I-Movies : Movie TicketBooking System

ProjectDescription:

Users can effortlessly browse movies, selectshowtime, and chooseseats from the comfort of their homes. purchase a robust client-server architecture with Express.js and MongoDB, theapp ensures efficientdata 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 providesa comprehensive guidefor setup, development, and understanding of the application's technical architecture and functionalities.

Scenario

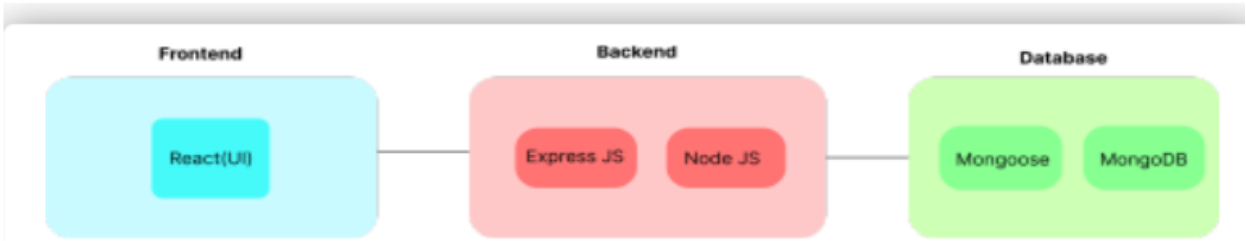
Imagine Sarah, a movie enthusiast, excitedly expecting the release of an anticipated film. With a busy schedule, she prefers the convenience of booking her movie tickets online. Logging into the iMovies app, Sarah navigates effortlessly through the user-friendly interface, browsing the list of usable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

Upon finding the movie she has been eagerly awaiting, Sarah selects her preferred cinema locationand showtime. The app displays a seating layout, allowing her to choose the perfect seats for an optimalviewing see. With a few clicks, Sarahconfirms 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, ensuringa hassle-free experience from start to finish.

Thanks to the iMovies app's intuitive design and robust features, Sarah enjoys a seamless moviegoingexperience, making her next cinemaouting 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, Watchlist, 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 iMoviesapp 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 easilyby using RESTfulApi. On the backend side, we employ Express.js frameworks to handle the server-side logic and communication.

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

PREREQUISITES

Todevelop the iMoviesapp, essential prerequisites include Node.js and npm for server-side JavaScript, MongoDBfor data storage,Express.js for backendAPI 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.

a. **Download**: https://nodejs.org/en/download/
b. **Installation Guide**: https://nodejs.org/en/download/package-manager/

2. MongoDB

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

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

3. Express.js

A Node.jsframework for backendAPI development.

1. Install via
   npm: bash
   'npm install express'

4. React.js

JavaScript library for building dynamic UI's.

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

HTML, CSS, and JavaScript

Front-end developers need fundamental knowledge.

6. Database Connectivity

Use **Mongoose** (ODMfor MongoDB) to connect Node.jswith MongoDB.

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

7. Firebase Storage (for Images)

To upload/store images:

a. **Create a FirebaseProject**: https://console.firebase.google.com/

b. **Install Firebase**

SDK: 3. bash

'npm install Firebase'

Enable Storage in Firebase Console and configure:
javascript 'import { getStorage, ref, uploadBytesResumable, getDownloadURL } from
"Firebase/storage";'

Version Control (Git):

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

9. Development Environment

Choose an IDE:

a. **VS Code**: https://code.visualstudio.com/download
b. **Sublime Text**: https://www.sublimetext.com/download
c. **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 commandsto set up the basicfolder structure: bash
mkdirimovie-app
cd imovie-app

Set Up React Frontend

Create your frontendwith 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 commandsto set up your Node.js backend: bash mkdir Backend cd Backend
npm init -y

Create Backend Structure

While still in the Backendfolder, 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 VersionControl

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

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

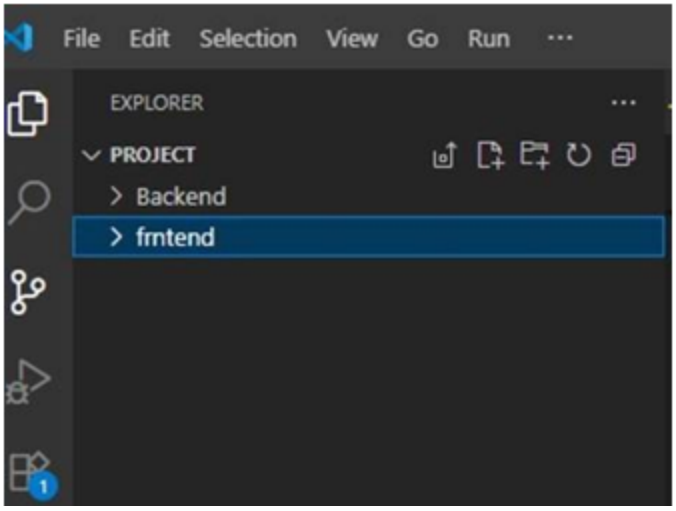
Install Backend Dependencies

Run these commands to installrequired packages: bash
cd Backend

npm install express mongoose cors dotenv ■
Start Development Servers Open two terminal windows and run: bash

Terminal 1 (frontend) cd frontendnpm start
Terminal 2 (backend) cd Backend

node server.js



2. Backend Development

Initialize Express Server

2. Navigate to your Backendfolder and installrequired dependencies: bash
cd Backend

npm install express cors body-parser mongoose jsonwebtoken dotenv bcryptjs

3. Create a new index.jsfile as your main serverfile: bash
   touch index.js

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

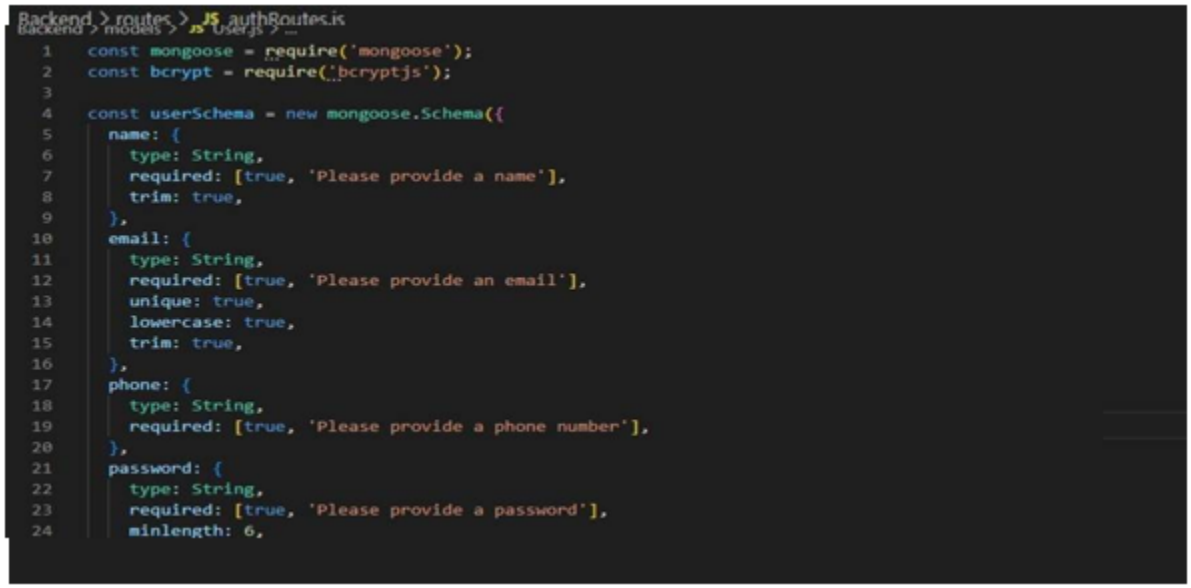
a. Basic Server Setup

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

Authentication Setup

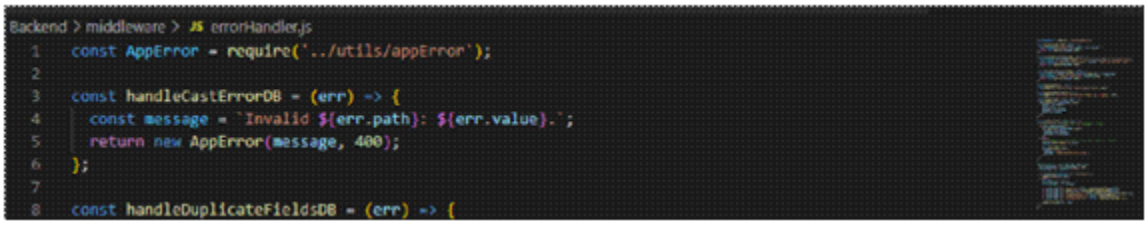
2. Create a models/User.js file for user schema:

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



Error HandlingMiddleware

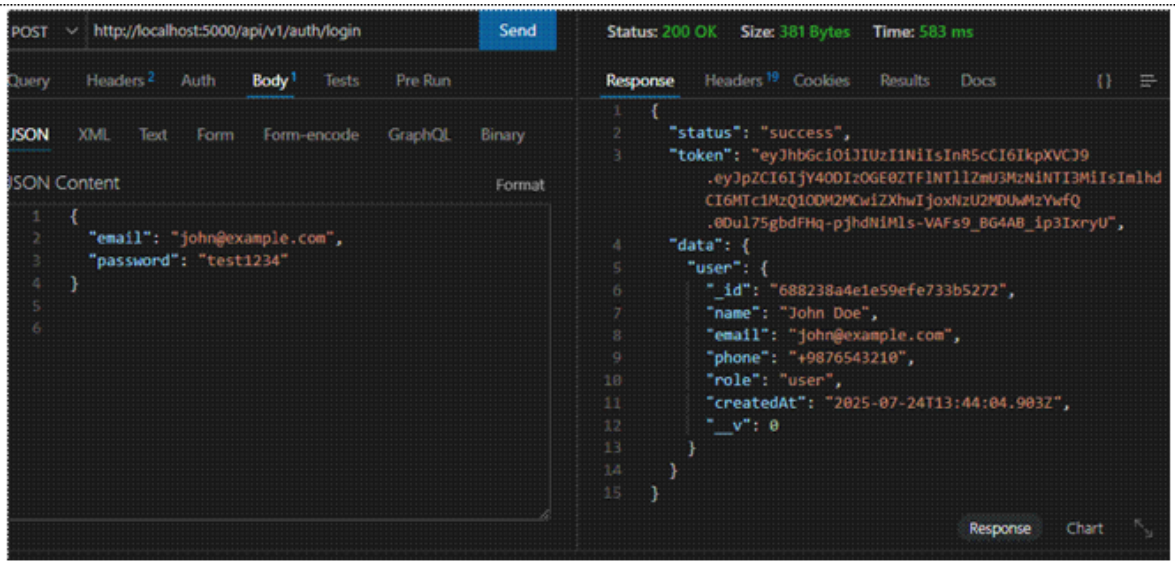
1. Add this to your server.js after all routes:



Start Your Server

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

1. **Testing The Backendby using the thunder client:**

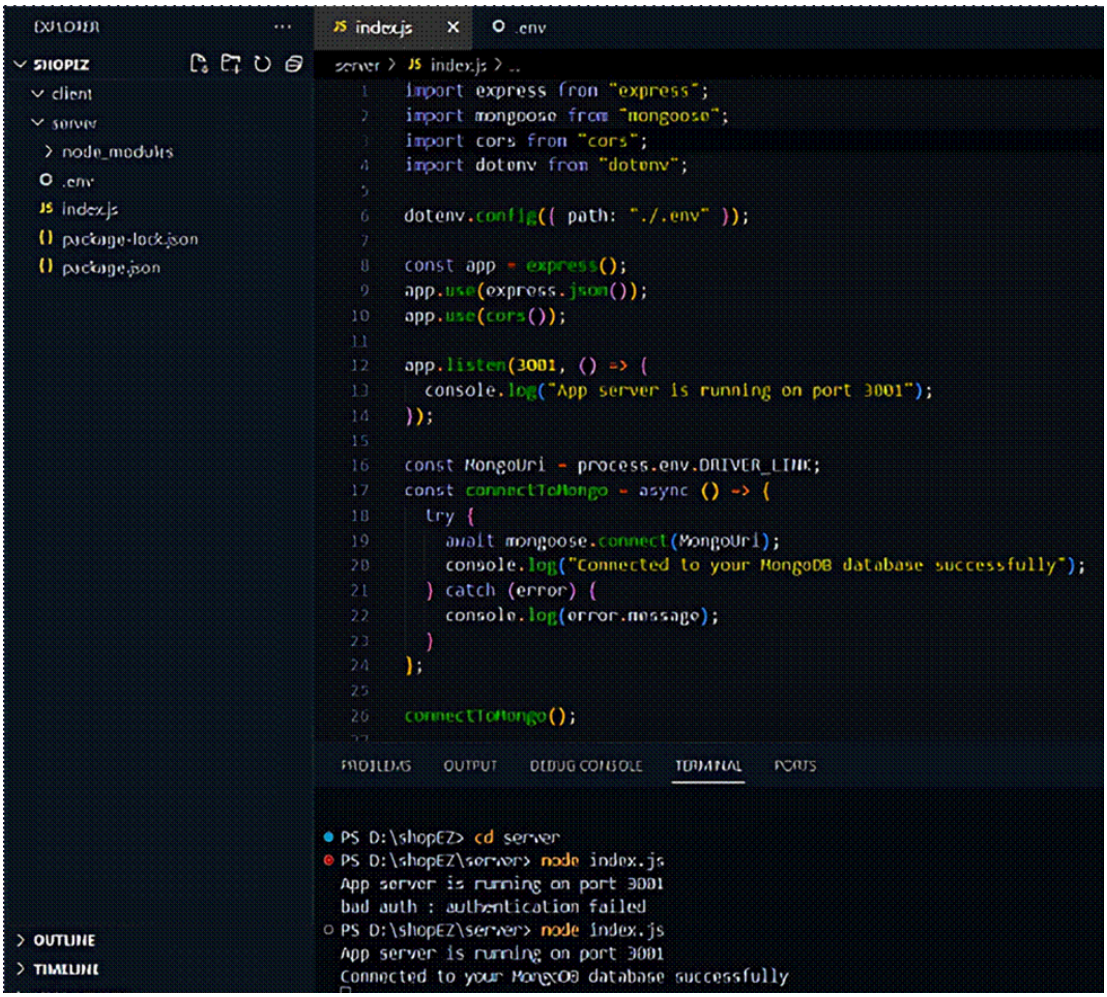


3. Database Development

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, ensuringdata integrity and security for user accounts.



Theater Schema:

4. Schema: theaterSchema

5. Model: 'Theater'

6. The theater schema capturesdetails about theatersincluding names, IDs, locations, seat prices, and seat layouts.

7. It facilitates the management of theater information such as seatingarrangements 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 schemarepresents individual movieshowings 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 schemarecords user preferences by storing user email and movie ID pairs for favorite movies.

19. It enables usersto bookmark and access theirpreferred 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 schemacaptures details of user bookingsincluding 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);
```

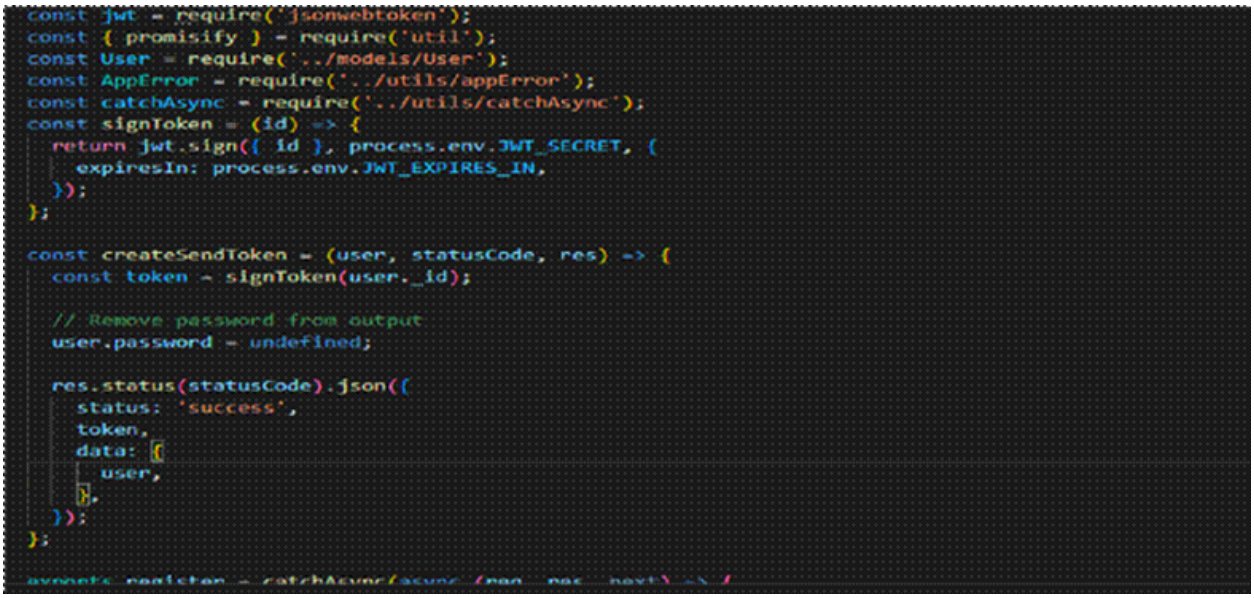
Admin Schema:

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, providingaccess to privileged functionalities for managing the application.



4. Frontend development

1. Setup React Application

Step 1: Create React App

Run this commandin your project root:

bash

npm create vite@latest
cd frontend

Step 2: Install EssentialLibraries

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

a. axios: For API calls
b. react-router-dom: For routing
c. @mui/material: UI components

Step 3: Folder Structure

frntend/

```
├── node_modules/          # All npm packages (ignoredin Git)
├── src/
│   ├── components/
│   │   ├── FeedbackForm.jsx
│   │   ├── Footer.jsx
│   │   └── Header.jsx
│   │
│   ├── pages/
│   │   ├── AdminDashboard.jsx
│   │   ├── BookingConfirmation.jsx
│   │   ├── CinemaSelection.jsx
│   │   ├── Home.jsx
│   │   ├── Login.jsx
│   │   ├── MovieDetails.jsx
│   │   ├── Payment.jsx
│   │   ├── Register.jsx
│   │   ├── SeatSelection.jsx
│   │   └── UserDashboard.jsx
│   │
```

```
│   ├── services/
│   │   ├── authService.js
│   │   └── movieService.js
│   │
│   ├── App.jsx
│   ├── index.css
│   └── main.jsx
│
├── .gitignore
├── index.html
├── package-lock.json
├── package.json
├── postcss.config.js
├── tailwind.config.js
├── vite.config.js
└── vite.config.ts
```

2. Design UI Components

Step1:CreateCore Components

1. **Header.jsx**

   Navigation bar with logo

   Menu items (Home,Movies, Theaters) User auth section

frntend > src > components > Header.jsx > Header

```jsx
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

```jsx
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

a. Configure Tailwind in tailwind.config.js
b. Add base styles in index.css
c. Use utility classes for components

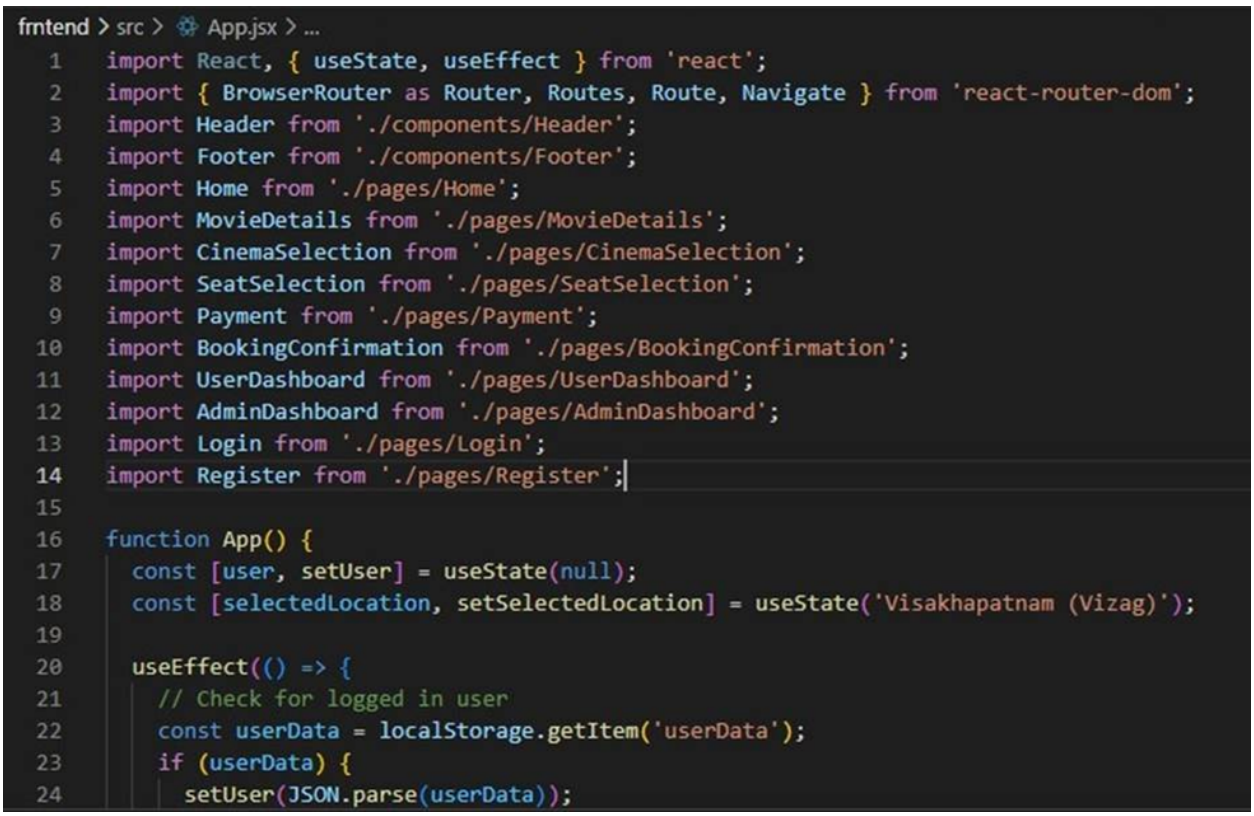
frntend > src > # index.css > ...

```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 routesin 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. UserDashboard(/dashboard)
11. Admin Dashboard (/admin)



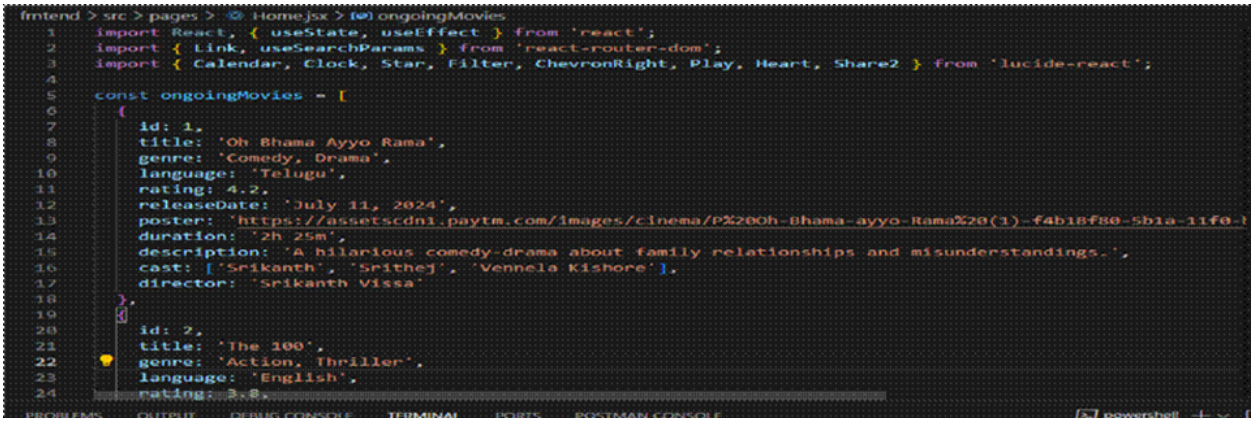
3. Implement Frontend Logic

Step1: API Services

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

frntend > src > pages > MovieDetails.jsx > ...

```jsx
import React, { useState } from 'react';
import { useParams, Link, useNavigate } from 'react-router-dom';
import { Star, Calendar, Clock, Users, Play, Heart, Share2, X } from 'lucide-react';

// Trailer Modal Component
const TrailerModal = ({ isOpen, onClose, trailerUrl, movieTitle }) => {
  if (!isOpen) return null;

  return (
    <div className="fixed inset-0 bg-black bg-opacity-75 flex items-center justify-center z-50 p-4">
      <div className="bg-white rounded-lg max-w-4xl w-full max-h-[90vh] overflow-hidden">
        <div className="flex justify-between items-center p-4 border-b">
          <h3 className="text-lg font-semibold">{movieTitle} - Official Trailer</h3>
          <button
            onClick={onClose}
            className="p-2 hover:bg-gray-100 rounded-full transition-colors"
          >
            <X className="h-5 w-5" />
          </button>
        </div>
        <div className="aspect-video">
          <iframe
            width="100%"
            height="100%"
```

Seat Selection:

c. Fetch available seats

d. Track selected seats

e. Calculate the total price

frntend > src > pages > SeatSelection.jsx > ...

```jsx
import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { ArrowLeft, Users, IndianRupee } from 'lucide-react';

const SeatSelection = ({ user }) => {
  const { movieId, cinemaId, showtime } = useParams();
  const navigate = useNavigate();

  const [selectedSeats, setSelectedSeats] = useState([]);
  const [loading, setLoading] = useState(false);

  // Check if user is logged in
  useEffect(() => {
    if (!user) {
      if (window.confirm('Please login to book tickets. Would you like to login now?')) {
        navigate('/login');
      } else {
        navigate(-1);
      }
    }
  }, [user, navigate]);

  // Mock seat data - in real app, this would come from API
  const seatLayout = {
```

Step3: State Management

1. Use Context API for:

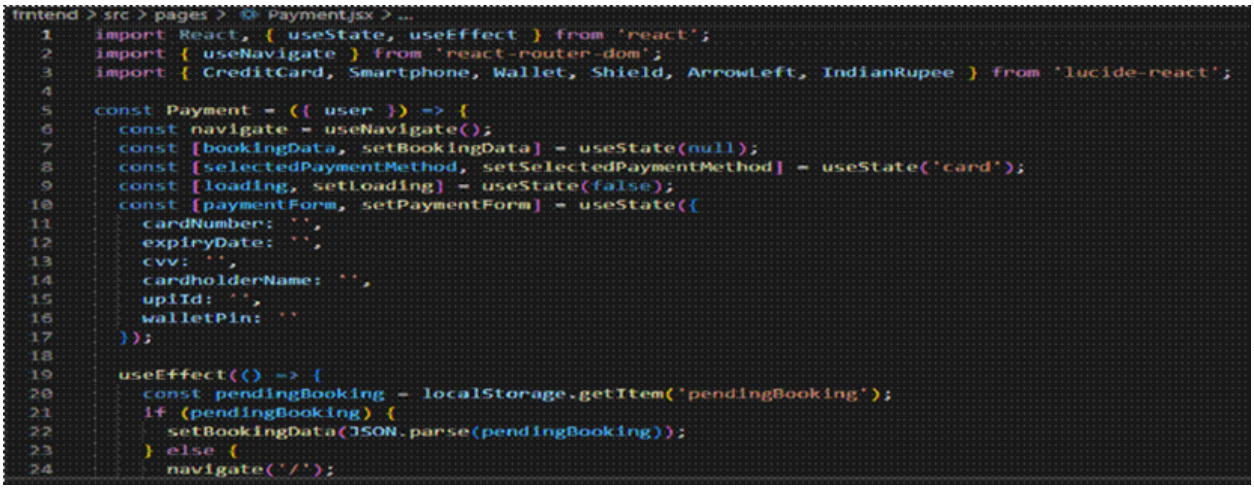
1. User authentication
2. Booking process

3. UI preferences

```
frntend > src > pages > UserDashboard.jsx > UserDashboard > useEffect() callback > mockBookings > poster
import React, { useState, useEffect } from 'react';
import { Calendar, Clock, MapPin, Star, Ticket, Download } from 'lucide-react';

const UserDashboard = ({ user }) => {
  const [activeTab, setActiveTab] = useState('bookings');
  const [bookings, setBookings] = useState([]);

  useEffect(() => {
    // Mock bookings data - in real app, fetch from API
    const mockBookings = [
      {
        id: 'BK001',
        movieTitle: 'Oh Bhama Ayyo Rama',
        cinema: 'INOX CMR Central',
        location: 'Visakhapatnam',
        date: '2024-07-15',
        time: '07:30 PM',
        seats: ['D5', 'D6'],
        amount: 500,
        status: 'confirmed',
        poster: 'https://images.filmibeat.com/img/popcorn/movie_posters/ohbhamaayyorama-20250324125
      },
      {
        id: 'BK002',
```

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

2. . Payment Integration
    a. Setup payment form
    b. Connect to paymentgateway ● Handle success/failure



```
frntend > src > pages > Payment.jsx > ...
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { CreditCard, Smartphone, Wallet, Shield, ArrowLeft, IndianRupee } from 'lucide-react';

const Payment = ({ user }) => {
  const navigate = useNavigate();
  const [bookingData, setBookingData] = useState(null);
  const [selectedPaymentMethod, setSelectedPaymentMethod] = useState('card');
  const [loading, setLoading] = useState(false);
  const [paymentForm, setPaymentForm] = useState({
    cardNumber: '',
    expiryDate: '',
    cvv: '',
    cardholderName: '',
    upiId: '',
    walletPin: ''
  });

  useEffect(() => {
    const pendingBooking = localStorage.getItem('pendingBooking');
    if (pendingBooking) {
      setBookingData(JSON.parse(pendingBooking));
    } else {
      navigate('/');
```

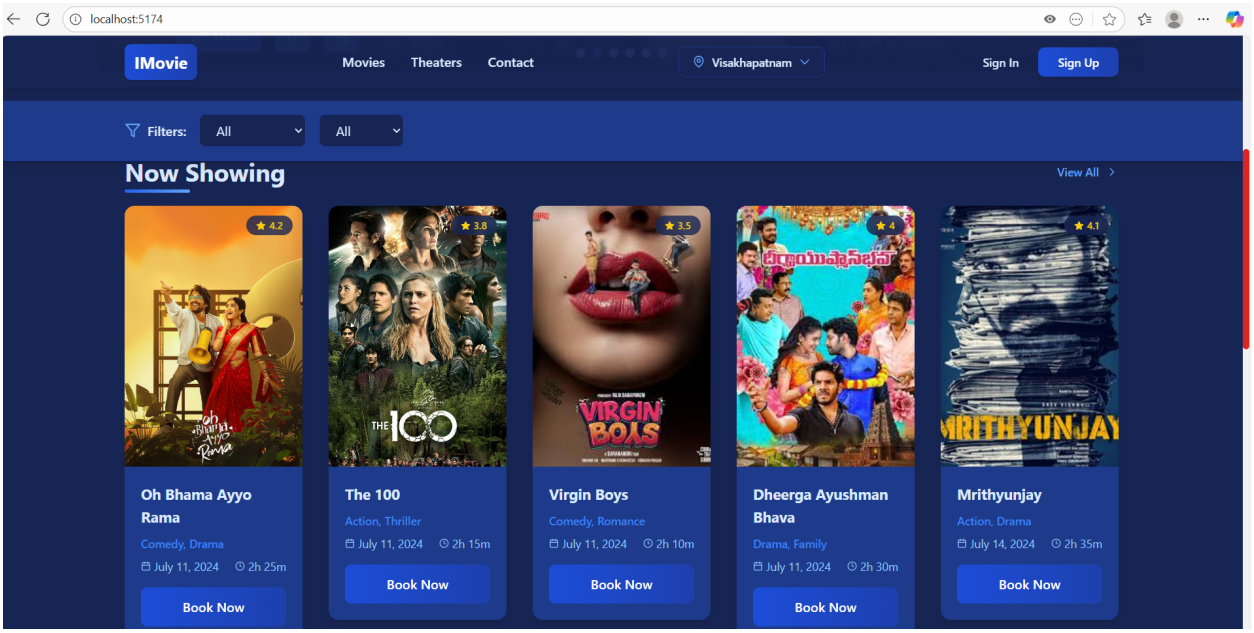
Step4: Run the Application

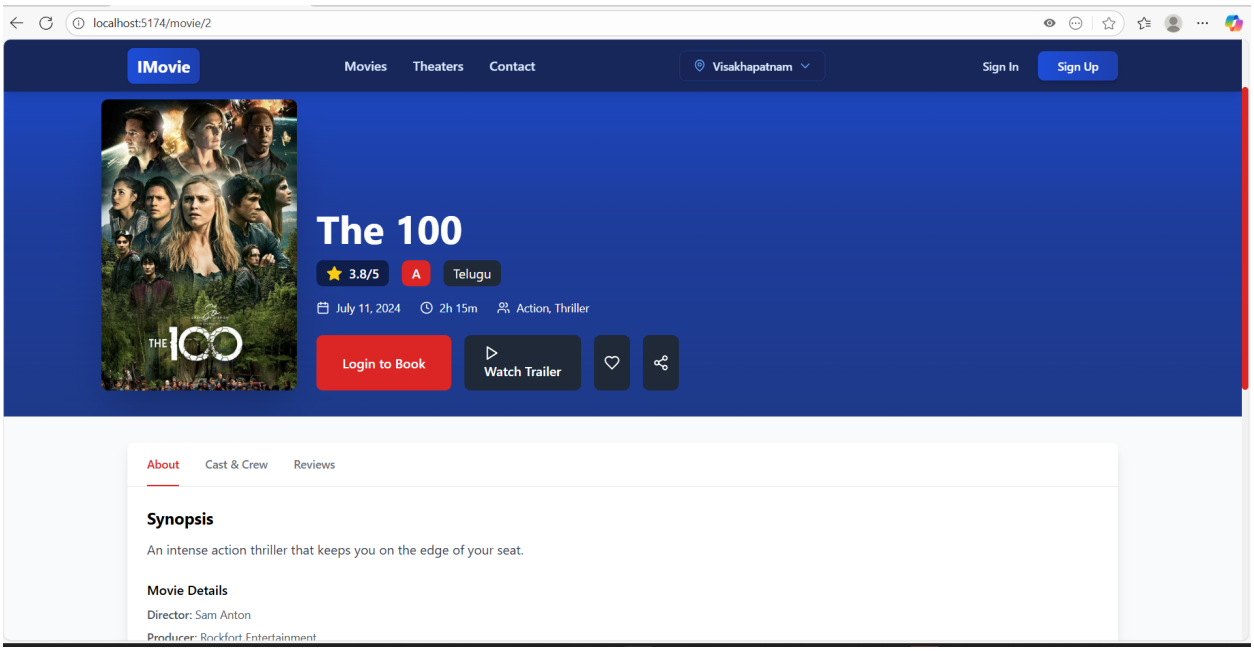
bash
npmrun
dev

```
PS D:\project> cd frntend
PS D:\project\frntend> npm run dev

> vite-react-typescript-starter@0.0.0 dev
> vite


  VITE v7.0.5  ready in 944 ms


  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

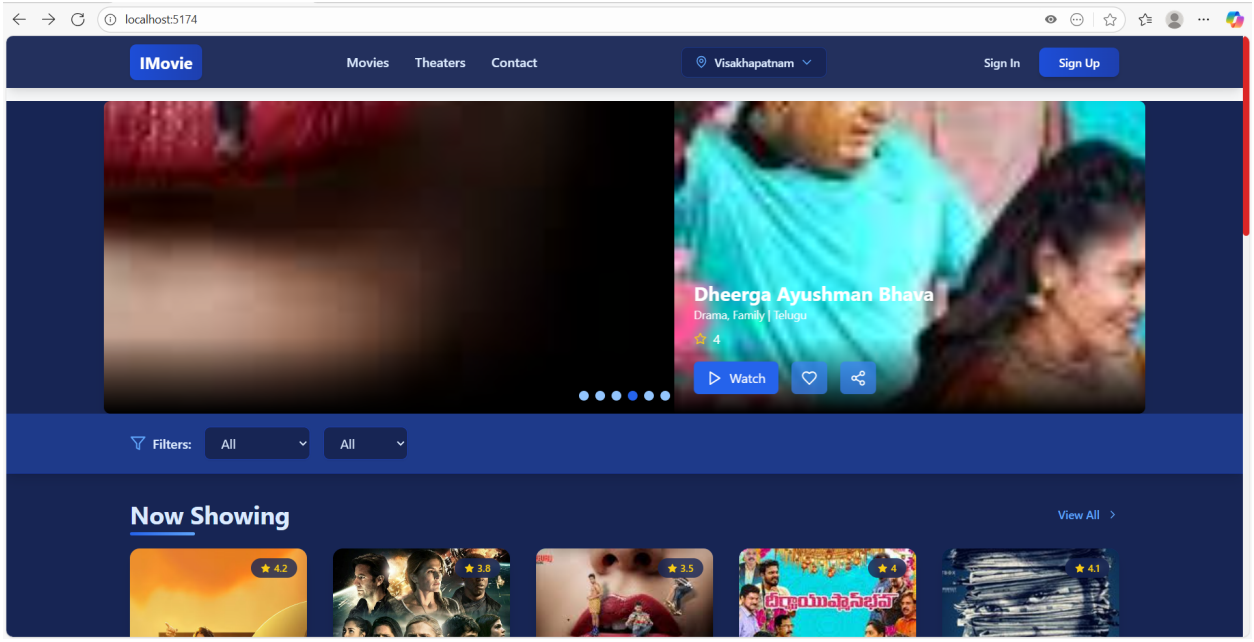
5.Project Implementation

Movie details page

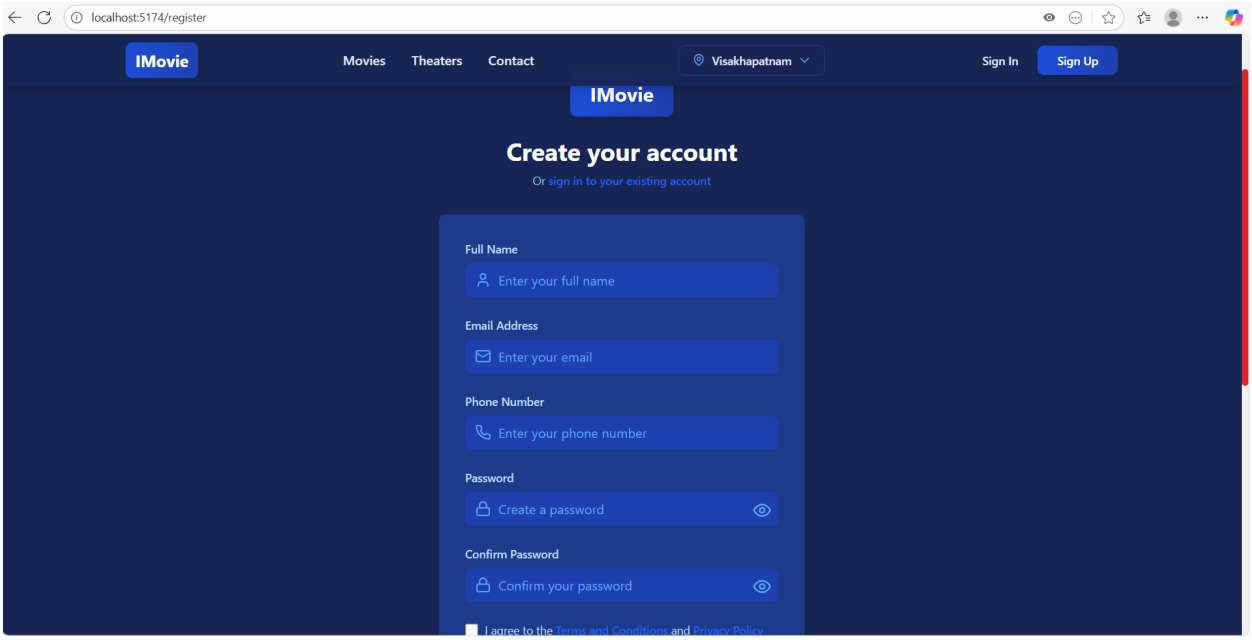




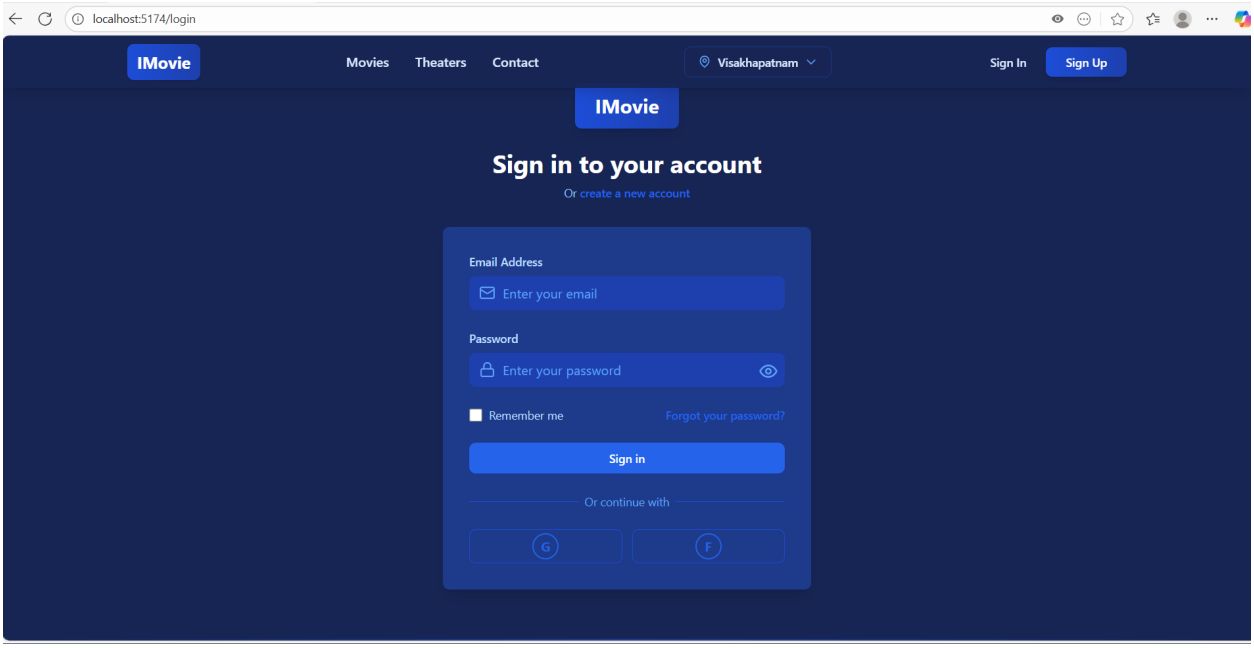
Home page



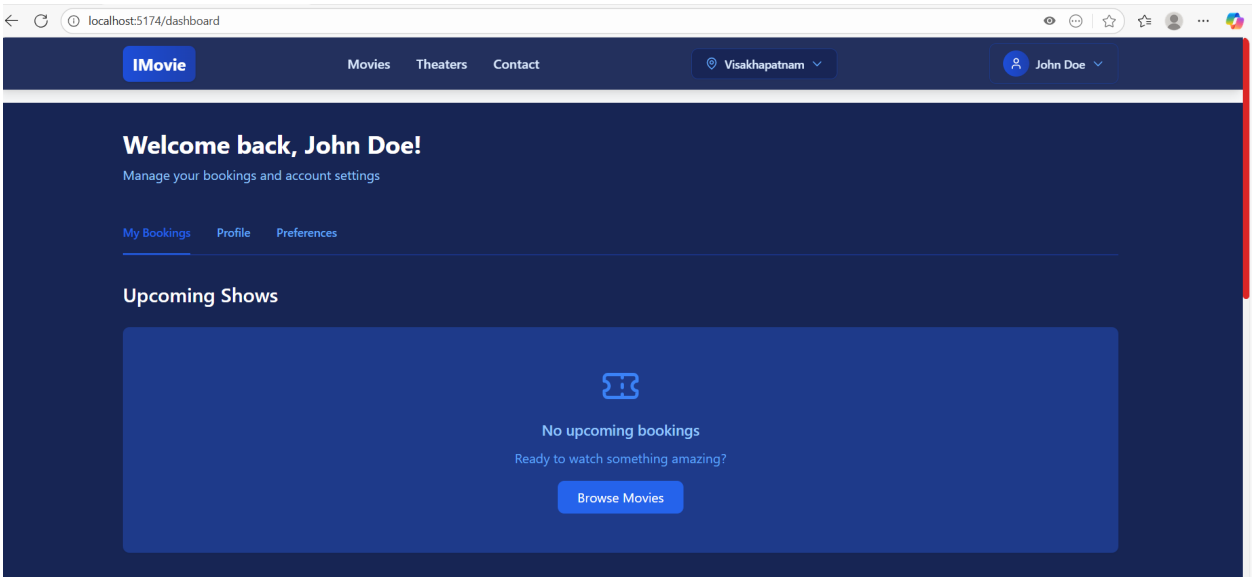
Sign up page



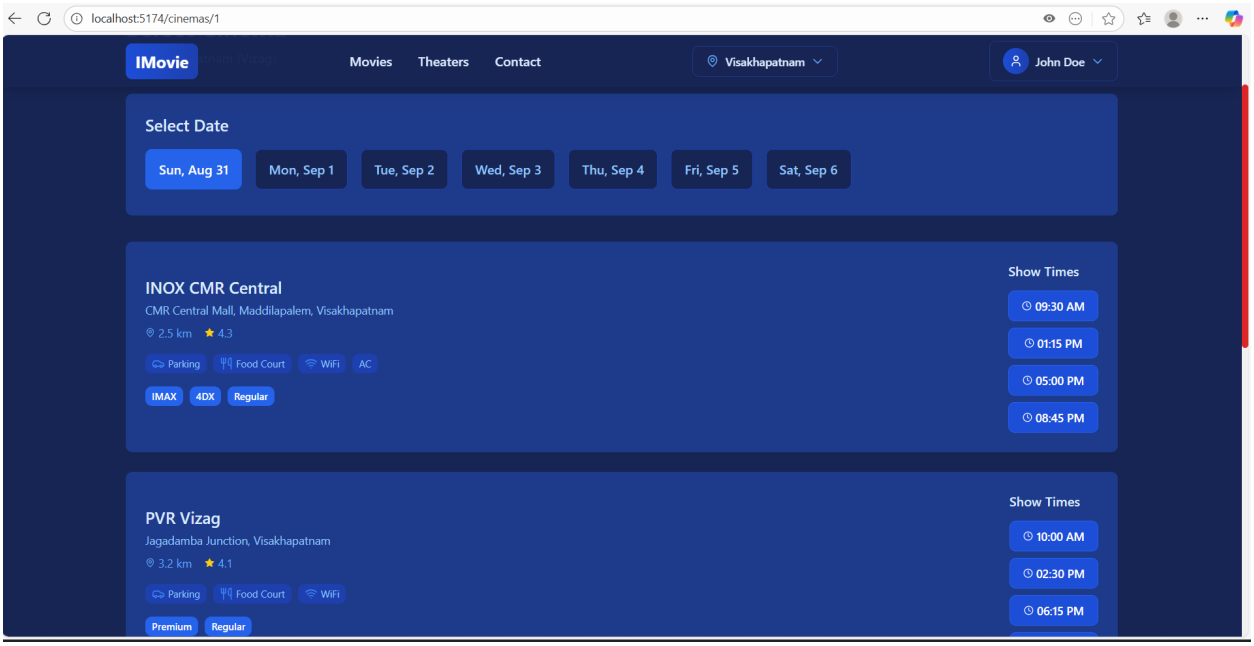
Sign in page



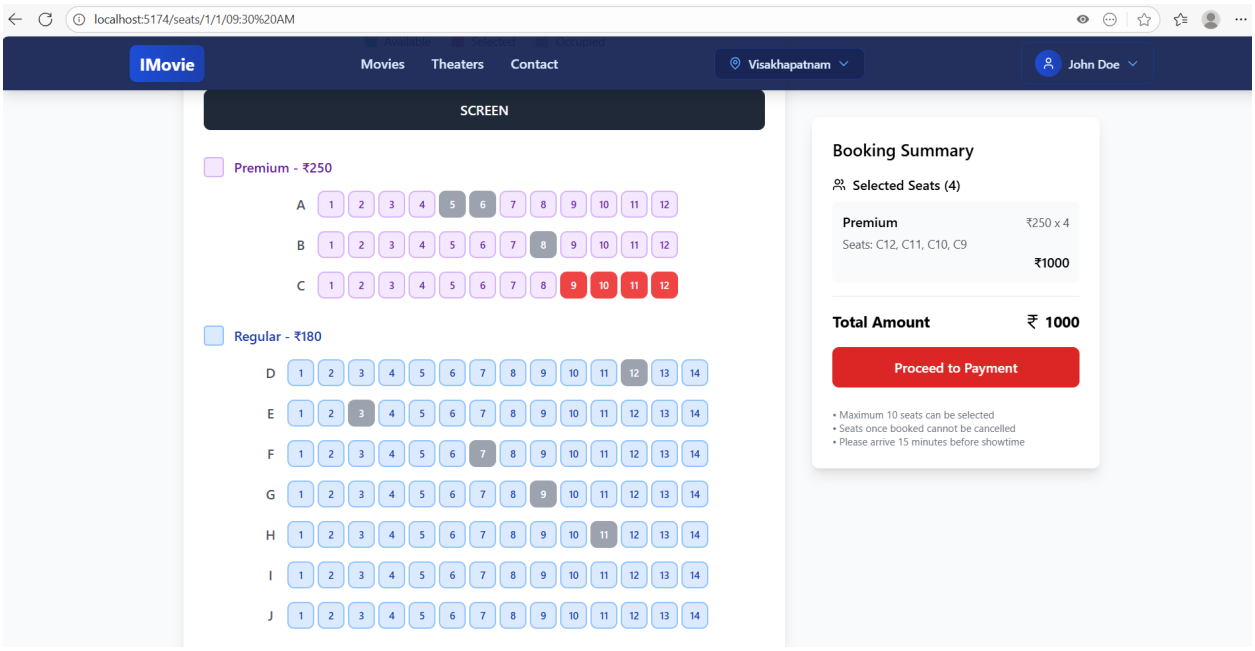
User ProfilePage



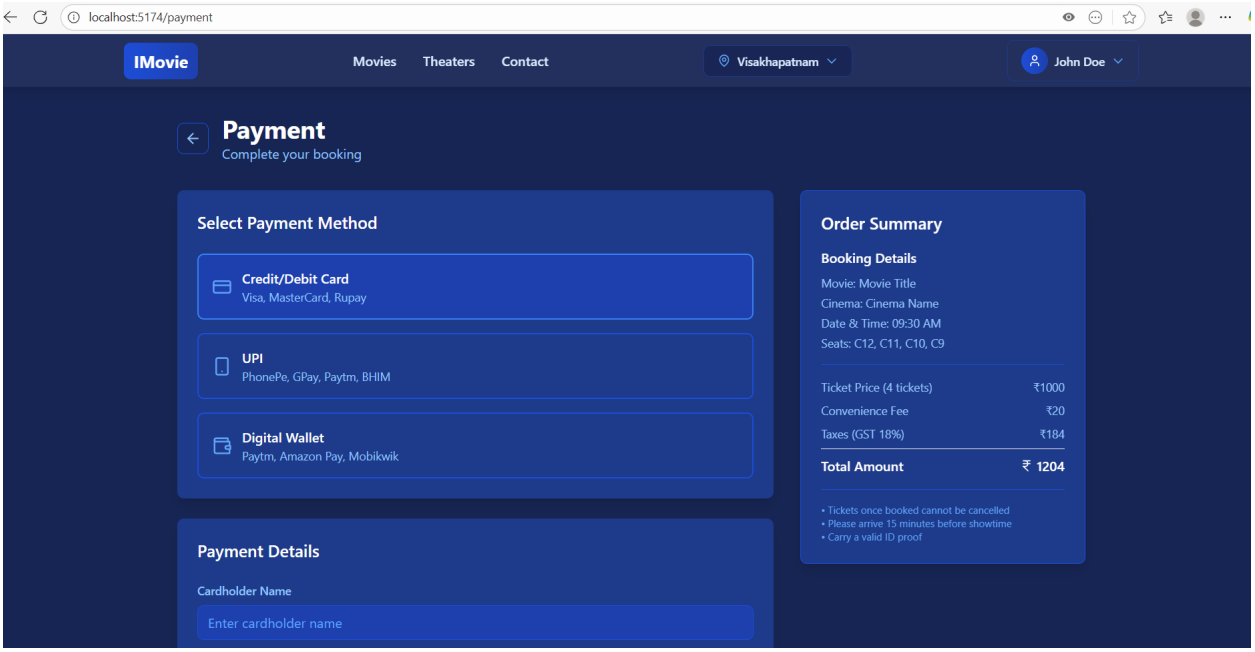
Bookings Page

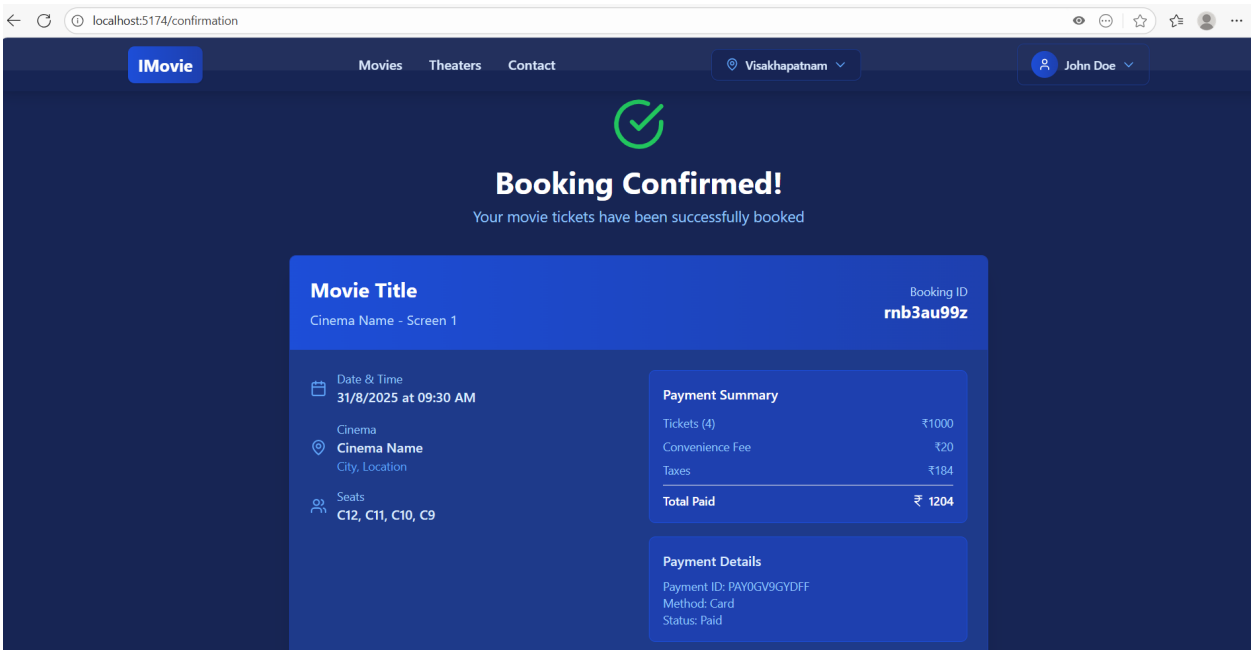


Seat Selection page

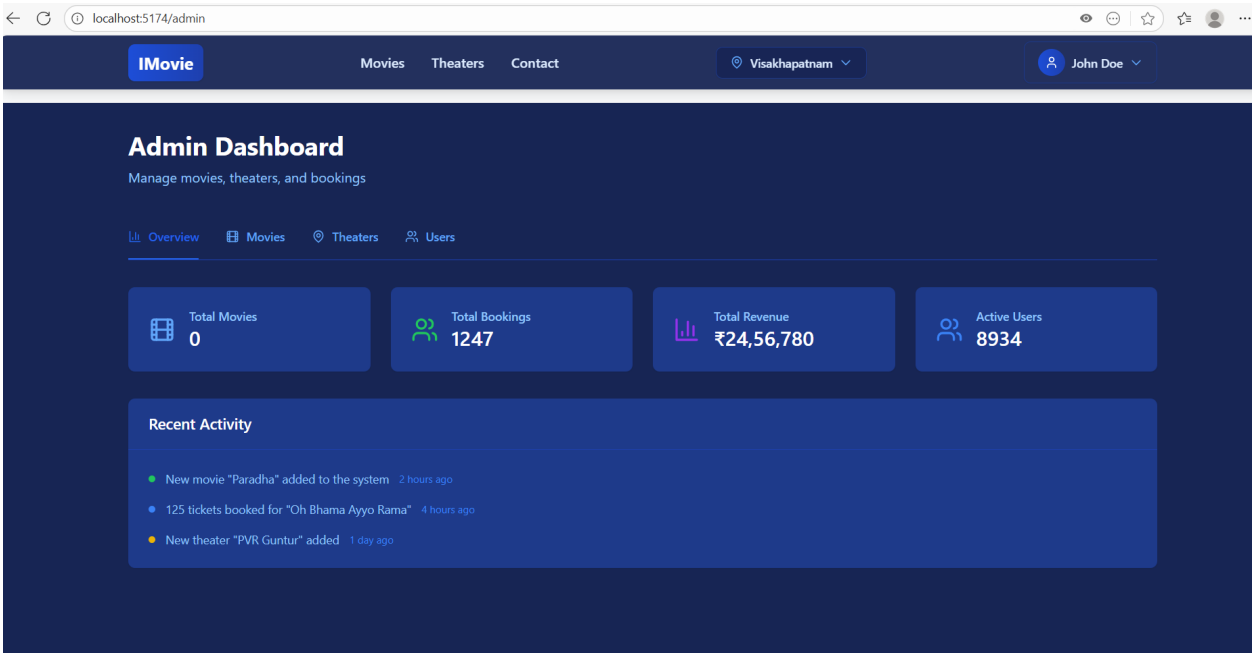


Ticket PaymentSuccess page

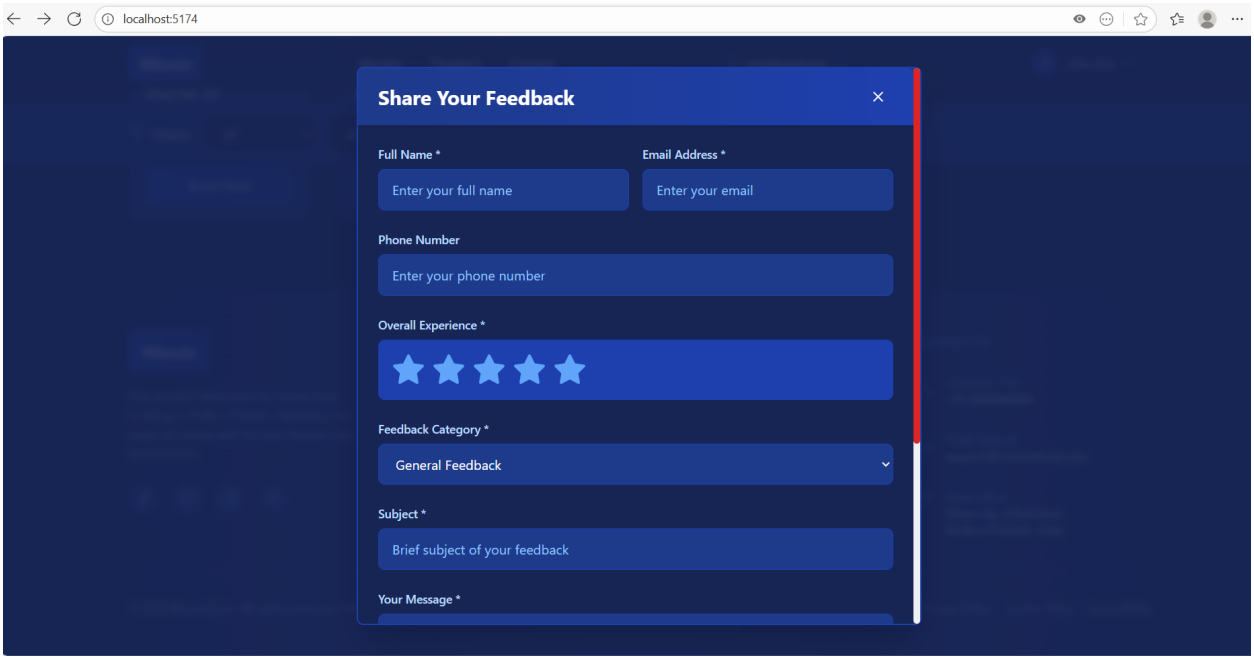




Admin DashBoardPage



Feedback Page



Conclusion

I-Movies is a cutting-edge onlinemovie ticket bookingplatform 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 offersseamless 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.