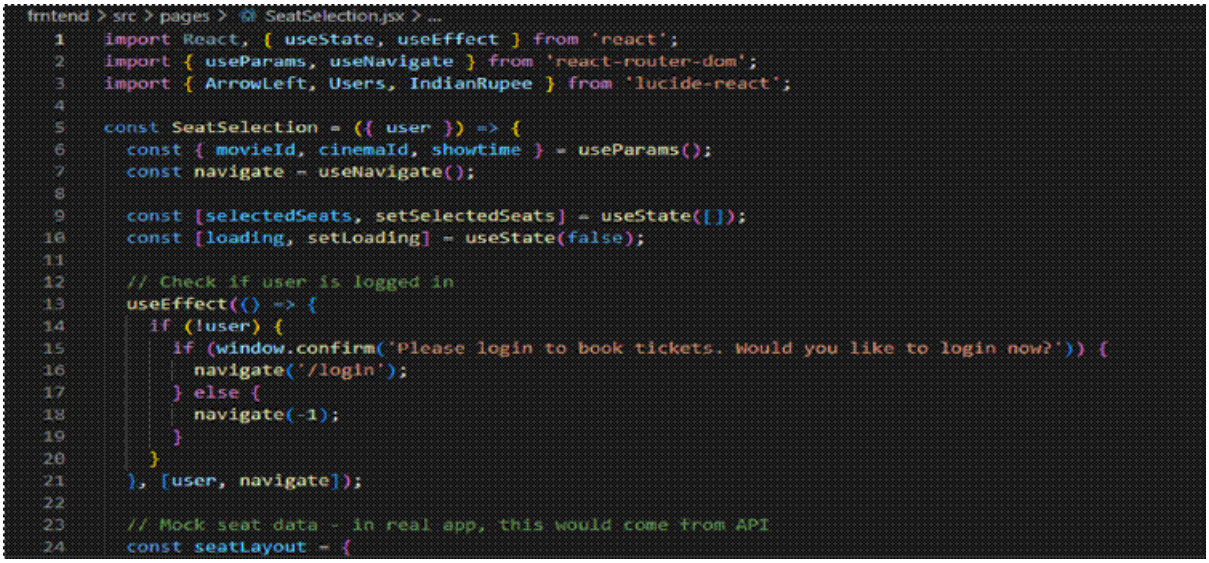
a. Fetch and display featured movies ● Implement carousel for promot



2. Movie Details:
    a. Fetch movie data by ID
    b. Display showtime availability

Seat Selection:

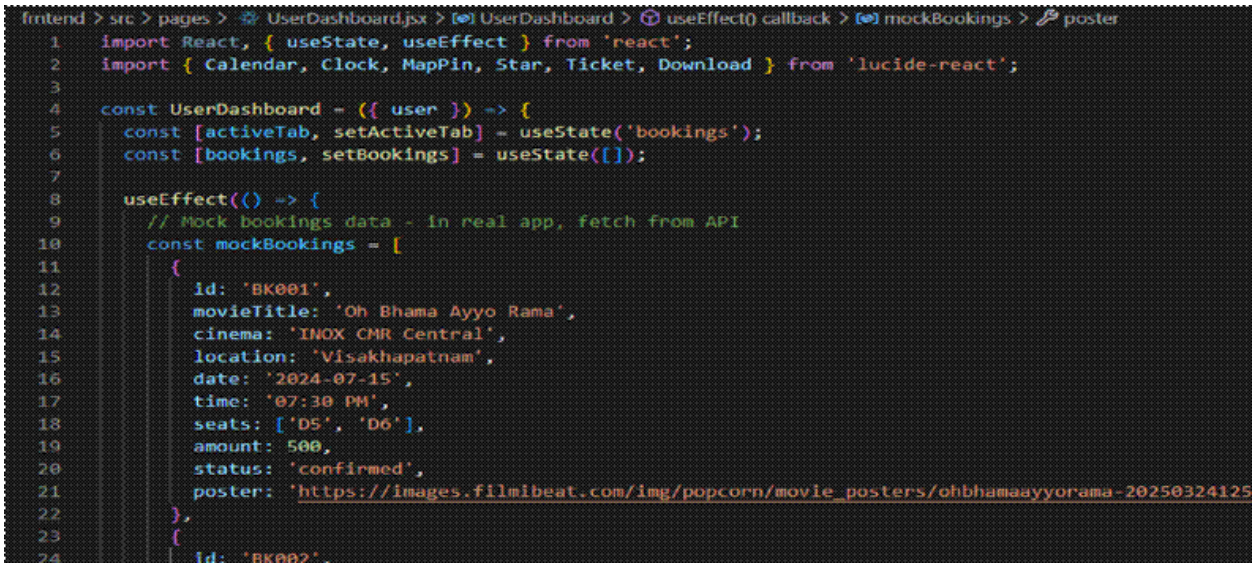
c. Fetch available seats
d. Track selected seats
e. Calculate the total price



Step3: State Management

1. Use Context API for:

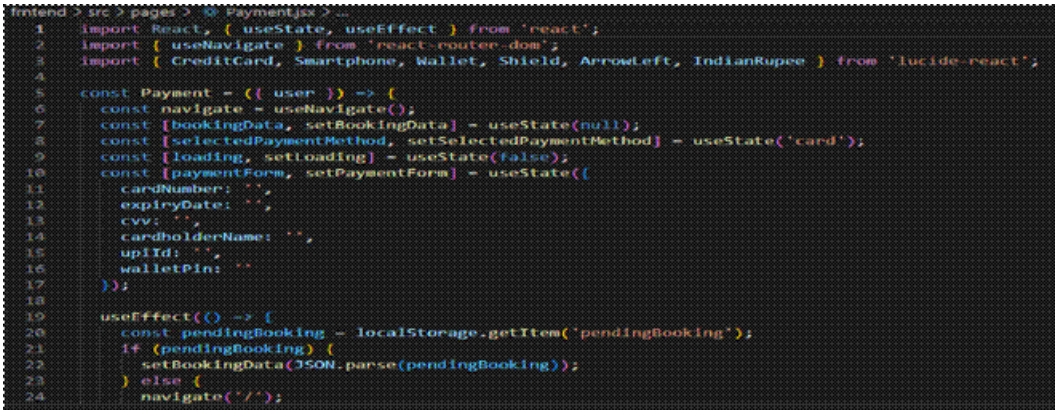
1. User authentication
2. Booking process
3. UI preferences



1. . Authentication Flow
    a. Implement login/logout
    b. Store JWT tokens
    c. Protected routes

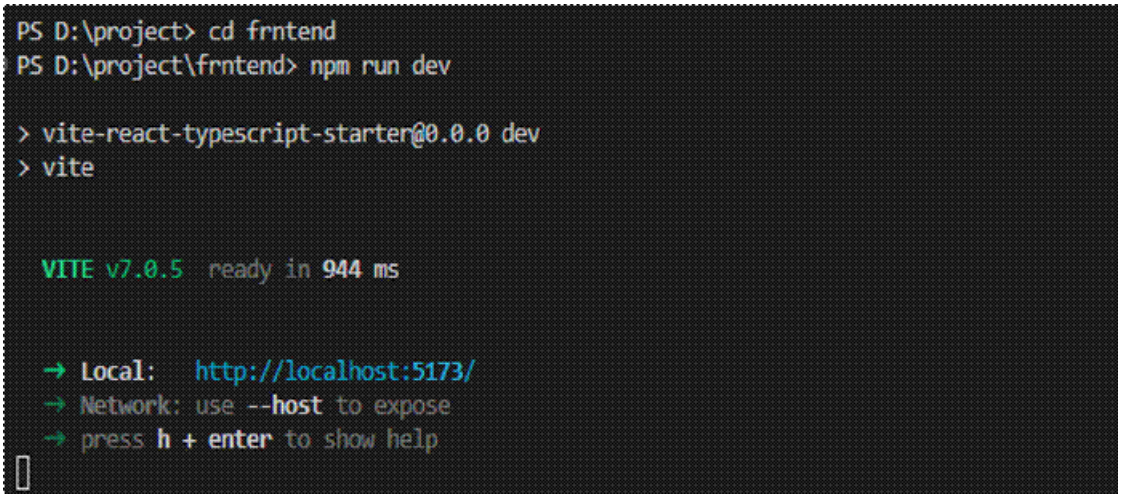
2. . Payment Integration

a. Setup payment form
b. Connect to payment gateway ● Handle success/failure



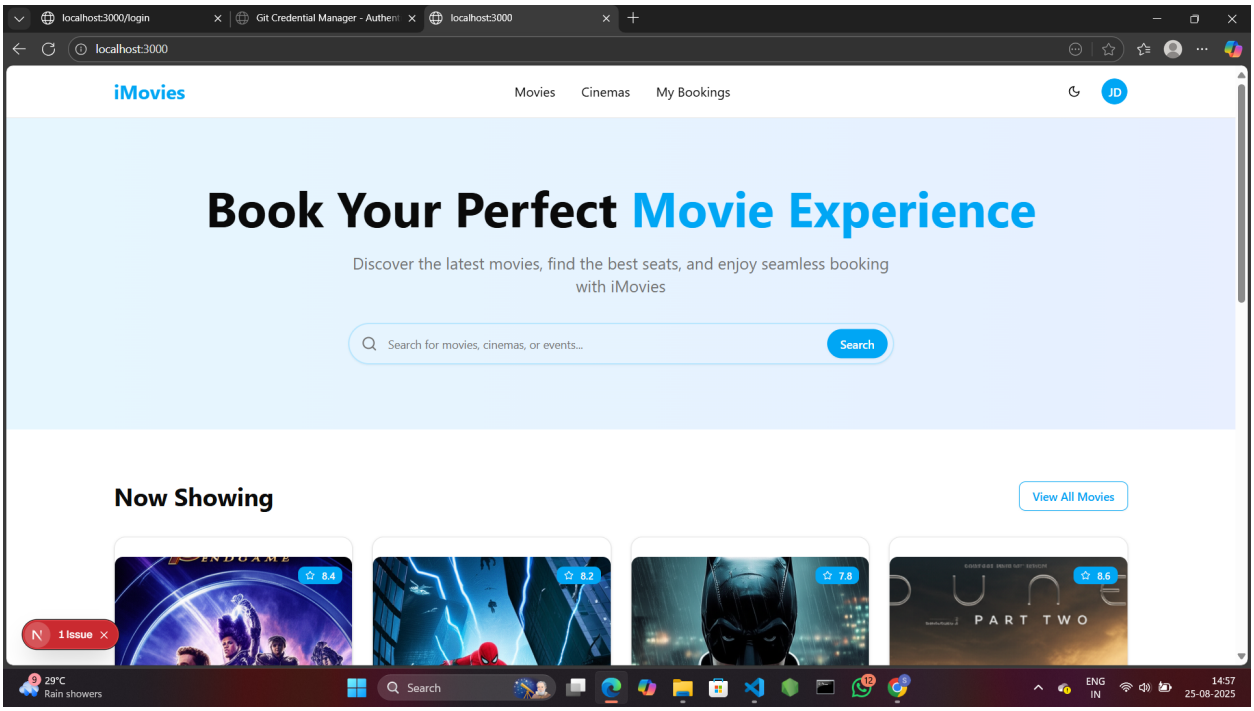
Step4: Run the Application

bash
npm run
dev

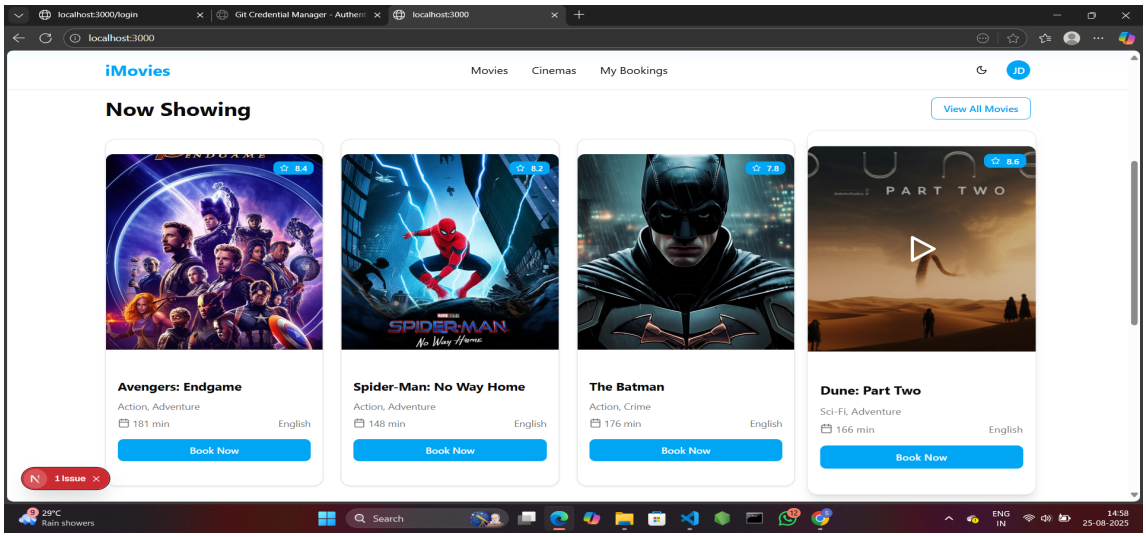


5.Project Implementation

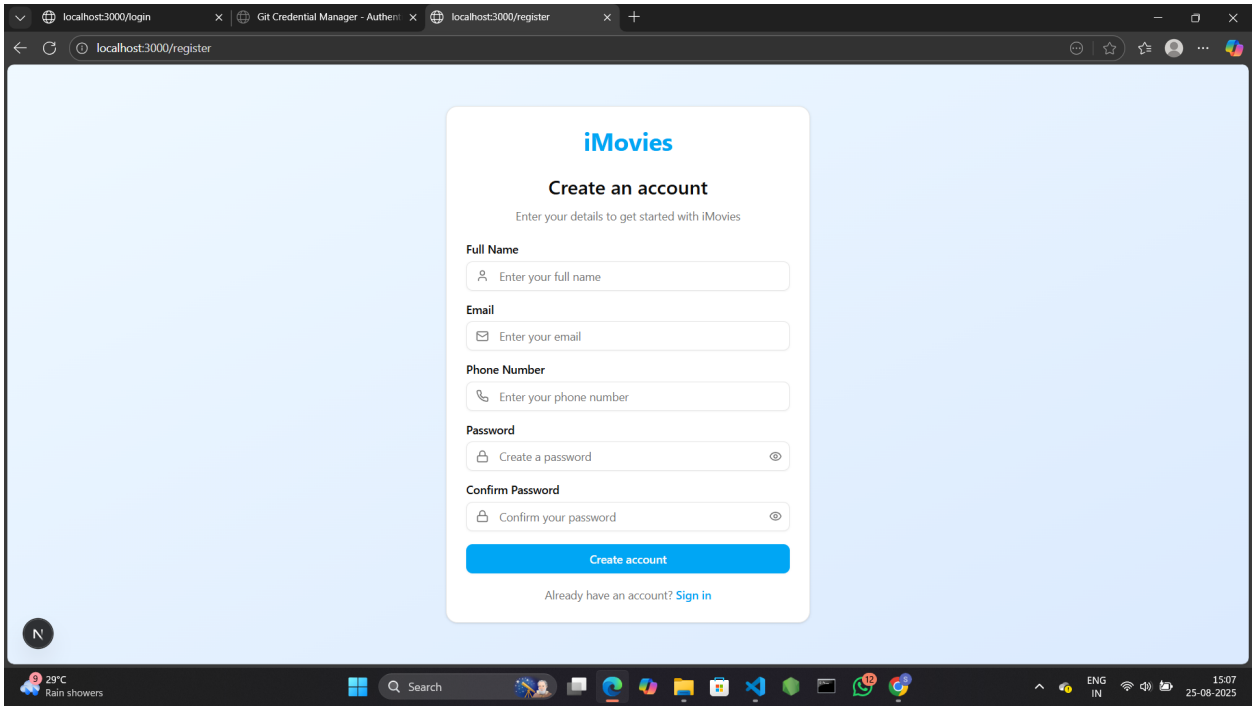
Movie details page



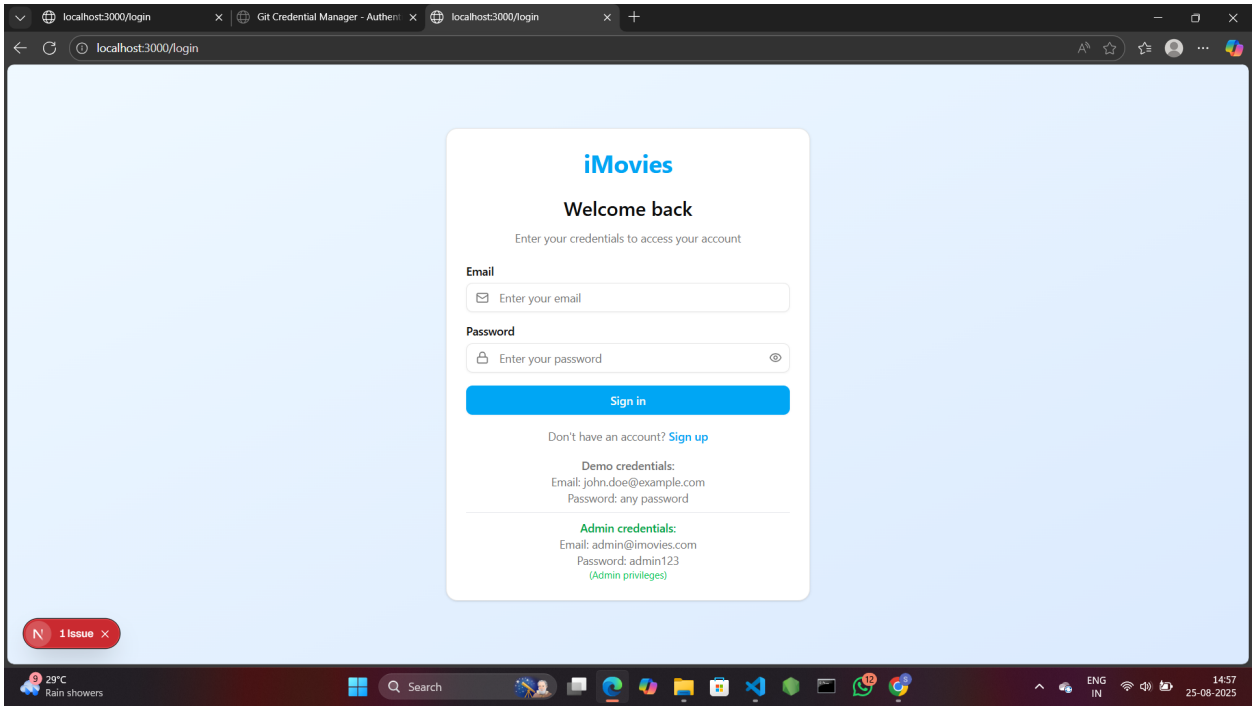
Home page



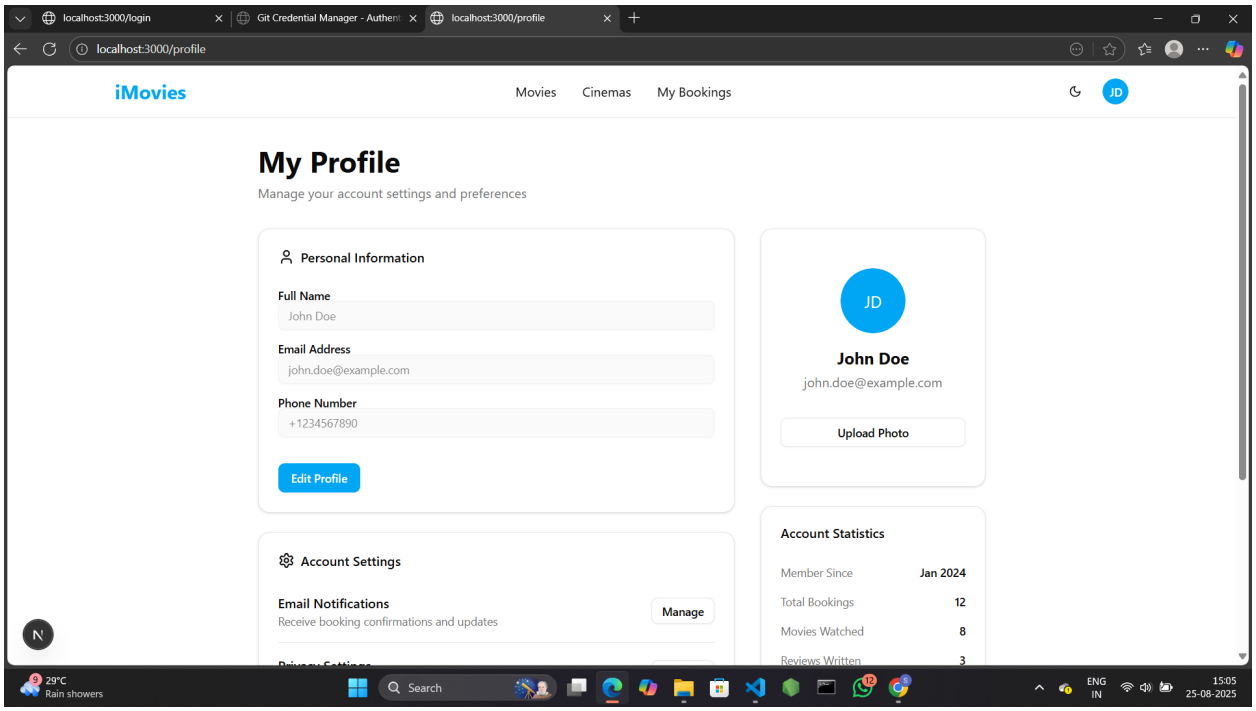
Sign up page



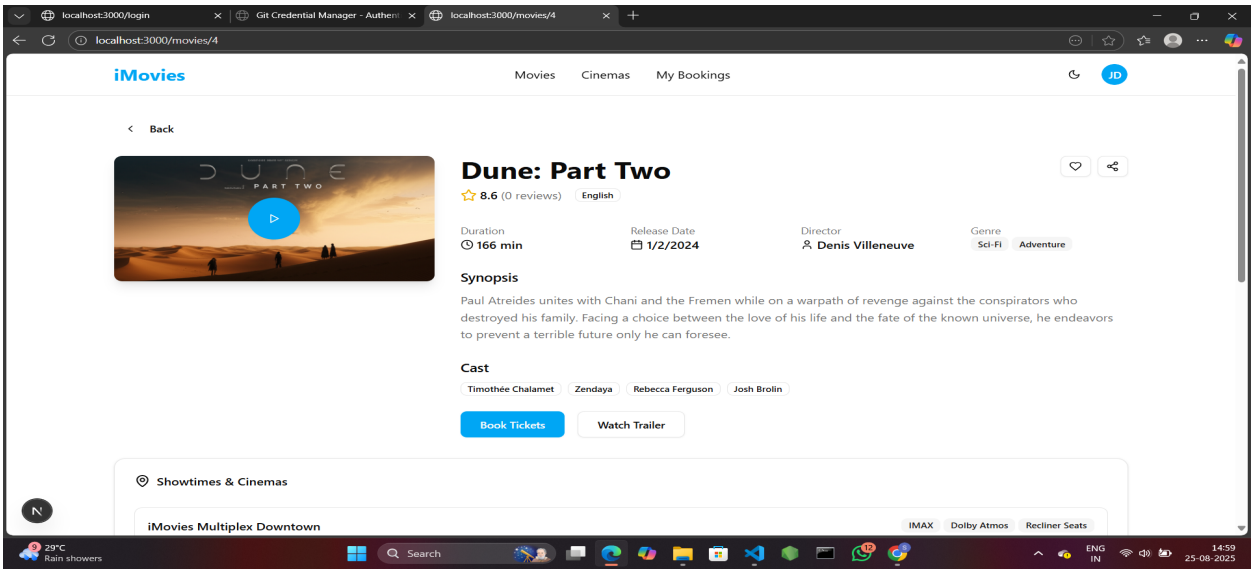
Sign in page

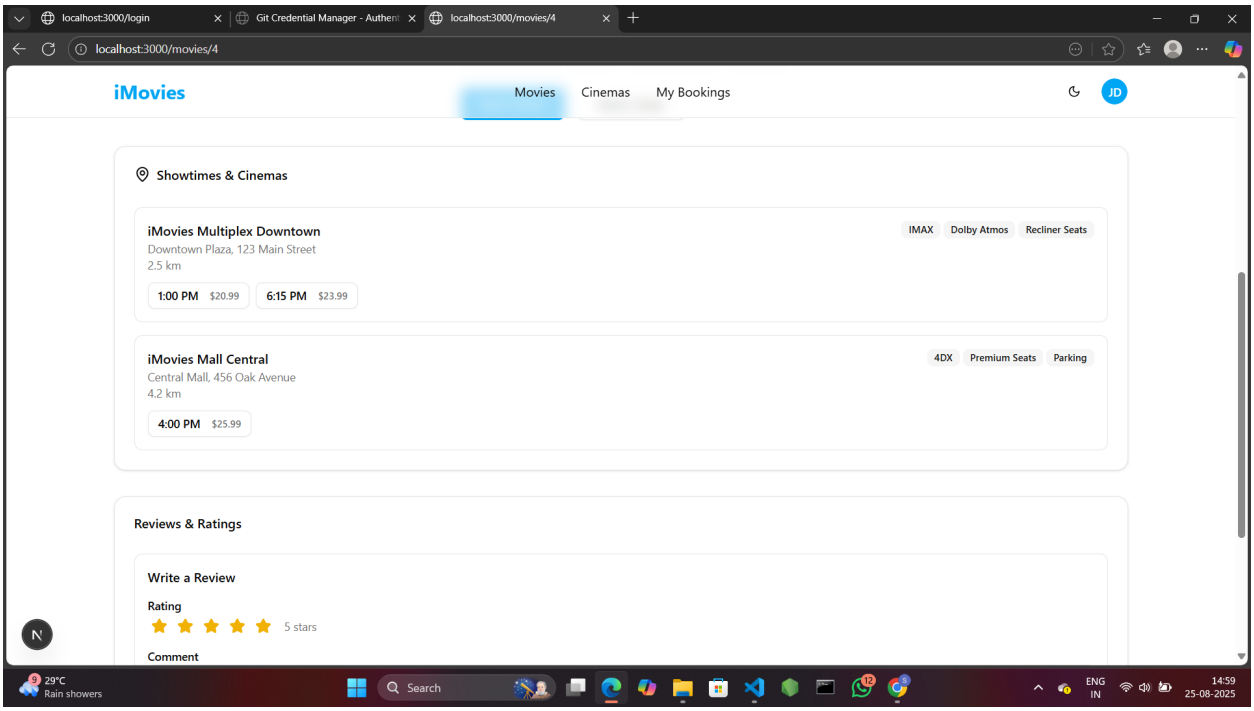


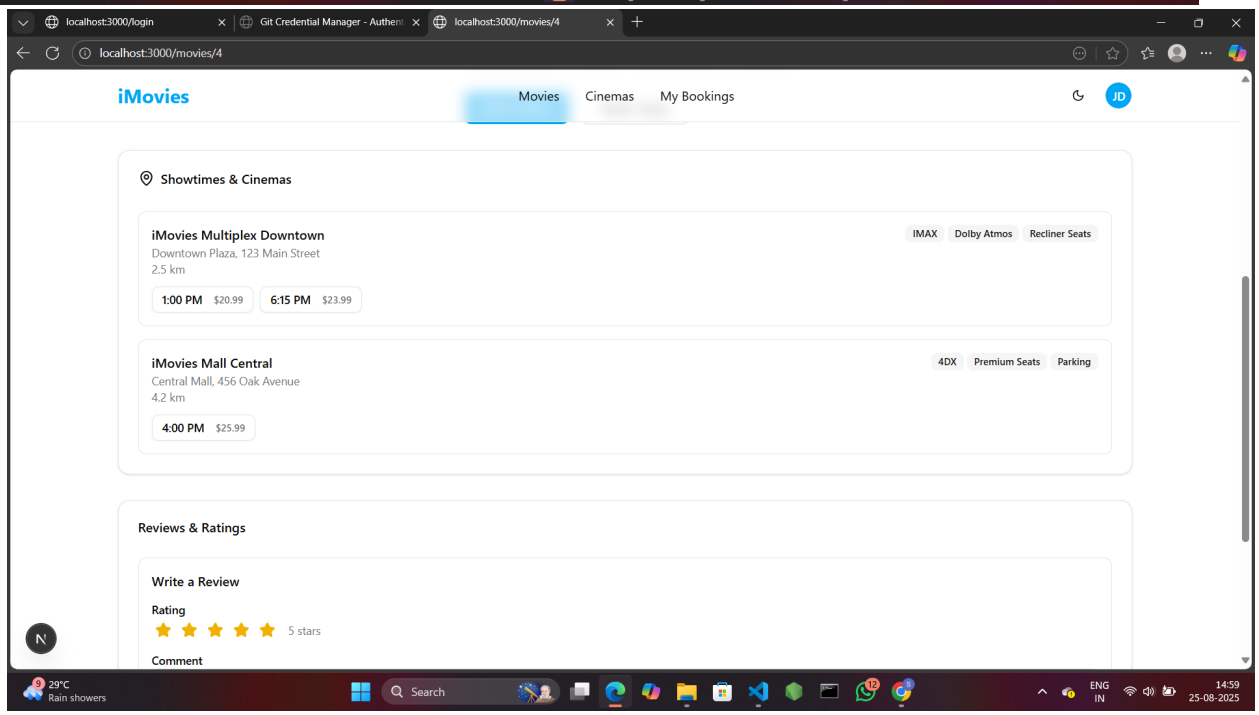
User Profile Page



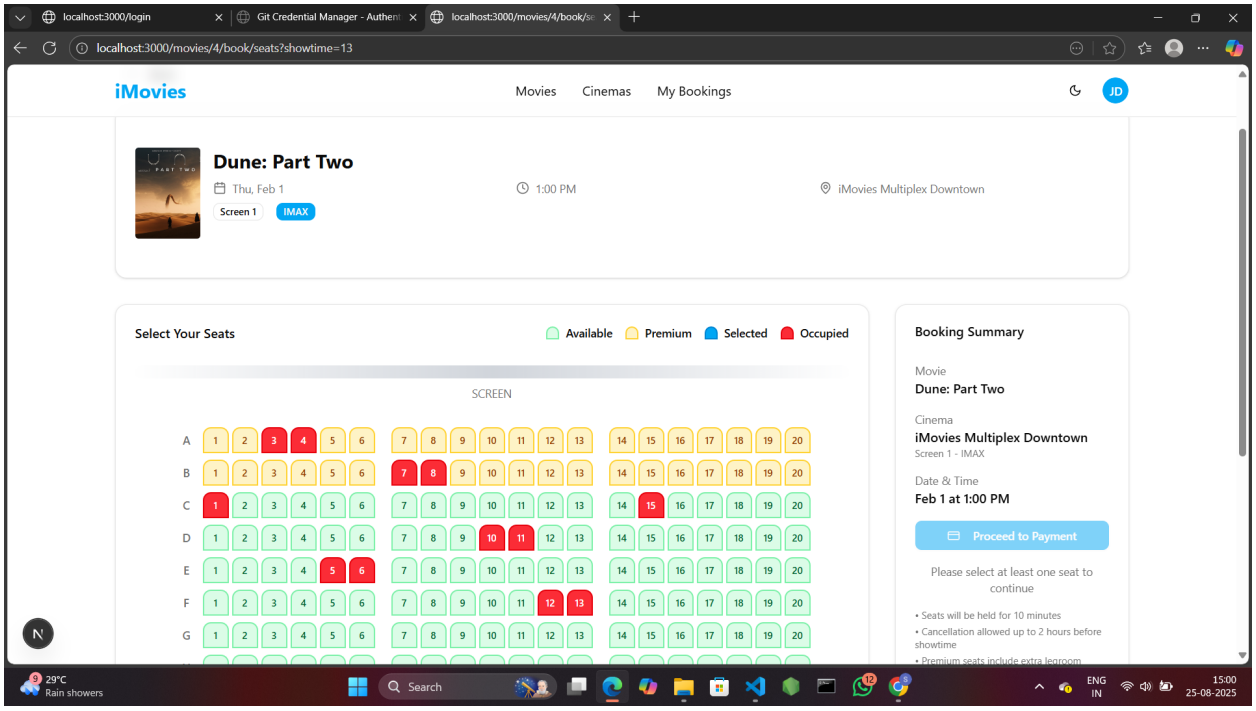
Bookings Page



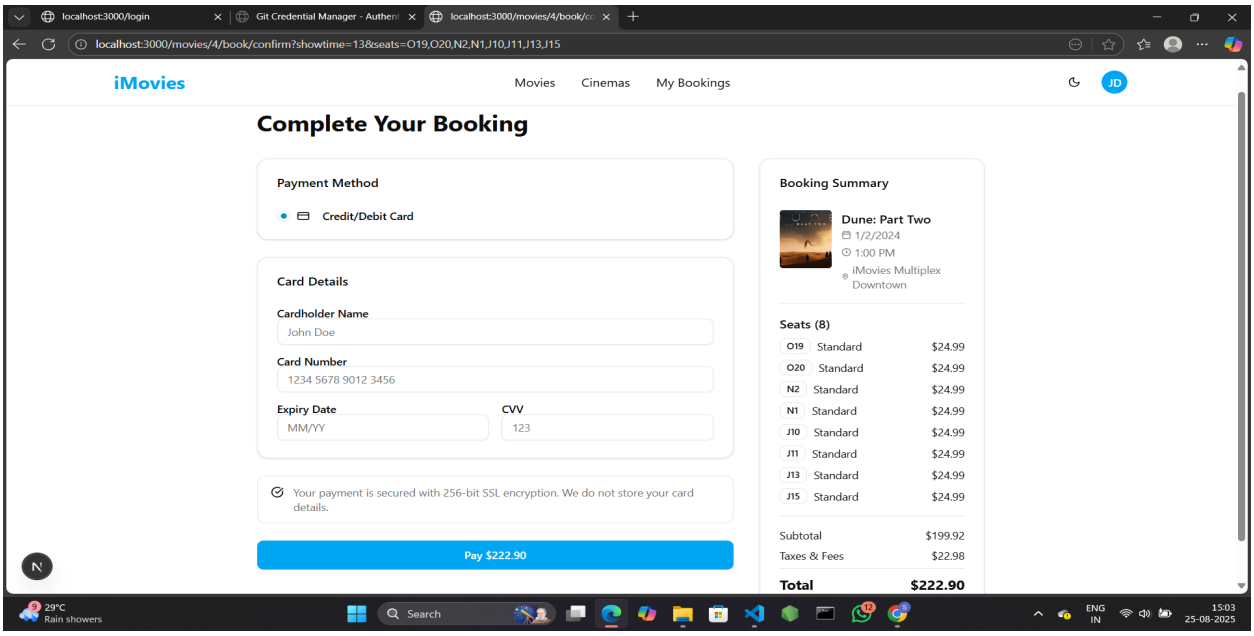

localhost:3000/login
Git Credential Manager - Authent
localhost:3000/movies/4
localhost:3000/movies/4
iMovies
Movies
Cinemas
My Bookings
JD
Showtimes & Cinemas
iMovies Multiplex Downtown
Downtown Plaza, 123 Main Street
2.5 km
IMAX
Dolby Atmos
Recliner Seats
1:00 PM $20.99
6:15 PM $23.99
iMovies Mall Central
Central Mall, 456 Oak Avenue
4.2 km
4DX
Premium Seats
Parking
4:00 PM $25.99
Reviews & Ratings
Write a Review
Rating
5 stars
Comment
29°C
Rain showers
Search
ENG
IN
14:59
25-08-2025

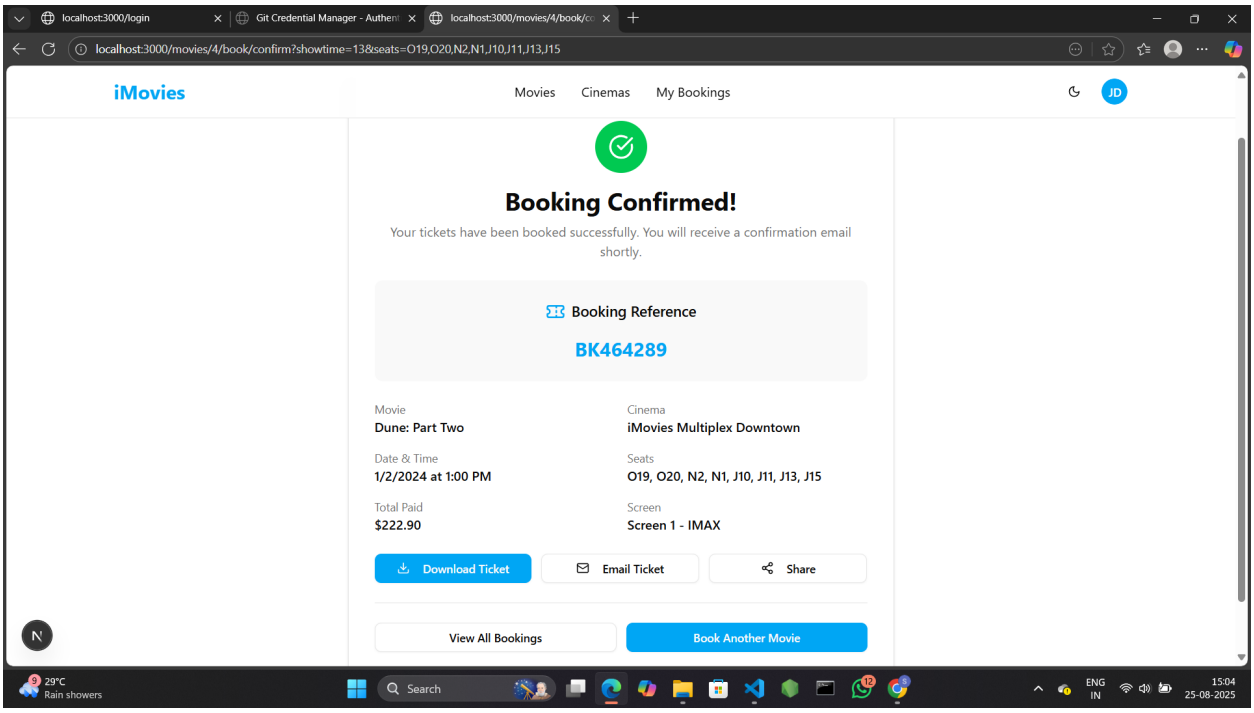

localhost:3000/login
Git Credential Manager - Authent
localhost:3000/movies/4
localhost:3000/movies/4
iMovies
Movies
Cinemas
My Bookings
JD
Showtimes & Cinemas
iMovies Multiplex Downtown
Downtown Plaza, 123 Main Street
2.5 km
IMAX
Dolby Atmos
Recliner Seats
1:00 PM $20.99
6:15 PM $23.99
iMovies Mall Central
Central Mall, 456 Oak Avenue
4.2 km
4DX
Premium Seats
Parking
4:00 PM $25.99
Reviews & Ratings
Write a Review
Rating
5 stars
Comment
29°C
Rain showers
Search
ENG
IN
14:59
25-08-2025

Seat Selection page

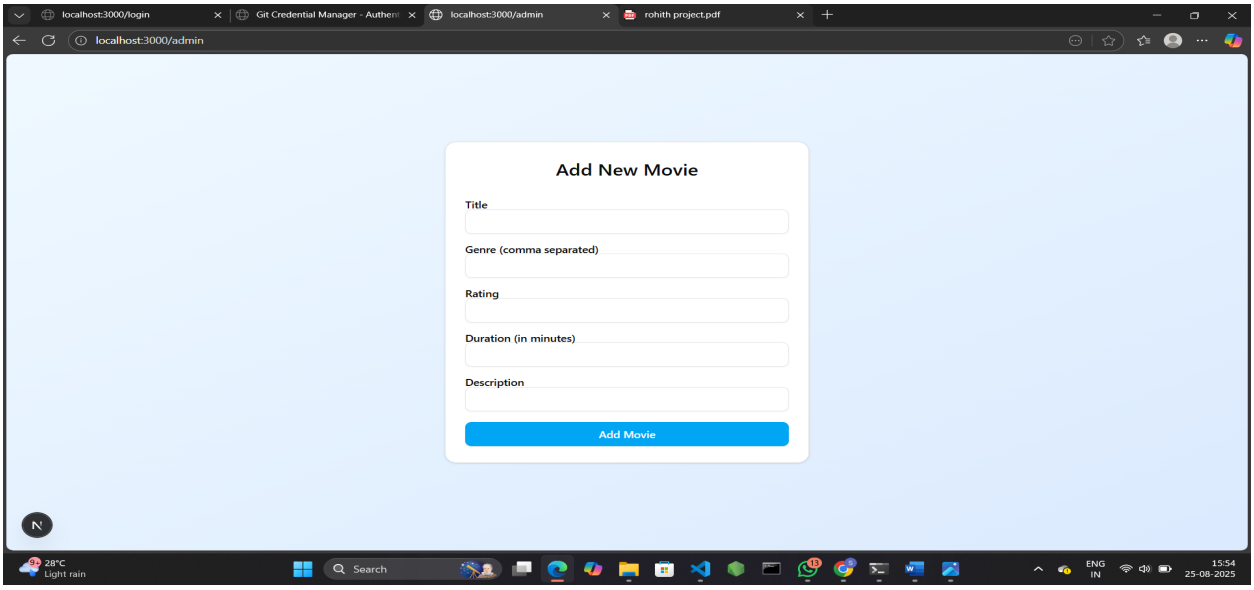


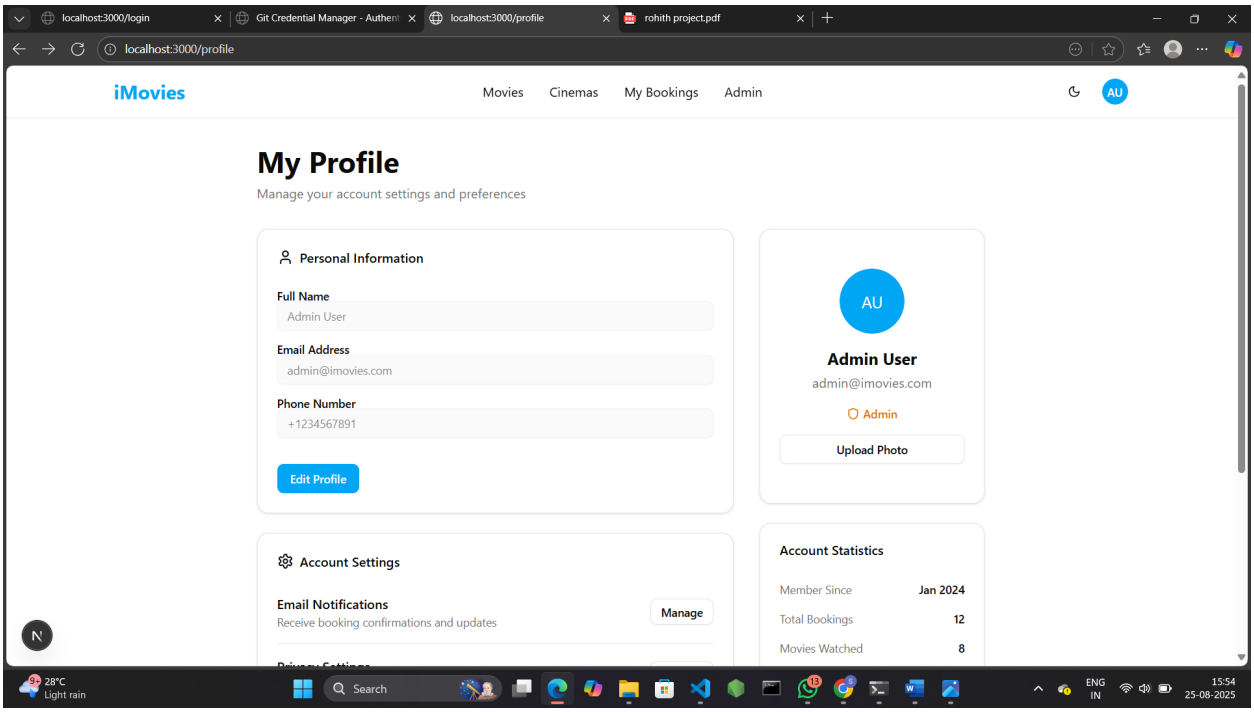
Ticket Payment Success page

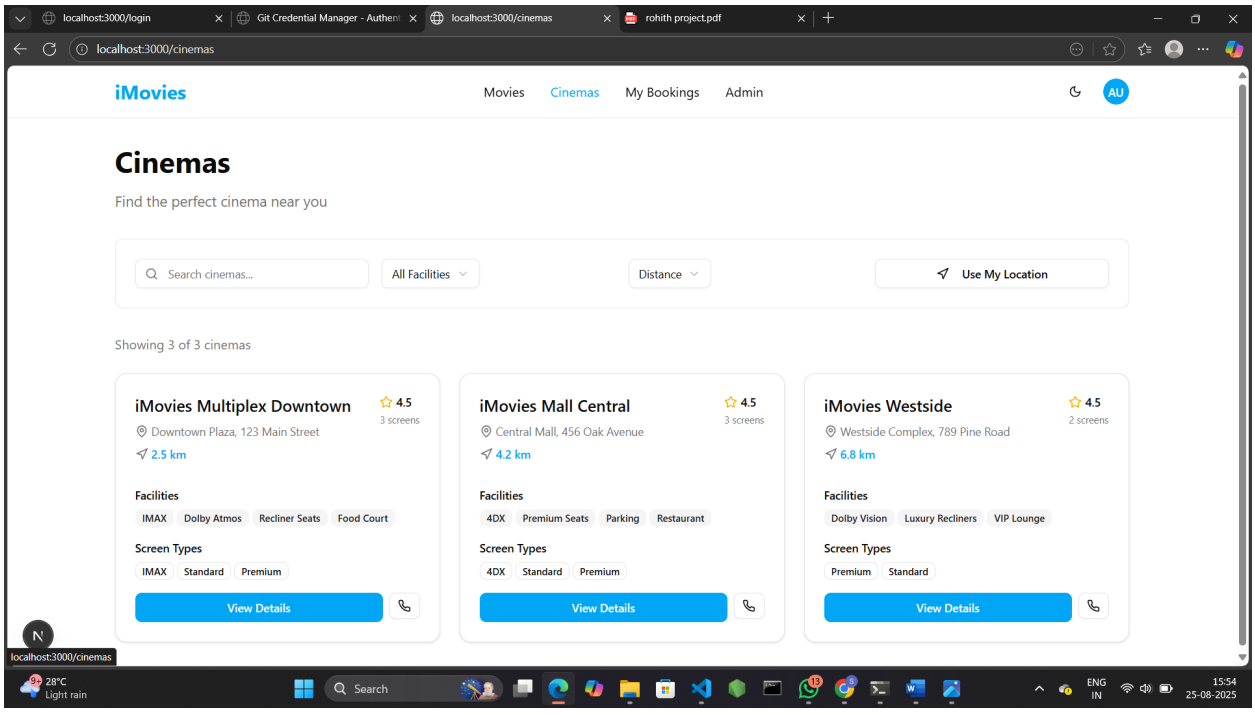




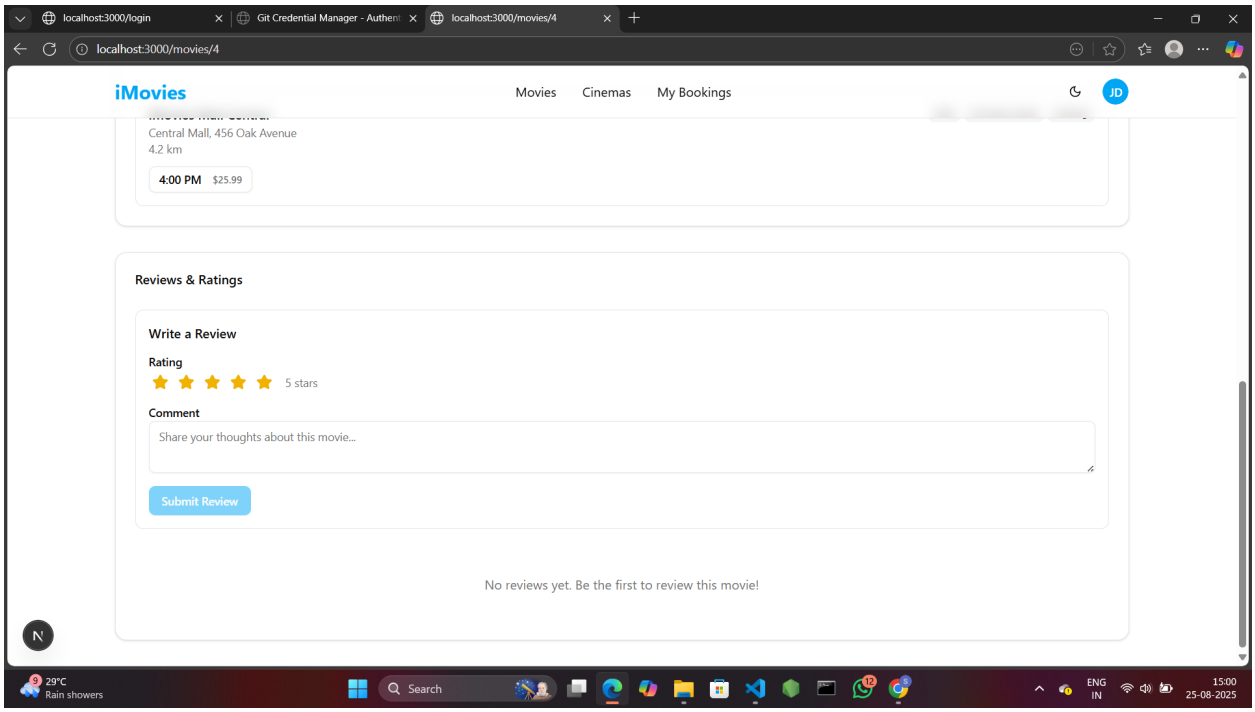
Admin DashBoard Page




localhost:3000/login
Git Credential Manager - Authent
localhost:3000/profile
rohith project.pdf
localhost:3000/profile
iMovies
Movies
Cinemas
My Bookings
Admin
AU
My Profile
Manage your account settings and preferences
Personal Information
Full Name
Admin User
Email Address
admin@imovies.com
Phone Number
+1234567891
Edit Profile
AU
Admin User
admin@imovies.com
Admin
Upload Photo
Account Settings
Email Notifications
Receive booking confirmations and updates
Manage
Account Statistics
Member Since
Jan 2024
Total Bookings
12
Movies Watched
8
28°C
Light rain
Search
ENG
IN
15:54
25-08-2025


localhost:3000/login
Git Credential Manager - Authent
localhost:3000/cinemas
rohith project.pdf
localhost:3000/cinemas
iMovies
Movies
Cinemas
My Bookings
Admin
AU
Cinemas
Find the perfect cinema near you
Search cinemas...
All Facilities
Distance
Use My Location
Showing 3 of 3 cinemas
iMovies Multiplex Downtown
4.5
3 screens
Downtown Plaza, 123 Main Street
2.5 km
Facilities
IMAX
Dolby Atmos
Recliner Seats
Food Court
Screen Types
IMAX
Standard
Premium
View Details
iMovies Mall Central
4.5
3 screens
Central Mall, 456 Oak Avenue
4.2 km
Facilities
4DX
Premium Seats
Parking
Restaurant
Screen Types
4DX
Standard
Premium
View Details
iMovies Westside
4.5
2 screens
Westside Complex, 789 Pine Road
6.8 km
Facilities
Dolby Vision
Luxury Recliners
VIP Lounge
Screen Types
Premium
Standard
View Details
localhost:3000/cinemas
28°C
Light rain
Search
ENG
IN
15:54
25-08-2025

Feedback Page



Conclusion

I-Movies is a cutting-edge online movie ticket booking platform designed to revolutionize the cinema experience. Built with React and Vite for a lightning-fast, responsive interface, and powered by MongoDB for efficient data management, the system offers seamless movie browsing, real-time seat selection, and instant booking confirmations. Users can effortlessly create accounts, view show times, select their preferred seats, and receive digital tickets via email—all within a sleek, user-friendly environment. The platform's robust backend ensures smooth performance even during peak hours, while its mobile-optimized design guarantees accessibility across all devices. By combining modern web technologies with intuitive functionality, I-Movies delivers a convenient, scalable solution that eliminates the hassles of traditional ticket purchasing, making movie outings more enjoyable than ever before. The system has undergone rigorous testing to guarantee reliability, security,

and an exceptional user experience from start to finish.