

# I-Movies : Movie Ticket Booking System

## Project Description:

Users can effortlessly browse movies, select showtime, and choose seats from the comfort of their homes. The app features a robust client-server architecture with Express.js and MongoDB, ensuring efficient data storage and retrieval. With features like user management, 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.

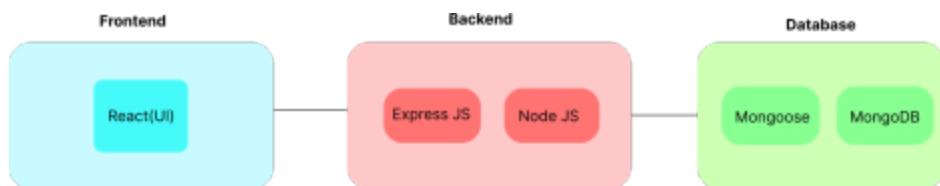
## Scenario

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.



## 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 the backend

easily by using RESTful Api.

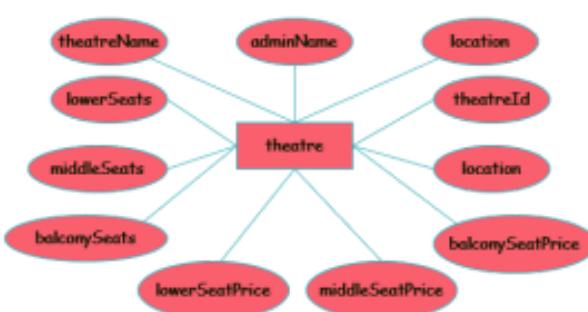
On the backend side, we employ Express.js frameworks to handle the server-side logic and communication.

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](https://nodejs.org) and Git from [git-scm.com](https://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.jsThis creates:  
• models/ for database schemas  
• 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 Backend Dependencies

Run these commands to install required packages:

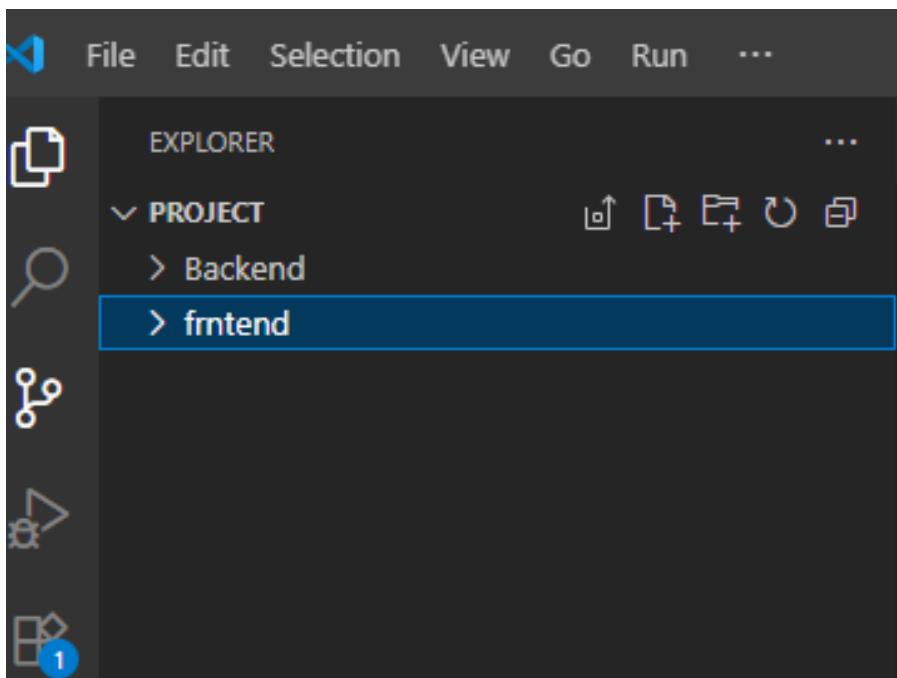
```
bash  
cd Backend
```

```
npm install express mongoose cors dotenv
```

### ■ Start Development Servers

Open two terminal windows and run:

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



## 2. Backend Development

- 
- **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/');
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,
25   }
26 });

```

- 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, there's a configuration panel for a POST request to `http://localhost:5000/api/v1/auth/login`. The 'Body' tab is selected, showing a JSON payload:

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

On the right, the response details are shown: Status: 200 OK, Size: 381 Bytes, Time: 583 ms. The 'Response' tab is selected, displaying the JSON response:1 {
2 "status": "success",
3 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
4 .eyJpZCI6IjY4ODIzOGE0ZTF1NTl1ZmU3MzNiNTI3MiIsImhlhd
5 CI6MTc1MzQ1ODM2MCwiZXhwIjoxNzU2MDUwMzYwfQ
6 .0Du175gbdBHQ-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.

The screenshot shows a terminal window with the following content:

```

> OUTLINE
> TIMELINE
> NPM SCRIPTS

● 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

```

Below the terminal, there is a code editor window showing the contents of index.js:

```

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();

```

The code editor interface includes tabs for FOLDERS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS.

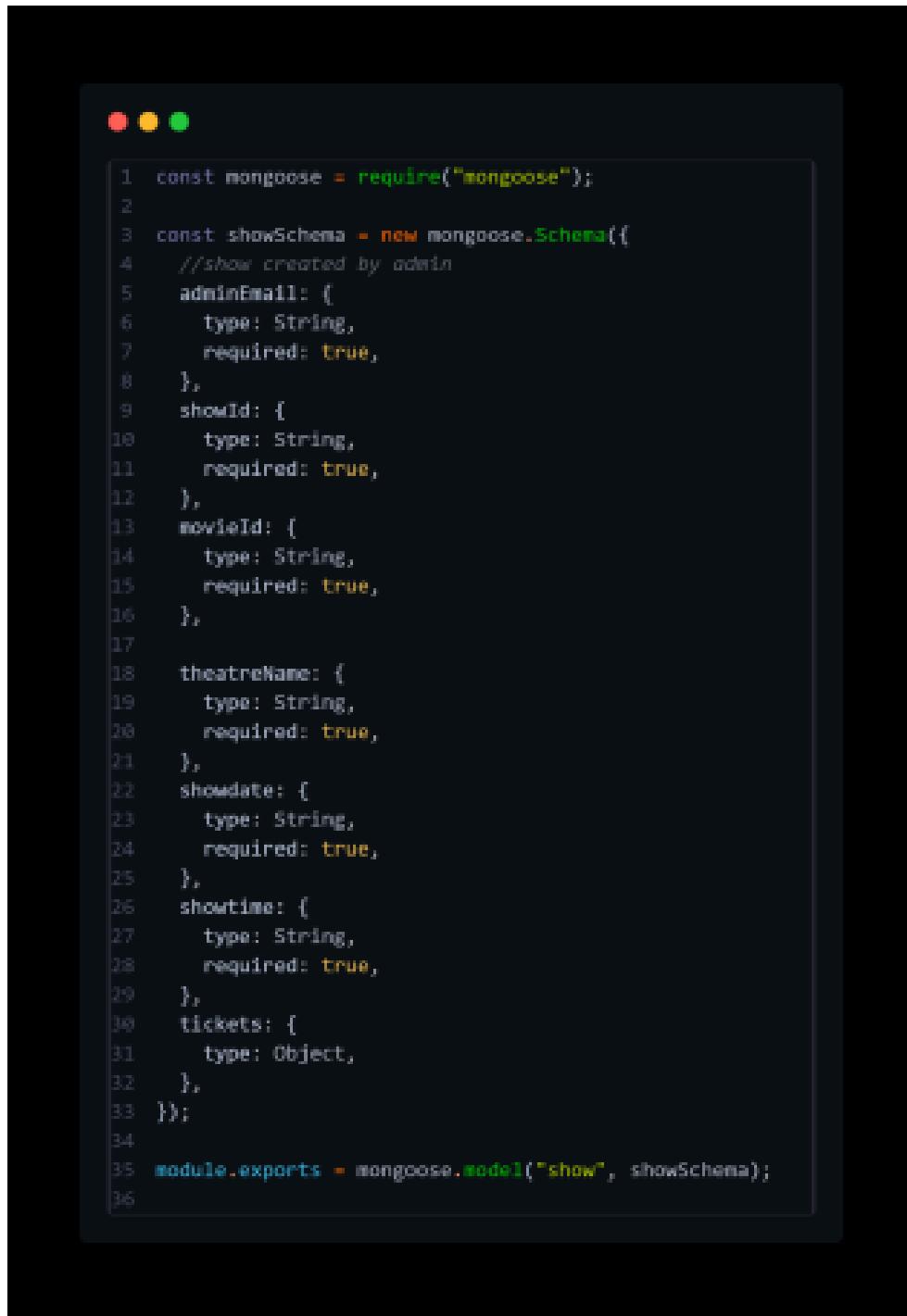
### Theater Schema:

- 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   },
48 });
49
50 module.exports = mongoose.model("Theatre", theatreSchema);
51
```

### 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 Favorite schema records user preferences by storing user email and movie ID pairs for favorite movies.
- It enables users to bookmark and 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 Booking schema captures details of user bookings including booking ID, 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,
    ],
  });
};

events.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

```
frontend > src > components > Header.jsx > Header
  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   'Visakhapatnam (Vizag)',
  7   'Vijayawada',
  8   'Guntur',
  9   'Nellore',
 10   'Tirupati',
 11   'Kurnool',
 12   'Rajahmundry',
 13   'Eluru',
 14   'Anantapur',
 15   'Kadapa',
 16   'Chittoor',
 17   'Srikakulam',
 18   'Narasaraopet',
 19   'Mangalagiri',
 20   'Kakinada',
 21   'Tadepalligudem',
 22   'Bhimavaram'
 23 ];
 24
```

- **Footer.jsx**

Copyright information

Contact links

Social media icons

```

frntend > src > components > Footer.jsx > Footer
1 import React, { useState } from 'react';
2 import { Link } from 'react-router-dom';
3 import { Facebook, Twitter, Instagram, Youtube, Mail, Phone, MapPin, Star } from 'lucide-react';
4 import FeedbackForm from './FeedbackForm';
5
6 const Footer = () => {
7   const [showFeedback, setShowFeedback] = useState(false);
8
9   return (
10     <>
11       <footer className="bg-gradient-to-r from-gray-900 via-gray-800 to-gray-900 text-white">
12         <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-12">
13           <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8">
14             {/* Company Info */}
15             <div className="space-y-4">
16               <div className="flex items-center space-x-2">
17                 <div className="bg-gradient-to-r from-red-600 to-red-700 text-white px-4 py-3 rounded-full w-10 h-10 flex items-center justify-center">
18                   IMovie
19                 </div>
20               </div>
21               <p className="text-gray-300 text-sm leading-relaxed">
22                 Your premier destination for movie ticket booking in Andhra Pradesh.
23                 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 > ...
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) => {

```

## Step 2: Implement Styling

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

```
frntend > src > # index.css > ...
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 */
```

## Step 3: Set Up Routing

### Define routes in App.jsx

- Home (/)

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

```
fmtend > 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 Login from './pages/Login';
14 import Register from './pages/Register';

15
16 function App() {
17   const [user, setUser] = useState(null);
18   const [selectedLocation, setSelectedLocation] = useState('Visakhapatnam (Vizag)');
19
20   useEffect(() => {
21     // Check for logged in user
22     const userData = localStorage.getItem('userData');
23     if (userData) {
24       setUser(JSON.parse(userData));
25     }
26   }, []);
27
28   return (
29     <Router>
30       <Routes>
31         <Route path="/" element={<Home />} />
32         <Route path="/movies" element={<MovieDetails />} />
33         <Route path="/cinemas/:movield" element={<CinemaSelection />} />
34         <Route path="/book/:showtimeld" element={<SeatSelection />} />
35         <Route path="/payment" element={<Payment />} />
36         <Route path="/confirmation" element={<BookingConfirmation />} />
37         <Route path="/login" element={<Login />} />
38         <Route path="/register" element={<Register />} />
39         <Route path="/dashboard" element={<UserDashboard />} />
40         <Route path="/admin" element={<AdminDashboard />} />
41       </Routes>
42     </Router>
43   );
44 }

45 export default App;
```

### 3. Implement Frontend Logic

#### Step 1: 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 carousel for promotional movies

```
fmtend > src > pages > Home.jsx > ongoingMovies
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: 'Oh Bhama Ayyo Rama',
9      genre: 'Comedy, Drama',
10     language: 'Telugu',
11     rating: 4.2,
12     releaseDate: 'July 11, 2024',
13     poster: 'https://assetscdn1.paytm.com/images/cinema/P%20oh-Bhama-ayyo-Rama%20(1)-f4b18f80-5b1a-11f0-b',
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     {
21       id: 2,
22       title: 'The 100',
23       genre: 'Action, Thriller',
24       language: 'English',
25       rating: 3.8,
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

2. Movie Details:

- Fetch movie data by ID
- Display showtime availability

```

frontend > src > pages > MovieDetails.jsx > ...
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
5  // Trailer Modal Component
6  const TrailerModal = ({ isOpen, onClose, trailerUrl, movieTitle }) => {
7    if (!isOpen) return null;
8
9    return (
10      <div className="fixed inset-0 bg-black bg-opacity-75 flex items-center justify-center z-50 p-4">
11        <div className="bg-white rounded-lg max-w-4xl w-full max-h-[90vh] overflow-hidden">
12          <div className="flex justify-between items-center p-4 border-b">
13            <h3 className="text-lg font-semibold">{movieTitle} - Official Trailer</h3>
14            <button
15              onClick={onClose}
16              className="p-2 hover:bg-gray-100 rounded-full transition-colors"
17            >
18              <X className="h-5 w-5" />
19            </button>
20          </div>
21          <div className="aspect-video">
22            <iframe
23              width="100%"
24              height="100%">

```

## Seat Selection:

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

```

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 = {

```

### Step 3: State Management

1. Use Context API for:

- User authentication
- Booking process
- UI preferences

```
frtend > src > pages > UserDashboard.jsx > useEffect() callback > mockBookings > poster
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: 'Oh Bhama Ayyo Rama',
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://images.filmibeat.com/img/popcorn/movie_posters/ohbhamaayyorama-20250324125
22        },
23        {
24          id: 'BK002'.

```

2 . Authentication Flow

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

3 . Payment Integration

- Setup payment form
- Connect to payment gateway

- Handle success/failure

```
frntend > src > pages > ⚘ Payment.jsx > ...
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   });
27
28   const handleBooking = () => {
29     const bookingData = {
30       cardNumber: paymentForm.cardNumber,
31       expiryDate: paymentForm.expiryDate,
32       cvv: paymentForm.cvv,
33       cardholderName: paymentForm.cardholderName,
34       upiId: paymentForm.upiId,
35       walletPin: paymentForm.walletPin
36     };
37
38     // Perform booking logic here
39
40     // Set booking data to local storage
41     localStorage.setItem('pendingBooking', JSON.stringify(bookingData));
42
43     // Navigate back to home page
44     navigate('/');
45   };
46
47   return (
48     <div>
49       <h1>Payment</h1>
50       <form>
51         <input type="text" value={paymentForm.cardNumber} />
52         <input type="text" value={paymentForm.expiryDate} />
53         <input type="text" value={paymentForm.cvv} />
54         <input type="text" value={paymentForm.cardholderName} />
55         <input type="text" value={paymentForm.upiId} />
56         <input type="text" value={paymentForm.walletPin} />
57         <button type="button" onClick={handleBooking}>Book</button>
58       </form>
59     </div>
60   );
61 }
```

## Step 4: Run the Application

bash

npm run 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

The screenshot displays the movie search interface and two main sections: "Now Showing" and "Coming Soon".

**Now Showing:**

- Oh Rama Ayyo Rama**: Comedy, Drama. Release Date: 17 July 11, 2024. Duration: 02h 30m. Book Now.
- The 100**: Action, Thriller. Release Date: 17 July 11, 2024. Duration: 02h 10m. Book Now.
- Virgin Boys**: Comedy, Romance. Release Date: 17 July 11, 2024. Duration: 02h 10m. Book Now.
- Dheenga Ayushman Bhava**: Drama, Family. Release Date: 17 July 11, 2024. Duration: 02h 30m. Book Now.
- Mrithyunjay**: Action, Drama. Release Date: 17 July 14, 2024. Duration: 02h 30m. Book Now.

**Coming Soon:**

- Meghalu Cheppina Prema Katha**: Romance, Drama. Release Date: 17 July 24, 2024. Duration: 02h 20m. Coming Soon.
- Hari Hara Veera Mallu**: Historical, Action. Release Date: 17 July 24, 2024. Duration: 02h 40m. Coming Soon.
- Mahavatara Narasimha**: Mythology, Action. Release Date: 17 July 24, 2024. Duration: 02h 30m. Coming Soon.
- Selfish**: Thriller, Drama. Release Date: 17 July 27, 2024. Duration: 02h 10m. Coming Soon.
- Paradha**: Action, Romance. Release Date: 17 July 28, 2024. Duration: 02h 20m. Coming Soon.

The image shows a movie page for 'Oh Bhama Ayyo Rama'. At the top is a large banner featuring a close-up of a man and a woman smiling. Below the banner is a smaller thumbnail image of the movie's cast. The title 'Oh Bhama Ayyo Rama' is displayed prominently in white text. Below the title are ratings (4.2/5), age rating (U/A), and genre (Telugu). Release date (July 11, 2024), runtime (2h 25m), and genres (Comedy, Drama) are also listed. Buttons for 'Book Tickets' and 'Watch Trailer' are present, along with social sharing icons.

**About** Cast & Crew Reviews

**Synopsis**

A hilarious comedy-drama about family relationships and misunderstandings. The story revolves around a middle-class family and their amusing adventures filled with love, laughter, and life lessons.

**Movie Details**

Director: Srikanth Vissa

## Home page

The image shows the homepage of IMovie. At the top, there is a navigation bar with the IMovie logo, a search bar, location dropdown (Visakhapatnam), and user profile (John Doe). The main header reads 'Welcome to IMovie' and 'Book your favorite movies in Visakhapatnam (Vizag)'. Below the header are buttons for 'Latest Movies', 'Best Showtimes', and 'Premium Experience'. The main content area features a section titled 'Now Showing' with five movie posters. Each poster includes the movie title and its rating. A 'View All >' link is located at the bottom right of this section.

IMovie

Search for movies, theaters...

Visakhapatnam John Doe

Welcome to IMovie

Book your favorite movies in Visakhapatnam (Vizag)

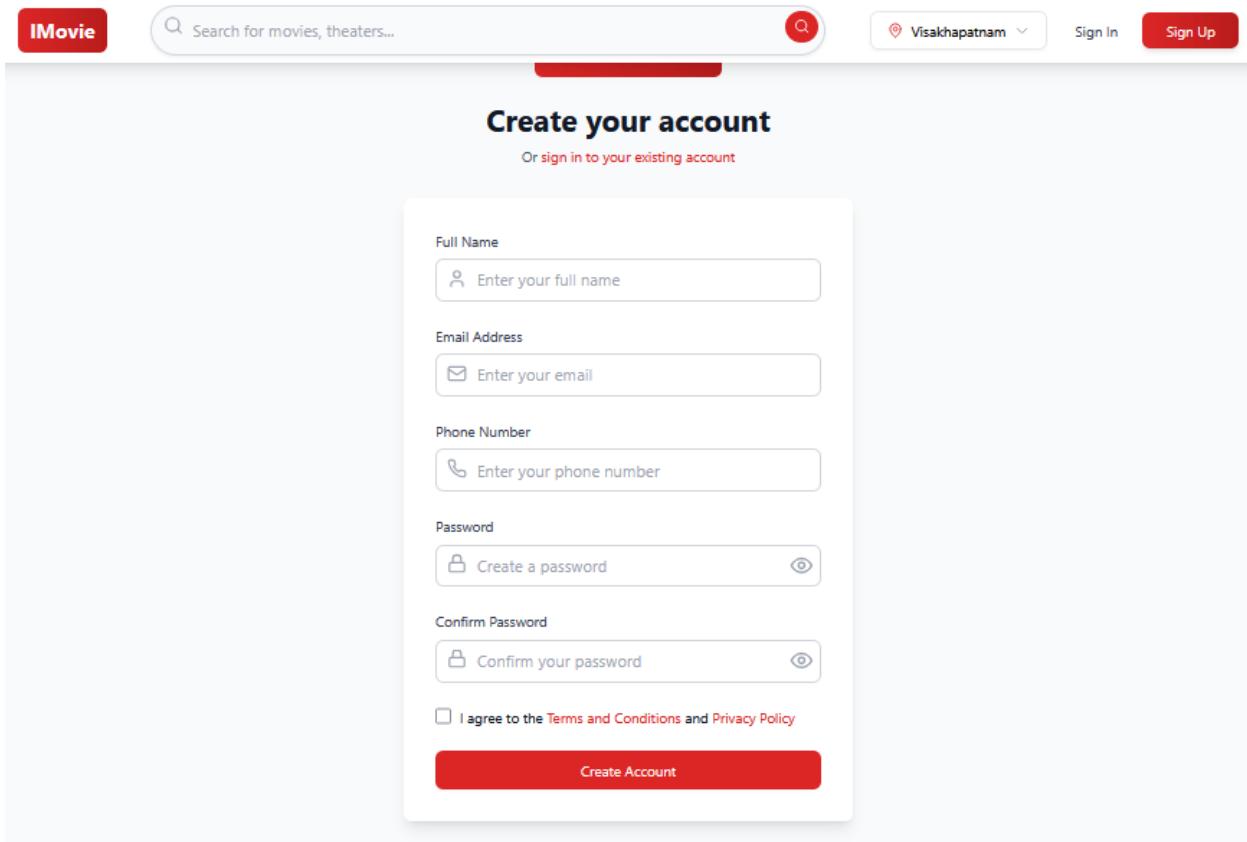
Latest Movies Best Showtimes Premium Experience

Now Showing

View All >

4.2 THE ICO 3.8 VIRGIN BOAS 3.5 4.1

# Sign up page



The screenshot shows the IMovie sign-up page. At the top, there is a red header bar with the IMovie logo on the left, a search bar in the center containing the placeholder "Search for movies, theaters...", and a red search icon on the right. To the right of the search bar are location settings ("Visakhapatnam") and links for "Sign In" and "Sign Up". Below the header is a large white form area with a red header titled "Create your account". Underneath it, a link says "Or sign in to your existing account". The form contains six 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 an eye icon), "Confirm Password" (placeholder "Confirm your password" with an eye icon), and a checkbox labeled "I agree to the Terms and Conditions and Privacy Policy". At the bottom of the form is a large red "Create Account" button.

IMovie

Search for movies, theaters...

Visakhapatnam

Sign In

Sign Up

## Create your account

Or sign in to your existing account

Full Name

Enter your full name

Email Address

Enter your email

Phone Number

Enter your phone number

Password

Create a password

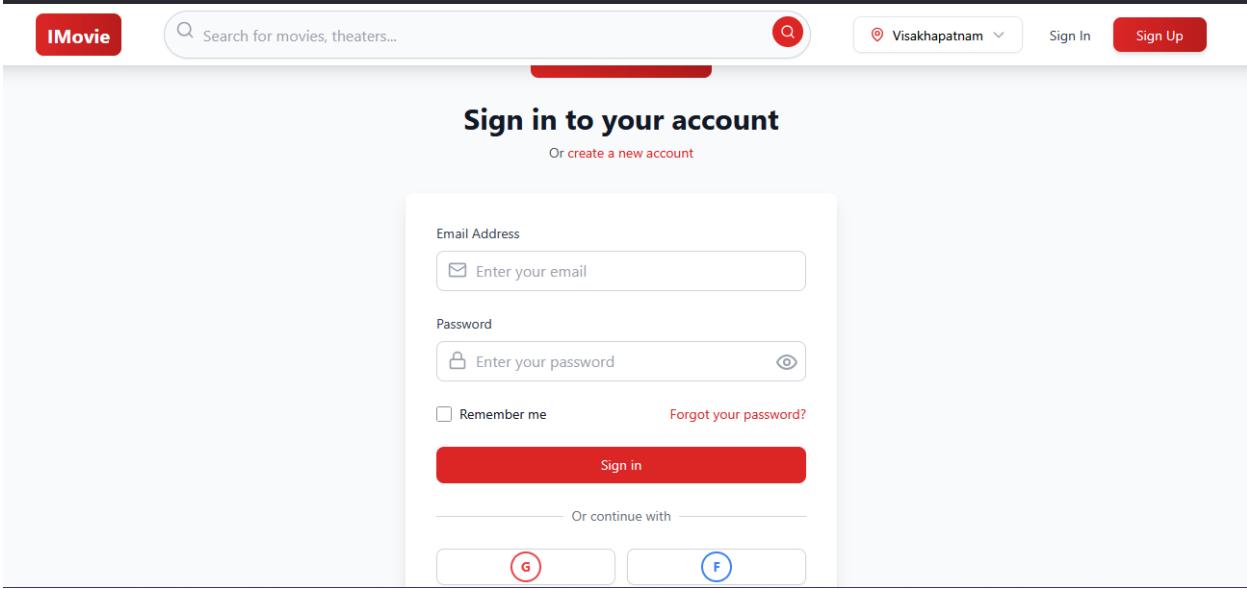
Confirm Password

Confirm your password

I agree to the [Terms and Conditions](#) and [Privacy Policy](#)

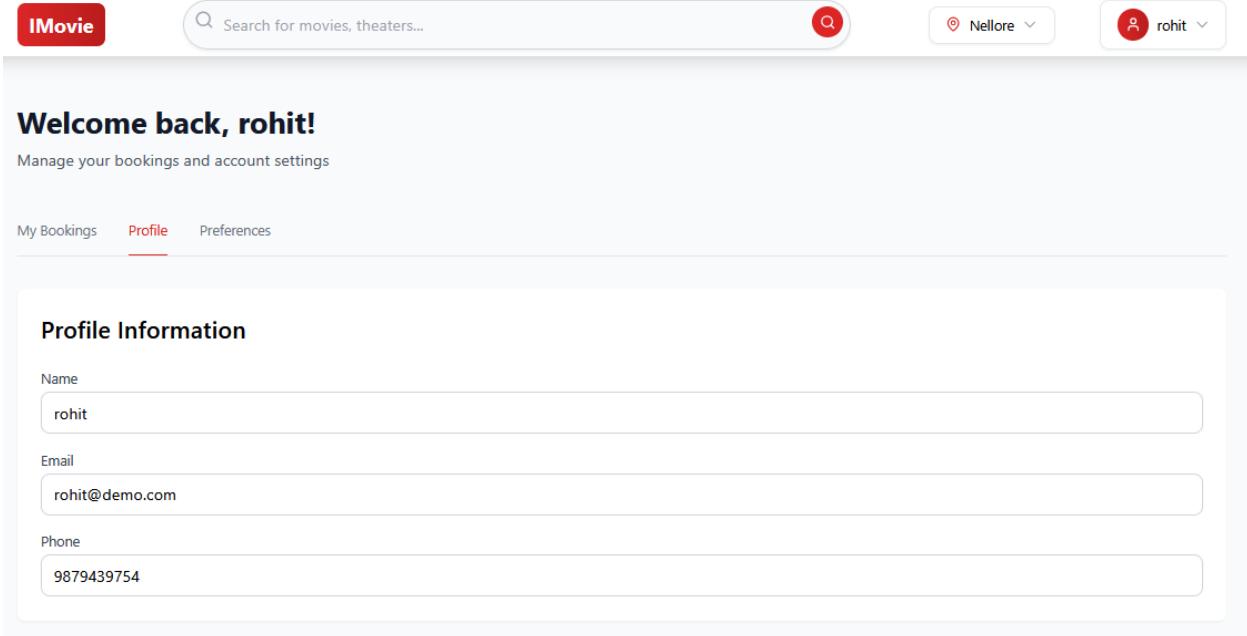
Create Account

## Sign in page



The screenshot shows the IMovie sign-in page. 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 "Visakhapatnam", and links for "Sign In" and "Sign Up". Below the navigation bar, the main title "Sign in to your account" is displayed in bold black text, followed by the sub-instruction "Or [create a new account](#)". The sign-in form consists of two input fields: "Email Address" and "Password", both with placeholder text ("Enter your email" and "Enter your password"). There is also a "Remember me" checkbox and a "Forgot your password?" link. A large red "Sign in" button is centered below the password field. Below the sign-in form, there is a section titled "Or continue with" featuring two social media icons: a red "G" for Google and a blue "F" for Facebook.

## User Profile Page



The screenshot shows the IMovie user profile page for a user named "rohit". At the top, the IMovie logo, search bar, location dropdown set to "Nellore", and user profile icon are visible. The main header says "Welcome back, rohit!" and provides a link to "Manage your bookings and account settings". Below the header, there are three tabs: "My Bookings", "Profile" (which is currently selected), and "Preferences". The "Profile Information" section contains three input fields: "Name" (with "rohit" entered), "Email" (with "rohit@demo.com" entered), and "Phone" (with "9879439754" entered).

**IMovie**

Search for movies, theaters...

Nellore

rohit

## Welcome back, rohit!

Manage your bookings and account settings

My Bookings   Profile   Preferences

### Upcoming Shows

No upcoming bookings

Ready to watch something amazing?

Browse Movies

### Booking History

Movie Title	Cinema Hall	Date	Time	Seat Details	Price	Status
Oh Bhama Ayyo Rama	INOX CMR Central, Visakhapatnam	15/7/2024	07:30 PM	Seats: D5, D6	₹500	Confirmed



Download Ticket   Rate Movie   Booking ID: BK001

# Bookings Page

**IMovie**

Visakhapatnam rohit

## Select Cinema

Visakhapatnam (Vizag)

### Select Date

Sun, Jul 27 Mon, Jul 28 Tue, Jul 29 Wed, Jul 30 Thu, Jul 31 Fri, Aug 1 Sat, Aug 2

**INOX CMR Central**  
CMR Central Mall, Maddilapalem, Visakhapatnam  
 2.5 km 4.3

**Show Times**

09:30 AM  
 01:15 PM  
 05:00 PM  
 08:45 PM

**PVR Vizag**  
Jagadamba Junction, Visakhapatnam  
 3.2 km 4.1

**Show Times**

10:00 AM  
 02:30 PM  
 06:15 PM  
 09:30 PM

## Seat Selection page

The screenshot shows the IMovie Seat Selection page. At the top, there is a search bar with placeholder text "Search for movies, theaters..." and a location dropdown set to "Visakhapatnam". On the right, there is a user profile for "rohit".

The main area is titled "Select Seats" and shows a seating chart for a movie screen. The chart includes a legend: green for Available, red for Selected, and grey for Occupied. The seating is arranged in rows A through J and columns 1 through 14. Seats C9 and C10 are selected (red).

**Booking Summary:**

- Selected Seats (2):** Premium Seats: C9, C10. Price: ₹500.
- Total Amount:** ₹ 500. **Proceed to Payment** button.

Instructions at the bottom of the summary:

- Maximum 10 seats can be selected.
- Seats once booked cannot be cancelled.
- Please arrive 15 minutes before showtime.

## Ticket Payment Success page

The screenshot shows the IMovie Ticket Payment Success page. At the top, there is a search bar with placeholder text "Search for movies, theaters..." and a location dropdown set to "Visakhapatnam". On the right, there is a user profile for "rohit".

The main area is titled "Payment" and shows the "Complete your booking" step. It includes a "Select Payment Method" section with options for Credit/Debit Card, UPI (PhonePe, GPay, Paytm, BHIM), and Digital Wallet (Paytm, Amazon Pay, MobiKwik). The UPI option is highlighted with a red border.

**Order Summary:**

Booking Details	
Movie: Movie Title	₹500
Cinema: Cinema Name	
Date & Time: 05:00 PM	
Seats: C9, C10	
Ticket Price (2 tickets)	₹500
Convenience Fee	₹10
Taxes (GST 18%)	₹92
<b>Total Amount</b>	<b>₹ 602</b>

Instructions at the bottom of the summary:

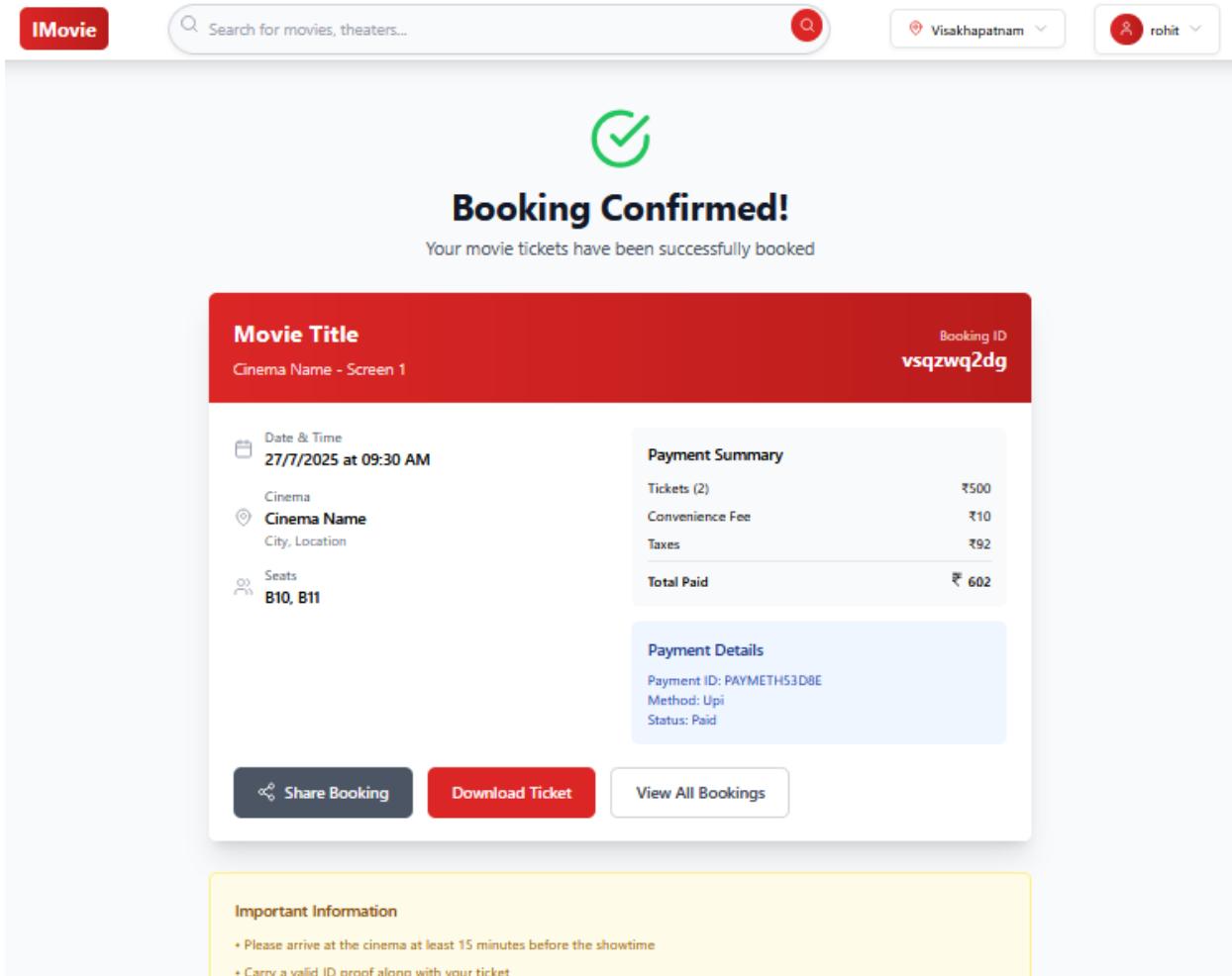
- Tickets once booked cannot be cancelled.
- Please arrive 15 minutes before showtime.
- Carry a valid ID proof.

**Payment Details:**

UPI ID: 89743907@ybl

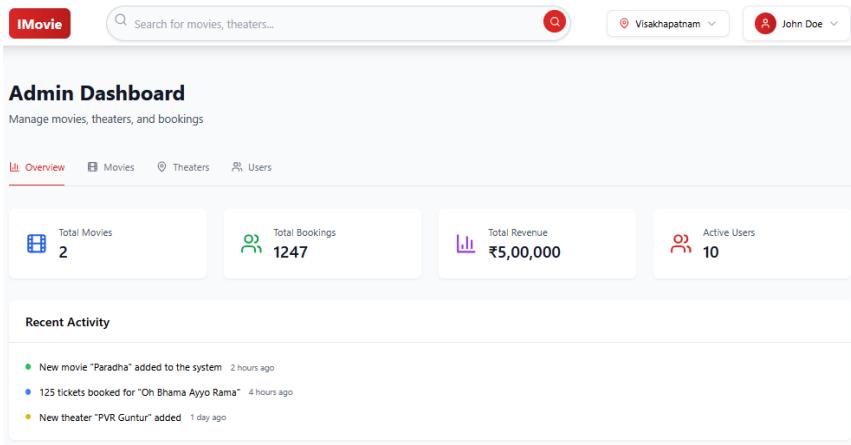
A note: Your payment information is encrypted and secure.

**Pay ₹ 602** button.



The screenshot shows a booking confirmation page for a movie ticket. At the top, there is a red header bar with the IMovie logo, a search bar, location dropdown (Visakhapatnam), and user profile (rohit). Below the header, a green checkmark icon and the text "Booking Confirmed!" are displayed. A sub-header says "Your movie tickets have been successfully booked". The main content area has a red header "Movie Title" and a sub-header "Cinema Name - Screen 1". To the right, the "Booking ID" is shown as "vsqzwq2dg". The booking details include: Date & Time (27/7/2025 at 09:30 AM), Cinema (Cinema Name), Seats (B10, B11). On the right, a "Payment Summary" table shows: Tickets (2) ₹500, Convenience Fee ₹10, Taxes ₹92, and Total Paid ₹ 602. Below this is a "Payment Details" section with Payment ID: PAYMETH53D8E, Method: Upi, and Status: Paid. At the bottom are three buttons: "Share Booking" (dark grey), "Download Ticket" (red), and "View All Bookings" (light grey). A yellow box labeled "Important Information" contains two bullet points: "Please arrive at the cinema at least 15 minutes before the showtime" and "Carry a valid ID proof along with your ticket".

## Admin DashBoard Page



The screenshot shows the Admin Dashboard page. At the top, there is a red header bar with the IMovie logo, a search bar, location dropdown (Visakhapatnam), and user profile (John Doe). Below the header, the title "Admin Dashboard" is displayed, followed by the subtitle "Manage movies, theaters, and bookings". There is a navigation bar with tabs: Overview (selected), Movies, Theaters, and Users. The main content area features four summary cards: "Total Movies" (2), "Total Bookings" (1247), "Total Revenue" (₹5,00,000), and "Active Users" (10). Below these cards is a "Recent Activity" section showing three items: "New movie 'Paradha' added to the system" (2 hours ago), "125 tickets booked for 'Oh Bhama Ayyo Rama'" (4 hours ago), and "New theater 'PVR Guntur' added" (1 day ago).

IMovie

Search for movies, theaters...

Visakhapatnam

John Doe

## Admin Dashboard

Manage movies, theaters, and bookings

Overview Movies Theaters Users

### Movie Management

+ Add Movie

MOVIE	GENRE	LANGUAGE	RATING	STATUS	ACTIONS
Oh Bhama Ayyo Rama 2h 25m	Comedy, Drama	Telugu	4.2/5	active	
The 100 2h 15m	Action, Thriller	Telugu	3.8/5	active	

IMovie

Search for movies, theaters...

Visakhapatnam

John Doe

### Add New Movie

Movie Name

Language

Type of Movie (Genre)

Poster

No file selected.

Banner

No file selected.

Actress

Director

Music Director

Producer

**IMovie**

Search for movies, theaters...

Visakhapatnam

John Doe

## Admin Dashboard

Manage movies, theaters, and bookings

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

## Feedback Page

### Share Your Feedback

Full Name \*

Email Address \*

Phone Number

Overall Experience \*

Feedback Category \*

General Feedback

Subject \*

Brief subject of your feedback

Your Message \*

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

Cancel

Submit Feedback

## **Conclusion**

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