

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

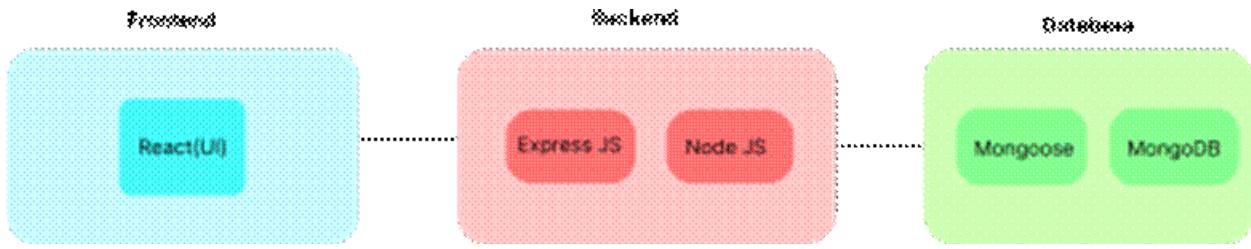
Imagine Sarah, a movie enthusiast, excitedly expecting the release of an anticipated film. With a busy schedule, she prefers the convenience of booking her movie tickets online. Logging into the iMovies app, Sarah navigates effortlessly through the user-friendly interface, browsing the list of usable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

Technical Architecture



In This Architecture Diagram:

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

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

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

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

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

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

smooth and immersive experience for all users.

ER-Diagram



PREREQUISITES

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

1. Node.js and npm

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

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

2. MongoDB

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

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

3. Express.js

A Node.js framework for backend API development.

- Install via npm:
bash
'npm install express'

4. React.js

JavaScript library for building dynamic UI's.

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

5. HTML, CSS, and JavaScript

Front-end developers need fundamental knowledge.

6. Database Connectivity

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

- **Guide:** <https://www.section.io/engineering-education/nodejs-mongoose-smongodb/>

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:

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



1. Project setup and configuration

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

■ Install Node.js and Git

Run these commands one by one in your terminal:

```
bash  
# Verify  
installations  
'node --version'  
npm --version  
git --version
```

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

■ Create Project Structure

Execute these commands to set up the basic folder structure:

```
bash  
mkdir imovie-app  
cd imovie-app
```

■ Set Up React Frontend

Create your frontend with this command:

```
bash  
npx create-react-app frontend  
This will generate:
```

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

■ Initialize Backend

Run these commands to set up your Node.js backend:

```
bash mkdir Backend cd Backend  
npm init -y
```

■ Create Backend Structure

While still in the Backend folder, run:

```
bash mkdir  
models routes  
touch server.js  
This creates:
```

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

■ Set Up Version Control

Navigate back to your project root and initialize Git:

```
bash cd ..  
git init  
touch  
.gitignore
```

Add these lines to your .gitignore file:

```
text  
node_modules/  
.env
```

■ Install Backend Dependencies

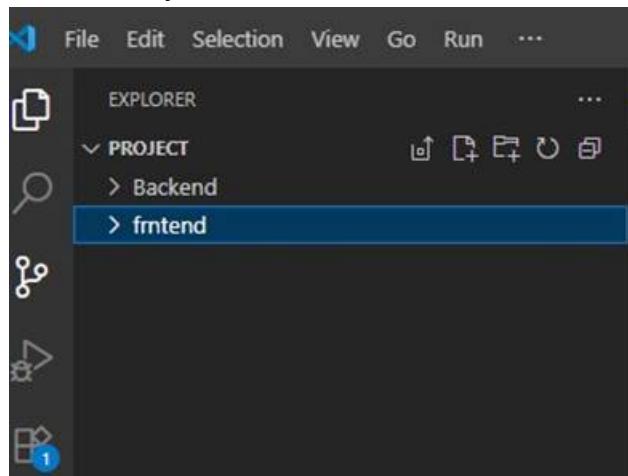
Run these commands to install required packages:

```
bash  
cd Backend  
npm install express mongoose cors
```

dotenv ■ Start Development Servers

Open two terminal windows and run: bash

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



2. Backend Development

-

1. **Initialize Express Server**
2. Navigate to your Backend folder and install required dependencies:

```
bash
```

```
cd Backend
```

```
npm install express cors body-parser mongoose jsonwebtoken dotenv bcryptjs
```

3. Create a new index.js file as your main server file:

```
bash
```

```
touch index.js
```

4. **Configure Environment Variables**

5. Create a .env file in your Backend folder:

```
bash
```

```
touch .env
```

6. Add these configurations to your .env file: env

```
PORT=5000
```

```
MONGODB_URI=mongodb://localhost:27017/imovie_db
```

```
JWT_SECRET=your_strong_secret_key_here
```

- **Basic Server Setup**

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

Authentication Setup

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

```
backend > models > JS User.js > ...
1  const mongoose = require('mongoose');
2
3  const usersSchema = new mongoose.Schema({
4    name: {
5      type: String,
6      required: true,
7      trim: true
8    },
9    email: {
10      type: String,
11      required: true,
12      unique: true,
13      trim: true
14    },
15    phone: {
16      type: String,
17      required: true
18    },
19    isAdmin: {
20      type: Boolean,
21      default: false
22    }
},
```

3. 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

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

Start Your Server

2. Run the backend server with:

bash

node server.js

■ Testing The Backend by using the thunder client:

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:5000/api/v1/auth/login
- Send** button is highlighted.
- Body** tab is selected, showing JSON content:

```
{  
  "email": "john@example.com",  
  "password": "test1234"  
}
```
- Status:** 200 OK
- Size:** 381 Bytes
- Time:** 583 ms
- Response** tab is selected, displaying the response body:

```
{  
  "status": "success",  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXCVJ9.  
  .eyJpZC16IJY40DIZoGE0ZTFINT1lZmU3MzN1NTI3MiIsImhd  
  CI6MTc1MzQ1ODM2MCwiZXhwIjoxNzU2MDUwMzYwFQ  
.0Du175gbfHq-pjhdNmMs-VAFs9_BG4AB_ip3IxryU",  
  "data": {  
    "user": {  
      "_id": "688238a4e1e59efe733b5272",  
      "name": "John Doe",  
      "email": "john@example.com",  
      "phone": "+9876543210",  
      "role": "user",  
      "createdAt": "2025-07-24T13:44:04.903Z",  
      "__v": 0  
    }  
  }  
}
```

3.Database Development

Creating Data Schemas

User Schema:

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

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

Theater Schema:

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

```
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   adminPhone: {
49     type: String,
50     required: true,
51   },
52   showList: [
53     {
54       movieTitle: {
55         type: String,
56         required: true,
57       },
58       screeningDate: {
59         type: Date,
60         required: true,
61       },
62       screeningTime: {
63         type: String,
64         required: true,
65       },
66       ticketPrice: {
67         type: Number,
68         required: true,
69       },
70       ticketCount: {
71         type: Number,
72         required: true,
73       },
74       theaterId: {
75         type: String,
76         required: true,
77       }
78     }
79   ],
80   showListLength: {
81     type: Number,
82     required: true,
83   }
84 });
85
86 module.exports = mongoose.model("Theatre", theatreSchema);
```

Show Schema:

8. Schema: showSchema

9. Model: 'Show'

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

11. It organizes and stores data related to movie screenings, enabling users to

browse and book show-times effectively.

```
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   theatreName: {
18     type: String,
19     required: true,
20   },
21   showdate: {
22     type: String,
23     required: true,
24   },
25   showtime: {
26     type: String,
27     required: true,
28   },
29   tickets: {
30     type: Object,
31   },
32 });
33
34 module.exports = mongoose.model("show", showSchema);
35
```

Movie Schema:

12. Schema: movieSchema

13. Model: 'Movie'

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

IDs.

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

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

Favorite Schema:

16. Schema: favoriteSchema

17. Model: 'Favorite'

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

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

```
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:

20. Schema: bookingModel

21. Model: 'Booking'

22. The Booking schema captures details of user bookings including booking ID, user email, show ID, and ticket data.

23. It facilitates the management and tracking of user reservations, ensuring a

smooth booking process and accurate record-keeping.

```
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:

24. Schema: adminSchema

25. Model: 'Admin'

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

27. It represents administrative users in the system, providing access to privileged

functionalities for managing the application.

```
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,
        ],
    });
};

module.exports = 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/iconsmaterial  
npm install -D tailwindcss Postcss autoprefixer  
npx tailwindcss init -p  
  1. axios: For API calls  
  2. react-router-dom: For routing  
  3. @mui/material: UI components
```

Step 3: Folder Structure

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

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

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

2. Design UI Components

Step 1: Create Core

Components • Header.jsx

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

```

fmtend > 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

```

fmtend > 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             {/*
15               Company Info */
16             <div className="space-y-4">
17               <div className="flex items-center space-x-2">
18                 <div className="bg-gradient-to-r from-red-600 to-red-700 text-white px-4 py-3 rounded" style={{ width: 100 }}>
19                   IMovie
20                 </div>
21               </div>
22             <p className="text-gray-300 text-sm leading-relaxed">
23               Your premier destination for movie ticket booking in Andhra Pradesh.
24               Experience the magic of cinema with the best theaters and latest movies.
25             </p>
26           </div>
27         </div>
28       </footer>
29     </>
30   );
31 }
32
33 export default Footer;

```

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) => {
```

Step2: Implement Styling

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

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

Step3: Set Up Routing

Define routes in App.jsx

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

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

```
frntend > src > App.jsx > ...
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));

3. Implement Frontend Logic

Step1: API Services

1. Create services/movieService.js:
 - fetchMovies()
 - a. getMovieDetails(id)
 - b. getShowtimes(movield)
2. Create services/bookingService.js:
 - a. bookSeats(showtimeld, seats)
 - b. getBookingHistory()

Step 2: Data Binding

1. Home Page:

- a. Fetch and display featured movies • Implement carousel for promotional movies

```
frtend > src > pages > Home.jsx > (m) 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.pvtm.com/Images/cinema/P%20Oh-Bhama-ayyo-Rama%20(1)-f4b10f80-5b1a-11f0-1',
14    duration: '2h 25m',
15    description: 'A hilarious comedy-drama about family relationships and misunderstandings.',
16    cast: ['Srikanth', 'Srinivas', 'Vennela Kishore'],
17    director: 'Srikanth Vissa'
18  },
19  {
20    id: 2,
21    title: 'The 100',
22    genre: 'Action, Thriller',
23    language: 'English',
24    rating: 3.8,
```

2. Movie Details:

- a. Fetch movie data by ID
- b. Display showtime availability

```
frtend > 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:

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

```

frtend > 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 = {

```

Step3: State Management

1. Use Context API for:

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

```

frtend > src > pages > UserDashboard.jsx > UserDashboard > 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',

```

1. . Authentication Flow

- a. Implement login/logout
- b. Store JWT tokens
- c. Protected routes

2. . Payment Integration

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

```
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}
```

Step4: Run the Application

bash

npm run

dev

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

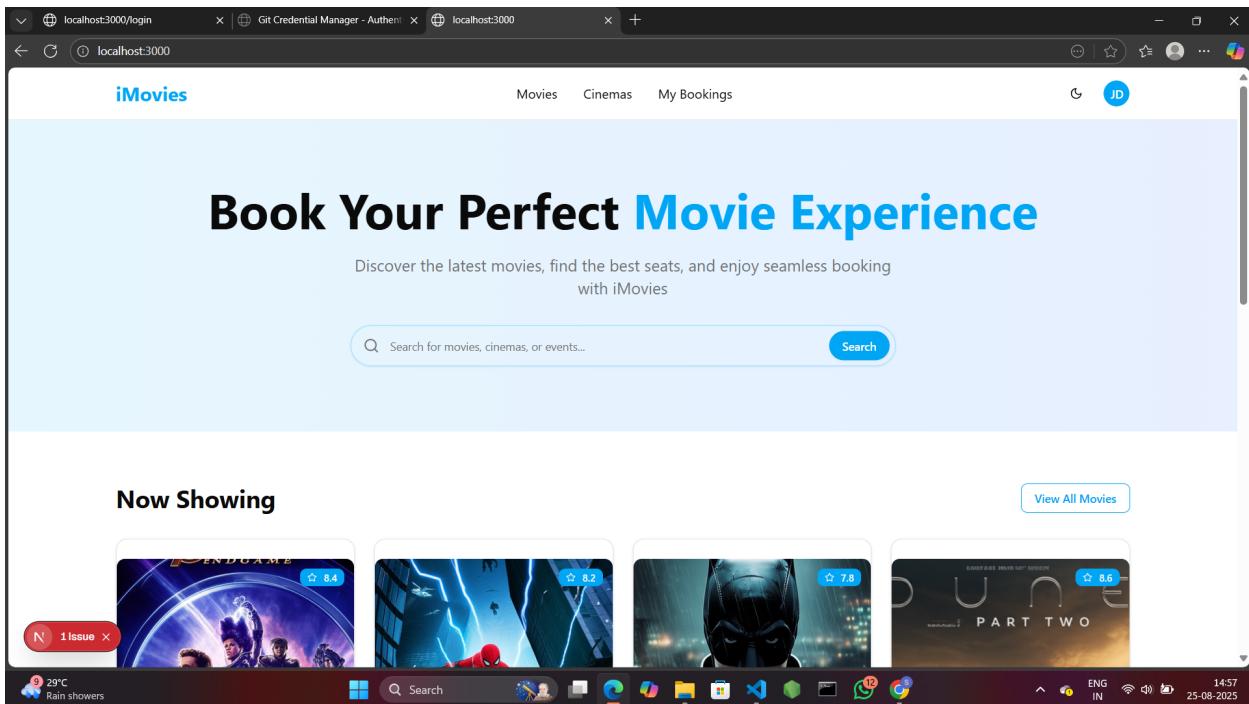
> vite-react-typescript-starter@0.0.0 dev
> vite

VITE v7.0.5 ready in 944 ms

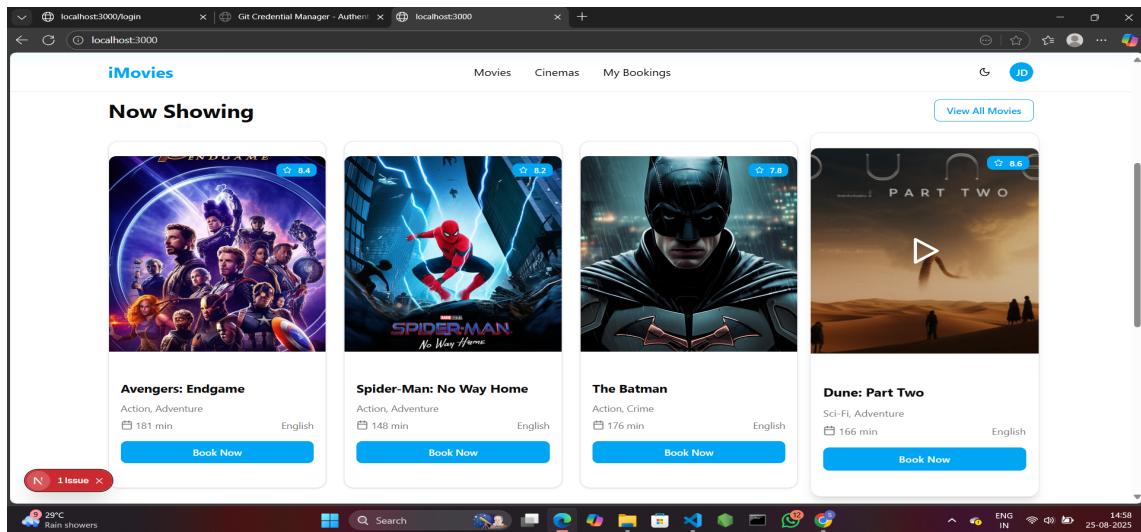
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
[]
```

5. Project Implementation

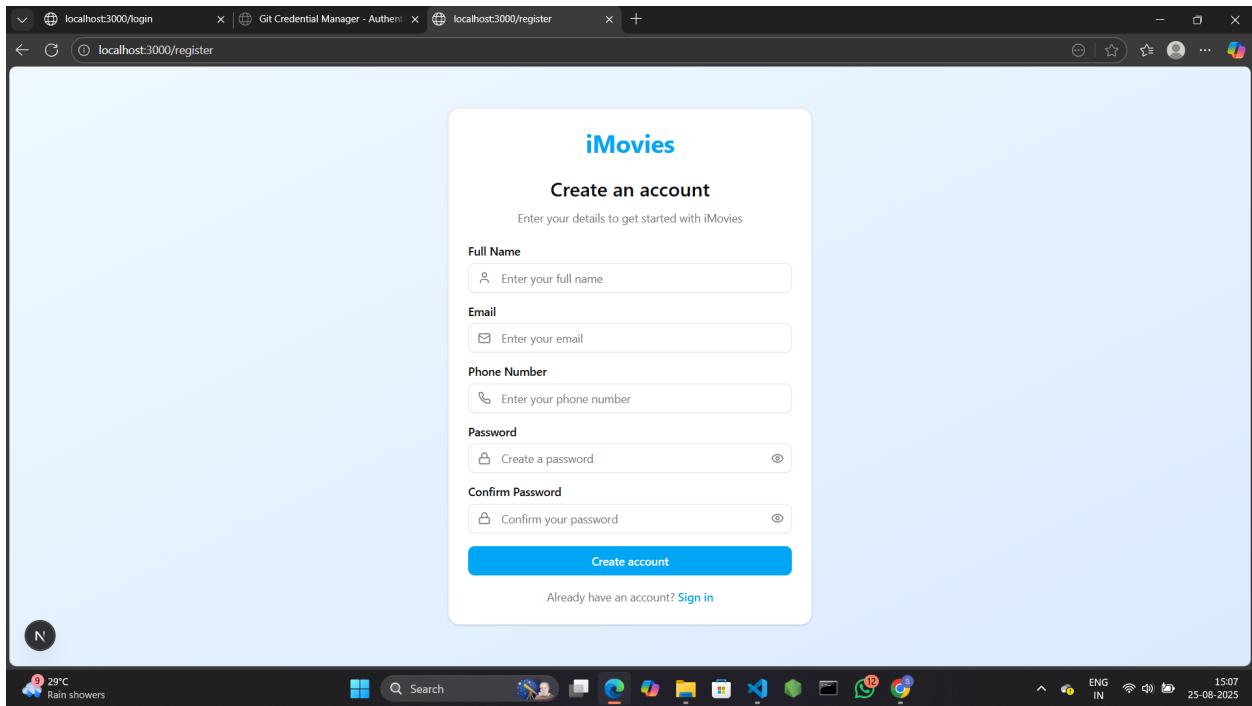
Movie details page



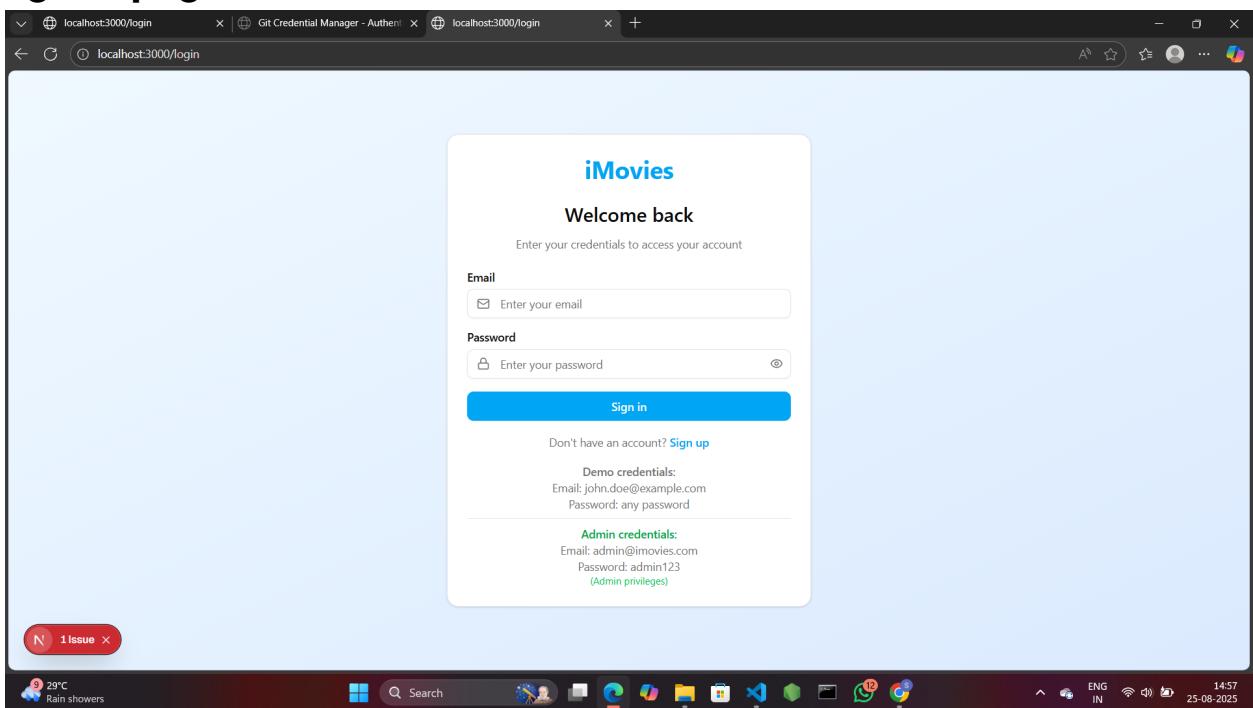
Home page



Sign up page



Sign in page



User Profile Page

The screenshot shows the 'My Profile' section of the iMovies application. On the left, there's a 'Personal Information' form with fields for Full Name (John Doe), Email Address (john.doe@example.com), and Phone Number (+1234567890). Below it is an 'Edit Profile' button. To the right is a profile summary card for 'John Doe' (john.doe@example.com) featuring a blue circular icon with 'JD'. It includes a 'Upload Photo' button. Further down is an 'Account Statistics' section showing 'Member Since Jan 2024', 'Total Bookings 12', 'Movies Watched 8', and 'Reviews Written 3'. At the bottom of the page, there are sections for 'Email Notifications' (Manage button) and 'Privacy Settings'. The browser taskbar at the top shows tabs for 'localhost:3000/login', 'Git Credential Manager - Authent...', and 'localhost:3000/profile'. The system tray at the bottom indicates a weather of 29°C Rain showers, network status, battery level, and the date/time 25-08-2025.

Bookings Page

The screenshot shows the movie details page for 'Dune: Part Two'. The main image is a desert landscape with silhouettes of figures. The title 'Dune: Part Two' is displayed with a rating of 8.6 (0 reviews) and an English subtitle. Below the title, it says 'Duration 166 min', 'Release Date 1/2/2024', 'Director Denis Villeneuve', and 'Genre Sci-Fi Adventure'. A 'Synopsis' section describes the plot. The 'Cast' section lists Timothée Chalamet, Zendaya, Rebecca Ferguson, and Josh Brolin. There are 'Book Tickets' and 'Watch Trailer' buttons. At the bottom, a 'Showtimes & Cinemas' section lists 'iMovies Multiplex Downtown' with options for IMAX, Dolby Atmos, and Recliner Seats. The browser taskbar and system tray are visible at the top and bottom respectively.

Showtimes & Cinemas

iMovies Multiplex Downtown
Downtown Plaza, 123 Main Street
2.5 km

IMAX Dolby Atmos Recliner Seats

1:00 PM \$20.99 6:15 PM \$23.99

iMovies Mall Central
Central Mall, 456 Oak Avenue
4.2 km

4DX Premium Seats Parking

4:00 PM \$25.99

Showtimes & Cinemas

iMovies Multiplex Downtown
Downtown Plaza, 123 Main Street
2.5 km

IMAX Dolby Atmos Recliner Seats

1:00 PM \$20.99 6:15 PM \$23.99

iMovies Mall Central
Central Mall, 456 Oak Avenue
4.2 km

4DX Premium Seats Parking

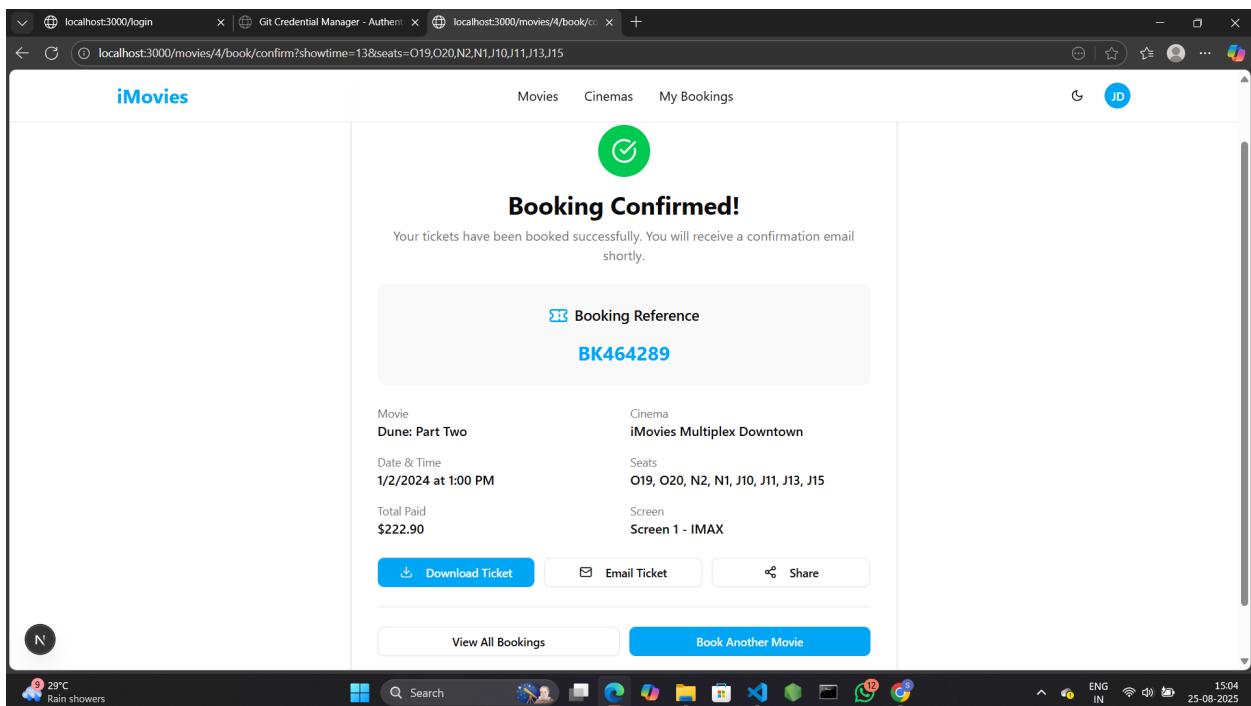
4:00 PM \$25.99

Seat Selection page

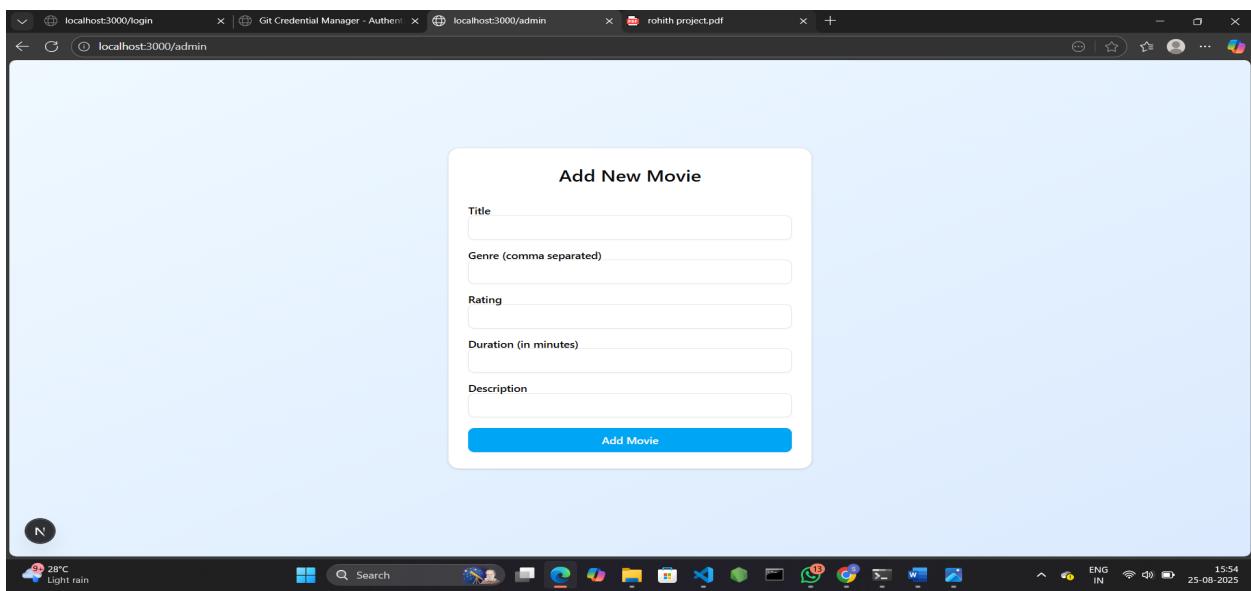
The screenshot shows the iMovies website interface for seat selection. At the top, there are tabs for 'Movies', 'Cinemas', and 'My Bookings'. The main content area displays a movie poster for 'Dune: Part Two', the date 'Thu, Feb 1', the time '1:00 PM', and the location 'iMovies Multiplex Downtown'. Below this, a seating chart titled 'Select Your Seats' shows a grid of 20 seats arranged in 7 rows (A-G) and 3 columns. Seats are color-coded: green for available, yellow for premium, blue for selected, and red for occupied. A legend at the top right of the seating chart identifies these colors. To the right of the seating chart is a 'Booking Summary' box containing movie details, showtime, and a 'Proceed to Payment' button. A message below the summary asks the user to select at least one seat. The bottom of the screen shows a taskbar with various icons and system status.

Ticket Payment Success page

The screenshot shows the iMovies website interface after a booking has been completed. The title 'Complete Your Booking' is displayed. On the left, there's a 'Payment Method' section with a radio button selected for 'Credit/Debit Card'. Below it is a 'Card Details' section with fields for 'Cardholder Name' (John Doe), 'Card Number' (1234 5678 9012 3456), 'Expiry Date' (MM/YY), and 'CVV' (123). A note states that payment is secured with 256-bit SSL encryption. At the bottom of this section is a large blue button labeled 'Pay \$222.90'. To the right is a 'Booking Summary' box showing the movie 'Dune: Part Two' on 1/2/2024 at 1:00 PM at 'iMovies Multiplex Downtown'. It also lists the selected seats (O19, O20, N2, N1, J10, J11, J13, J15) and their details. A summary table at the bottom shows the Subtotal (\$199.92), Taxes & Fees (\$22.98), and Total (\$222.90). The bottom of the screen shows a taskbar with various icons and system status.



Admin DashBoard Page



iMovies

Movies Cinemas My Bookings Admin

My Profile

Manage your account settings and preferences

Personal Information

Full Name: Admin User

Email Address: admin@imovies.com

Phone Number: +1234567891

Edit Profile

Admin User

admin@imovies.com

Admin

Upload Photo

Account Settings

Email Notifications: Receive booking confirmations and updates **Manage**

Account Statistics

| Member Since | Jan 2024 |
|----------------|----------|
| Total Bookings | 12 |
| Movies Watched | 8 |

28°C Light rain

Search

25-08-2025

iMovies

Movies **Cinemas** My Bookings Admin

Cinemas

Find the perfect cinema near you

Search cinemas... All Facilities Distance Use My Location

Showing 3 of 3 cinemas

iMovies Multiplex Downtown ★ 4.5 3 screens
Downtown Plaza, 123 Main Street
2.5 km
Facilities: IMAX, Dolby Atmos, Recliner Seats, Food Court
Screen Types: IMAX, Standard, Premium
View Details

iMovies Mall Central ★ 4.5 3 screens
Central Mall, 456 Oak Avenue
4.2 km
Facilities: 4DX, Premium Seats, Parking, Restaurant
Screen Types: 4DX, Standard, Premium
View Details

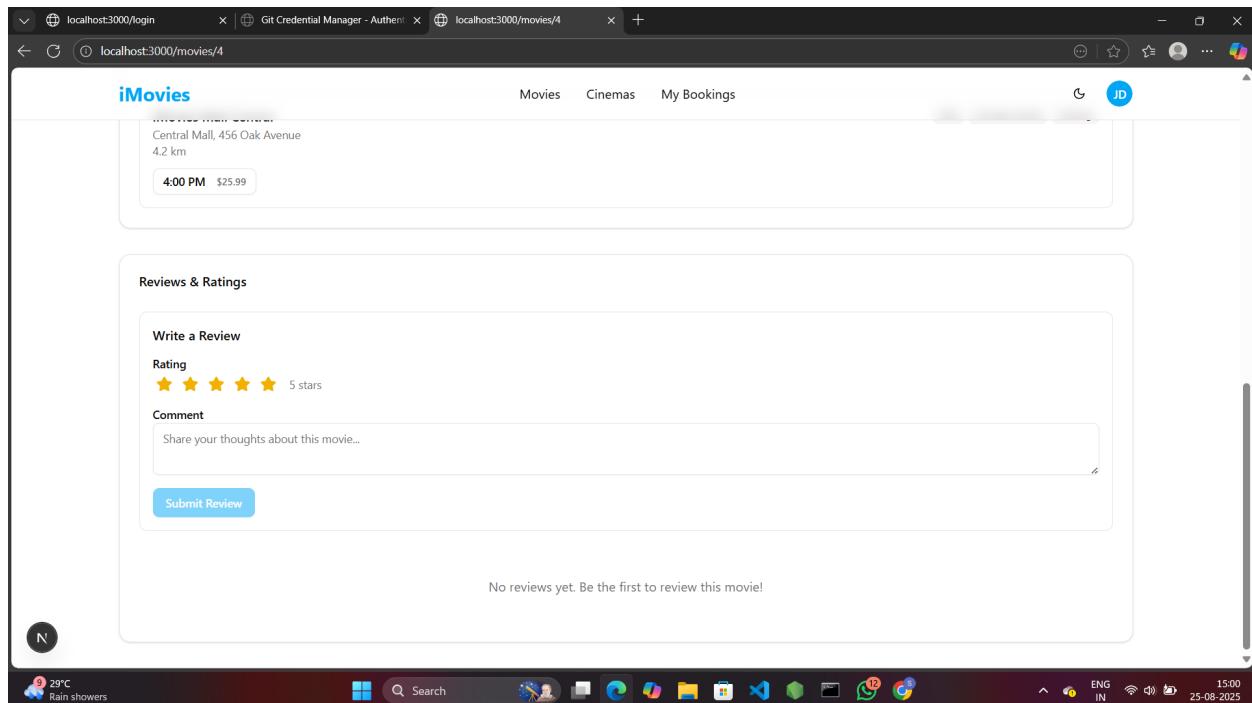
iMovies Westside ★ 4.5 2 screens
Westside Complex, 789 Pine Road
6.8 km
Facilities: Dolby Vision, Luxury Recliners, VIP Lounge
Screen Types: Premium, Standard
View Details

28°C Light rain

Search

25-08-2025

Feedback Page



Conclusion

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

and an exceptional user experience from start to finish.