I-Movies : MovieTicket Booking System

Project Description:

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

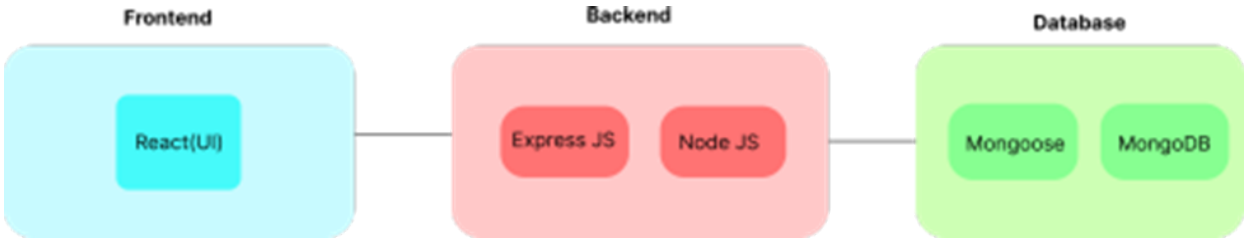
Imagine Sarah, a movie enthusiast, excitedly expecting the release of an anticipated film. With a busy schedule, she prefers the convenience of booking her movie tickets online. Logging into the iMovies app, Sarah navigates effortlessly through the user-friendly interface, browsing the list of usable 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 movie- going experience, making her next cinema outing memorable and stress-free.

Technical Architecture



In This Architecture Diagram:

- 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.,
- 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.

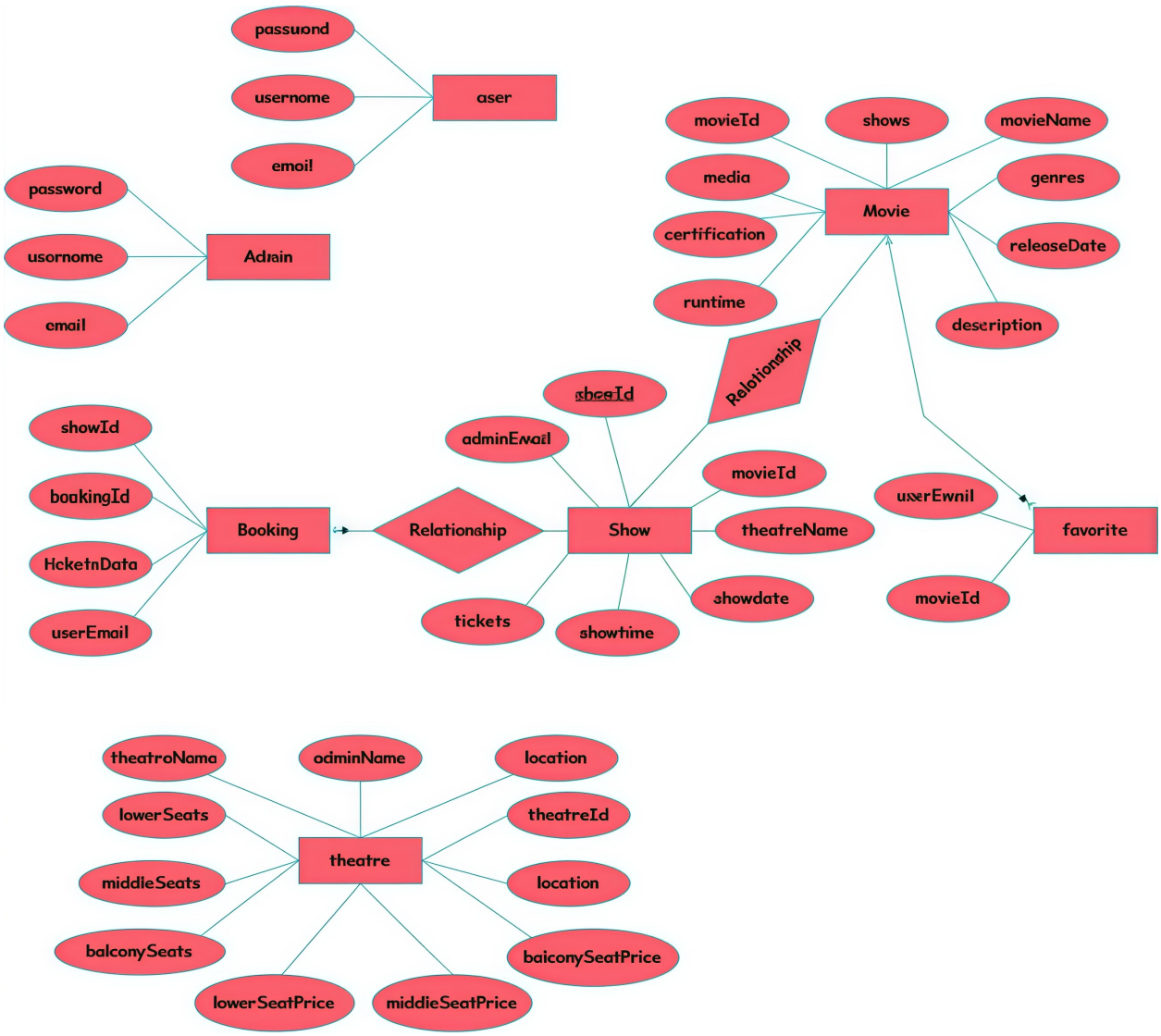
The technical architecture of the iMovies app follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses not only the user interface and presentation but also incorporates the axiom library to connect with backend easily by using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



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 detailedinstructions.

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

2. MongoDB

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

- **Download (Community Edition)**: https://www.mongodb.com/try/download/community
- **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** (ODMfor MongoDB) to connect Node.jswith MongoDB.

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

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 Storagein FirebaseConsole and configure:

   javascript

   'import { getStorage, ref, uploadBytesResumable, getDownloadURL } from "Firebase/storage";'

8. Version Control (Git)

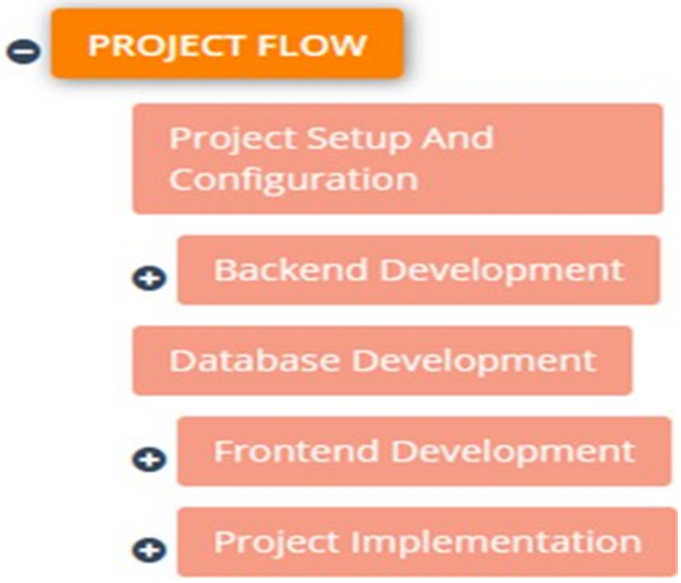
- **Download Git**: https://git-scm.com/downloads

9. Development Environment

Choose an IDE:

- **VS Code**: https://code.visualstudio.com/download
- **Sublime Text**: https://www.sublimetext.com/download

WebStorm: https://www.jetbrains.com/webstorm/download



1. Project setup and configuration

The project setupinvolves installing Node.jsand Git, creatinga structured foldersystem 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:
  - frontend/src/for React components
  - frontend/public/ for static assets
  - Allnecessary configurationfiles
- **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:

- models/ for databaseschemas
- routes/ for API endpoints
- 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 BackendDependencies**

Run these commands to install required packages:

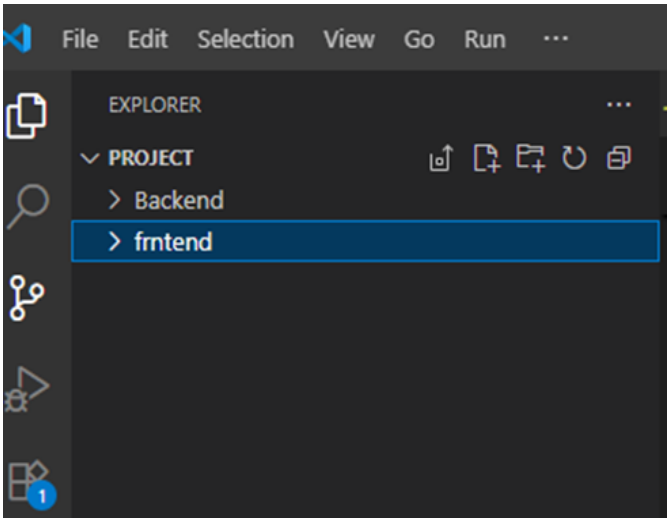
bash
cd Backend
npm install express mongoose cors dotenv

- **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

- **Initialize Express Server**
  - Navigate to your Backend folder and installrequired dependencies:

    bash
    cd Backend

    npm install express cors body-parser mongoose jsonwebtoken dotenv bcryptjs
  - Create a new index.js file as your main server file:

    bash
    touch index.js
- **Configure Environment Variables**
  - Create a .env file in your Backend folder:

    bash
    touch .env
  - Add theseconfigurations to your .env file:

    env

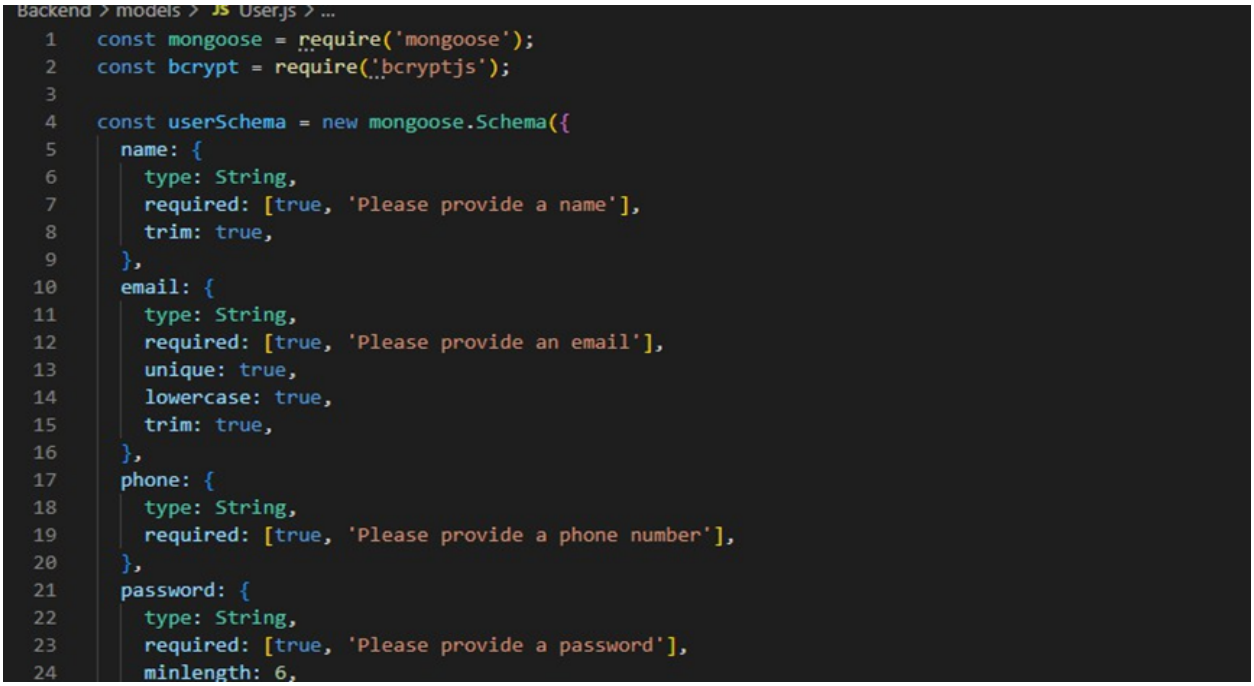
PORT=5000
MONGODB_URI=mongodb://localhost:27017/imovie_db

JWT_SECRET=your_strong_secret_key_here

- **Basic Server Setup**
  - Add this code to your indexserver.js file:

Authentication Setup

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



- 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**
  - 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 = () =>
```

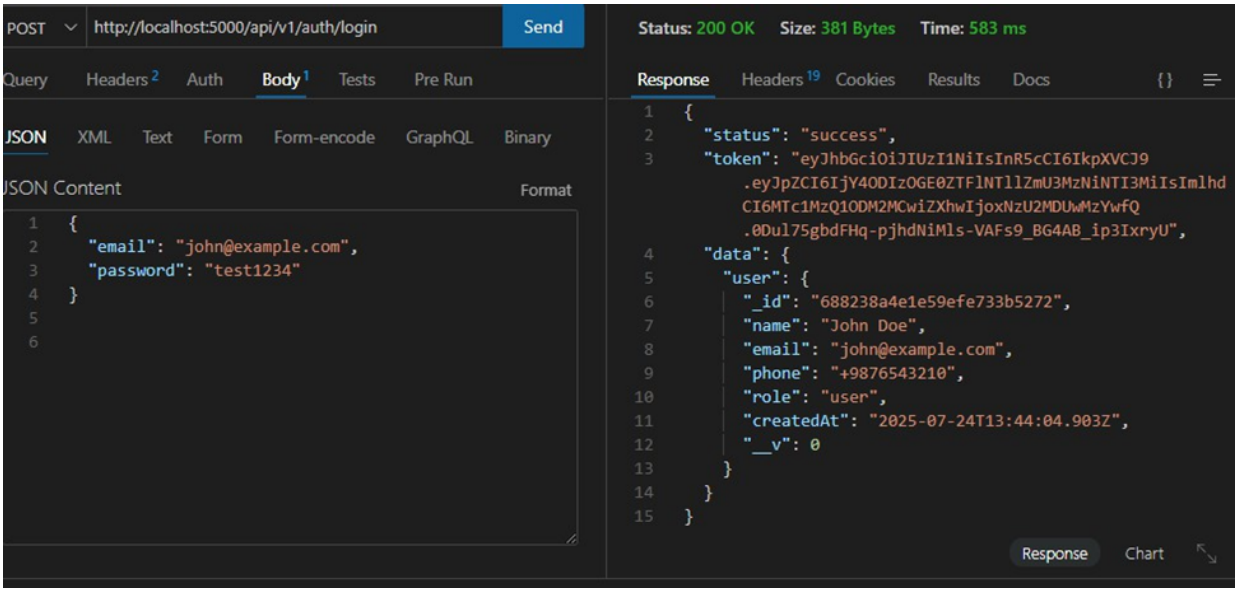
Start Your Server

- Run the backend server with:

bash

node server.js

- **Testing The Backend by using the thunder client:**

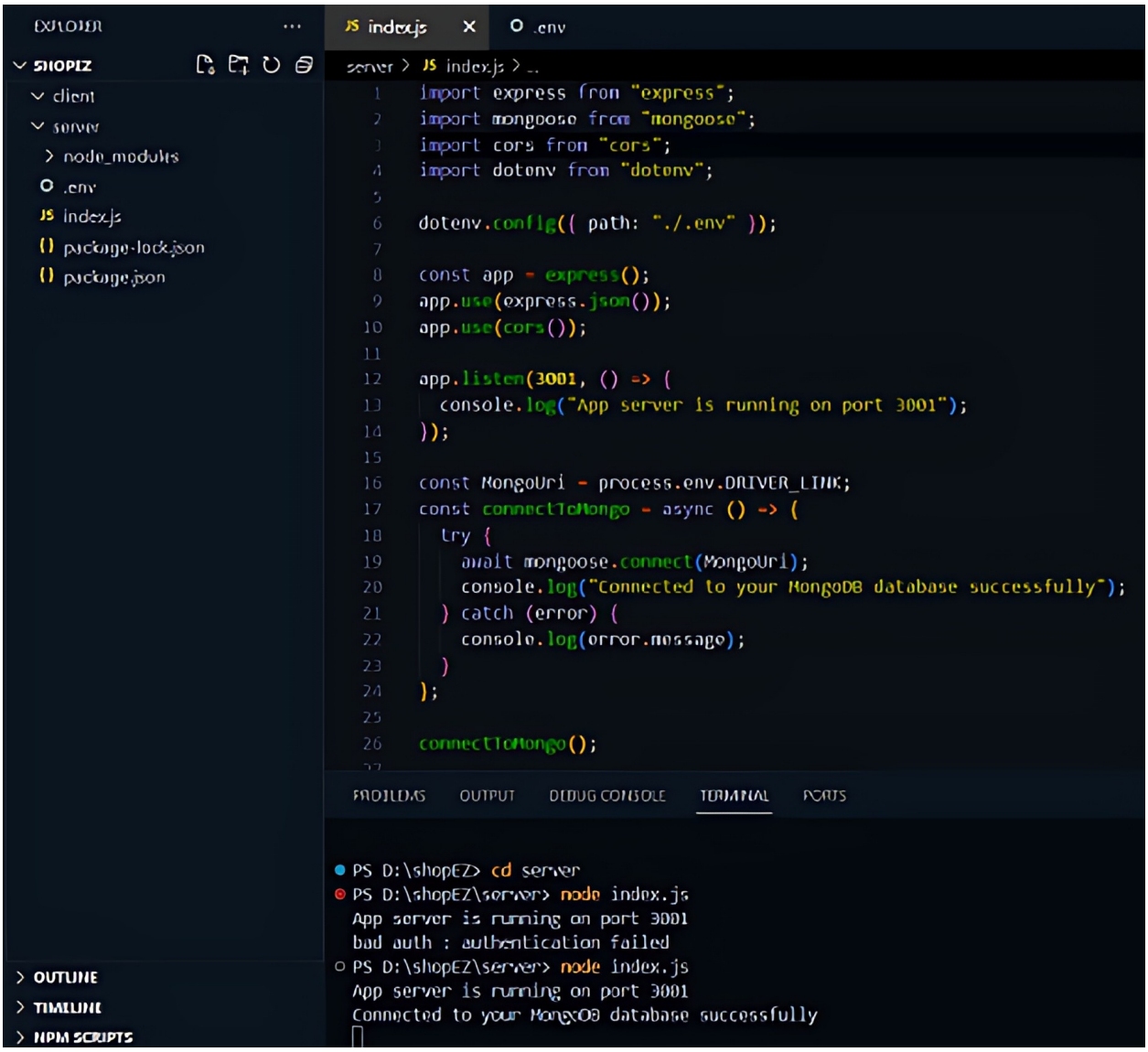


3. Database Development

Creating Data Schemas

User Schema:

- Schema: userSchema
- Model: 'User'
- The User schema definesfields like username,email, and password with specified constraints such as minimumand maximum lengthsand uniqueness.
- It represents user data in the application, ensuring data integrity and security for user accounts.



Theater Schema:

- Schema: theaterSchema
- Model: 'Theater'
- The theater schema captures details about theaters including names, IDs, locations, seat prices, and seat layouts.
- 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:

- Schema: showSchema

- Model: 'Show'

- The Show schema represents individual movie showings with attributes such as admin email, movie ID, theater name, date, time, and ticket details.

- 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:

- Schema: movieSchema

- Model: 'Movie'

- The Movie schema defines properties for movies including name, description, genres, release date, runtime, certification, media format, and associated show IDs.

- 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:

- Schema: favoriteSchema

- Model: 'Favorite'

- The Favoriteschema records user preferences by storing user email and movie ID pairs for favorite movies.

- It enablesusers to bookmarkand 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:

- Schema: bookingModel

- Model: 'Booking'

- The Bookingschema captures detailsof user bookings including bookingID, user email, show ID, and ticket data.

- 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);
```

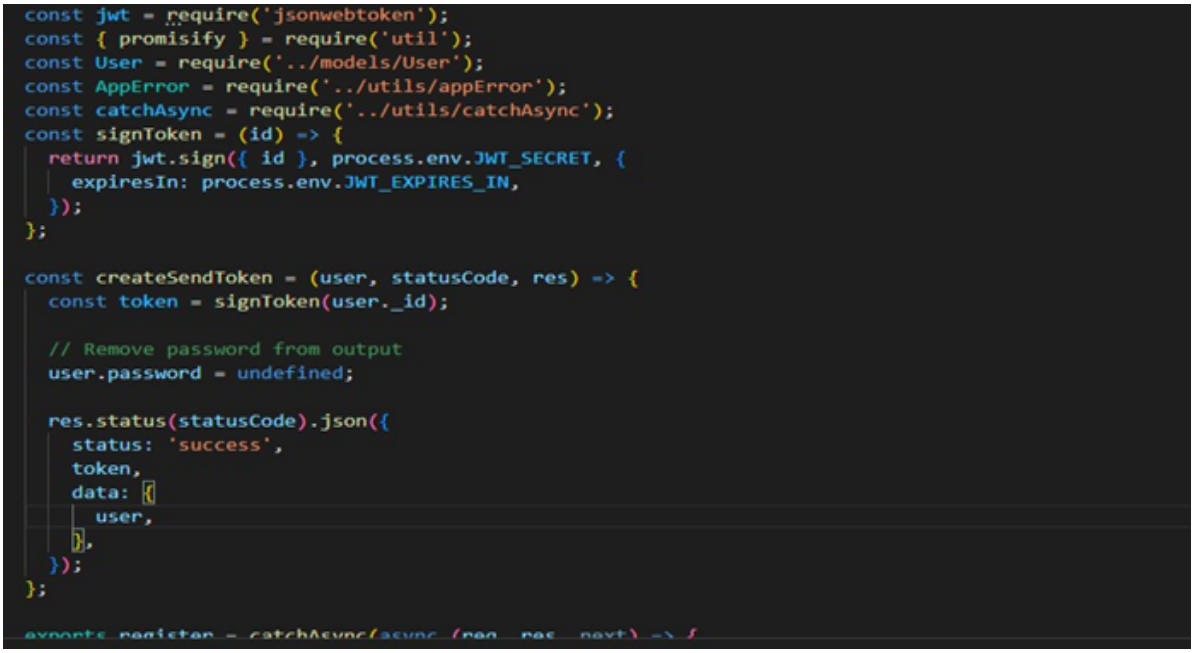
Admin Schema:

- Schema: adminSchema

- Model: 'Admin'

- The Admin schema defines fields for admin accounts such as username, email, and password with specified constraints.

- It represents administrative users in the system,providing access to privileged functionalities for managing the application.



```
const jwt = require('jsonwebtoken');
const { promisify } = require('util');
const User = require('../models/User');
const AppError = require('../utils/appError');
const catchAsync = require('../utils/catchAsync');
const signToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN,
  });
};

const createSendToken = (user, statusCode, res) => {
  const token = signToken(user._id);

  // Remove password from output
  user.password = undefined;

  res.status(statusCode).json({
    status: 'success',
    token,
    data: {
      user,
    },
  });
};

exports.register = catchAsync(async (req, res, next) => {
```

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/icons-material

npm install -D tailwindcss Postcss
autoprefixer

npx tailwindcss init -p

- axios: For API calls
- react-router-dom:Forrouting
- @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

Step 1: Create Core Components

- **Header.jsx**

  Navigation bar with logo

  Menu items(Home, Movies, Theaters)

  User auth section

Header.jsx

frntend > src > components > Header.jsx > locations

```jsx
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { Search, MapPin, User, Menu, X } from 'lucide-react';

const locations = [
'Kathmandu',
'Pokhara',
'Biratnagar',
'Bharatpur',
'Dharan',
'Butwal',
'Hetauda',
'Nepalgunj',
'Janakpur',
'Bhaktapur',
'Lalitpur',
'Itahari',
'Dhangadhi',
'Birgunj',
'Tansen',
'Ghorahi',
'Siddharthanagar'
];

```

- **Footer.jsx**

  Copyright information
  Contact links
  Social media icons

Footer.jsx

frntend > src > components > Footer.jsx > Footer

```jsx
import React, { useState } from 'react';
import { Facebook, Twitter, Instagram, Youtube, Rss } from 'lucide-react';
import FeedbackForm from './FeedbackForm';

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

  return (
    <footer className="bg-gray-800 text-white p-6">
      <div className="container mx-auto flex flex-col items-center space-y-4">
        {/* Social Media Icons */}
        <div className="flex space-x-4">
          <a href="https://facebook.com" target="_blank" rel="noopener noreferrer">
            <Facebook className="h-6 w-6 hover:text-gray-400" />
          </a>
          <a href="https://twitter.com" target="_blank" rel="noopener noreferrer">
            <Twitter className="h-6 w-6 hover:text-gray-400" />
          </a>
          <a href="https://instagram.com" target="_blank" rel="noopener noreferrer">
            <Instagram className="h-6 w-6 hover:text-gray-400" />
          </a>
          <a href="https://youtube.com" target="_blank" rel="noopener noreferrer">
            <Youtube className="h-6 w-6 hover:text-gray-400" />
          </a>
          <a href="/rss" target="_blank" rel="noopener noreferrer">
            <Rss className="h-6 w-6 hover:text-gray-400" />
```

- **FeedbackForm.jsx**

  Rating input
  Comment textarea
  Submission button

FeedbackForm.jsx

D:\project\frntend\src\components\FeedbackForm.jsx (preview) ...backForm

```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) => {
    setFormData(prev => ({
      ...prev,
```

Step2: Implement Styling

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

Step3: Set Up Routing

Define routes in App.jsx

- Home (/)
- Movies (/movies)
- Cinema Selection (/cinemas/:movieId)
- Seat Selection (/book/:showtimeId)
- Payment (/payment)
- Booking Confirmation (/confirmation)
- Login (/login)
- Register (/register)
- User Dashboard (/dashboard)
- 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 AddMovie from './pages/AddMovie';
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 token = localStorage.getItem('token');
    const userData = localStorage.getItem('user');
    if (token && userData) {
      setUser(JSON.parse(userData));
```

3.Implement Frontend Logic

Step1: API Services

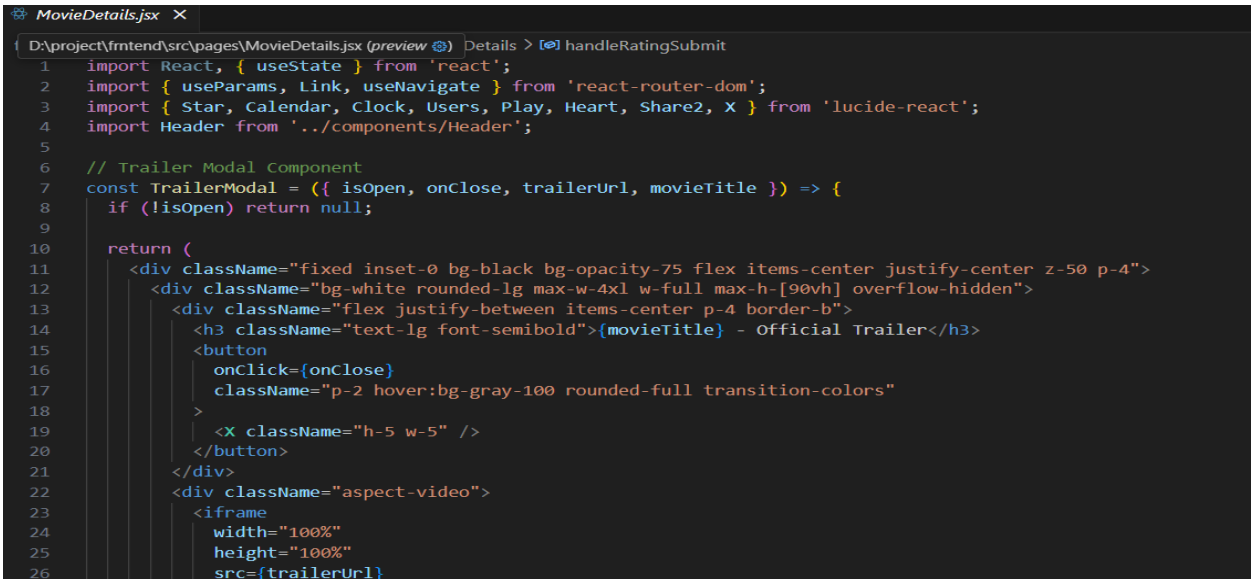
1. Create services/movieService.js:
    - fetchMovies()
    - getMovieDetails(id)
    - getShowtimes(movieId)
2. Create services/bookingService.js:
    - bookSeats(showtimeId, seats)
    - getBookingHistory()

Step 2: Data Binding

1. Home Page:
    - Fetch and display featured movies
    - Implement carouselfor promot

```
Home.jsx
D:\project\frntend\src\pages\Home.jsx (preview)
import React, { useState, useEffect } from 'react';
import { Link, useSearchParams } from 'react-router-dom';
import { Calendar, Clock, Star, Filter, ChevronRight, Play, Heart, Share2 } from 'lucide-react';

const ongoingMovies = [
  {
    id: 1,
    title: 'Animal',
    genre: 'Comedy, Drama',
    language: 'hindi',
    rating: 4.2,
    releaseDate: 'July 11, 2024',
    poster: 'https://upload.wikimedia.org/wikipedia/en/9/90/Animal_%282023_film%29_poster.jpg',
    duration: '2h 25m',
    description: 'A hilarious comedy-drama about family relationships and misunderstandings.',
    cast: ['Srikanth', 'Srithej', 'Vennela Kishore'],
    director: 'Srikanth Vissa'
  },
  {
    id: 2,
    title: 'Avatar',
    genre: 'Action, Thriller',
    language: 'English',
    rating: 3.8,
    releaseDate: 'July 11, 2024',
    poster: 'https://fr.web.img4.acsta.net/pictures/22/11/02/14/49/4565071.jpg',
```

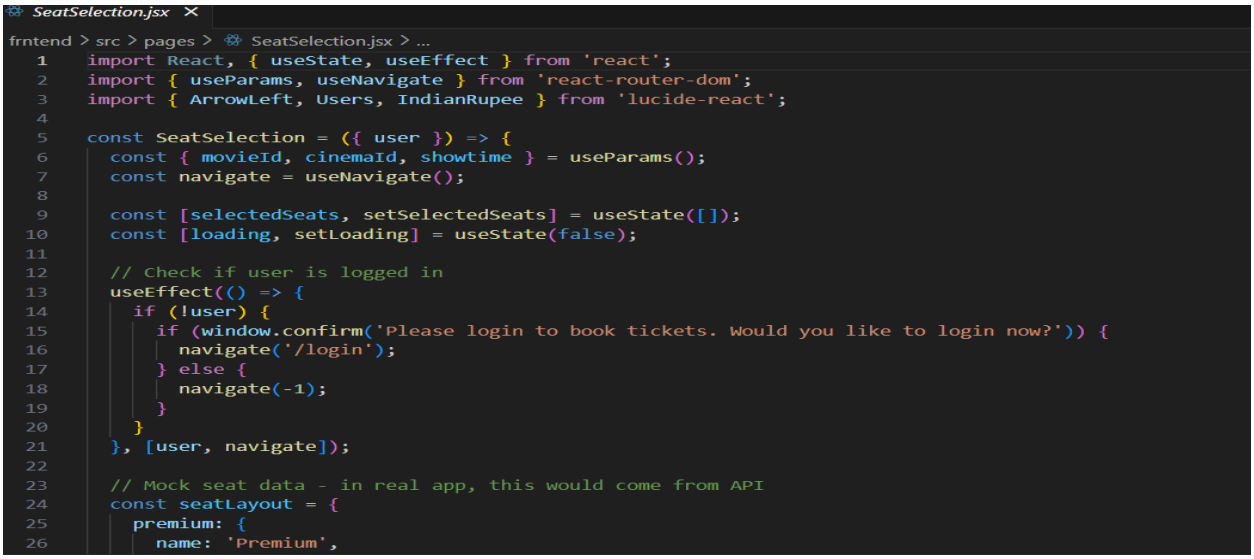
2. Movie Details:
    - Fetch moviedata by ID
    - Display showtime availability



```
MovieDetails.jsx
D:\project\frntend\src\pages\MovieDetails.jsx (preview) Details > handleRatingSubmit
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';
import Header from '../components/Header';

// 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%"
            src={trailerUrl}
```

Seat Selection:

- Fetch available seats
- Track selectedseats
- Calculate the total price



```jsx
// SeatSelection.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 = {
    premium: {
      name: 'Premium',
```

Step3: State Management

1. Use Context API for:
    - User authentication
    - Booking process
    - UI preferences

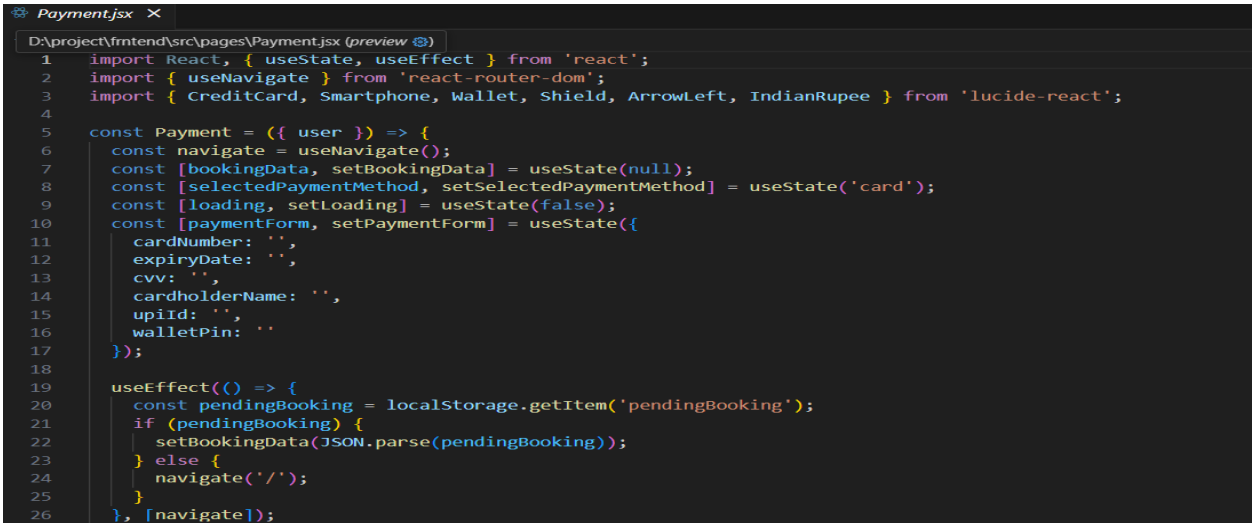
```jsx
// UserDashboard.jsx
import React, { useState, useEffect } from 'react';
import { Calendar, Clock, MapPin, Star, Ticket, Download } from 'lucide-react';

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: 'Saiyaara',
        cinema: 'INOX CMR Central',
        location: 'Visakhapatnam',
        date: '2024-07-15',
        time: '07:30 PM',
        seats: ['D5', 'D6'],
        amount: 500,
        status: 'confirmed',
        poster: 'https://myhdmovie365.s3.amazonaws.com/Top%20Banner%202_Mobile.jpg',
      },
      {
        id: 'BK002',
        movieTitle: 'Chhaava',
        cinema: 'PVR Vizag',
```

2 . Authentication Flow

- Implement login/logout
- Store JWT tokens
- Protectedroutes

3 . PaymentIntegration

- Setup payment form
- Connect to payment gateway
- Handle success/failure

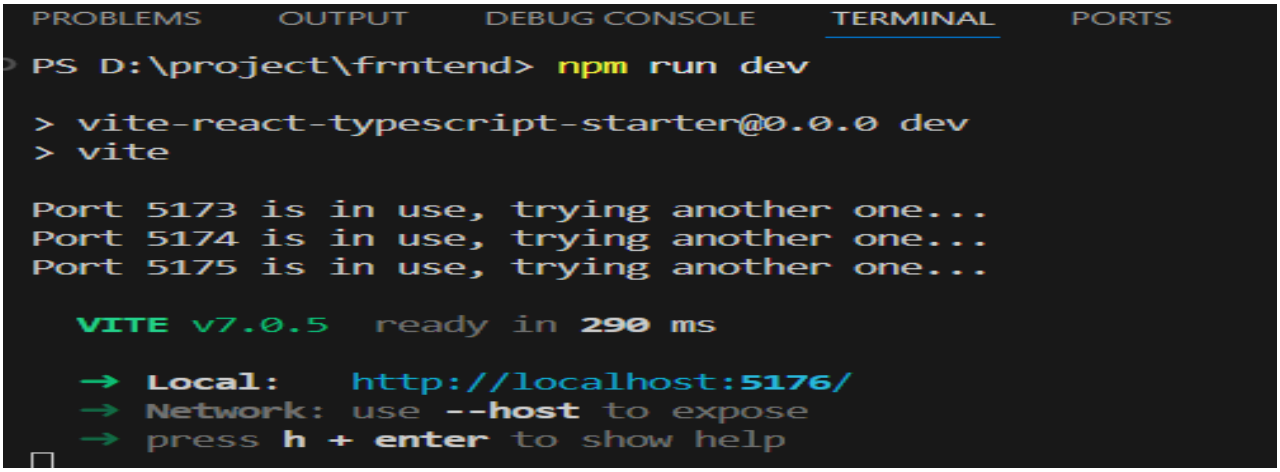


```jsx
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { CreditCard, Smartphone, Wallet, Shield, ArrowLeft, IndianRupee } from 'lucide-react';

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('/');
    }
  }, [navigate]);
```

Step4: Run the Application

bash

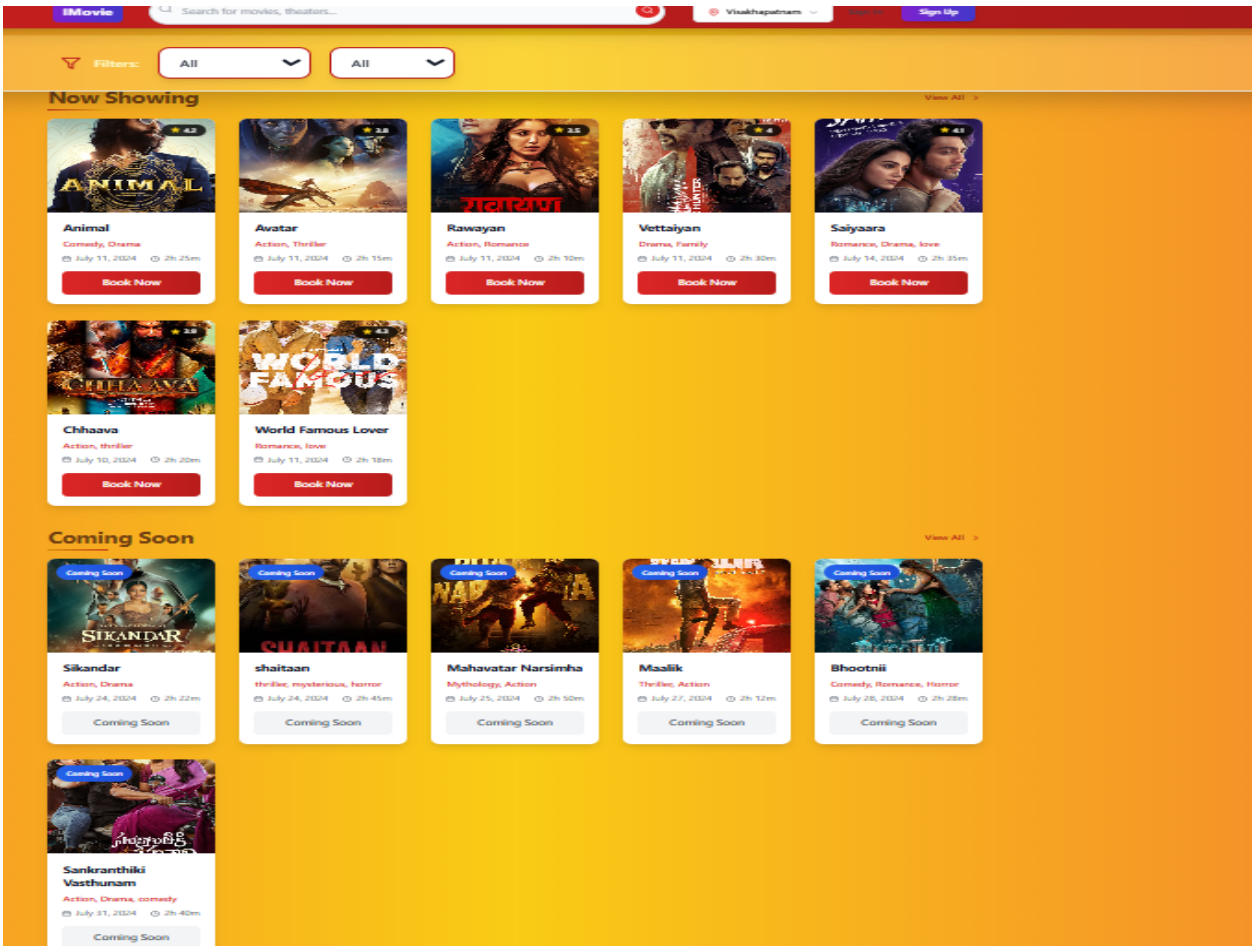
npm run dev

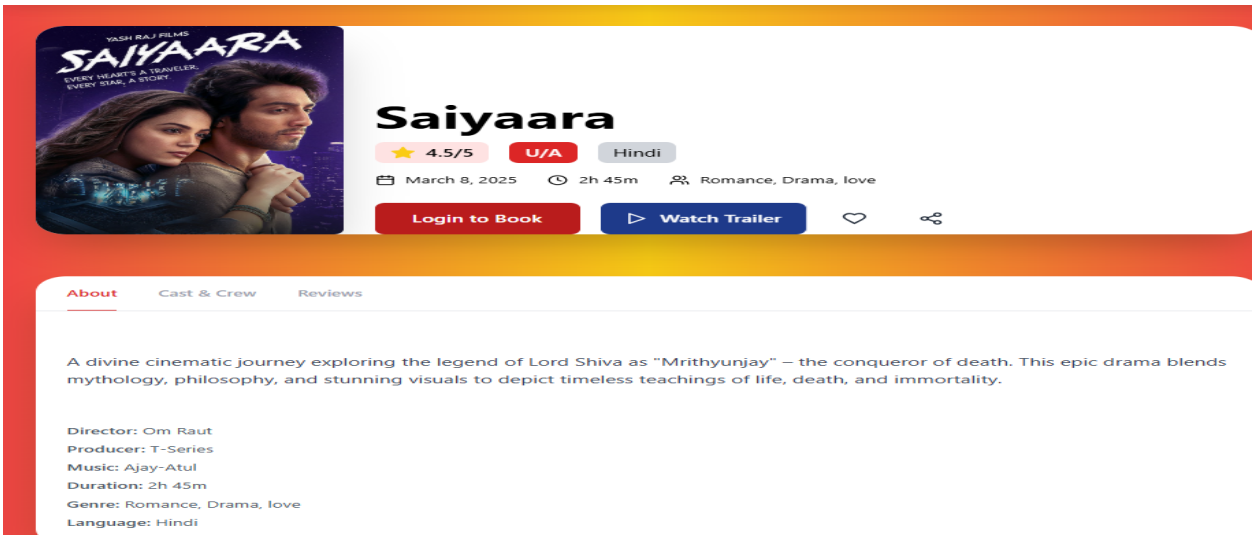


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\project\frntend> npm run dev

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

Port 5173 is in use, trying another one...
Port 5174 is in use, trying another one...
Port 5175 is in use, trying another one...

  VITE v7.0.5  ready in 290 ms

  ➜  Local:   http://localhost:5176/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

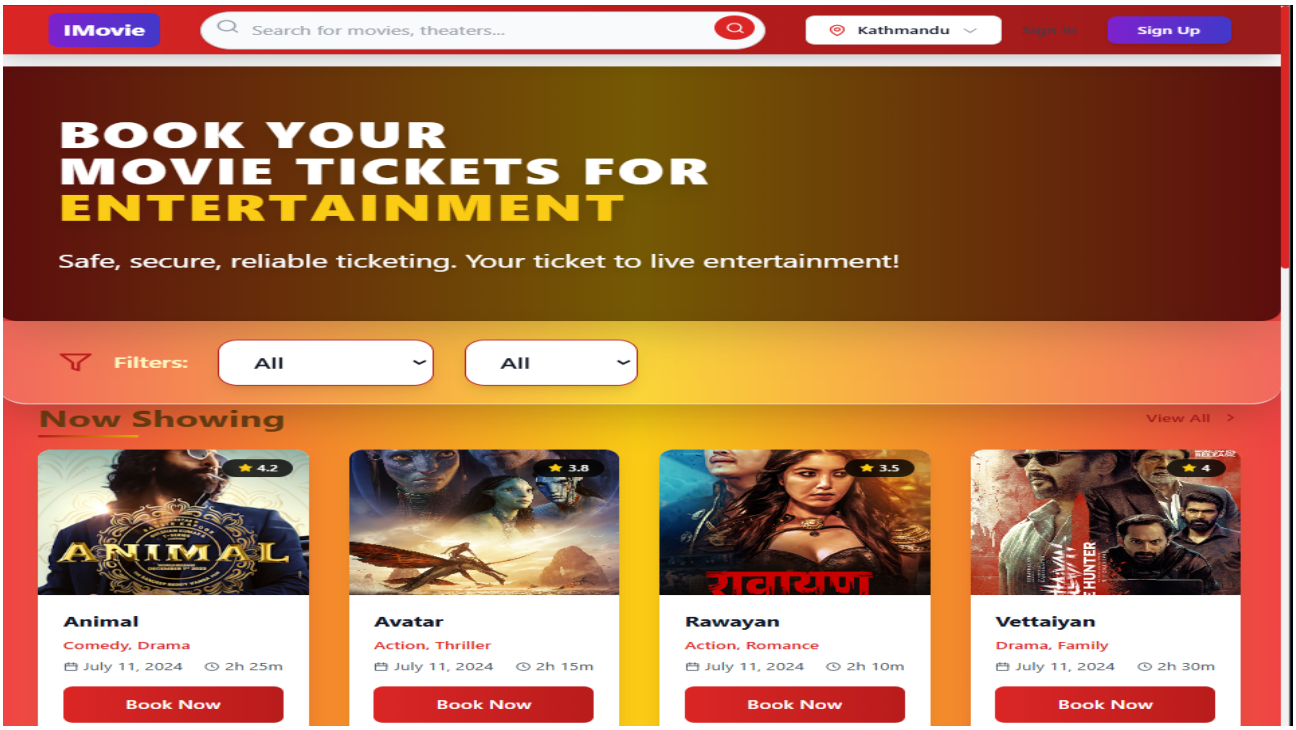
5. Project Implementation

Movie details page

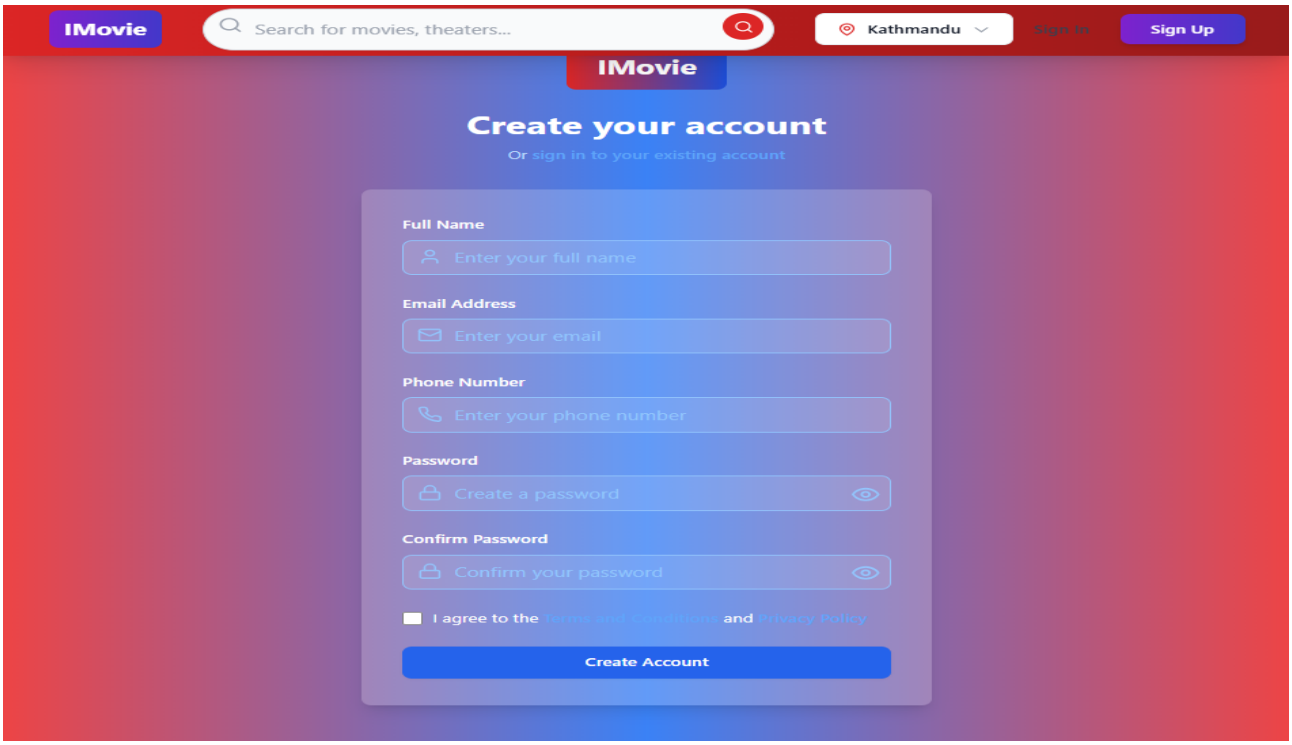




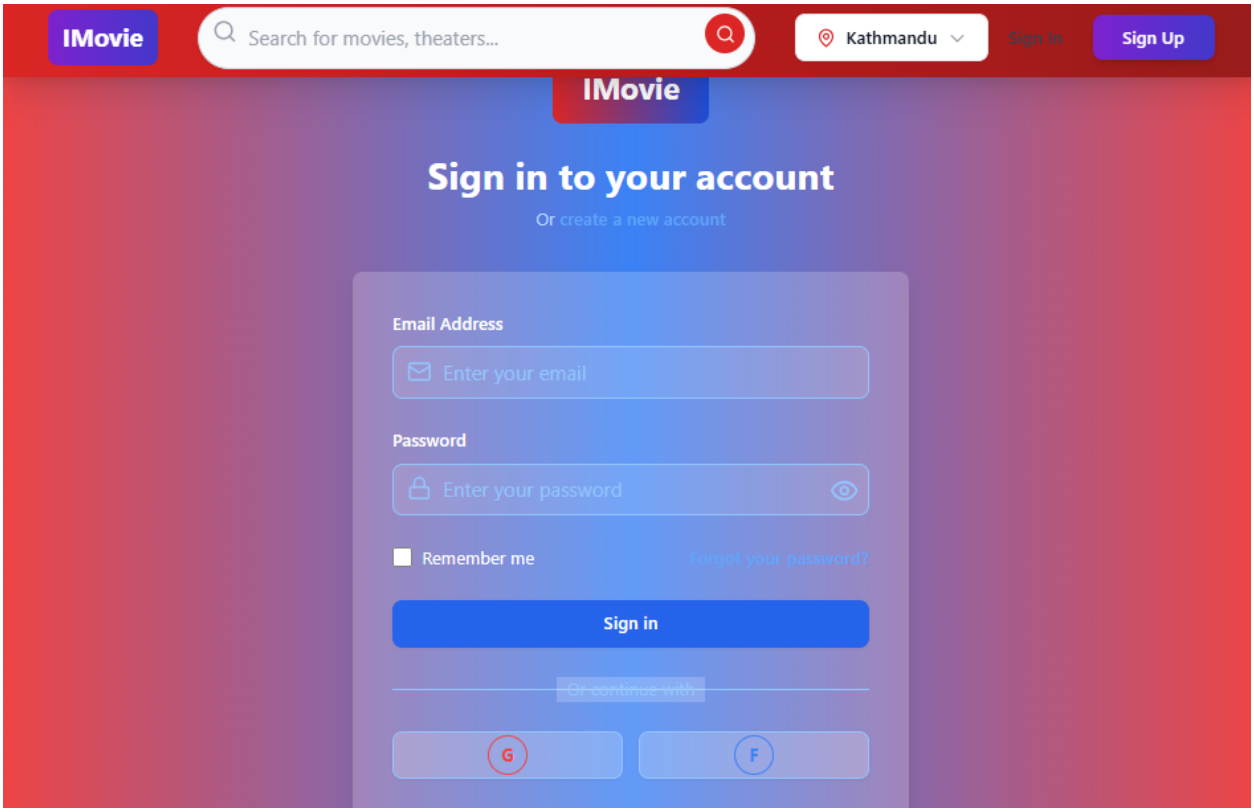
Home page



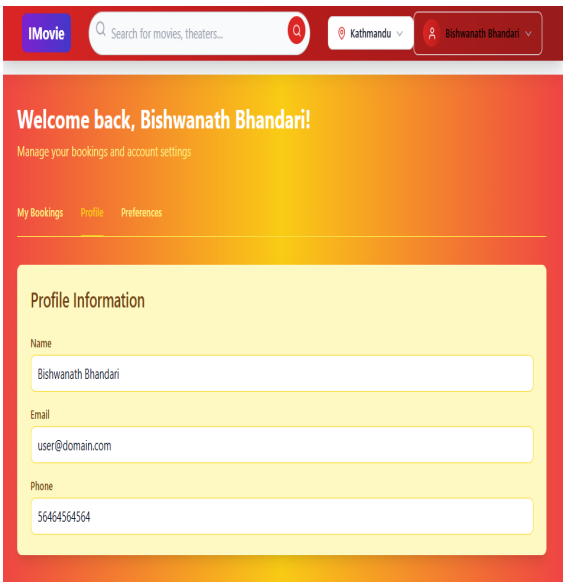
Sign up page

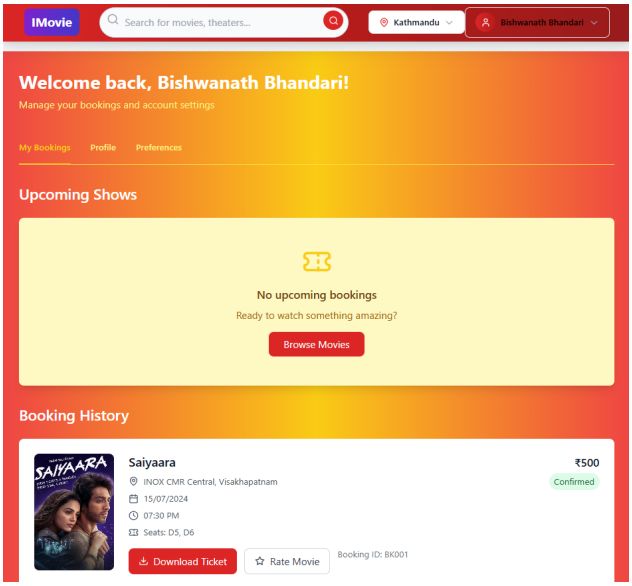


Sign in page

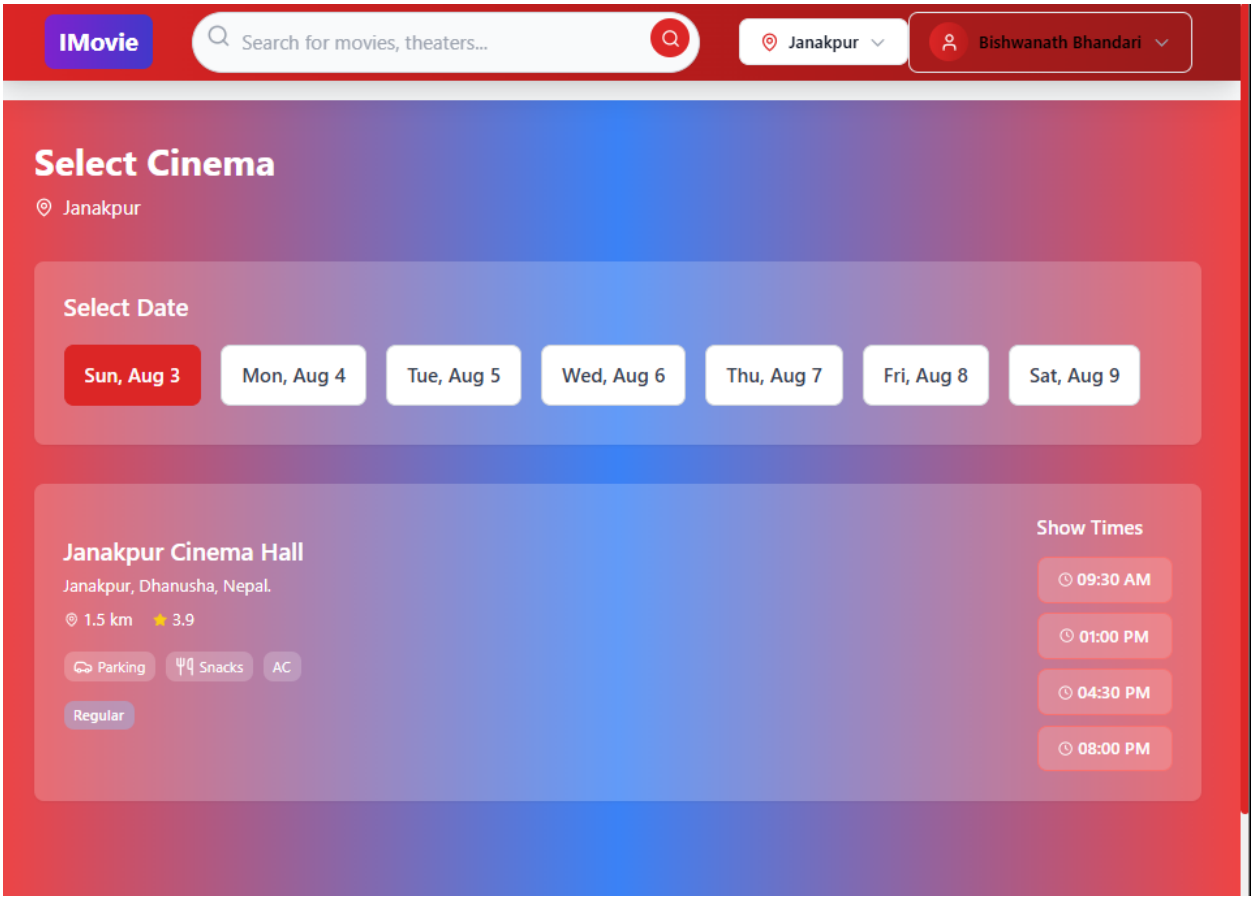


User Profile Page

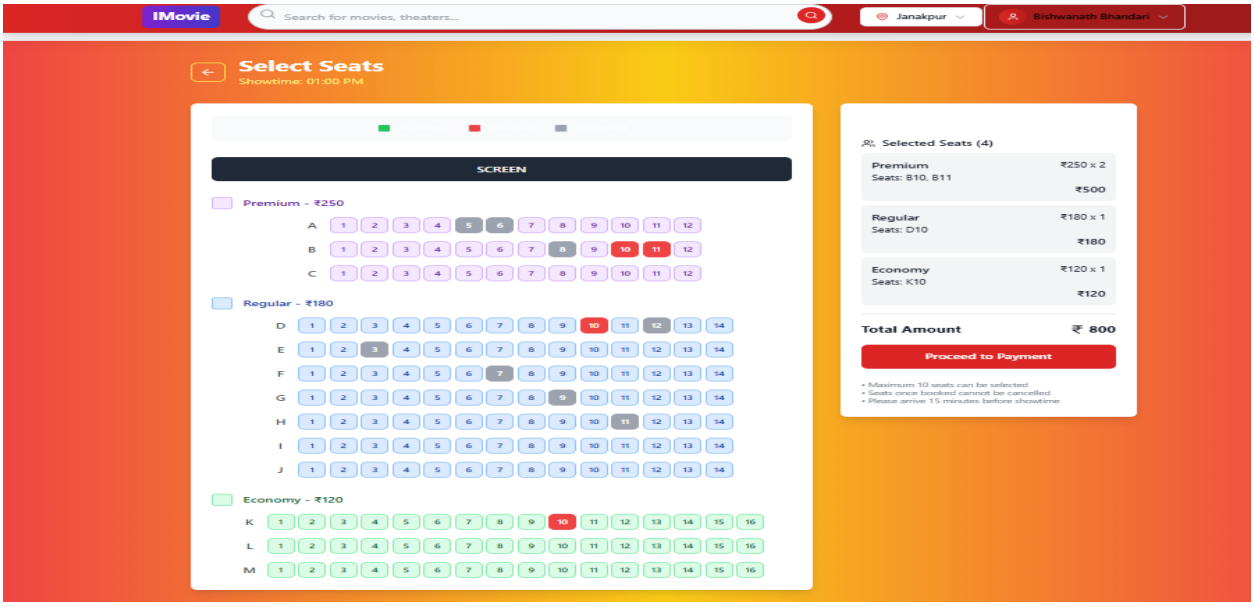




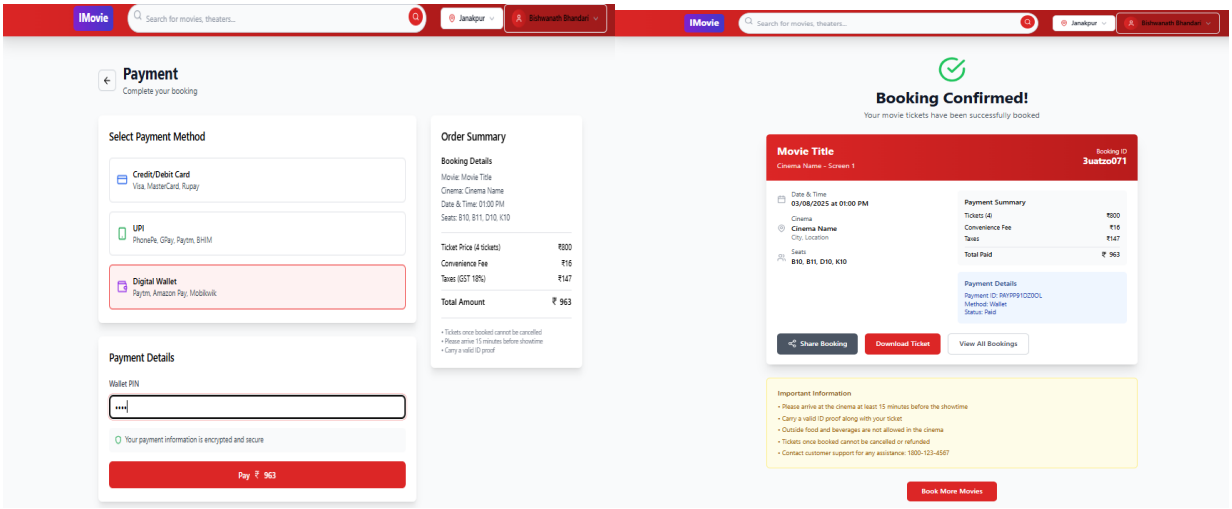
Bookings Page



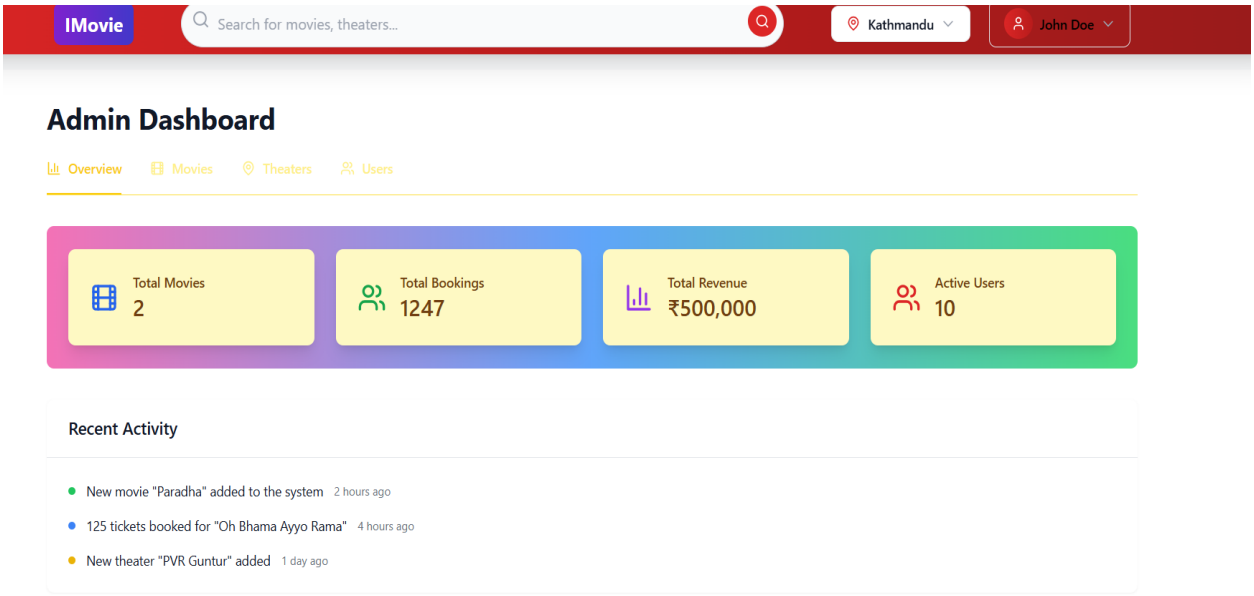
Seat Selection page

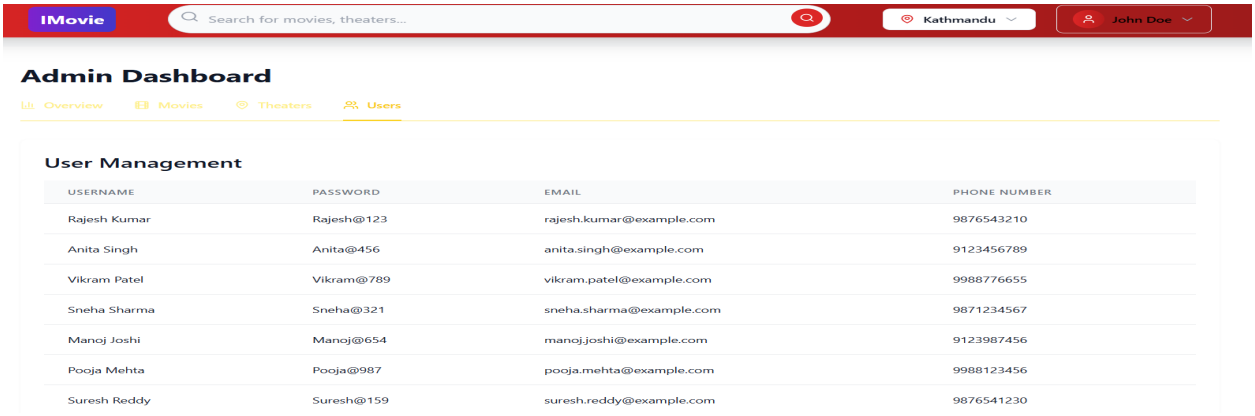


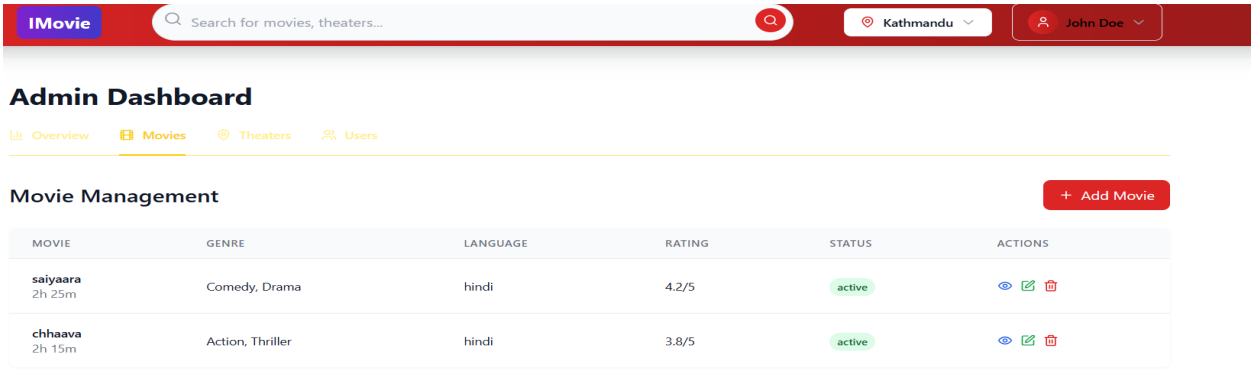
Ticket PaymentSuccess page

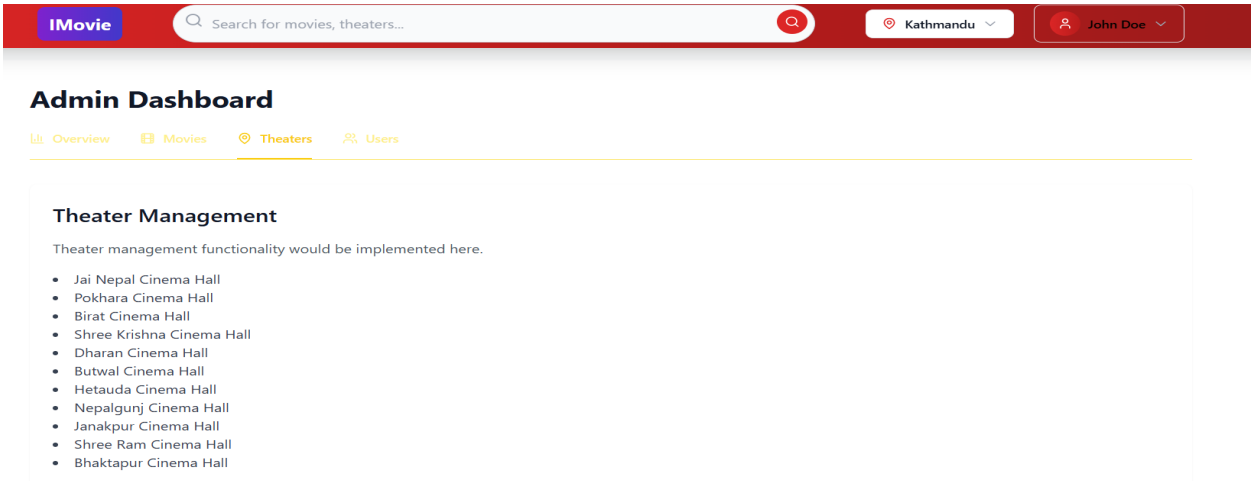


Admin DashBoardPage

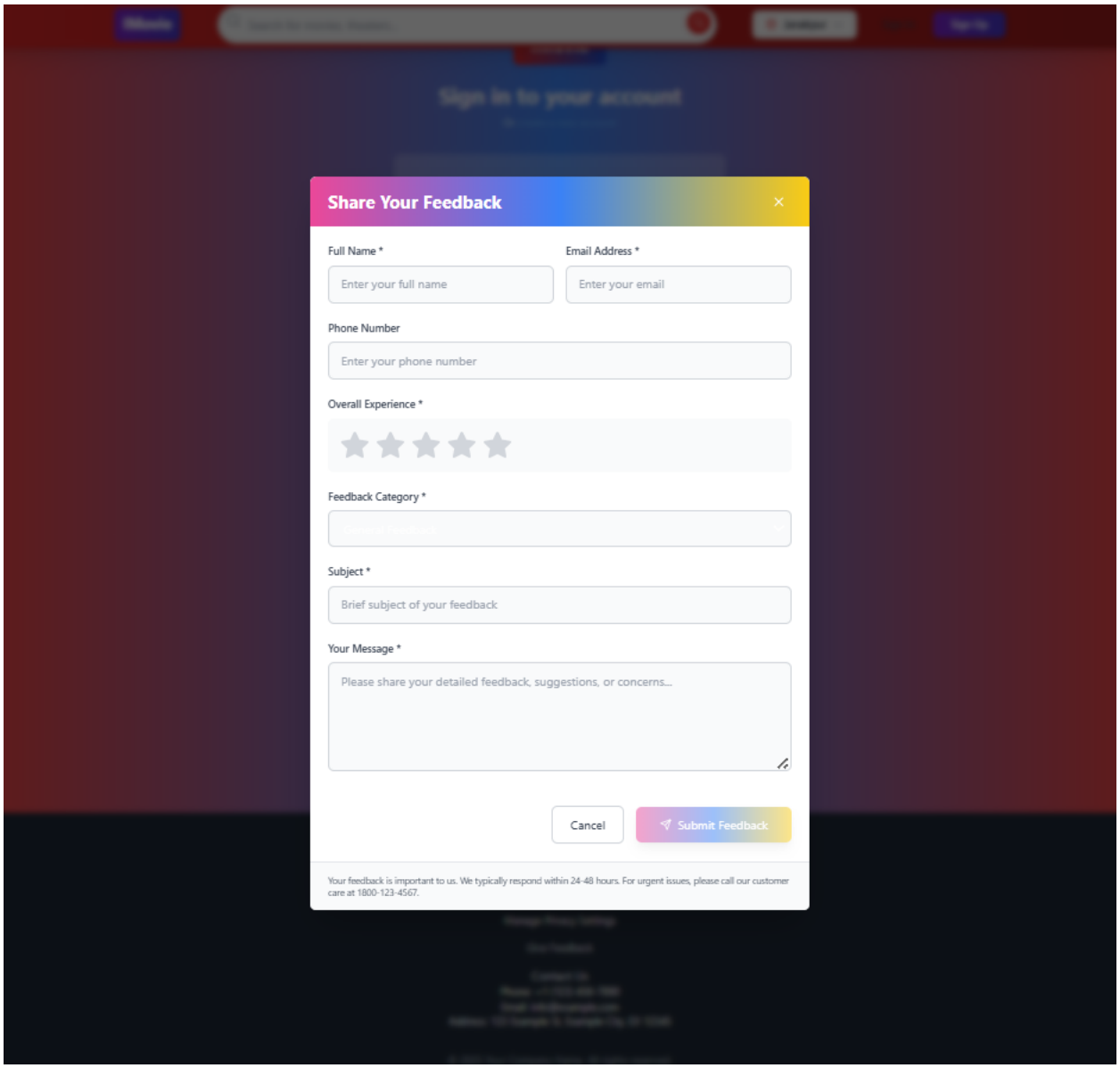



IMovie
Search for movies, theaters...
Kathmandu
John Doe
Admin Dashboard
Overview
Movies
Theaters
Users
User Management
USERNAME
PASSWORD
EMAIL
PHONE NUMBER
Rajesh Kumar
Rajesh@123
rajesh.kumar@example.com
9876543210
Anita Singh
Anita@456
anita.singh@example.com
9123456789
Vikram Patel
Vikram@789
vikram.patel@example.com
9988776655
Sneha Sharma
Sneha@321
sneha.sharma@example.com
9871234567
Manoj Joshi
Manoj@654
manoj.joshi@example.com
9123987456
Pooja Mehta
Pooja@987
pooja.mehta@example.com
9988123456
Suresh Reddy
Suresh@159
suresh.reddy@example.com
9876541230


IMovie
Search for movies, theaters...
Kathmandu
John Doe
Admin Dashboard
Overview
Movies
Theaters
Users
Movie Management
+ Add Movie
MOVIE
GENRE
LANGUAGE
RATING
STATUS
ACTIONS
saiyaara
2h 25m
Comedy, Drama
hindi
4.2/5
active
chhaava
2h 15m
Action, Thriller
hindi
3.8/5
active


IMovie
Search for movies, theaters...
Kathmandu
John Doe
Admin Dashboard
Overview
Movies
Theaters
Users
Theater Management
Theater management functionality would be implemented here.
Jai Nepal Cinema Hall
Pokhara Cinema Hall
Birat Cinema Hall
Shree Krishna Cinema Hall
Dharan Cinema Hall
Butwal Cinema Hall
Hetauda Cinema Hall
Nepalgunj Cinema Hall
Janakpur Cinema Hall
Shree Ram Cinema Hall
Bhaktapur Cinema Hall

Feedback Page



Conclusion

I-Movieз stands as a transformative leap in the realm of online movie ticket booking, seamlessly blending cutting-edge technology with user-centric design to redefine the cinema experience. By leveraging the speed and responsiveness of React and Vite, coupled with the robust data management of MongoDB, the platform delivers a lightning-fast, intuitive, and secure solution for movie enthusiasts. From effortless account creation and real-time seat selection to instant booking confirmations and digital tickets, I-Movieз eliminates the traditional hassles of ticket purchasing, offering unparalleled convenience and scalability. Its mobile-optimized design ensures accessibility across all devices, while its rigorously tested backend guarantees reliability even during peak hours. By prioritizing both functionality and user experience, I-Movieз not only simplifies the process of planning movie outings but also enhances the overall enjoyment of cinema-goers. With its innovative approach and commitment to excellence, I-Movieз sets a new standard for modern, hassle-free entertainment solutions, making every movie experience truly unforgettable.