I-Movies : Movie Ticket Booking System

Project Description:

Users can effortlessly browse movies, select showtime, and choose seats from the comfort of their homes. purchase a robust client-server architecture with Express.js and MongoDB, the app ensures efficient data storage and retrieval. With features like user hallmark, booking management, and real-time seat handiness tracking, it provides a personalized and convenient movie-going experience. This support provides a comprehensive guide for setup, development, and understanding of the application's technical architecture and functionalities.

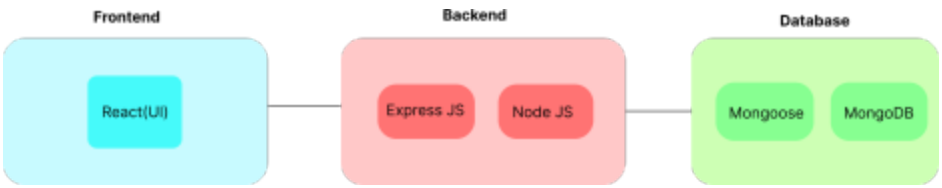
Scenario

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

Upon finding the movie she has been eagerly awaiting, Sarah selects her preferred cinema location and showtime. The app displays a seating layout, allowing her to choose the perfect seats for an optimal viewing see. With a few clicks, Sarah confirms her booking and proceeds to the payment section.

Using the secure payment gateway integrated into the iMovies app, Sarah completes her transaction smoothly, receiving a booking confirmation with all the details she needs for her upcoming movie night. As the date approaches, Sarah can easily access her booking history through the app, ensuring a hassle-free experience from start to finish.

Thanks to the iMovies app's intuitive design and robust features, Sarah enjoys a seamless movie going experience, making her next cinema outing memorable and stress-free.



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 officialwebsite. 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.firebase.google.com/

2. **Install Firebase SDK**:

3. bash
'npm install Firebase'**Enable Storage** in Firebase Console and configure:
javascript
'import { getStorage, ref, uploadBytesResumable, getDownloadURL } from "Firebase/storage";'
8. Version Control (Git):
● **Download Git**: https://git-scm.com/downloads
9. Development Environment
Choose an IDE:
● **VS Code**: https://code.visualstudio.com/download
● **Sublime Text**: https://www.sublimetext.com/download
● **WebStorm**: https://www.jetbrains.com/webstorm/download

1. Project setup and configuration

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

■ **Install Node.js and Git**
Run these commands one by one in your terminal:
bash
Verify installations
'node --version'
npm --version
git --version

If any command fails, download Node.js from nodejs.org and Git from git-scm.com.
■ **Create Project Structure**
Execute these commands to set up the basic folder structure:

bash

mkdir imovie-app

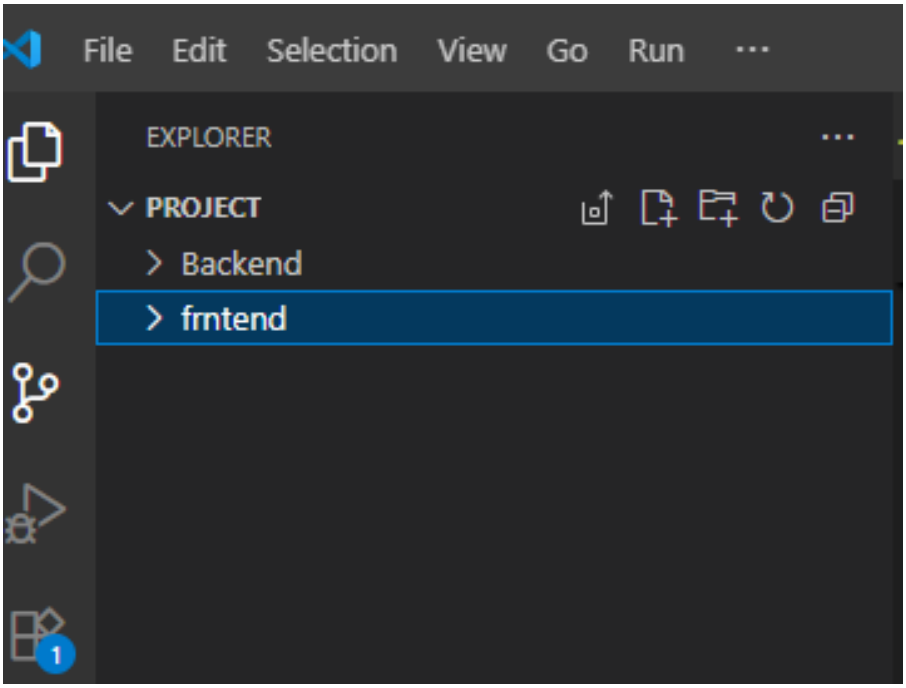
cd imovie-app
■ **Set Up React Frontend**
Create your frontend with this command:
bash
npx create-react-app frontend
This will generate:
● frontend/src/ for React components
● frontend/public/ for static assets
● All necessary configuration files
■ **Initialize Backend**
Run these commands to set up your Node.js backend:
bash
mkdir Backend
cd Backend
npm init -y
■ **Create Backend Structure**
While still in the Backend folder, run:
bash
mkdir models routes
touch server.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 > ...

```
const dotenv = require('dotenv');

// Handle uncaught exceptions
process.on('uncaughtException', (err) => {
  console.log('UNCAUGHT EXCEPTION! 💥 Shutting down...');
  console.log(err.name, err.message);
  process.exit(1);
});

// Load environment variables
dotenv.config();

// Check if MongoDB URI is configured
if (!process.env.MONGODB_URI) {
  console.log('MONGODB_URI not found in environment variables, using default: mongodb://127.0.0.1:27017/i
}

const app = require('./app');
const connectDB = require('./config/database');

// Connect to MongoDB with error handling
connectDB().catch(err => {
  console.error('Failed to connect to MongoDB:', err.message);
```

Authentication Setup

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

Backend > models > JS User.js > ...

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please provide a name'],
    trim: true,
  },
  email: {
    type: String,
    required: [true, 'Please provide an email'],
    unique: true,
    lowercase: true,
    trim: true,
  },
  phone: {
    type: String,
    required: [true, 'Please provide a phone number'],
  },
  password: {
    type: String,
    required: [true, 'Please provide a password'],
    minlength: 6,
```

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

Backend > routes > JS authRoutes.js

```
const express = require('express');
const authController = require('../controllers/authController');

const router = express.Router();

// Public routes
router.post('/register', authController.register);
router.post('/login', authController.login);

// Protected routes
router.use(authController.protect);
router.get('/me', authController.getMe);

module.exports = router;
```

■ **Error Handling Middleware**

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

Backend > middleware > JS errorHandler.js

```
const AppError = require('../utils/appError');

const handleCastErrorDB = (err) => {
  const message = `Invalid ${err.path}: ${err.value}.`;
  return new AppError(message, 400);
};

const handleDuplicateFieldsDB = (err) => {
  const value = err.errmsg?.match(/(["'])(\\?.)*?\1/)?.[0] || 'duplicate value';
  const message = `Duplicate field value: ${value}. Please use another value!`;
  return new AppError(message, 400);
};

const handleValidationErrorDB = (err) => {
  const errors = Object.values(err.errors).map((el) => el.message);
  const message = `Invalid input data. ${errors.join('. ')}`;
  return new AppError(message, 400);
};

const handleJWTError = () =>
  new AppError('Invalid token. Please log in again!', 401);

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

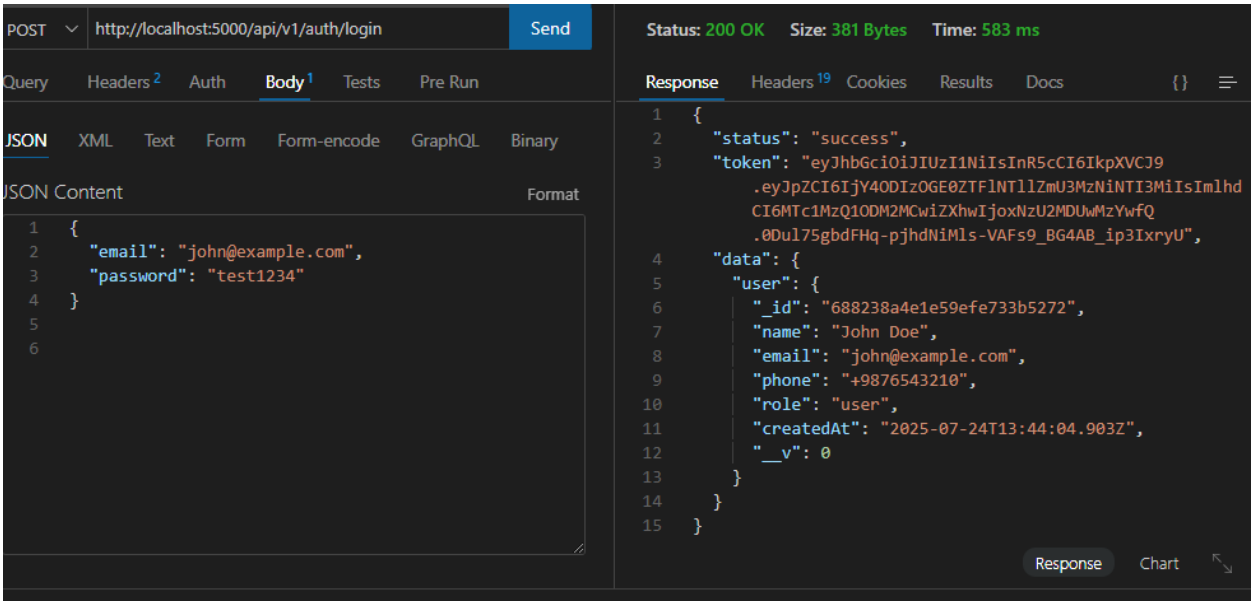
Start Your Server

● Run the backend server with:

bash

node server.js

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

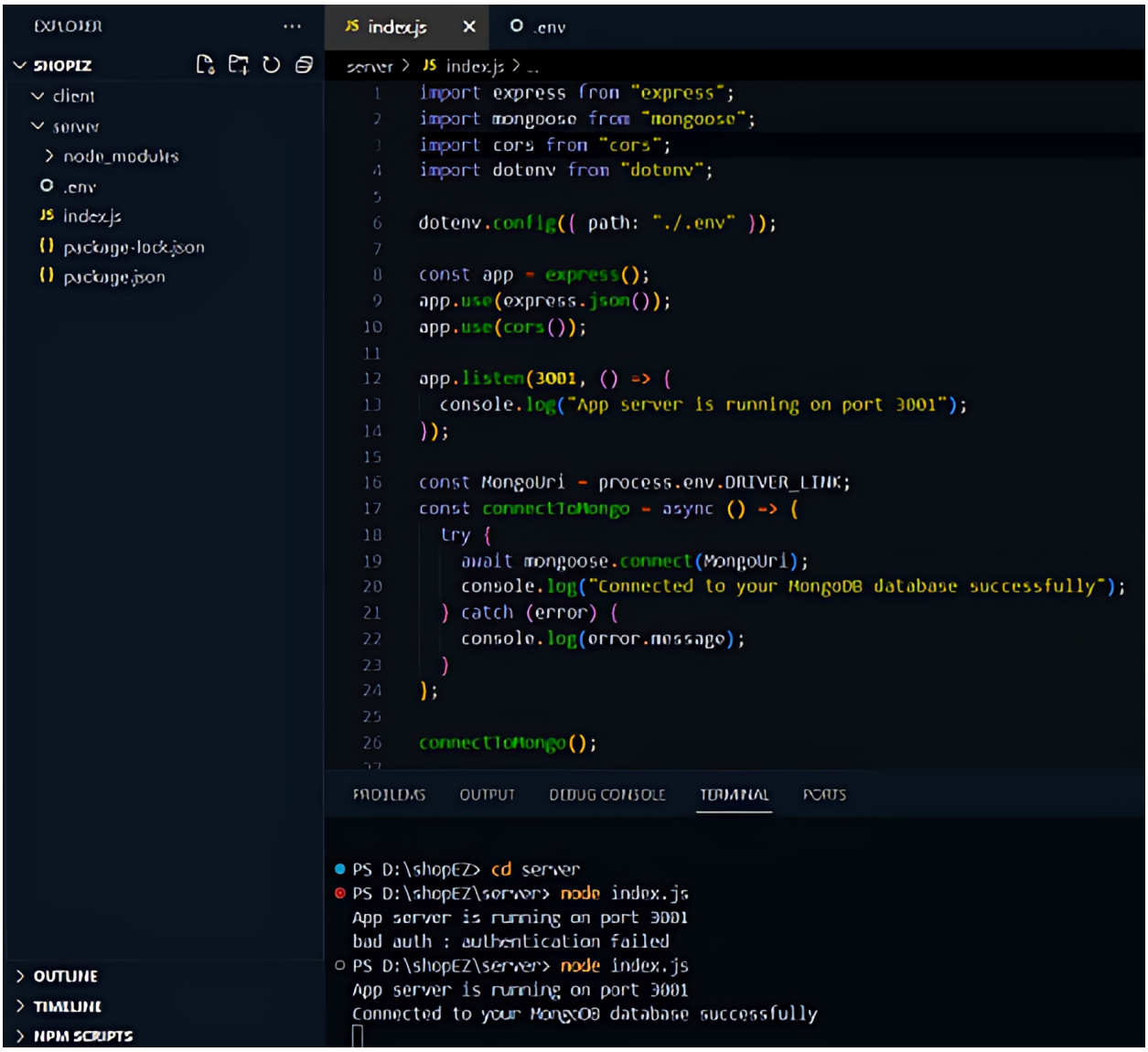


3.Database Development

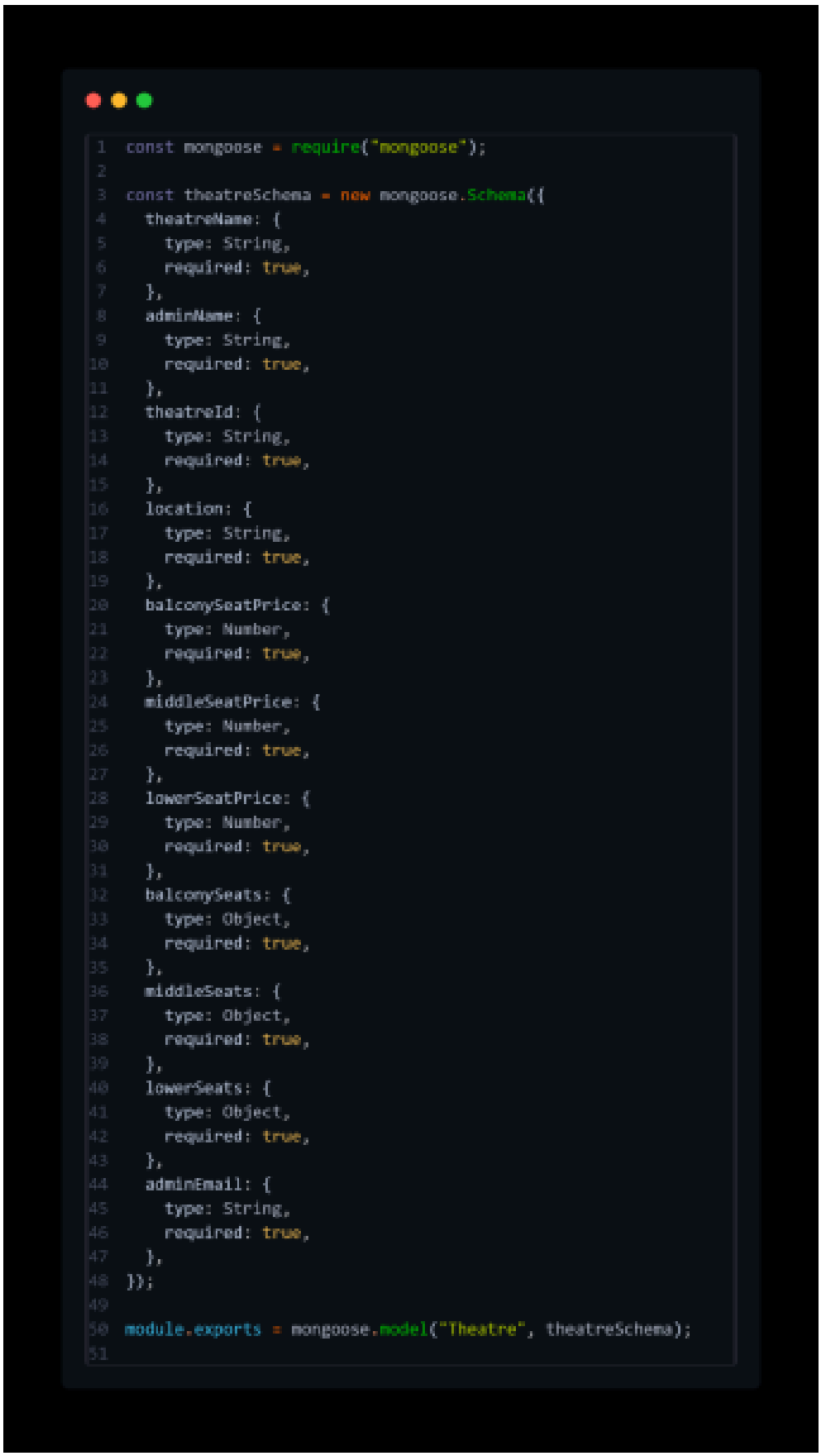
Creating Data Schemas

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.



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



```
const mongoose = require("mongoose");

const theatreSchema = new mongoose.Schema({
  theatreName: {
    type: String,
    required: true,
  },
  adminName: {
    type: String,
    required: true,
  },
  theatreId: {
    type: String,
    required: true,
  },
  location: {
    type: String,
    required: true,
  },
  balconySeatPrice: {
    type: Number,
    required: true,
  },
  middleSeatPrice: {
    type: Number,
    required: true,
  },
  lowerSeatPrice: {
    type: Number,
    required: true,
  },
  balconySeats: {
    type: Object,
    required: true,
  },
  middleSeats: {
    type: Object,
    required: true,
  },
  lowerSeats: {
    type: Object,
    required: true,
  },
  adminEmail: {
    type: String,
    required: true,
  },
});

module.exports = mongoose.model("Theatre", theatreSchema);

```

Show Schema:
- Schema: showSchema
- Model: 'Show'
- The Show schema represents individual movie showings with attributes such as admin email, movie ID, theater name, date, time, and ticket details.
- It organizes and stores data related to movie screenings, enabling users to browse and book show-times effectively.

```
const mongoose = require("mongoose");

const showSchema = new mongoose.Schema({
  //show created by admin
  adminEmail: {
    type: String,
    required: true,
  },
  showId: {
    type: String,
    required: true,
  },
  movieId: {
    type: String,
    required: true,
  },

  theatreName: {
    type: String,
    required: true,
  },
  showdate: {
    type: String,
    required: true,
  },
  showtime: {
    type: String,
    required: true,
  },
  tickets: {
    type: Object,
  },
});

module.exports = mongoose.model("show", showSchema);

```

Movie Schema:
- Schema: movieSchema
- Model: 'Movie'
- The Movie schema defines properties for movies including name, description, genres, release date, runtime, certification, media format, and associated show IDs.
- It encapsulates movie data, facilitating organization and retrieval of information about available films for users to explore and book.

```
const mongoose = require("mongoose");

const movieSchema = new mongoose.Schema({
  movieName: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  genres: {
    type: String,
    required: true,
  },
  releaseDate: {
    type: String,
    required: true,
  },
  runtime: {
    type: Number,
    required: true,
  },
  certification: {
    type: String,
    required: true,
  },
  media: {
    type: String,
    required: true,
  },
  movieId: {
    type: String,
    required: true,
  },
  //contains showid's
  shows: [{ type: String }],
});

module.exports = mongoose.model("Movie", movieSchema);

```

Favorite Schema:
- Schema: favoriteSchema
- Model: 'Favorite'
- The 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.

```
const mongoose = require("mongoose");

const favoriteSchema = new mongoose.Schema({
  userEmail: {
    type: String,
    required: true,
  },
  movieId: {
    type: String,
    required: true,
  },
});

module.exports = mongoose.model("Favorite",
favoriteSchema);

```

Booking Schema:
- Schema: bookingModel
- Model: 'Booking'
- The 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.

```
const mongoose = require("mongoose");

const bookingModel = new mongoose.Schema({
  bookingId: {
    type: String,
    required: true,
  },
  userEmail: {
    type: String,
    required: true,
  },
  showId: {
    type: String,
    required: true,
  },
  ticketsData: {
    type: Object,
    requires: true,
  },
});

module.exports = mongoose.model("Booking",
bookingModel);

```

Admin Schema:
- Schema: adminSchema
- Model: 'Admin'
- The Admin schema defines fields for admin accounts such as username, email, and password with specified constraints.
- It represents administrative users in the system, providing access to privileged functionalities for managing the application.

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

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

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

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

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

4.Frontend development

1. Setup React Application

Step 1: Create React App

Run this command in your project root:

bash

npm create vite@latest

cd frontend

Step 2: Install Essential Libraries

bash

npm install axios react-router-dom react-icons react-toastify @mui/material @mui/icons material

npm install -D tailwindcss Postcss autoprefixer

npx tailwindcss init -p

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

```
frntend > 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
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import { Facebook, Twitter, Instagram, Youtube, Mail, Phone, MapPin, Star } from 'lucide-react';
import FeedbackForm from './FeedbackForm';

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

  return (
    <>
      <footer className="bg-gradient-to-r from-gray-900 via-gray-800 to-gray-900 text-white">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-12">
          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8">
            {/* Company Info */}
            <div className="space-y-4">
              <div className="flex items-center space-x-2">
                <div className="bg-gradient-to-r from-red-600 to-red-700 text-white px-4 py-3 roun
                  IMovie
                </div>
              </div>
              <p className="text-gray-300 text-sm leading-relaxed">
                Your premier destination for movie ticket booking in Andhra Pradesh.
                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 > ...
import React, { useState } from 'react';
import { X, Star, Send } from 'lucide-react';

const FeedbackForm = ({ onClose }) => {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    phone: '',
    rating: 0,
    category: 'general',
    subject: '',
    message: ''
  });
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleRatingClick = (rating) => {
```

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 > ...

```css
@tailwind base;
@tailwind components;
@tailwind utilities;

/* Custom scrollbar */
::-webkit-scrollbar {
  width: 8px;
}

::-webkit-scrollbar-track {
  background: #f1f1f1;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb {
  background: #dc2626;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
  background: #b91c1c;
}

/* Smooth scrolling */
```

Step 3: Set Up Routing

Define routes in App.jsx

- Home (/)

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

frntend > src > App.jsx > ...

```jsx
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import Header from './components/Header';
import Footer from './components/Footer';
import Home from './pages/Home';
import MovieDetails from './pages/MovieDetails';
import CinemaSelection from './pages/CinemaSelection';
import SeatSelection from './pages/SeatSelection';
import Payment from './pages/Payment';
import BookingConfirmation from './pages/BookingConfirmation';
import UserDashboard from './pages/UserDashboard';
import AdminDashboard from './pages/AdminDashboard';
import Login from './pages/Login';
import Register from './pages/Register';

function App() {
  const [user, setUser] = useState(null);
  const [selectedLocation, setSelectedLocation] = useState('Visakhapatnam (Vizag)');

  useEffect(() => {
    // Check for logged in user
    const userData = localStorage.getItem('userData');
    if (userData) {
      setUser(JSON.parse(userData));
```

3. Implement Frontend Logic

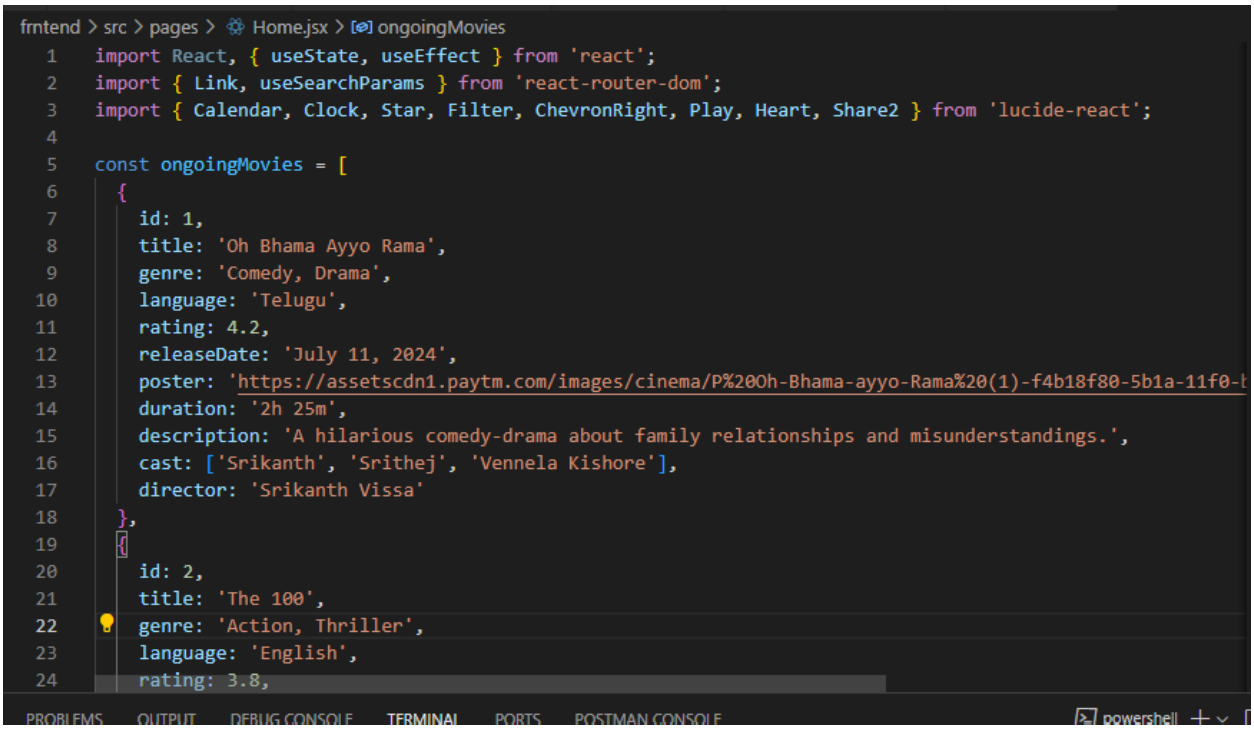
Step 1: API Services

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 promot



2. Movie Details:
● Fetch movie data by ID
● Display showtime availability

frntend > src > pages > MovieDetails.jsx > ...

```jsx
import React, { useState } from 'react';
import { useParams, Link, useNavigate } from 'react-router-dom';
import { Star, Calendar, Clock, Users, Play, Heart, Share2, X } from 'lucide-react';

// Trailer Modal Component
const TrailerModal = ({ isOpen, onClose, trailerUrl, movieTitle }) => {
  if (!isOpen) return null;

  return (
    <div className="fixed inset-0 bg-black bg-opacity-75 flex items-center justify-center z-50 p-4">
      <div className="bg-white rounded-lg max-w-4xl w-full max-h-[90vh] overflow-hidden">
        <div className="flex justify-between items-center p-4 border-b">
          <h3 className="text-lg font-semibold">{movieTitle} - Official Trailer</h3>
          <button
            onClick={onClose}
            className="p-2 hover:bg-gray-100 rounded-full transition-colors"
          >
            <X className="h-5 w-5" />
          </button>
        </div>
        <div className="aspect-video">
          <iframe
            width="100%"
            height="100%"
```

Seat Selection:

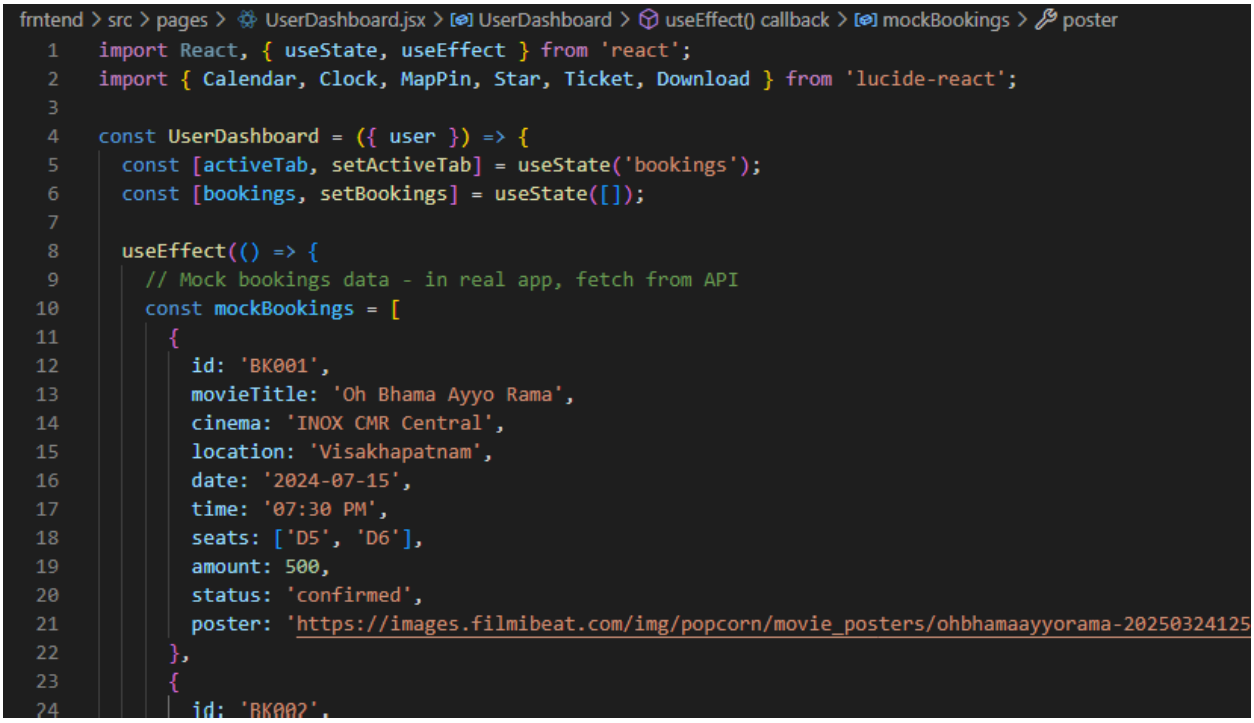
- Fetch available seats
- Track selected seats
- Calculate the total price

frntend > src > pages > SeatSelection.jsx > ...

```jsx
import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { ArrowLeft, Users, IndianRupee } from 'lucide-react';

const SeatSelection = ({ user }) => {
  const { movieId, cinemaId, showtime } = useParams();
  const navigate = useNavigate();

  const [selectedSeats, setSelectedSeats] = useState([]);
  const [loading, setLoading] = useState(false);

  // Check if user is logged in
  useEffect(() => {
    if (!user) {
      if (window.confirm('Please login to book tickets. Would you like to login now?')) {
        navigate('/login');
      } else {
        navigate(-1);
      }
    }
  }, [user, navigate]);

  // Mock seat data - in real app, this would come from API
  const seatLayout = {
```

Step 3: State Management

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



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('/');
```

Step 4: Run the Application

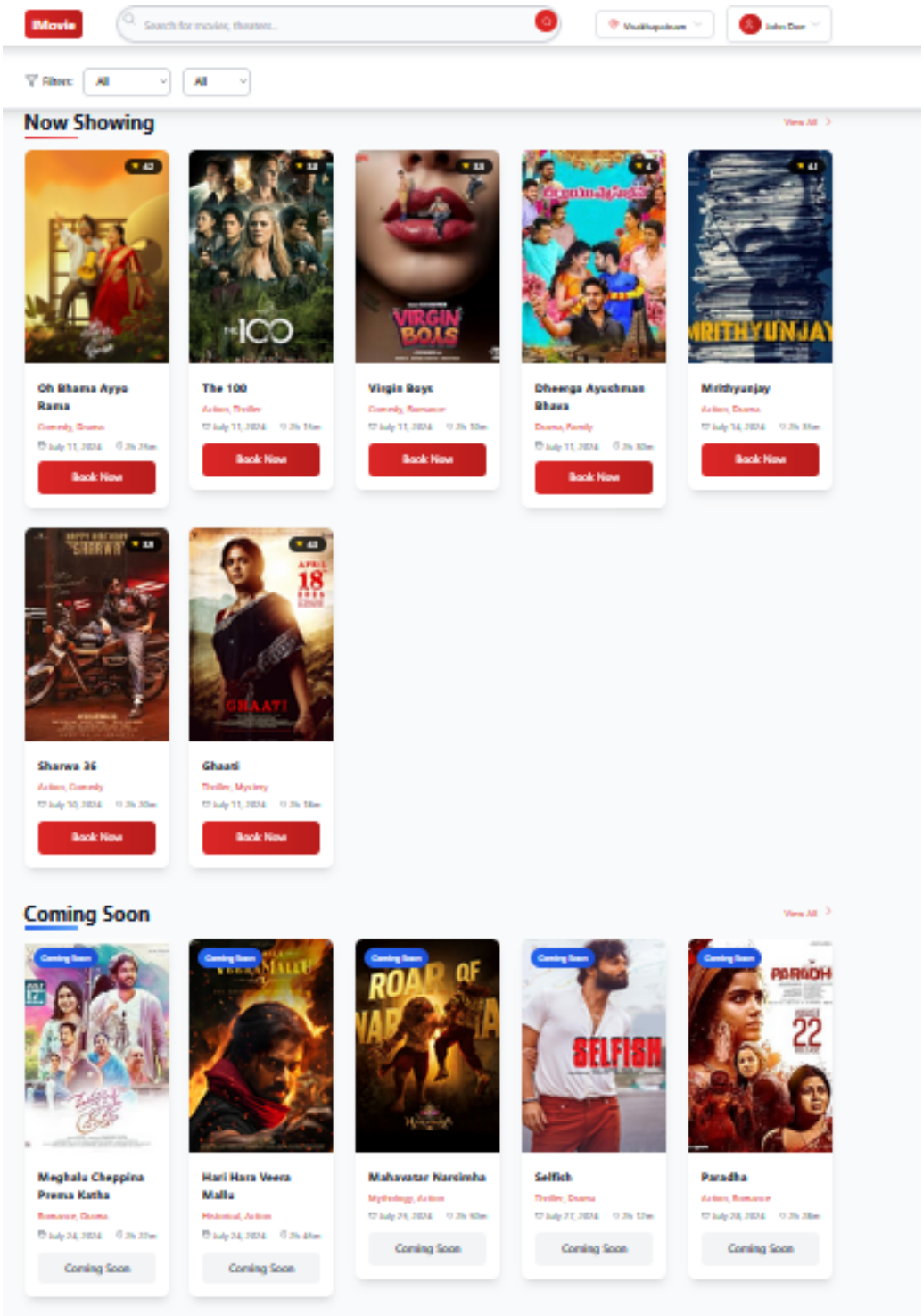
bash

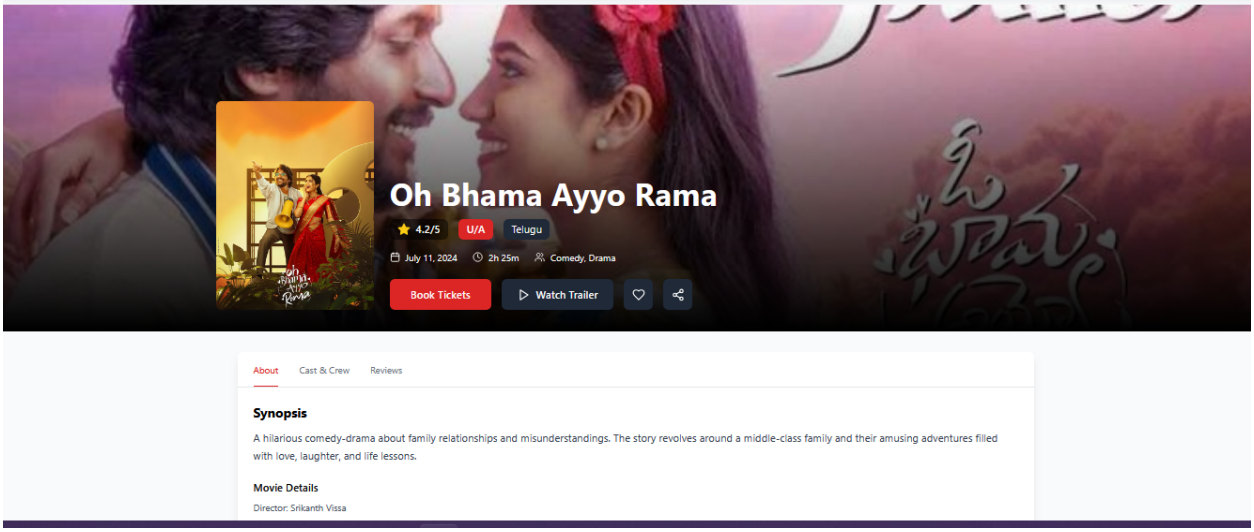
npm run dev

```
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
```

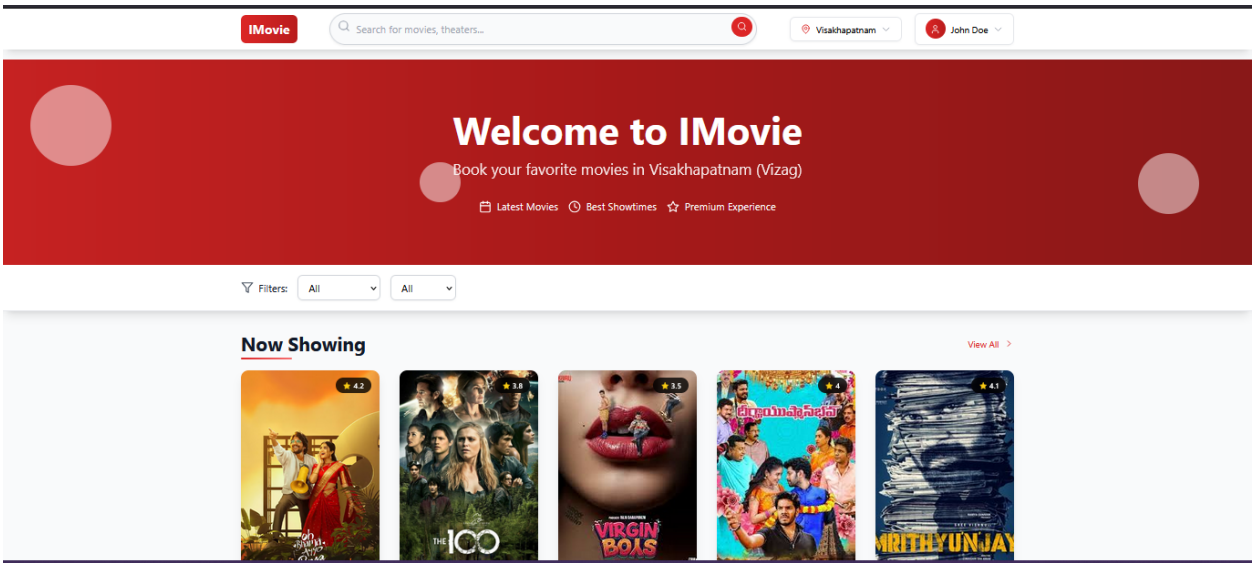
5.Project Implementation

Movie details page

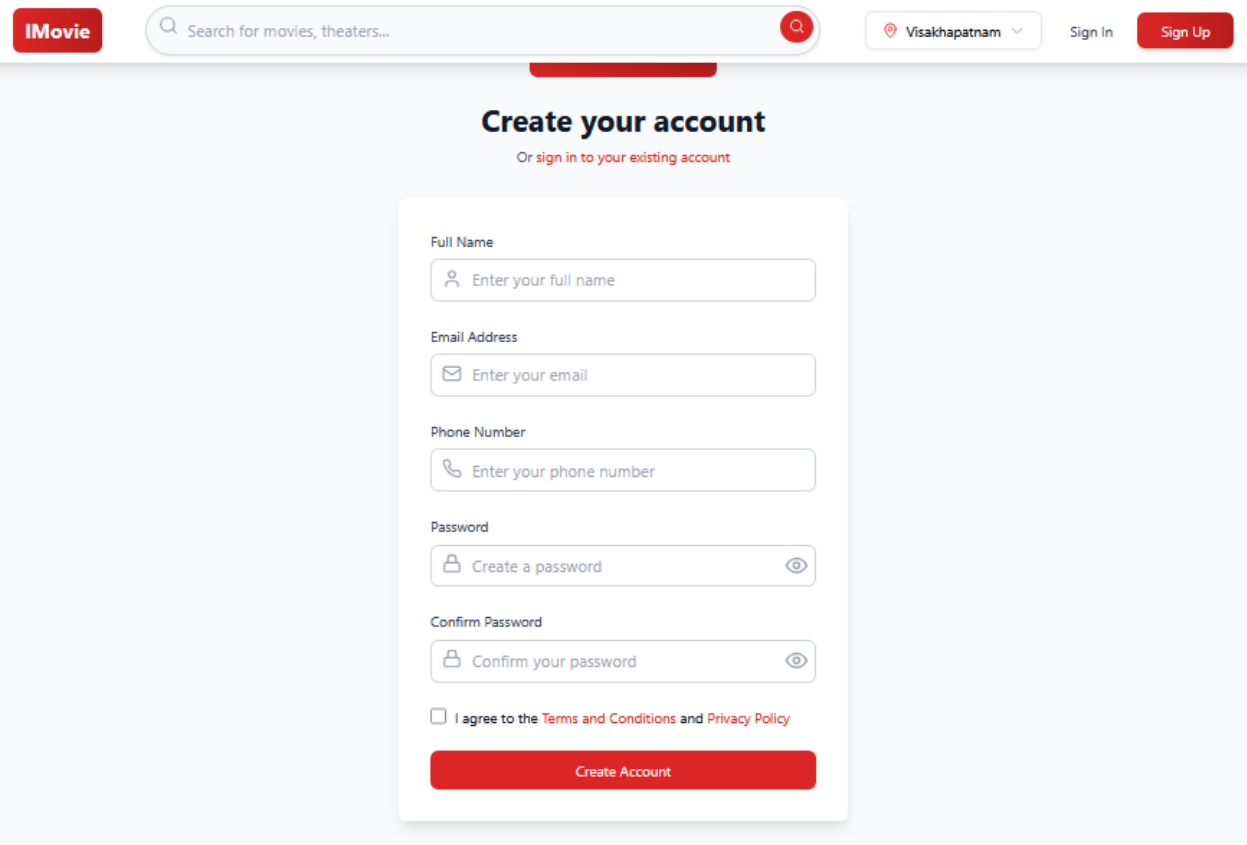



Oh Bhama Ayyo Rama
4.2/5
U/A
Telugu
July 11, 2024
2h 25m
Comedy, Drama
Book Tickets
Watch Trailer
About
Cast & Crew
Reviews
Synopsis
A hilarious comedy-drama about family relationships and misunderstandings. The story revolves around a middle-class family and their amusing adventures filled with love, laughter, and life lessons.
Movie Details
Director: Srikanth Vissa

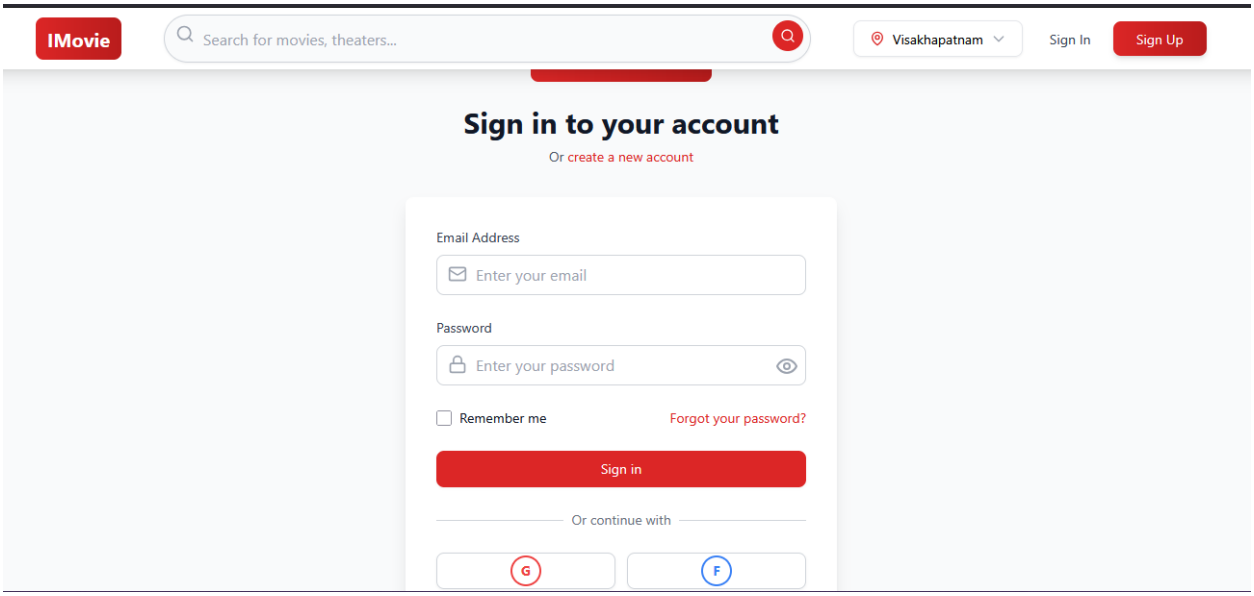
Home page


IMovie
Search for movies, theaters...
Visakhapatnam
John Doe
Welcome to IMovie
Book your favorite movies in Visakhapatnam (Vizag)
Latest Movies
Best Showtimes
Premium Experience
Filters:
All
All
Now Showing
View All
4.2
3.8
3.5
4
4.1
VIRGIN BOYS

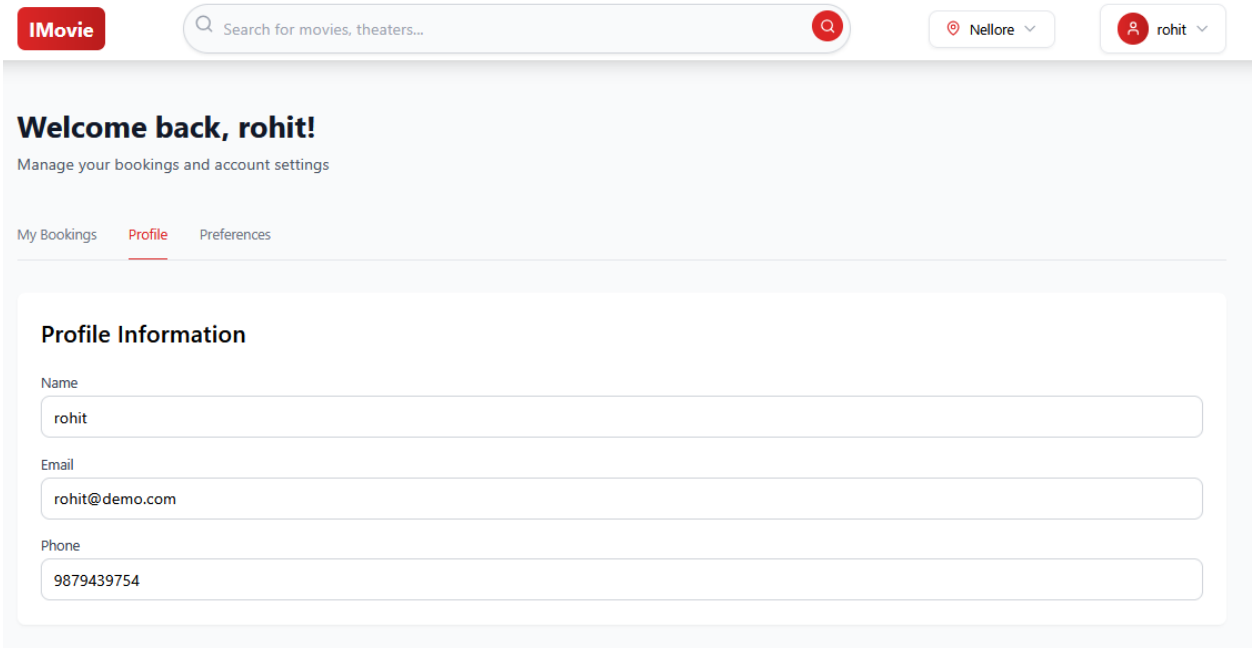
Sign up page


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



User Profile Page



IMovie

Search for movies, theaters...

Nellore

rohit

Welcome back, rohit!

Manage your bookings and account settings

My Bookings | Profile | Preferences

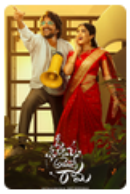
Upcoming Shows

No upcoming bookings

Ready to watch something amazing?

Browse Movies

Booking History



Oh Bhama Ayyo Rama

₹500

INOX CMR Central, Visakhapatnam

Confirmed

15/7/2024

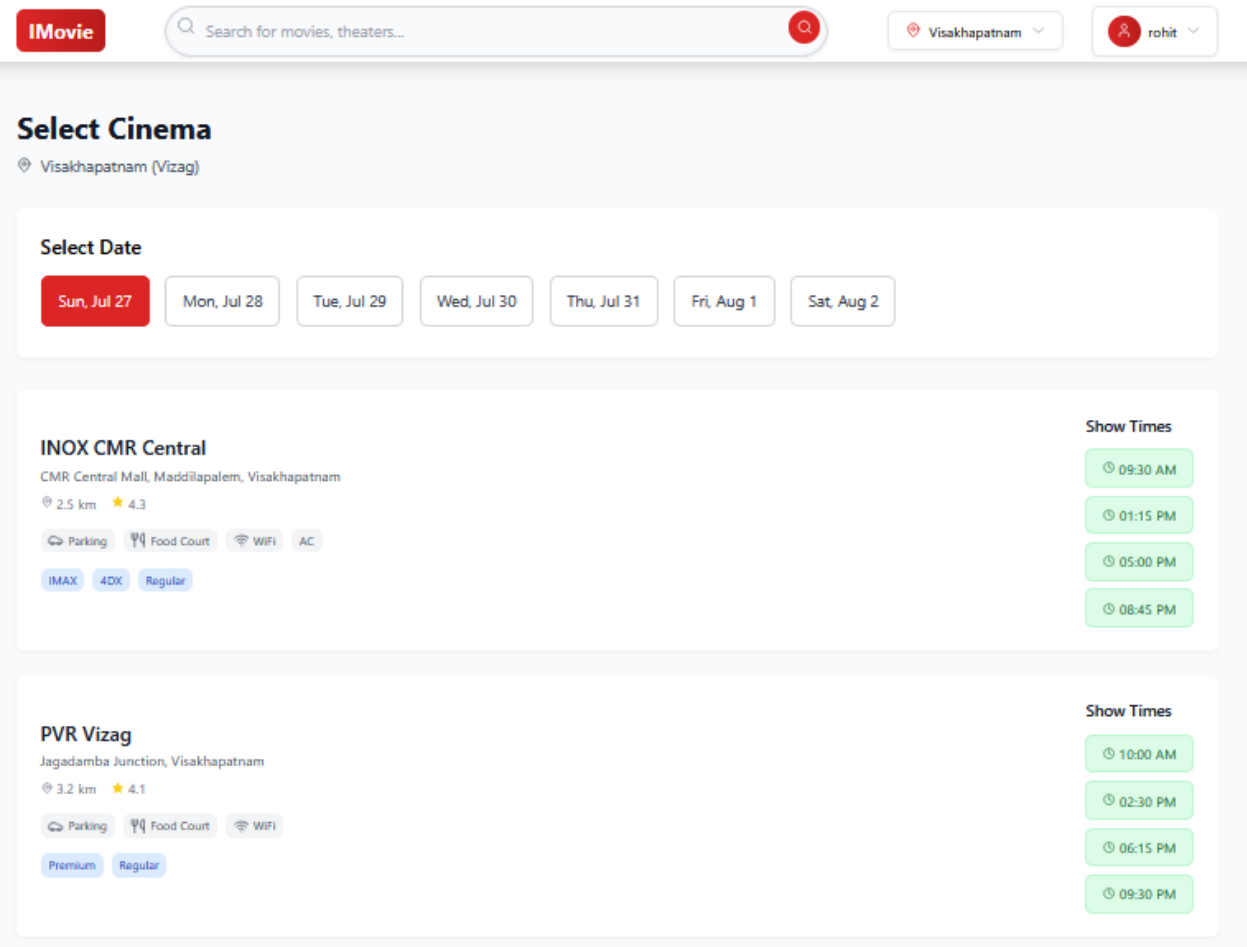
07:30 PM

Seats: D5, D6

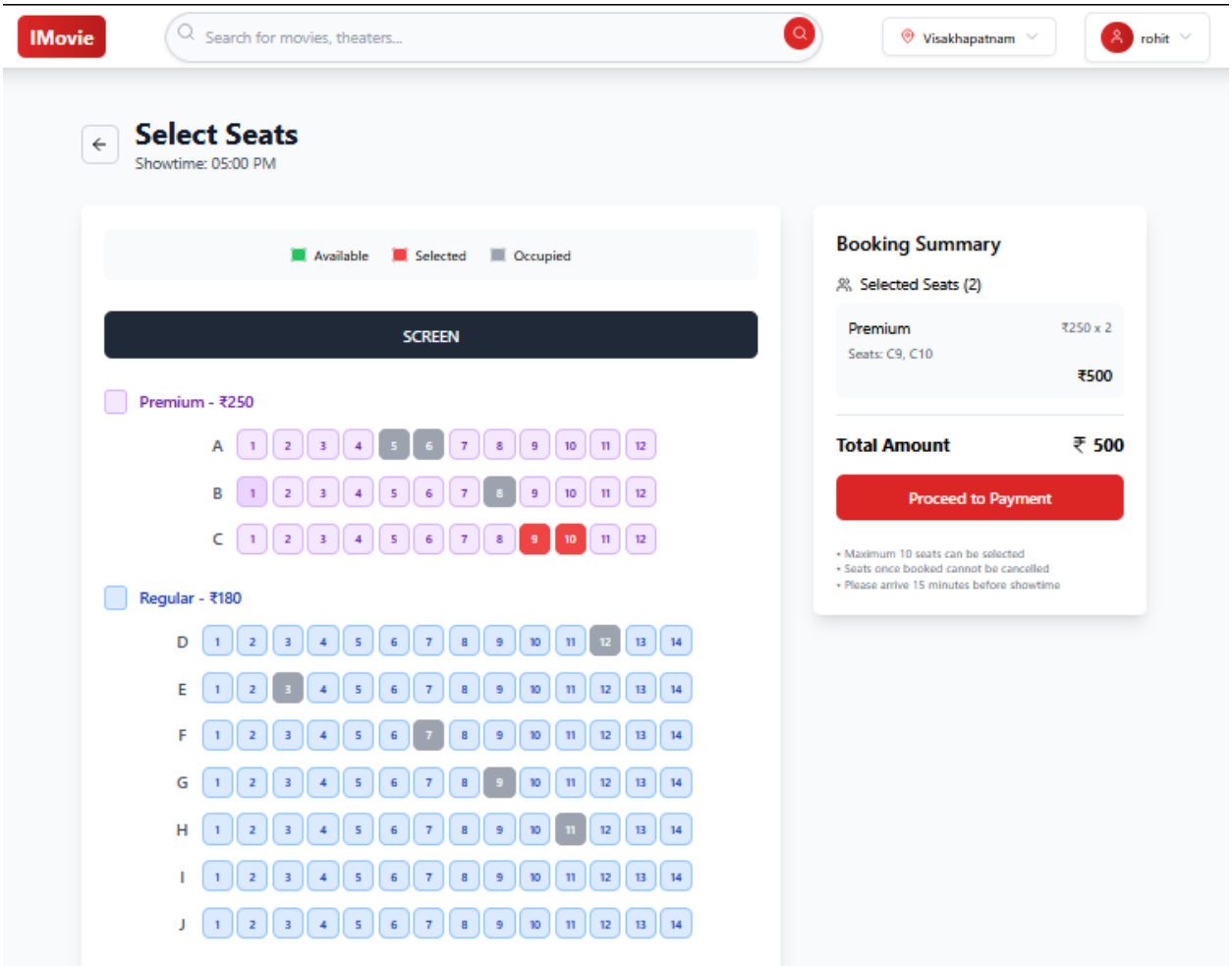
Download Ticket | Rate Movie

Booking ID: BK001

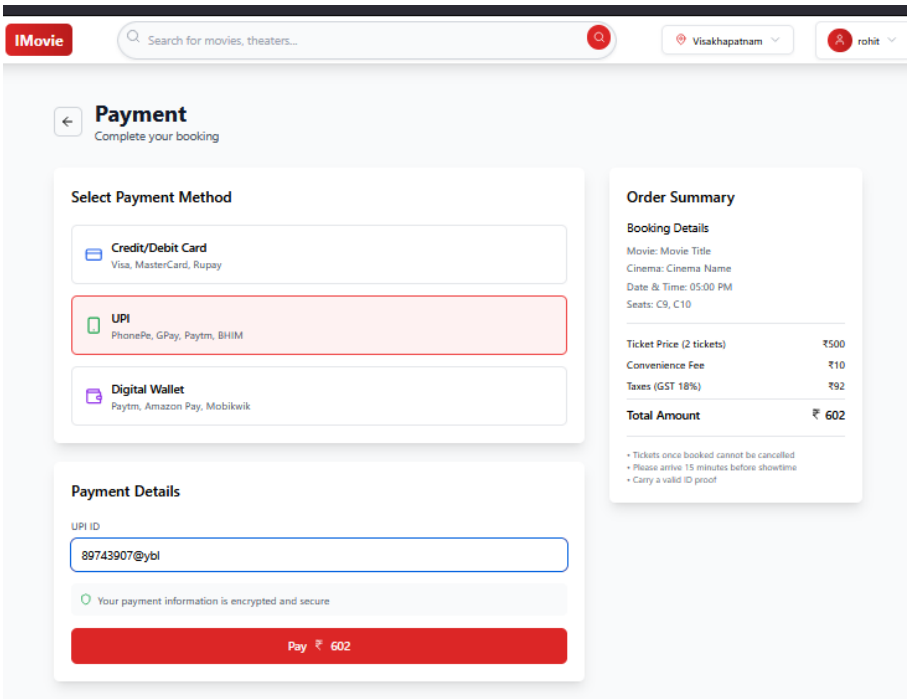
Bookings Page

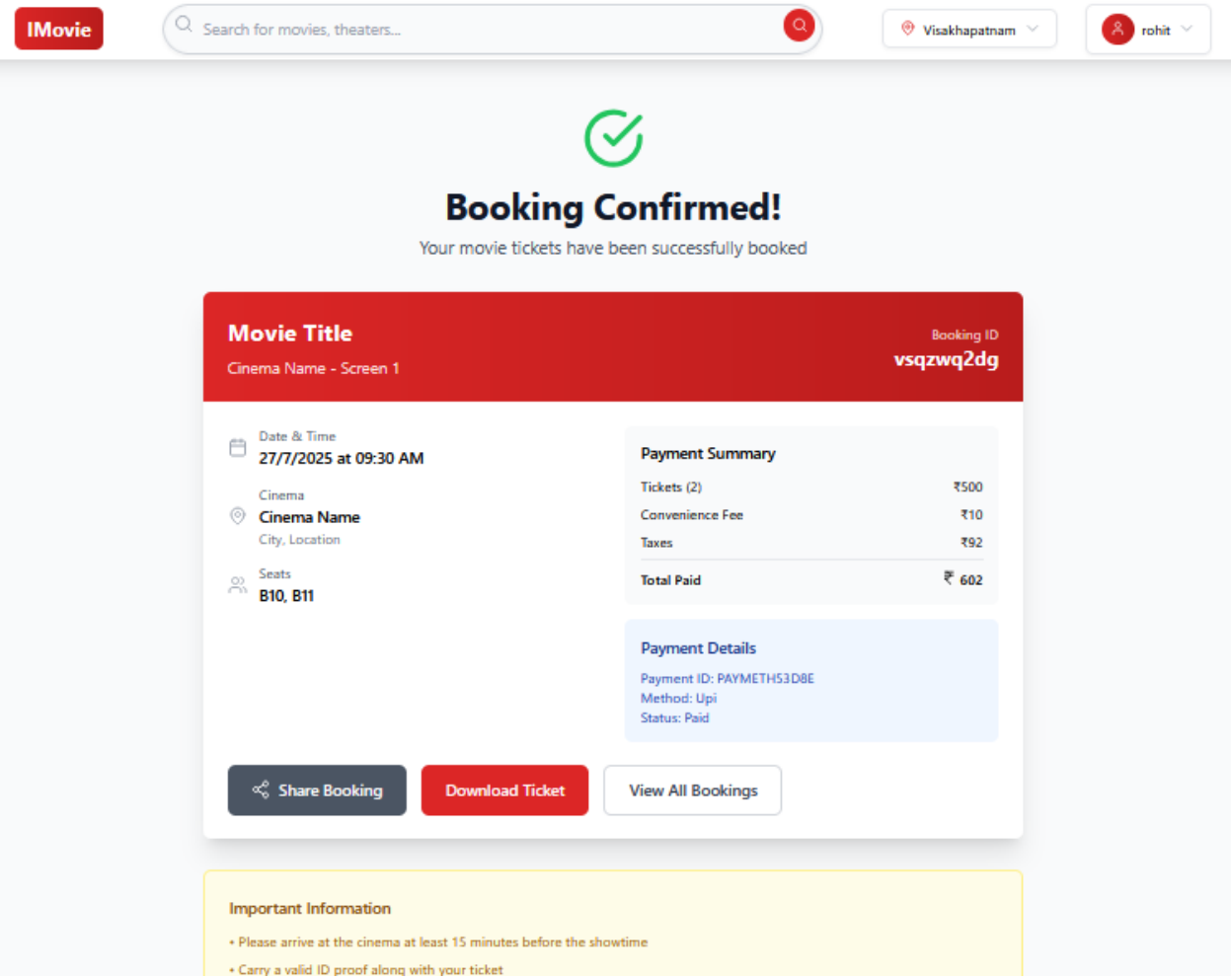


Seat Selection page

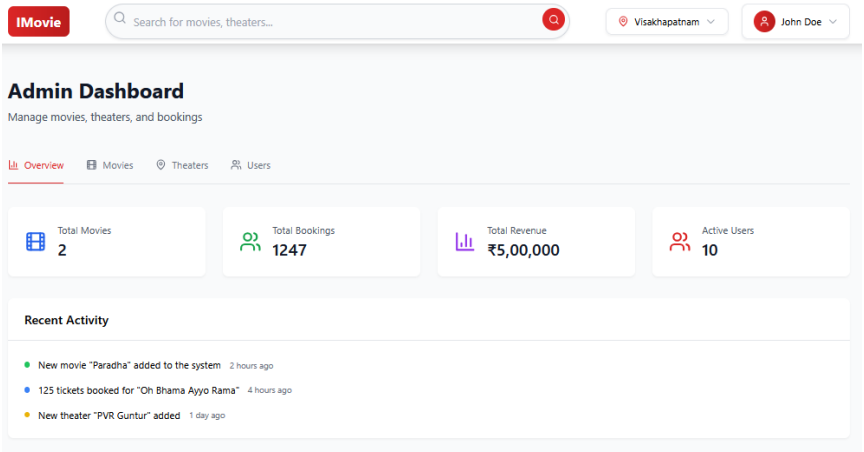


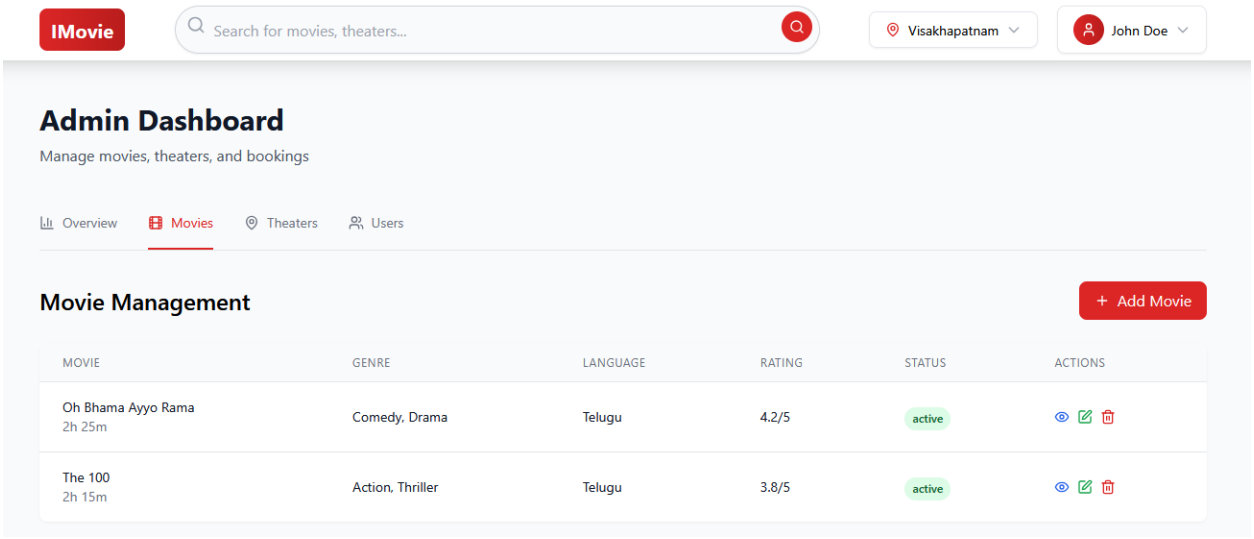
Ticket Payment Success page

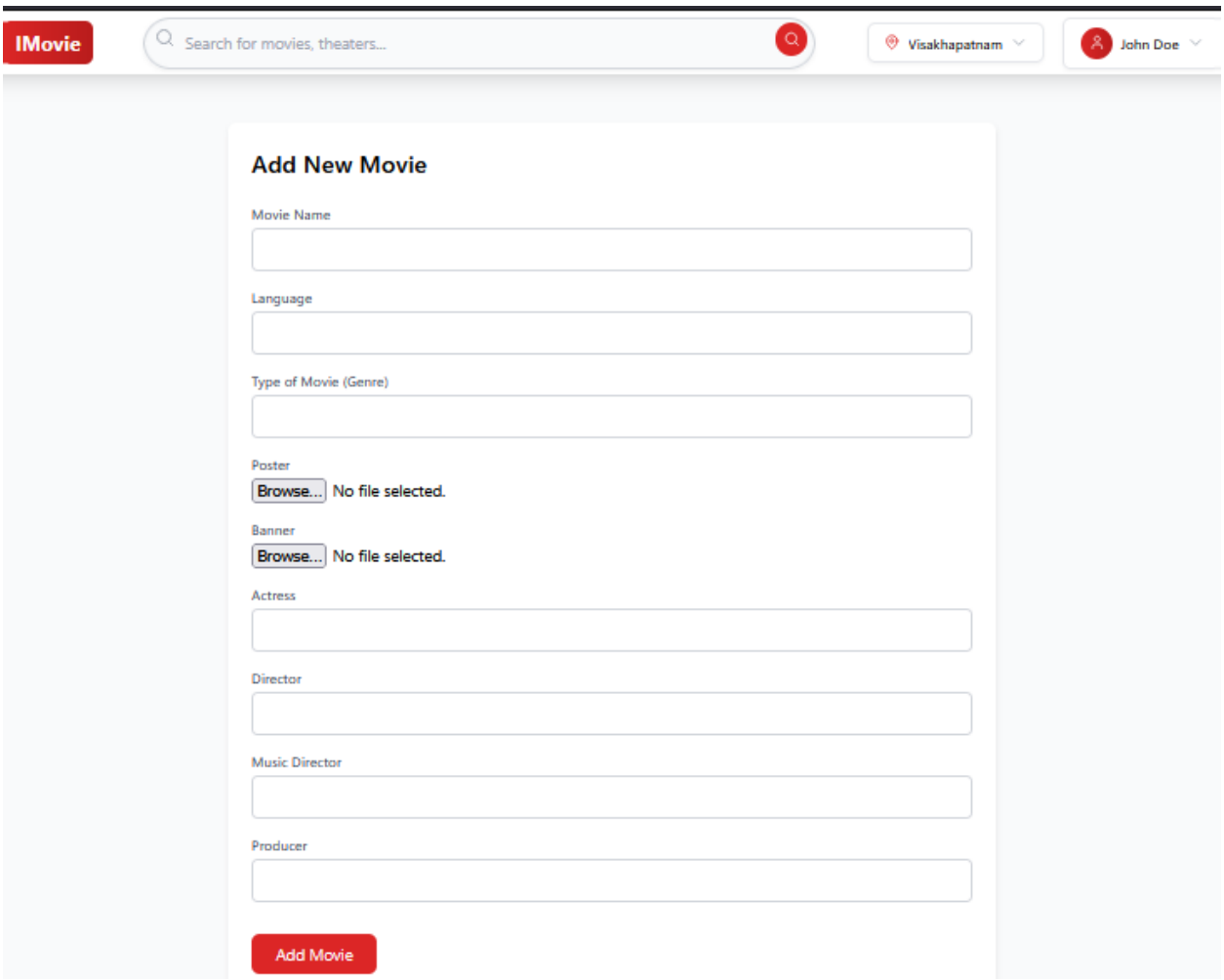


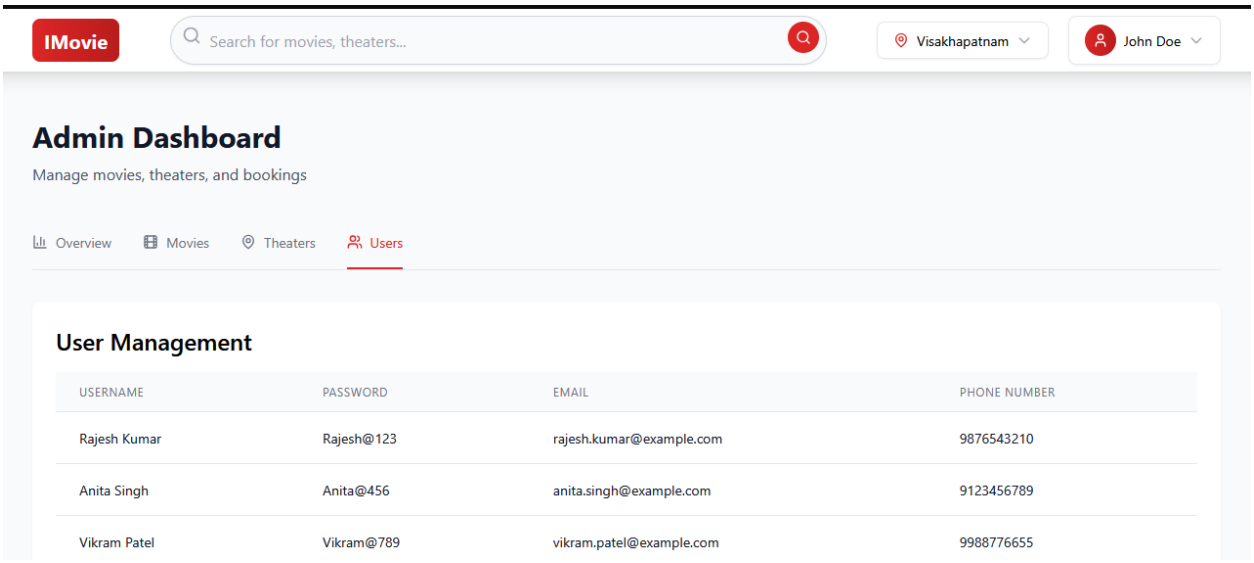


Admin DashBoard Page

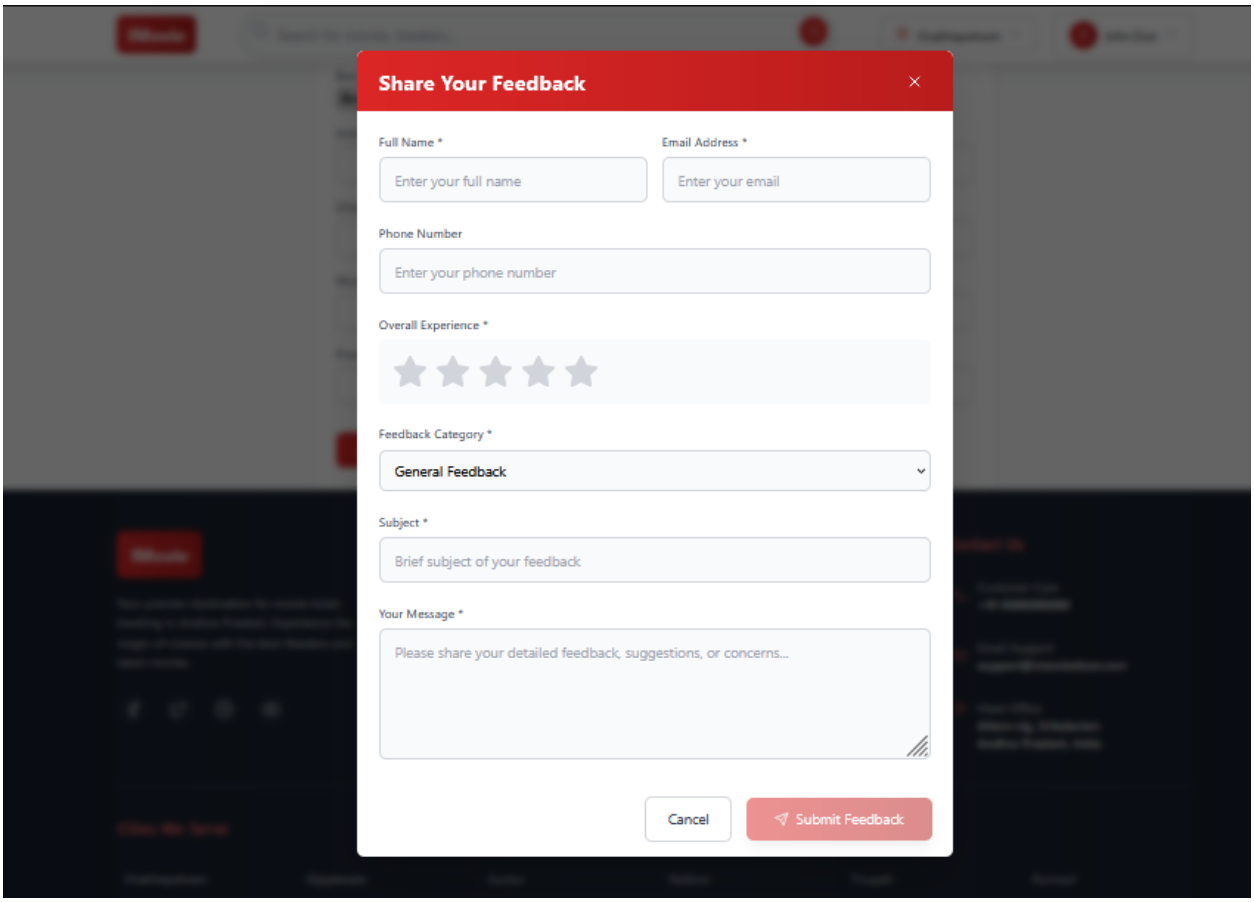



IMovie
Search for movies, theaters...
Visakhapatnam
John Doe
Admin Dashboard
Manage movies, theaters, and bookings
Overview
Movies
Theaters
Users
Movie Management
+ Add Movie
MOVIE
GENRE
LANGUAGE
RATING
STATUS
ACTIONS
Oh Bhama Ayyo Rama
2h 25m
Comedy, Drama
Telugu
4.2/5
active
The 100
2h 15m
Action, 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
Browse...
No file selected.
Banner
Browse...
No file selected.
Actress
Director
Music Director
Producer
Add Movie



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.