

I-Movies : MovieTicket Booking System

Project Description:

Users can effortlessly browse movies, select showtime, and choose seats from the comfort of their homes. Using 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 availability tracking, it provides a personalized and convenient movie-going experience. This document provides a comprehensive guide for setup, development, and understanding of the application's technical architecture and functionalities.

Scenario

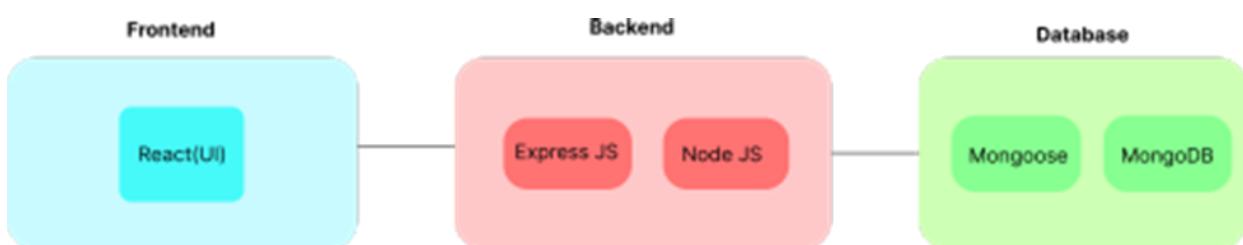
Imagine Sarah, a movie enthusiast, excitedly expecting the release of an anticipated film. With a busy schedule, she prefers the convenience of booking her movie tickets online. Logging into the iMovies app, Sarah navigates effortlessly through the user-friendly interface, browsing the list of usable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

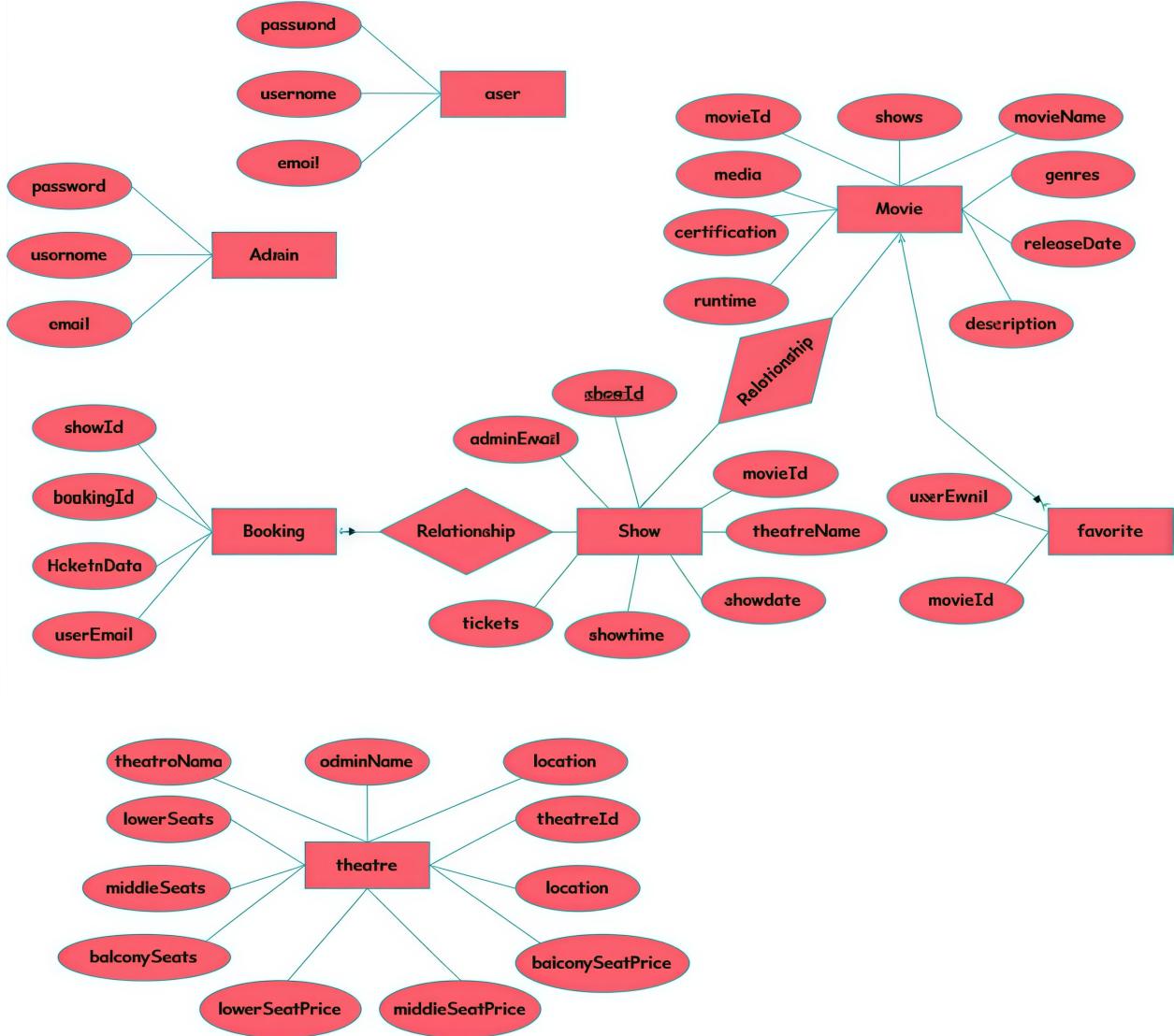
The technical architecture of the iMovies app follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses not only the user interface and presentation but also incorporates the axiom library to connect with backend easily by using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

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

2. MongoDB

Set up a MongoDB database for 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** (ODM for MongoDB) to connect Node.js with 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.firebaseio.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:

- **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 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:

- frontend/src/ for React components
- frontend/public/ for static assets
- 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:

- 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)
```

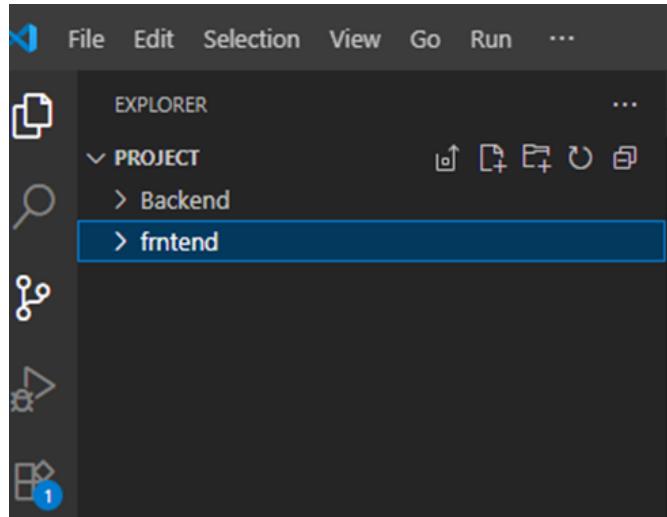
```
cd frontend
```

```
npm start
```

```
#Terminal 2 (backend)
```

```
cd Backend
```

```
node server.js
```



2. Backend Development

- **Initialize Express Server**

- Navigate to your Backend folder and install required 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 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**

- Add this code to your indexserver.js file:

```
Backend > JS server.js > ...
1  const dotenv = require('dotenv');
2
3  // Handle uncaught exceptions
4  process.on('uncaughtException', (err) => {
5    console.log('UNCAUGHT EXCEPTION! ⚡ Shutting down...');
6    console.log(err.name, err.message);
7    process.exit(1);
8  });
9
10 // Load environment variables
11 dotenv.config();
12
13 // Check if MongoDB URI is configured
14 if (!process.env.MONGODB_URI) {
15   console.log('MONGODB_URI not found in environment variables, using default: mongodb://127.0.0.1:27017/j
16 }
17
18 const app = require('./app');
19 const connectDB = require('./config/database');
20
21 // Connect to MongoDB with error handling
22 connectDB().catch(err => {
23   console.error('Failed to connect to MongoDB:', err.message);
24   process.exit(1);
25 })
```

Authentication Setup

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

```
Backend > models > JS User.js > ...
1  const mongoose = require('mongoose');
2  const bcrypt = require('bcryptjs');
3
4  const userSchema = new mongoose.Schema({
5    name: {
6      type: String,
7      required: [true, 'Please provide a name'],
8      trim: true,
9    },
10   email: {
11     type: String,
12     required: [true, 'Please provide an email'],
13     unique: true,
14     lowercase: true,
15     trim: true,
16   },
17   phone: {
18     type: String,
19     required: [true, 'Please provide a phone number'],
20   },
21   password: {
22     type: String,
23     required: [true, 'Please provide a password'],
24     minlength: 6,
```

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

```
Backend > routes > JS authRoutes.js
1  const express = require('express');
2  const authController = require('../controllers/authController');
3
4  const router = express.Router();
5
6  // Public routes
7  router.post('/register', authController.register);
8  router.post('/login', authController.login);
9
10 // Protected routes
11 router.use(authController.protect);
12 router.get('/me', authController.getMe);
13
14 module.exports = router;
```

■ Error Handling Middleware

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

```
Backend > middleware > JS errorHandler.js
1  const AppError = require('../utils/appError');
2
3  const handleCastErrorDB = (err) => {
4    const message = `Invalid ${err.path}: ${err.value}.`;
5    return new AppError(message, 400);
6  };
7
8  const handleDuplicateFieldsDB = (err) => {
9    const value = errerrmsg?.match(/([""])(\\?.)*?\1/.?.[0] || 'duplicate value';
10   const message = `Duplicate field value: ${value}. Please use another value!`;
11   return new AppError(message, 400);
12 };
13
14 const handleValidationErrorDB = (err) => {
15   const errors = Object.values(err.errors).map((el) => el.message);
16   const message = `Invalid input data. ${errors.join(' ')}`;
17   return new AppError(message, 400);
18 };
19
20 const handleJWTError = () =>
21   new AppError('Invalid token. Please log in again!', 401);
22
23 const handleJWTExpiredError = () =>
24   new AppError('Your token has expired! Please log in again.', 401);
```

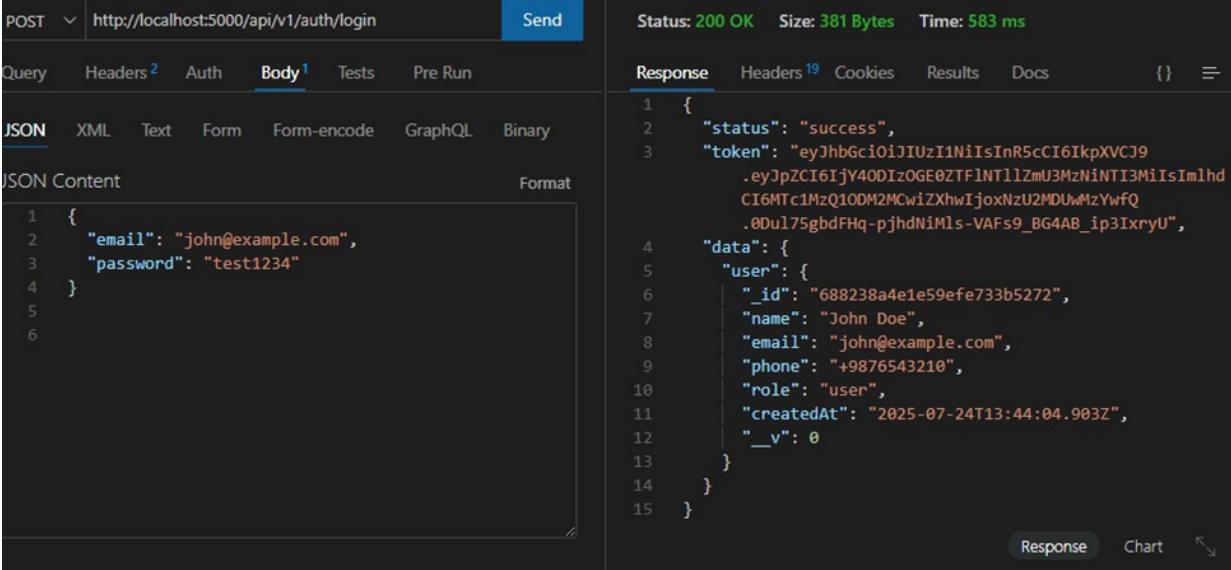
Start Your Server

- Run the backend server with:

bash

node server.js

■ Testing The Backend by using the thunder client:



The screenshot shows the Thunder client interface. On the left, a POST request is being sent to `http://localhost:5000/api/v1/auth/login`. The 'Body' tab is selected, displaying a JSON payload:

```
1 {
2   "email": "john@example.com",
3   "password": "test1234"
4 }
```

On the right, the response is shown with a status of **200 OK**, size of **381 Bytes**, and time of **583 ms**. The 'Response' tab is selected, displaying the JSON response:1 {
2 "status": "success",
3 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
4 .eyJpZCI6IjY4ODIzOGE0ZTF1NTllZmU3MzMzNlNTI3MiIsImhd
5 CI6MTc1MzQ1ODM2MCwiZXhwIjoxNzU2MDUwMzYwfQ
6 .0Dul75gbdFHq-pjhdNiMls-VAFs9_BG4AB_ip3IxryU",
7 "data": {
8 "user": {
9 "_id": "688238a4e1e59efe733b5272",
10 "name": "John Doe",
11 "email": "john@example.com",
12 "phone": "+9876543210",
13 "role": "user",
14 "createdAt": "2025-07-24T13:44:04.903Z",
15 "__v": 0
16 }
17 }
18 }

3. Database Development

Creating Data Schemas

User Schema:

- Schema: userSchema
- Model: 'User'
- 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.

```

server > JS index.js > ..
1 import express from "express";
2 import mongoose from "mongoose";
3 import cors from "cors";
4 import dotenv from "dotenv";
5
6 dotenv.config({ path: "./.env" });
7
8 const app = express();
9 app.use(express.json());
10 app.use(cors());
11
12 app.listen(3001, () => {
13   console.log("App server is running on port 3001");
14 });
15
16 const MongoURI = process.env.DRIVER_LINK;
17 const connectToMongo = async () => {
18   try {
19     await mongoose.connect(MongoURI);
20     console.log("Connected to your MongoDB database successfully");
21   } catch (error) {
22     console.log(error.message);
23   }
24 };
25
26 connectToMongo();
27

```

FOLDERS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

● PS D:\shopEZ> cd server
● PS D:\shopEZ\server> node index.js
App server is running on port 3001
bad auth : authentication failed
○ PS D:\shopEZ\server> node index.js
App server is running on port 3001
Connected to your MongoDB database successfully

```

> OUTLINE > TIMELINE > NPM SCRIPTS

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.

```
1 const mongoose = require("mongoose");
2
3 const theatreSchema = new mongoose.Schema({
4   theatreName: {
5     type: String,
6     required: true,
7   },
8   adminName: {
9     type: String,
10    required: true,
11  },
12   theatreId: {
13     type: String,
14     required: true,
15  },
16   location: {
17     type: String,
18     required: true,
19  },
20   balconySeatPrice: {
21     type: Number,
22     required: true,
23  },
24   middleSeatPrice: {
25     type: Number,
26     required: true,
27  },
28   lowerSeatPrice: {
29     type: Number,
30     required: true,
31  },
32   balconySeats: {
33     type: Object,
34     required: true,
35  },
36   middleSeats: {
37     type: Object,
38     required: true,
39  },
40   lowerSeats: {
41     type: Object,
42     required: true,
43  },
44   adminEmail: {
45     type: String,
46     required: true,
47   }
});
```

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.

```
1 const mongoose = require("mongoose");
2
3 const showSchema = new mongoose.Schema({
4   //show created by admin
5   adminEmail: {
6     type: String,
7     required: true,
8   },
9   showId: {
10     type: String,
11     required: true,
12   },
13   movieId: {
14     type: String,
15     required: true,
16   },
17
18   theatreName: {
19     type: String,
20     required: true,
21   },
22   showdate: {
23     type: String,
24     required: true,
25   },
26   showtime: {
27     type: String,
28     required: true,
29   },
30   tickets: {
31     type: Object,
32   },
33 });
34
35 module.exports = mongoose.model("show", showSchema);
36
```

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.

```
1 const mongoose = require("mongoose");
2
3 const movieSchema = new mongoose.Schema({
4   movieName: {
5     type: String,
6     required: true,
7   },
8   description: {
9     type: String,
10    required: true,
11   },
12   genres: {
13     type: String,
14     required: true,
15   },
16   releaseDate: {
17     type: String,
18     required: true,
19   },
20   runtime: {
21     type: Number,
22     required: true,
23   },
24   certification: {
25     type: String,
26     required: true,
27   },
28   media: {
29     type: String,
30     required: true,
31   },
32   movieId: {
33     type: String,
34     required: true,
35   },
36   //contains showid's
37   shows: [{ type: String }],
38 });
39
40 module.exports = mongoose.model("Movie", movieSchema);
41
```

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.

```
1 const mongoose = require("mongoose");
2
3 const favoriteSchema = new mongoose.Schema({
4   userEmail: {
5     type: String,
6     required: true,
7   },
8   movieId: {
9     type: String,
10    required: true,
11   },
12 });
13
14 module.exports = mongoose.model("Favorite",
15   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.

```
1 const mongoose = require("mongoose");
2
3 const bookingModel = new mongoose.Schema({
4   bookingId: {
5     type: String,
6     required: true,
7   },
8   userEmail: {
9     type: String,
10    required: true,
11   },
12   showId: {
13     type: String,
14     required: true,
15   },
16   ticketsData: {
17     type: Object,
18     requires: true,
19   },
20 });
21
22 module.exports = mongoose.model("Booking",
23   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: For routing
- @mui/material: UI components

Step 3: Folder Structure

```
frntend/
├── node_modules/      # All npm packages (ignored in 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 X
frntend > src > components > Header.jsx > locations
1 import React, { useState } from 'react';
2 import { Link, useNavigate } from 'react-router-dom';
3 import { Search, MapPin, User, Menu, X } from 'lucide-react';
4
5 const locations = [
6   'Kathmandu',
7   'Pokhara',
8   'Biratnagar',
9   'Bharatpur',
10  'Dharan',
11  'Butwal',
12  'Hetauda',
13  'Nepalganj',
14  'Janakpur',
15  'Bhaktapur',
16  'Lalitpur',
17  'Itahari',
18  'Dhangadhi',
19  'Birgunj',
20  'Tansen',
21  'Ghorahi',
22  'Siddharthanagar'
23 ];
24
```

- **Footer.jsx**

Copyright information

Contact links

Social media icons

```
❶ Footer.jsx ✘
frontend > src > components > Footer.jsx [❷] Footer
1 import React, { useState } from 'react';
2 import { Facebook, Twitter, Instagram, Youtube, Rss } from 'lucide-react';
3 import FeedbackForm from './FeedbackForm';
4
5 const Footer = () => {
6   const [showFeedback, setShowFeedback] = useState(false);
7
8   return (
9     <footer className="bg-gray-800 text-white p-6">
10      <div className="container mx-auto flex flex-col items-center space-y-4">
11        {/* Social Media Icons */}
12        <div className="flex space-x-4">
13          <a href="https://facebook.com" target="_blank" rel="noopener noreferrer">
14            <Facebook className="h-6 w-6 hover:text-gray-400" />
15          </a>
16          <a href="https://twitter.com" target="_blank" rel="noopener noreferrer">
17            <Twitter className="h-6 w-6 hover:text-gray-400" />
18          </a>
19          <a href="https://instagram.com" target="_blank" rel="noopener noreferrer">
20            <Instagram className="h-6 w-6 hover:text-gray-400" />
21          </a>
22          <a href="https://youtube.com" target="_blank" rel="noopener noreferrer">
23            <Youtube className="h-6 w-6 hover:text-gray-400" />
24          </a>
25          <a href="/rss" target="_blank" rel="noopener noreferrer">
26            <Rss className="h-6 w-6 hover:text-gray-400" />

```

- **FeedbackForm.jsx**

Rating input

Comment textarea

Submission button

```
❶ FeedbackForm.jsx ✘
D:\project\frntend\src\components\FeedbackForm.jsx (preview ⓘ) backForm
1 import React, { useState } from 'react';
2 import { X, Star, Send } from 'lucide-react';
3
4 const FeedbackForm = ({ onClose }) => {
5   const [formData, setFormData] = useState({
6     name: '',
7     email: '',
8     phone: '',
9     rating: 0,
10    category: 'general',
11    subject: '',
12    message: ''
13  });
14   const [isSubmitting, setIsSubmitting] = useState(false);
15
16   const handleInputChange = (e) => {
17     const { name, value } = e.target;
18     setFormData(prev => ({
19       ...prev,
20       [name]: value
21     }));
22   };
23
24   const handleRatingClick = (rating) => {
25     setFormData(prev => ({
26       ...prev,
```

Step2: Implement Styling

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



The screenshot shows a code editor window with the title '# index.css 5 X'. The file path is 'D:\project\frmtend\src\index.css (preview)'. The code in the editor is:

```
1  @tailwind base;
2  @tailwind components;
3  @tailwind utilities;
4
5  /* Custom scrollbar */
6  ::-webkit-scrollbar {
7    width: 8px;
8  }
9
10 ::-webkit-scrollbar-track {
11   background: #f1f1f1;
12   border-radius: 4px;
13 }
14
15 ::-webkit-scrollbar-thumb {
16   background: #dc2626;
17   border-radius: 4px;
18 }
19
20 ::-webkit-scrollbar-thumb:hover {
21   background: #b91c1c;
22 }
23
24 /* Smooth scrolling */
25 html {
26   scroll-behavior: smooth;
```

Step3: Set Up Routing

Define routes in App.jsx

- Home (/)
- Movies (/movies)
- Cinema Selection (/cinemas/:movield)
- Seat Selection (/book/:showtimeld)
- Payment (/payment)
- Booking Confirmation (/confirmation)
- Login (/login)
- Register (/register)
- User Dashboard (/dashboard)
- Admin Dashboard (/admin)

```

frntend > src > App.jsx > ...
1 import React, { useState, useEffect } from 'react';
2 import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
3 import Header from './components/Header';
4 import Footer from './components/Footer';
5 import Home from './pages/Home';
6 import MovieDetails from './pages/MovieDetails';
7 import CinemaSelection from './pages/CinemaSelection';
8 import SeatSelection from './pages/SeatSelection';
9 import Payment from './pages/Payment';
10 import BookingConfirmation from './pages/BookingConfirmation';
11 import UserDashboard from './pages/UserDashboard';
12 import AdminDashboard from './pages/AdminDashboard';
13 import AddMovie from './pages/AddMovie';
14 import Login from './pages/Login';
15 import Register from './pages/Register';
16
17 function App() {
18   const [user, setUser] = useState(null);
19   const [selectedLocation, setSelectedLocation] = useState('Visakhapatnam (Vizag)');
20
21   useEffect(() => {
22     // Check for logged in user
23     const token = localStorage.getItem('token');
24     const userData = localStorage.getItem('user');
25     if (token && userData) {
26       setUser(JSON.parse(userData));

```

3.Implement Frontend Logic

Step1: API Services

1. Create services/movieService.js:

- fetchMovies()
- getMovieDetails(id)
- getShowtimes(movieId)

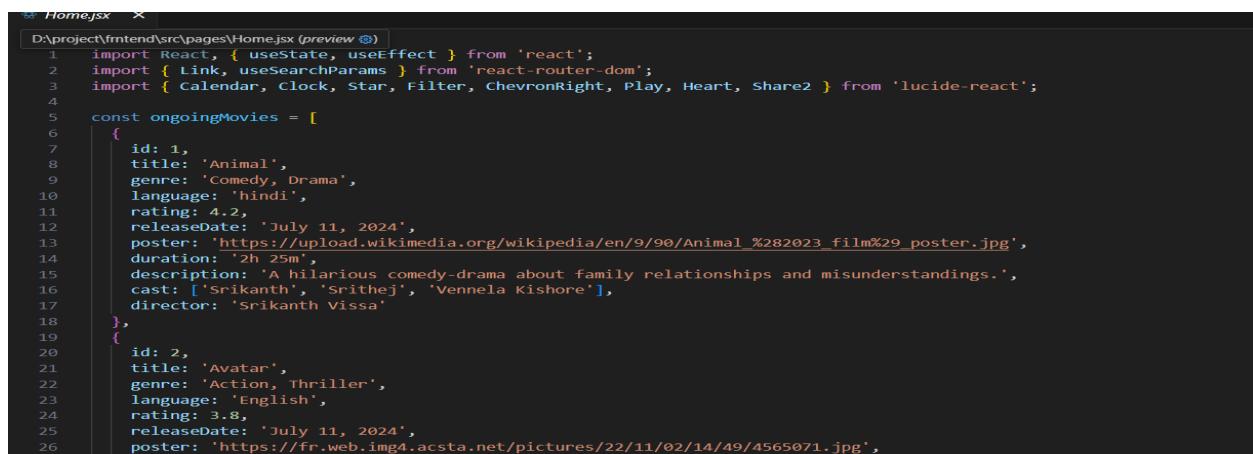
2. Create services/bookingService.js:

- bookSeats(showtimeId, seats)
- getBookingHistory()

Step 2: Data Binding

1. Home Page:

- Fetch and display featured movies
- Implement carouselfor promot

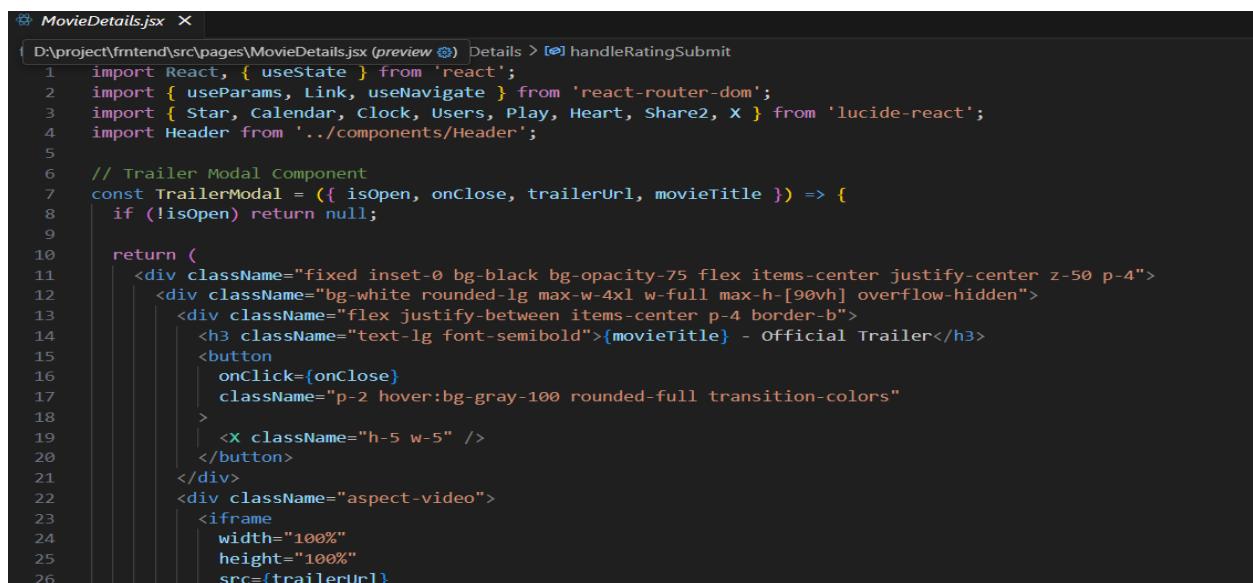


```
Home.jsx
D:\project\frntend\src\pages\Home.jsx (preview ⓘ)
1 import React, { useState, useEffect } from 'react';
2 import { Link, useSearchParams } from 'react-router-dom';
3 import { Calendar, Clock, Star, Filter, ChevronRight, Play, Heart, Share2 } from 'lucide-react';
4
5 const ongoingMovies = [
6   {
7     id: 1,
8     title: 'Animal',
9     genre: 'Comedy, Drama',
10    language: 'hindi',
11    rating: 4.2,
12    releaseDate: 'July 11, 2024',
13    poster: 'https://upload.wikimedia.org/wikipedia/en/9/90/Animal_%282023_film%29_poster.jpg',
14    duration: '2h 25m',
15    description: 'A hilarious comedy-drama about family relationships and misunderstandings.',
16    cast: ['Srikanth', 'Srithej', 'Vennela Kishore'],
17    director: 'Srikanth Vissa'
18  },
19  {
20    id: 2,
21    title: 'Avatar',
22    genre: 'Action, Thriller',
23    language: 'English',
24    rating: 3.8,
25    releaseDate: 'July 11, 2024',
26    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
1 import React, { useState } from 'react';
2 import { useParams, Link, useNavigate } from 'react-router-dom';
3 import { Star, Calendar, Clock, Users, Play, Heart, Share2, X } from 'lucide-react';
4 import Header from '../components/Header';
5
6 // Trailer Modal Component
7 const TrailerModal = ({ isOpen, onClose, trailerUrl, movieTitle }) => {
8   if (!isOpen) return null;
9
10  return (
11    <div className="fixed inset-0 bg-black bg-opacity-75 flex items-center justify-center z-50 p-4">
12      <div className="bg-white rounded-lg max-w-4xl w-full max-h-[90vh] overflow-hidden">
13        <div className="flex justify-between items-center p-4 border-b">
14          <h3 className="text-lg font-semibold">{movieTitle} - Official Trailer</h3>
15          <button
16            onClick={onClose}
17            className="p-2 hover:bg-gray-100 rounded-full transition-colors"
18          >
19            <X className="h-5 w-5" />
20          </button>
21        </div>
22        <div className="aspect-video">
23          <iframe
24            width="100%"
25            height="100%"
26            src={trailerUrl}
27          >
28        </div>
29      </div>
30    </div>
31  );
32}
```

Seat Selection:

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

```
SeatSelection.jsx X
frontend > src > pages > SeatSelection.jsx > ...
1 import React, { useState, useEffect } from 'react';
2 import { useParams, useNavigate } from 'react-router-dom';
3 import { ArrowLeft, Users, IndianRupee } from 'lucide-react';
4
5 const SeatSelection = ({ user }) => {
6   const { movieId, cinemaId, showtime } = useParams();
7   const navigate = useNavigate();
8
9   const [selectedSeats, setSelectedSeats] = useState([]);
10  const [loading, setLoading] = useState(false);
11
12  // Check if user is logged in
13  useEffect(() => {
14    if (!user) {
15      if (window.confirm('Please login to book tickets. Would you like to login now?')) {
16        navigate('/login');
17      } else {
18        navigate(-1);
19      }
20    }
21  }, [user, navigate]);
22
23  // Mock seat data - in real app, this would come from API
24  const seatLayout = {
25    premium: {
26      name: 'Premium',
27    },
28  };
29
30  const calculateTotalPrice = (seats) => {
31    const rates = {
32      'D5': 500,
33      'D6': 500,
34    };
35
36    return seats.reduce((total, seat) => total + rates[seat], 0);
37  };
38
39  const handleSeatSelect = (seat) => {
40    const newSelectedSeats = [...selectedSeats];
41
42    if (selectedSeats.includes(seat)) {
43      newSelectedSeats.splice(selectedSeats.indexOf(seat), 1);
44    } else {
45      newSelectedSeats.push(seat);
46    }
47
48    setSelectedSeats(newSelectedSeats);
49  };
50
51  const handleLoading = (status) => {
52    setLoading(status);
53  };
54
55  const handleLogout = () => {
56    localStorage.removeItem('user');
57    navigate('/login');
58  };
59
60  return (
61    <div>
62      <h1>Seat Selection</h1>
63      <p>Movie ID: {movieId}</p>
64      <p>Cinema ID: {cinemaId}</p>
65      <p>Showtime: {showtime}</p>
66
67      <table border="1">
68        <thead>
69          <tr>
70            <th>Seat Type</th>
71            <th>Status</th>
72          </tr>
73        </thead>
74        <tbody>
75          <tr>
76            <td>D5</td>
77            <td>Available</td>
78          </tr>
79          <tr>
80            <td>D6</td>
81            <td>Available</td>
82          </tr>
83        </tbody>
84      </table>
85
86      <button onClick={handleSeatSelect}>Select</button>
87      <button onClick={handleLogout}>Logout</button>
88    </div>
89  );
90
91  export default SeatSelection;
```

Step3: State Management

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

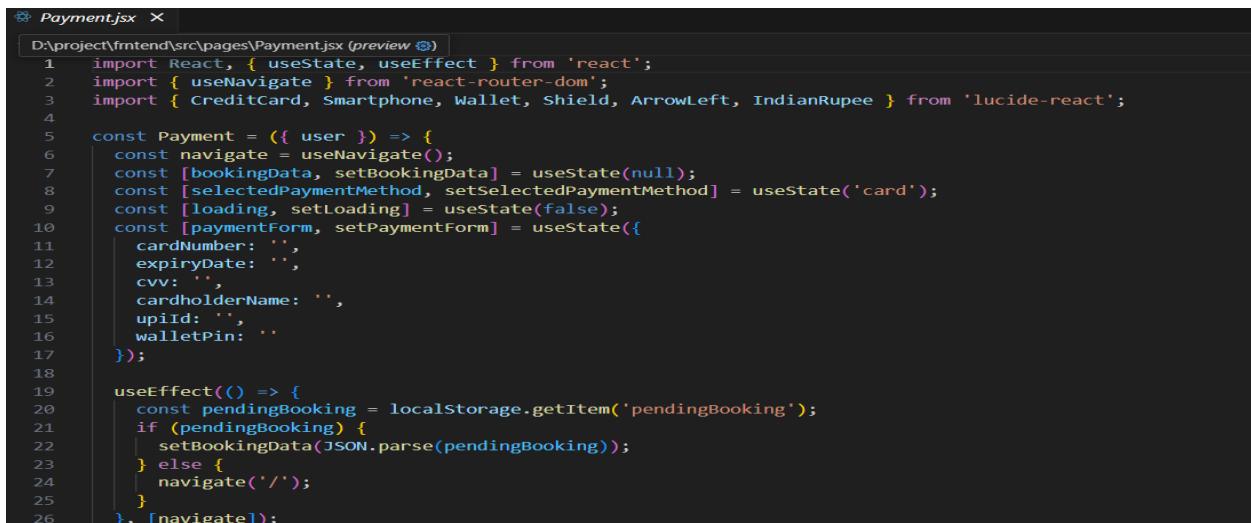
```
UserDashboard.jsx X
frontend > src > pages > UserDashboard.jsx > UserDashboard
1 import React, { useState, useEffect } from 'react';
2 import { Calendar, Clock, MapPin, Star, Ticket, Download } from 'lucide-react';
3
4 const UserDashboard = ({ user }) => {
5   const [activeTab, setActiveTab] = useState('bookings');
6   const [bookings, setBookings] = useState([]);
7
8   useEffect(() => {
9     // Mock bookings data - in real app, fetch from API
10    const mockBookings = [
11      {
12        id: 'BK001',
13        movieTitle: 'Saiyaara',
14        cinema: 'INOX CMR Central',
15        location: 'Visakhapatnam',
16        date: '2024-07-15',
17        time: '07:30 PM',
18        seats: ['D5', 'D6'],
19        amount: 500,
20        status: 'confirmed',
21        poster: 'https://myhdmovie365.s3.amazonaws.com/Top%20Banner%202_Mobile.jpg',
22      },
23      {
24        id: 'BK002',
25        movieTitle: 'Chhaava',
26        cinema: 'PVR Vizag',
27      },
28    ];
29
30    setBookings(mockBookings);
31  }, [activeTab]);
32
33  const handleTabChange = (tab) => {
34    setActiveTab(tab);
35  };
36
37  return (
38    <div>
39      <h1>User Dashboard</h1>
40      <p>User: {user}</p>
41
42      <ul style={{listStyleType: 'none', padding: 0}}>
43        <li
44          style={{border: '1px solid #ccc', padding: 5px, margin: 5px 0}}
45          >Bookings</li>
46        <li
47          style={{border: '1px solid #ccc', padding: 5px, margin: 5px 0}}
48          >Profile</li>
49        <li
50          style={{border: '1px solid #ccc', padding: 5px, margin: 5px 0}}
51          >Logout</li>
52        </ul>
53
54      <div>
55        <h2>Bookings</h2>
56        <table border="1">
57          <thead>
58            <tr>
59              <th>Booking ID</th>
60              <th>Movie Title</th>
61              <th>Cinema</th>
62              <th>Location</th>
63              <th>Date</th>
64              <th>Time</th>
65              <th>Seats</th>
66              <th>Amount</th>
67              <th>Status</th>
68            </tr>
69          </thead>
70          <tbody>
71            <tr>
72              <td>BK001</td>
73              <td>Saiyaara</td>
74              <td>INOX CMR Central</td>
75              <td>Visakhapatnam</td>
76              <td>2024-07-15</td>
77              <td>07:30 PM</td>
78              <td>D5, D6</td>
79              <td>500</td>
80              <td>Confirmed</td>
81            </tr>
82            <tr>
83              <td>BK002</td>
84              <td>Chhaava</td>
85              <td>PVR Vizag</td>
86              <td></td>
87              <td></td>
88              <td></td>
89              <td></td>
90              <td></td>
91              <td></td>
92            </tr>
93          </tbody>
94        </table>
95      </div>
96    </div>
97  );
98
99  export default UserDashboard;
```

2 . Authentication Flow

- Implement login/logout
- Store JWT tokens
- Protected routes

3 . Payment Integration

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

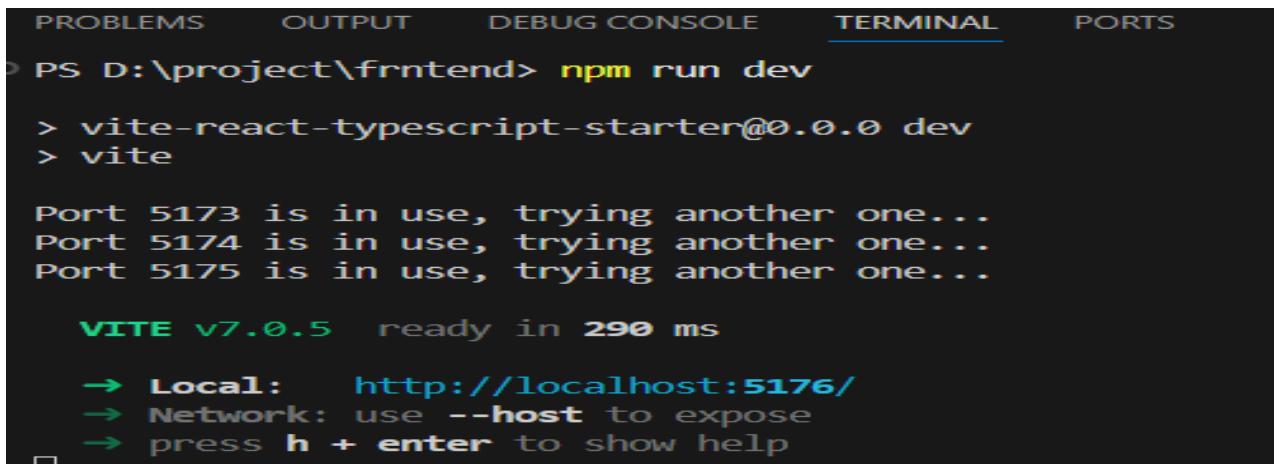


```
Payment.jsx
D:\project\frntend\src\pages\Payment.jsx (preview)
1 import React, { useState, useEffect } from 'react';
2 import { useNavigate } from 'react-router-dom';
3 import { CreditCard, Smartphone, Wallet, Shield, ArrowLeft, IndianRupee } from 'lucide-react';
4
5 const Payment = ({ user }) => {
6   const navigate = useNavigate();
7   const [bookingData, setBookingData] = useState(null);
8   const [selectedPaymentMethod, setSelectedPaymentMethod] = useState('card');
9   const [loading, setLoading] = useState(false);
10  const [paymentForm, setPaymentForm] = useState({
11    cardNumber: '',
12    expiryDate: '',
13    cvv: '',
14    cardholderName: '',
15    upiId: '',
16    walletPin: ''
17 });
18
19 useEffect(() => {
20   const pendingBooking = localStorage.getItem('pendingBooking');
21   if (pendingBooking) {
22     setBookingData(JSON.parse(pendingBooking));
23   } else {
24     navigate('/');
25   }
26 }, [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

The screenshot shows the movie search interface with a yellow header bar. The top navigation includes the iMovie logo, a search bar, and user account options (Visakhapatnam, Sign In, Sign Up). Below the header are two filter dropdowns labeled 'Filters' and 'All'. The main content area is divided into two sections: 'Now Showing' and 'Coming Soon'.
Now Showing: This section displays five movies with their posters, titles, genres, release dates, and runtimes. Each movie card includes a 'Book Now' button.

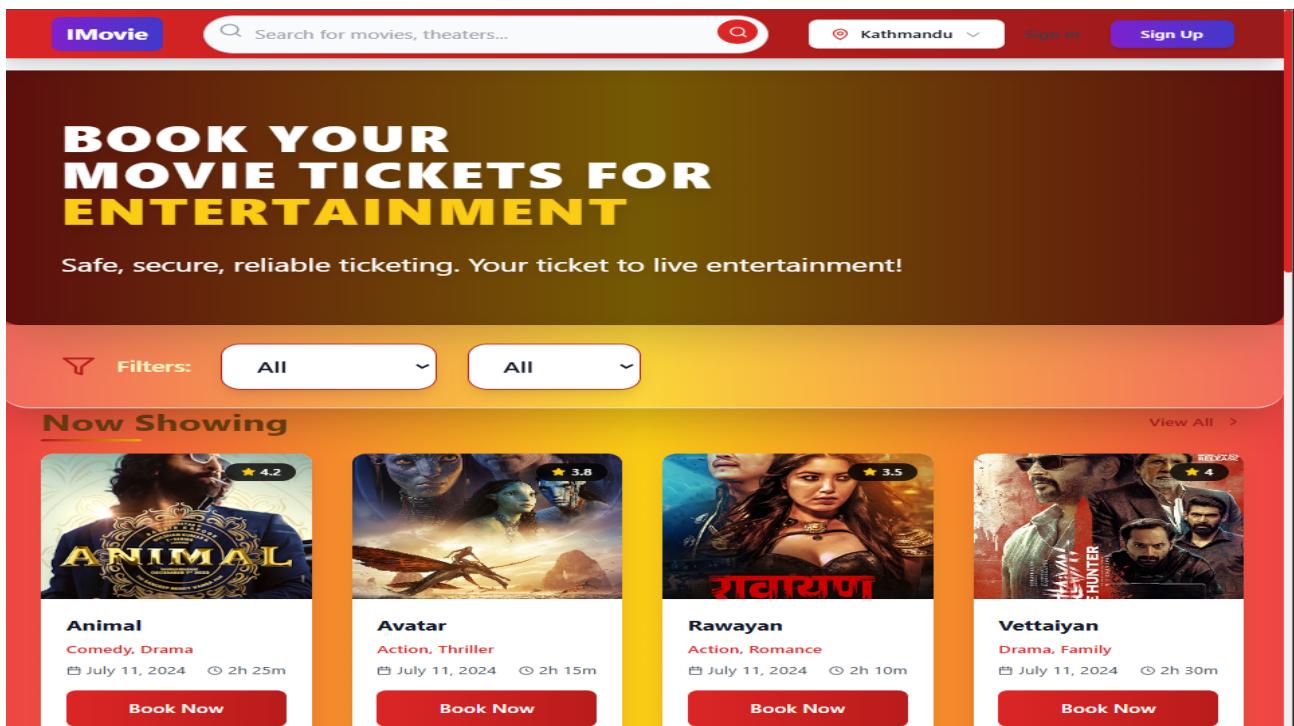
- Animal**: Comedy, Drama. Release: July 11, 2024, Runtime: 2h 25m.
- Avatar**: Action, Thriller. Release: July 11, 2024, Runtime: 2h 15m.
- Rawayan**: Action, Romance. Release: July 11, 2024, Runtime: 2h 10m.
- Vettaiyan**: Drama, Family. Release: July 11, 2024, Runtime: 2h 30m.
- Saiyaara**: Romance, Drama, love. Release: July 14, 2024, Runtime: 2h 35m.

Coming Soon: This section displays five movies with their posters, titles, genres, release dates, and runtimes. Each movie card includes a 'Coming Soon' button.

- Sikandar**: Action, Drama. Release: July 24, 2024, Runtime: 2h 22m.
- shaistaan**: Thriller, mysterious, horror. Release: July 24, 2024, Runtime: 2h 45m.
- Mahavitar Narsimha**: Mythology, Action. Release: July 25, 2024, Runtime: 2h 50m.
- Maalik**: Thriller, Action. Release: July 27, 2024, Runtime: 2h 12m.
- Bhootnii**: Comedy, Romance, Horror. Release: July 28, 2024, Runtime: 2h 28m.

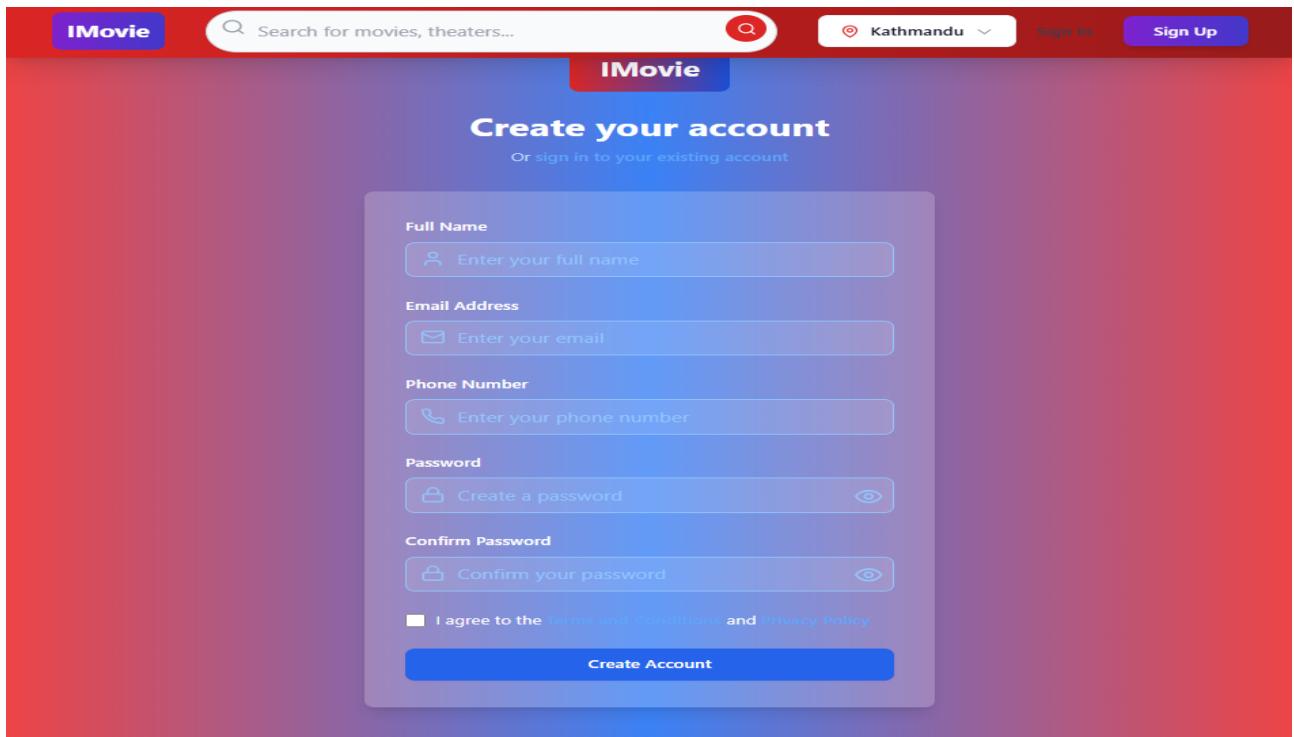
This screenshot shows a detailed movie page for 'Saiyaara'.
Movie Poster: The poster features a man and a woman in a close embrace, with text indicating it's a Yash Raj Films production and the title 'SAIYAARA' in large letters.
Title and Rating: The title 'Saiyaara' is prominently displayed in a large font. Below it are the rating '4.5/5', age rating 'U/A', and language 'Hindi'.
Release Info: The movie was released on March 8, 2025, with a runtime of 2h 45m. It falls under the genres of Romance, Drama, and love.
Action Buttons: Below the rating are buttons for 'Login to Book' (red) and 'Watch Trailer' (blue).
Navigation: At the bottom left, there are tabs for 'About' (which is currently selected), 'Cast & Crew', and 'Reviews'.
Description: A brief description states: 'A divine cinematic journey exploring the legend of Lord Shiva as "Mrithyunjay" – the conqueror of death. This epic drama blends mythology, philosophy, and stunning visuals to depict timeless teachings of life, death, and immortality.'
Technical Details: At the bottom, there is a list of technical details: Director: Om Raut, Producer: T-Series, Music: Ajay-Atul, Duration: 2h 45m, Genre: Romance, Drama, love, and Language: Hindi.

Home page



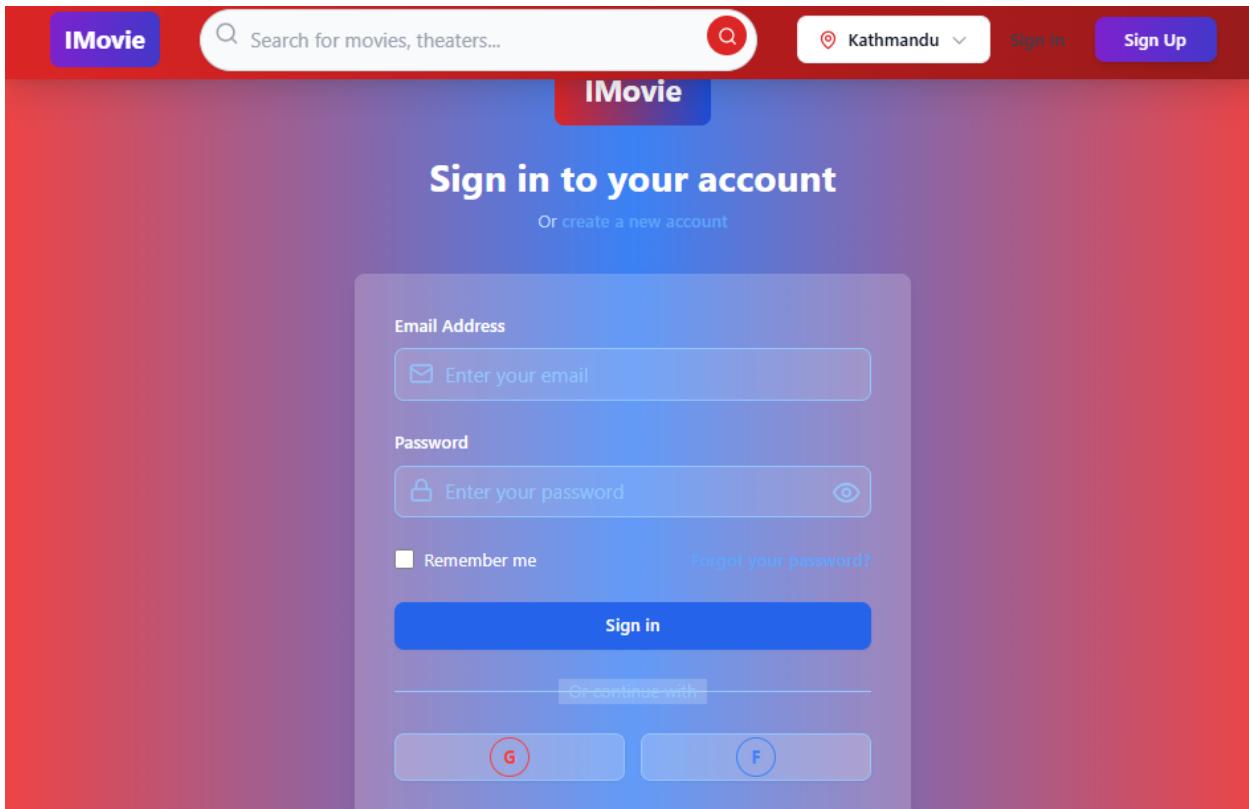
The screenshot shows the IMovie website homepage. At the top, there is a navigation bar with the IMovie logo, a search bar containing "Search for movies, theaters...", a location dropdown set to "Kathmandu", a "Sign In" button, and a "Sign Up" button. The main header features a large, bold text: "BOOK YOUR MOVIE TICKETS FOR ENTERTAINMENT". Below it, a subtext reads: "Safe, secure, reliable ticketing. Your ticket to live entertainment!". There are two dropdown menus labeled "Filters: All" and "All". The "Now Showing" section displays four movie cards: "Animal" (Comedy, Drama), "Avatar" (Action, Thriller), "Rawayan" (Action, Romance), and "Vettaiyan" (Drama, Family). Each card includes a "Book Now" button.

Sign up page



The screenshot shows the IMovie sign-up page. The top navigation bar is identical to the home page, featuring the IMovie logo, search bar, location dropdown, "Sign In" button, and "Sign Up" button. The main title is "Create your account" with a sub-instruction "Or sign in to your existing account". Below this, there are five input fields: "Full Name" (placeholder: "Enter your full name"), "Email Address" (placeholder: "Enter your email"), "Phone Number" (placeholder: "Enter your phone number"), "Password" (placeholder: "Create a password" with a visibility icon), and "Confirm Password" (placeholder: "Confirm your password" with a visibility icon). At the bottom, there is a checkbox for accepting terms and conditions and privacy policy, followed by a "Create Account" button.

Sign in page



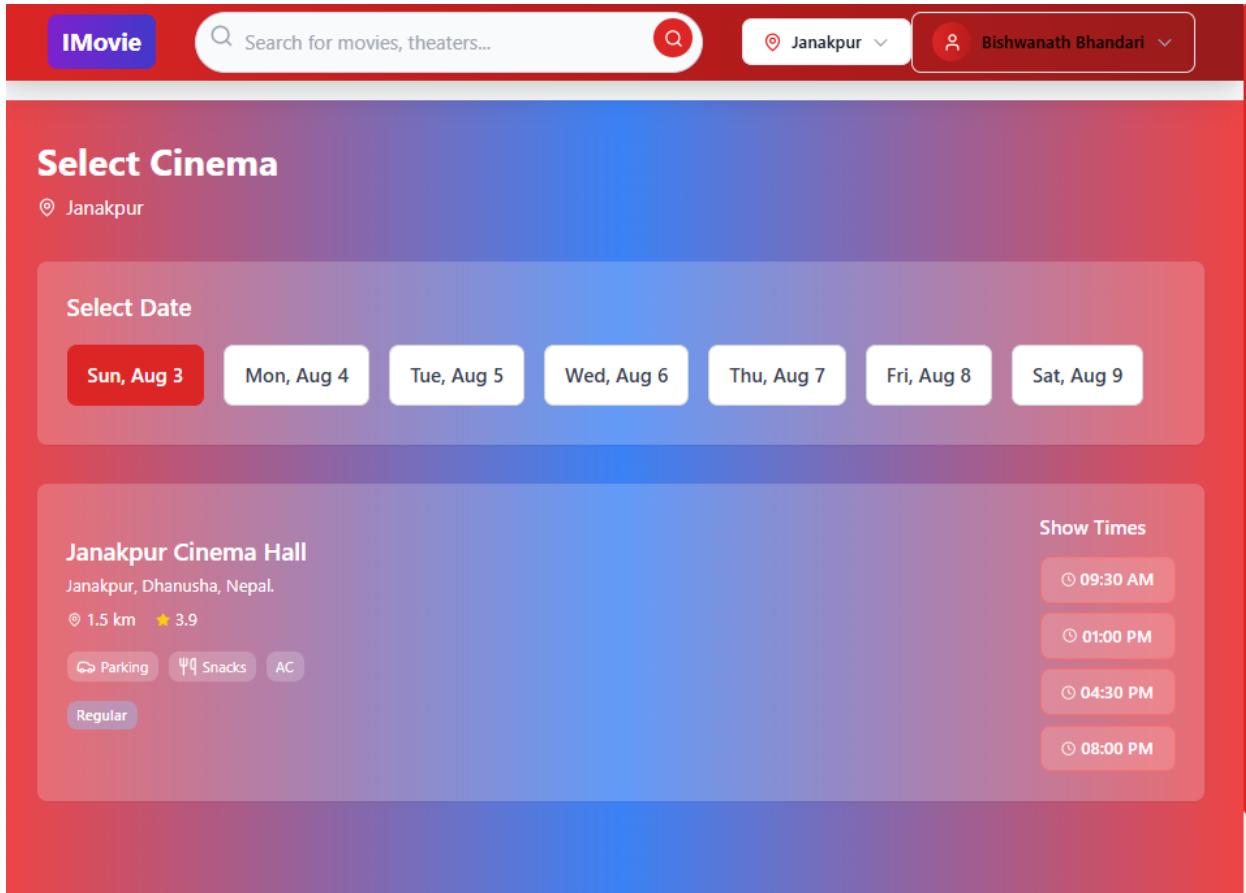
User Profile Page

The image compares two versions of the user profile page. Both versions have a red header bar with the IMovie logo, search bar, location dropdown (Kathmandu), and user profile (Bishwanath Bhandari).

Left Version (Yellow Gradient):
- Title: "Welcome back, Bishwanath Bhandari!"
- Subtitle: "Manage your bookings and account settings"
- Navigation tabs: "My Bookings" (selected), "Profile", "Preferences"
- Section: "Profile Information" with fields for Name (Bishwanath Bhandari), Email (user@domain.com), and Phone (5646456456).

Right Version (Orange Gradient):
- Title: "Welcome back, Bishwanath Bhandari!"
- Subtitle: "Manage your bookings and account settings"
- Navigation tabs: "My Bookings" (selected), "Profile", "Preferences"
- Section: "Upcoming Shows" showing a yellow box with "No upcoming bookings" and a "Browse Movies" button.
- Section: "Booking History" showing a booking for "Saiyaara" at INOX CMR Central, Visakhapatnam, on 15/07/2024 at 07:30 PM for Seats D5, D6. The status is "Confirmed" with a green badge. Buttons include "Download Ticket" and "Rate Movie".
- Booking ID: BK001

Bookings Page



The screenshot shows the 'Select Cinema' section of the Bookings Page. At the top, there is a search bar with placeholder text 'Search for movies, theaters...' and a location dropdown set to 'Janakpur'. Below the search bar, the location is also displayed as 'Janakpur'. A user profile for 'Bishwanath Bhandari' is shown on the right.

Select Date:

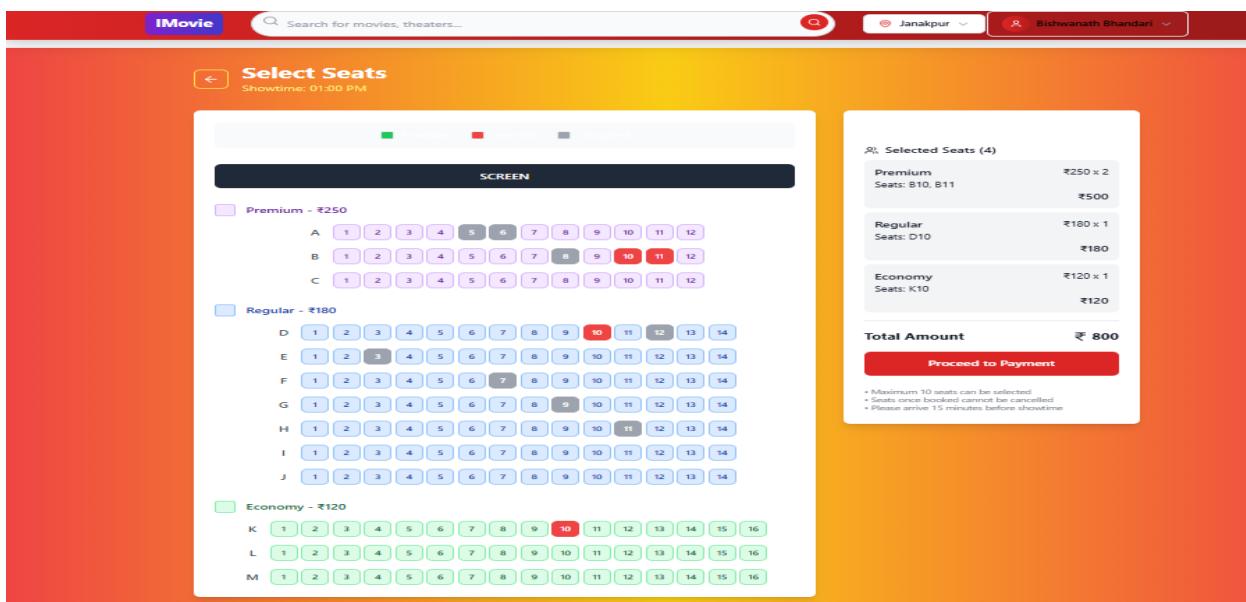
Sun, Aug 3 Mon, Aug 4 Tue, Aug 5 Wed, Aug 6 Thu, Aug 7 Fri, Aug 8 Sat, Aug 9

Janakpur Cinema Hall
Janakpur, Dhanusha, Nepal.
📍 1.5 km ★ 3.9
Parking Snacks AC
Regular

Show Times:

09:30 AM 01:00 PM 04:30 PM 08:00 PM

Seat Selection page



The screenshot shows the 'Select Seats' page. At the top, there is a search bar with placeholder text 'Search for movies, theaters...' and a location dropdown set to 'Janakpur'. Below the search bar, the location is also displayed as 'Janakpur'. A user profile for 'Bishwanath Bhandari' is shown on the right.

Select Seats
Showtime: 01:00 PM

SCREEN

Premium - ₹250

A	1	2	3	4	5	6	7	8	9	10	11	12
B	1	2	3	4	5	6	7	8	9	10	11	12
C	1	2	3	4	5	6	7	8	9	10	11	12

Regular - ₹180

D	1	2	3	4	5	6	7	8	9	10	11	12	13	14
E	1	2	3	4	5	6	7	8	9	10	11	12	13	14
F	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	1	2	3	4	5	6	7	8	9	10	11	12	13	14
H	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I	1	2	3	4	5	6	7	8	9	10	11	12	13	14
J	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Economy - ₹120

K	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Selected Seats (4)

Premium Seats: B10, B11	₹250 x 2	₹500
Regular Seats: D10	₹180 x 1	₹180
Economy Seats: K10	₹120 x 1	₹120

Total Amount ₹ 800

Proceed to Payment

* Maximum 10 seats can be selected
* Seats once booked cannot be canceled
* Seats are non-refundable & non-exchangeable

Ticket Payment Success page

The image shows two screenshots of a movie ticket booking website. The left screenshot is the 'Payment' page where a user selects a payment method (Credit/Debit Card, UPI, or Digital Wallet) and enters a wallet PIN. The right screenshot is the 'Booking Confirmed!' page, which displays the booking details, payment summary, and important information.

Payment Page Details:

- Booking Details:
 - Movie: Movie Title
 - Cinema: Cinema Name
 - Date & Time: 03/08/2023 at 01:00 PM
 - Seat: B10, B11, D10, K10
- Order Summary:
 - Ticket Price (4 Seats): ₹800
 - Convenience Fee: ₹16
 - Taxes (GST 18%): ₹147
 - Total Amount: ₹963
- Payment Method:
 - Credit/Debit Card: Visa, MasterCard, RuPay
 - UPI: PhonePe, GPay, Paytm, BHIM
 - Digital Wallet: Paytm, Amazon Pay, MobiKwik
- Payment Details:
 - Wallet PIN: [REDACTED]
 - Your payment information is encrypted and secure.
- Buttons: Pay ₹ 963

Booking Confirmed! Page Details:

- Movie Title: Cinema Name - Screen 1
- Booking ID: 3uatzo071
- Booking Details:
 - Date & Time: 03/08/2023 at 01:00 PM
 - Cinema: Cinema Name
 - City, Location: City, Location
 - Seat: B10, B11, D10, K10
- Payment Summary:

Tickets (4)	₹800	
Convenience Fee	₹16	
Taxes	₹147	
Total Paid	₹ 963	
- Payment Details:
 - Payment ID: 541999102001
 - Method: Wallet
 - Status: Paid
- Buttons: Share Booking, Download Ticket, View All Bookings
- Important Information:
 - Please arrive at the cinema at least 15 minutes before the showtime.
 - Carry a valid ID proof along with your ticket.
 - Outside food and beverages are not allowed in the cinema.
 - Tickets once booked cannot be cancelled or refunded.
 - Contact customer support for any assistance: 1800-123-4567.
- Buttons: Book More Movies

Admin Dashboard Page

The Admin Dashboard page provides a high-level overview of the system's performance and recent activity.

Admin Dashboard Overview:

- Overview: Overview, Movies, Theaters, Users
- Key Metrics:
 - Total Movies: 2
 - Total Bookings: 1247
 - Total Revenue: ₹500,000
 - Active Users: 10

Recent Activity:

- New movie "Paradha" added to the system 2 hours ago
- 125 tickets booked for "Oh Bhama Ayyo Rama" 4 hours ago
- New theater "PVR Guntur" added 1 day ago

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

Admin Dashboard

[Overview](#)
[Movies](#)
[Theaters](#)
[Users](#)

Movie Management

[+ Add Movie](#)

MOVIE	GENRE	LANGUAGE	RATING	STATUS	ACTIONS
saiyara 2h 25m	Comedy, Drama	hindi	4.2/5	active	
chhaava 2h 15m	Action, Thriller	hindi	3.8/5	active	

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

The image shows a feedback form titled "Share Your Feedback" centered over a blurred background of a website. The form has a pink header and a yellow footer. It includes fields for Full Name, Email Address, Phone Number, Overall Experience rating (5 stars), Feedback Category, Subject, and Your Message. A "Cancel" button and a "Submit Feedback" button are at the bottom. A note at the bottom of the form states: "Your feedback is important to us. We typically respond within 24-48 hours. For urgent issues, please call our customer care at 1800-123-4567." The background website features a red header with "Sign in to your account" and various navigation links.

Share Your Feedback

Full Name *

Email Address *

Phone Number

Overall Experience *

Feedback Category *

Subject *

Your Message *

Please share your detailed feedback, suggestions, or concerns...

Cancel  Submit Feedback

Your feedback is important to us. We typically respond within 24-48 hours. For urgent issues, please call our customer care at 1800-123-4567.

Message Privacy Settings
View Feedback
Contact Us
Phone: 123-456-7890
Email: info@customer.com
Address: 123 Market St, Anytown, CA 95010

Conclusion

I-Movie3 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-Movie3 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-Movie3 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-Movie3 sets a new standard for modern, hassle-free entertainment solutions, making every movie experience truly unforgettable.