

# I-Movies : Movie TicketBooking System

## ProjectDescription:

Users can effortlessly browse movies, select showtime, and choose seats from the comfort of their homes. The app uses 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 support provides a comprehensive guide for setup, development, and understanding of the application's technical architecture and functionalities.

## Scenario

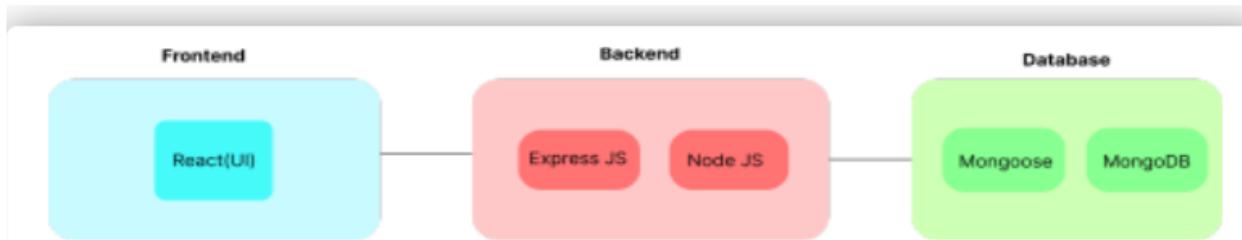
Imagine Sarah, a movie enthusiast, excitedly expecting the release of an anticipated film. With a busy schedule, she prefers the convenience of booking her movie tickets online. Logging into the iMovies app, Sarah navigates effortlessly through the user-friendly interface, browsing the list of usable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

Upon finding the movie she has been eagerly awaiting, Sarah selects her preferred cinema location and showtime. The app displays a seating layout, allowing her to choose the perfect seats for an optimal viewing 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, Watchlist, Movies page, Movie shows page, profile, Seat allotment page, Bookings page etc.,
2. The backend is represented by the "Backend" section, consisting of API endpoints for Users, Favorites, Shows, movies, bookings etc

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

The technical architecture of the 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.

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

## 2. MongoDB

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

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

## 3. Express.js

A Node.js framework for backend API development.

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

## 4. React.js

JavaScript library for building dynamic UI's.

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

## HTML, CSS, and JavaScript

Front-end developers need fundamental knowledge.

## 6. Database Connectivity

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

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

## 7. Firebase Storage (for Images)

To upload/store images:

a. **Create a Firebase Project:** <https://console.firebaseio.google.com/>

b. **Install Firebase**

**SDK:** 3. bash

```
'npm install Firebase'
```

**Enable Storage** in Firebase Console and configure:

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

**Version Control (Git):**

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

## 9. Development Environment

Choose an IDE:

a. **VS Code:** <https://code.visualstudio.com/download>

b. **Sublime Text:** <https://www.sublimetext.com/download>

c. **WebStorm:** <https://www.jetbrains.com/webstorm/download>

## PROJECT FLOW



# 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:

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

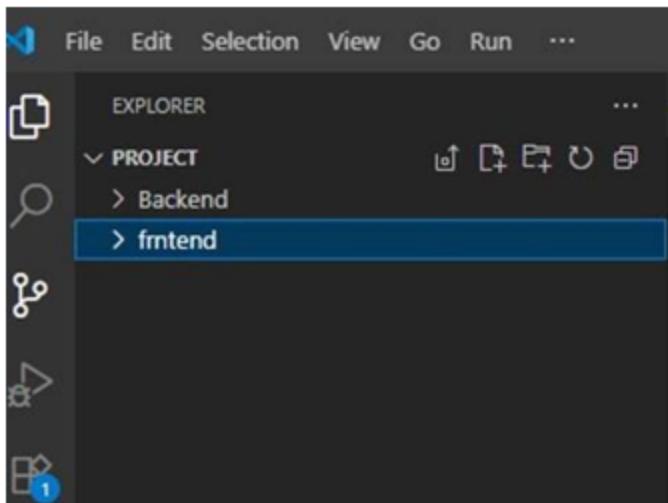
```
cd Backend
```

```
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

### 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 serverfile:  
bash  
touch index.js

### Configure Environment Variables

5. Create a .env file in your Backend folder:  
bash  
touch .env
6. Add these configurations to your .env file:  
env

```
PORT=5000
MONGODB_URL=mongodb://localhost:27017
/imovie_db
JWT_SECRET=your_strong_secret_key_here
```

### a. Basic Server Setup

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

### Authentication Setup

2. Create a models/User.js file for user schema:
3. Create a routes/auth.js file for authentication routes:

```
Backend > routes > js authRoutes.js
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,
```

### 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) => {
```

## Start Your Server

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

### 1. Testing The Backend by using the thunder client:

POST **http://localhost:5000/api/v1/auth/login** Send

Status: 200 OK Size: 381 Bytes Time: 583 ms

| Response                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Headers <sup>19</sup> | Cookies | Results | Docs | () | ≡ |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|---------|---------|------|----|---|
| <pre>1  { 2    "status": "success", 3    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. 4        .eyJpZC16IjY400IzOGE0ZTF1NTl1ZmU3MzN1NTI3MIIisImhd 5        .CI6MTc1MzQ1ODM2MCwiZXhwIjoxNzU2MDUwMzYwfQ 6        .00u175gbdFHq-pjhhdNIMls-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 }</pre> |                       |         |         |      |    |   |

Query Headers<sup>2</sup> Auth Body<sup>1</sup> Tests Pre-Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1  {
2    "email": "john@example.com",
3    "password": "test1234"
4  }
5
6
```

# 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.

The screenshot shows a code editor interface with a sidebar labeled 'EXPLORER' containing project files: 'SHOPIZ', 'client', 'server', 'node\_modules', '.env', 'index.js', 'package-lock.json', and 'package.json'. The main area displays the content of 'index.js' with syntax highlighting for JavaScript code. The code imports express, mongoose, cors, and dotenv, configures the app to use express and cors, and starts the server at port 3001. It also attempts to connect to a MongoDB database using the provided URI. The terminal below shows command-line interactions where the user runs 'node index.js' and the application logs its start and successful database connection.

```
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_URI;
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
```

TERMINAL

```
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
```

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 })
```

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
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:

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.

```
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:

16.Schema: favoriteSchema

17.Model: 'Favorite'

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

19.It enables usersto bookmark and access theirpreferred movies easily, enhancing their experience on the platform.

```
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 schemacaptures details of user bookingsincluding booking ID, user email, show ID, and ticket data.

23.It facilitates the management and tracking of user reservations, ensuring a smooth booking process and accurate record-keeping.

```
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,
    ],
  });
};

//dynamic protection - catchAsync/async/ensureAuth/auth.js

```

## 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
```

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

### **Step 3: Folder Structure**

```
frntend/  
    └── node_modules/      # All npm packages (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

### Step1: Create Core Components

#### 1. Header.jsx

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

```
frtend > 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

```
frtend > 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-8 h-8 flex items-center justify-center" data-lucide="IMovie">
18                 </div>
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.
24           </p>
25         </div>
26       </footer>
27     </>
28   );
29 }
```

## FeedbackForm.jsx

Rating input

Comment textarea

Submission button

```
frtend > 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

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

```
frtend > 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));
25     }
26   }, []);
27
28   return (
29     <Router>
30       <Routes>
31         <Route path="/" element={<Home />} />
32         <Route path="/movies" element={<MovieDetails />} />
33         <Route path="/movies/:id" element={<MovieDetails />} />
34         <Route path="/cinemas/:movield" element={<CinemaSelection />} />
35         <Route path="/book/:showtimeld" element={<SeatSelection />} />
36         <Route path="/payment" element={<Payment />} />
37         <Route path="/confirmation" element={<BookingConfirmation />} />
38         <Route path="/login" element={<Login />} />
39         <Route path="/register" element={<Register />} />
40         <Route path="/dashboard" element={<UserDashboard />} />
41         <Route path="/admin" element={<AdminDashboard />} />
42       </Routes>
43     </Router>
44   );
45 }
```

### 3. Implement Frontend Logic

#### Step1: API Services

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

#### Step 2: Data Binding

##### 1. Home Page:

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

```
frontend > 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-11f0-1',
 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: '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

```

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                <X className="h-5 w-5" />
18              </button>
19            </div>
20            <div className="aspect-video">
21              <iframe
22                width="100%"
23                height="100%">

```

### Seat Selection:

- c. Fetch available seats
- d. Track selected seats
- e. 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 = [

```

## Step3: State Management

1. Use Context API for:
  1. User authentication
  2. Booking process

### 3. UI preferences

```
frontend > 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',
25      }
26    ];
27    setBookings(mockBookings);
28  });
29}
```

## 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

```
frontend > 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  
npmrun  
dev
```

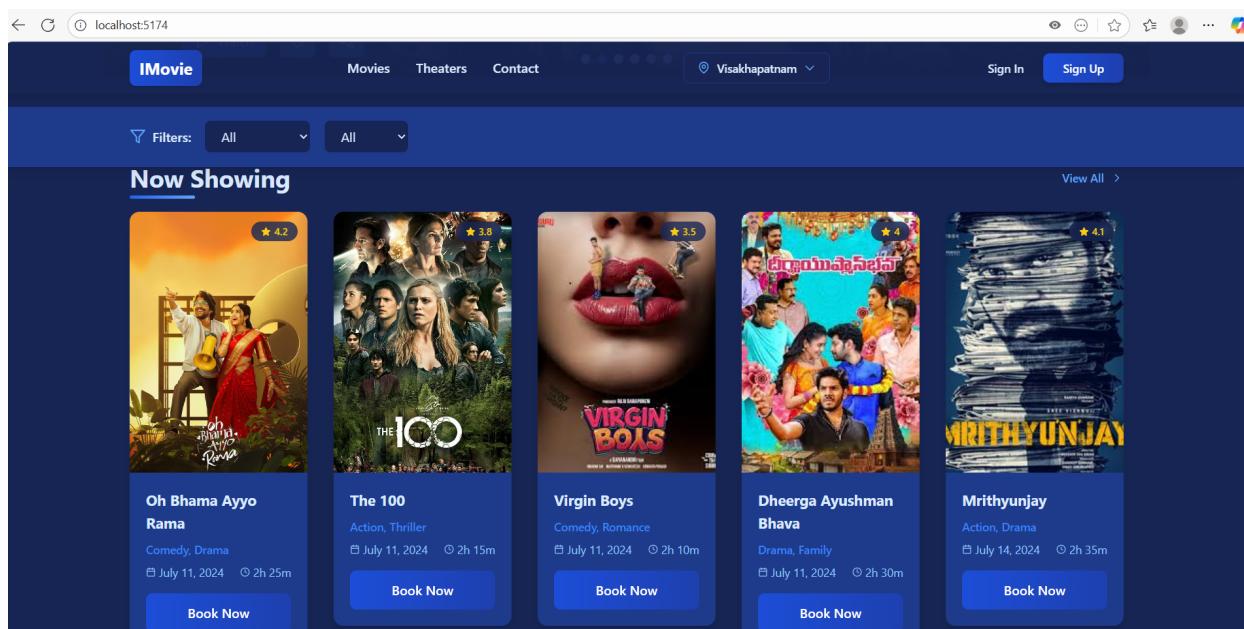
```
PS D:\project> cd frntend  
PS D:\project\frntend> npm run dev  
  
> vite-react-typescript-starter@0.0.0 dev  
> vite
```

```
VITE v7.0.5 ready in 944 ms
```

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

## 5. Project Implementation

### Movie details page



localhost:5174/movie/2

The 100

3.8/5 A Telugu

July 11, 2024 2h 15m Action, Thriller

Login to Book Watch Trailer

About Cast & Crew Reviews

**Synopsis**

An intense action thriller that keeps you on the edge of your seat.

**Movie Details**

Director: Sam Anton  
Producer: Rockfort Entertainment

## Home page

localhost:5174

Dheerga Ayushman Bhava

Drama, Family | Telugu

4 Watch

Filters: All All

Now Showing

View All >

## Sign up page

The screenshot shows the IMovie website's sign-up page. At the top, there is a navigation bar with links for 'Movies', 'Theaters', and 'Contact'. A dropdown menu for 'Visakhapatnam' is open. On the right side of the header, there are 'Sign In' and 'Sign Up' buttons. The main content area has a dark blue background with white text. It features a title 'Create your account' and 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'), and 'Confirm Password' (placeholder 'Confirm your password'). Each password field includes an eye icon for password visibility. At the bottom of the form, there is a checkbox labeled 'I agree to the [Terms and Conditions](#) and [Privacy Policy](#)'.

## Sign in page

The screenshot shows the IMovie website's sign-in page. The layout is similar to the sign-up page, with a dark blue header and white text. The title is 'Sign in to your account' and the sub-instruction is 'Or create a new account'. There are two input fields: 'Email Address' (placeholder 'Enter your email') and 'Password' (placeholder 'Enter your password'). Each password field has an eye icon. Below the password fields are two buttons: 'Remember me' (with a checked checkbox) and 'Forgot your password?'. In the center, there is a large blue 'Sign in' button. Below the sign-in form, there is a horizontal line with the text 'Or continue with' followed by two social media icons for Google (G) and Facebook (F).

## User ProfilePage

Welcome back, John Doe!

Manage your bookings and account settings

My Bookings   Profile   Preferences

Upcoming Shows

No upcoming bookings

Ready to watch something amazing?

Browse Movies

## Bookings Page

Select Date

Sun, Aug 31 Mon, Sep 1 Tue, Sep 2 Wed, Sep 3 Thu, Sep 4 Fri, Sep 5 Sat, Sep 6

**INOX CMR Central**  
CMR Central Mall, Maddilapalem, Visakhapatnam  
0.2.5 km ★ 4.3  
Parking Food Court WiFi AC  
IMAX 4DX Regular

Show Times

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

**PVR Vizag**  
Jagadamba Junction, Visakhapatnam  
0.3.2 km ★ 4.1  
Parking Food Court WiFi  
Premium Regular

Show Times

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

## Seat Selection page

The screenshot shows the IMovie website's seat selection interface. At the top, there are tabs for 'Movies', 'Theaters', and 'Contact'. The location is set to 'Visakhapatnam' and the user is 'John Doe'. The main area displays a seating chart for a screen, divided into two sections: 'Premium - ₹250' and 'Regular - ₹180'. The Premium section has rows A, B, and C with seats numbered 1 through 12. The Regular section has rows D through J with seats numbered 1 through 14. Seats C12, C11, C10, and C9 are highlighted in red, indicating they are selected. To the right, a 'Booking Summary' box shows the selected seats (C12, C11, C10, C9) and a total amount of ₹1000. A 'Proceed to Payment' button is visible.

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

**Booking Summary**

Selected Seats (4)

Premium ₹250 x 4  
Seats: C12, C11, C10, C9 ₹1000

Total Amount ₹ 1000

Proceed to Payment

• 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 website's payment success page. The top navigation and user information are identical to the previous screen. The main content is divided into two main sections: 'Select Payment Method' and 'Order Summary'. The 'Select Payment Method' section offers three options: 'Credit/Debit Card' (Visa, MasterCard, Rupay), 'UPI' (PhonePe, GPay, Paytm, BHIM), and 'Digital Wallet' (Paytm, Amazon Pay, MobiKwik). The 'Order Summary' section provides a detailed breakdown of the booking. It includes 'Booking Details' (Movie: Movie Title, Cinema: Cinema Name, Date & Time: 09:30 AM, Seats: C12, C11, C10, C9), a breakdown of costs (Ticket Price (4 tickets): ₹1000, Convenience Fee: ₹20, Taxes (GST 18%): ₹184), and a final 'Total Amount' of ₹ 1204. Below the summary, there are instructions: '• Tickets once booked cannot be cancelled', '• Please arrive 15 minutes before showtime', and '• Carry a valid ID proof'.

Payment Complete your booking

Select Payment Method

Credit/Debit Card Visa, MasterCard, Rupay

UPI PhonePe, GPay, Paytm, BHIM

Digital Wallet Paytm, Amazon Pay, MobiKwik

**Order Summary**

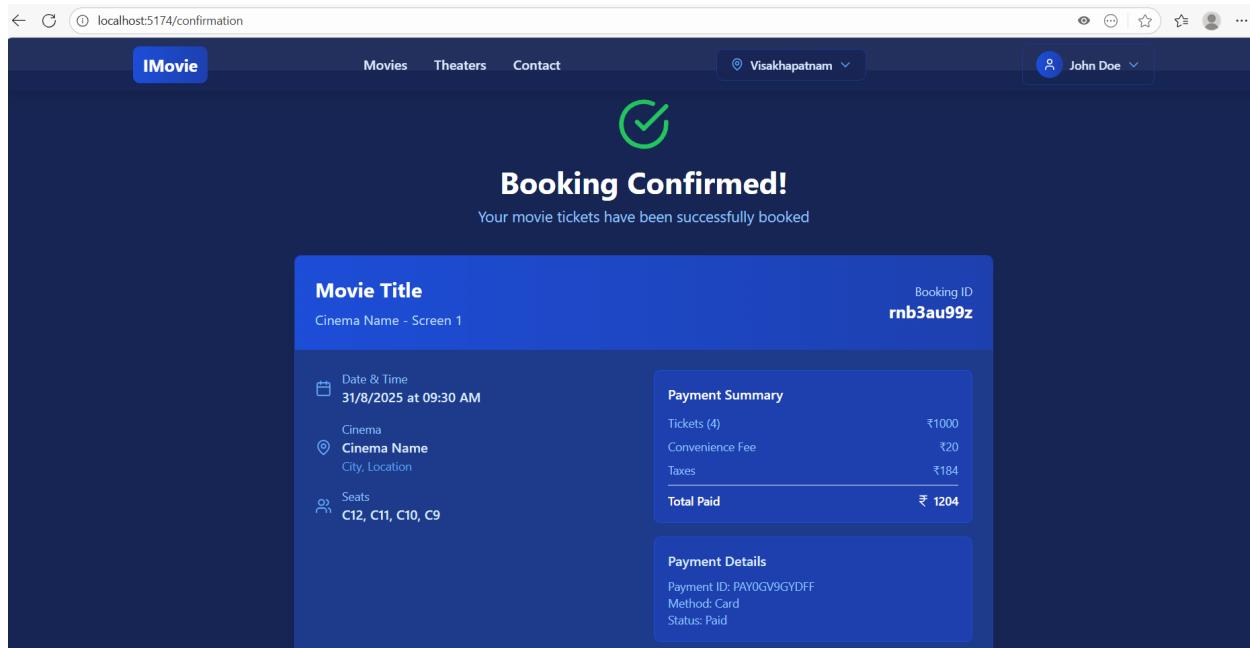
**Booking Details**

Movie: Movie Title  
Cinema: Cinema Name  
Date & Time: 09:30 AM  
Seats: C12, C11, C10, C9

Ticket Price (4 tickets) ₹1000  
Convenience Fee ₹20  
Taxes (GST 18%) ₹184

**Total Amount** ₹ 1204

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



## Admin DashBoardPage

A screenshot of the 'Admin Dashboard' page. The header is identical to the booking confirmation page. The main content area is titled 'Admin Dashboard' and includes a sub-header 'Manage movies, theaters, and bookings'. Below this is a navigation bar with tabs: 'Overview' (selected), 'Movies', 'Theaters', and 'Users'. Four summary cards are displayed: 'Total Movies' (0), 'Total Bookings' (1247), 'Total Revenue' (₹24,56,780), and 'Active Users' (8934). A 'Recent Activity' section lists 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).

# Feedback Page

The screenshot shows a feedback form titled "Share Your Feedback" on a dark blue background. The form includes fields for Full Name, Email Address, Phone Number, Overall Experience (with a 5-star rating), Feedback Category (set to "General Feedback"), Subject (Brief subject of your feedback), and Your Message (a large text area). The browser address bar at the top shows "localhost:5174".

Full Name \*

Email Address \*

Phone Number

Overall Experience \*

Feedback Category \*

Subject \*

Your Message \*

## 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.