**SkyAirlines Flight Booking**

**Application ✈️**

A MERN Stack Project

Submitted By -

SURENDAR E

7th Semester, Computer Science Engineering

Date: 30/11/2024

Abstract

**SkyAirlines** is a modern flight booking web application designed to simplify the process of searching and booking flights for users. Developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), this project incorporates responsive design, advanced flight filtering, and user authentication via Google and Facebook. The platform offers a visual seat selection feature to enhance user experience, alongside a robust backend for seamless API interactions. The project demonstrates the integration of full-stack technologies to create a real-world, scalable solution.

Introduction

* Problem Statement:  
  Traditional methods of flight booking can be time-consuming and lack modern functionalities like advanced filtering and seat selection.
* Objective:  
  The aim of this project is to build a user-friendly, responsive flight booking web application that enhances user experience with features like intuitive search, visual seat selection, and secure user authentication.
* Scope:  
  The application is designed for desktop and mobile users, offering a smooth and consistent experience across platforms.

Project Overview

* Purpose:

The **SkyAirlines** Flight Booking Application aims to provide users with a seamless experience for searching and booking flights. It features a user-friendly interface, real-time flight availability, and a secure booking process.

* Features:
  + Flight search based on source, destination, and travel date.
  + Real-time seat selection with visual interface.
  + User authentication and secure login.
  + Responsive and dynamic design for all devices.
  + Comprehensive flight details, including departure and arrival times.
  + Loading animations for smooth transitions.

Architecture

* Frontend:

Built with React.js, using reusable components and conditional rendering for dynamic content (e.g., flight results, seat selection). Smooth animations enhance user experience.

A screenshot of a computer program

Description automatically generated

* Backend:

Developed using **Node.js** and **Express.js**, with APIs to handle user authentication, flight search, and booking operations.

A screenshot of a computer program

Description automatically generated

* Database:

Utilized MongoDB Atlas for storing flight details, user data, and booking information. A schema is designed to optimize queries and ensure data integrity.

A screenshot of a computer

Description automatically generated

Setup Instructions

* Prerequisites:
  + Node.js
  + MongoDB (local or Atlas)
  + npm (Node Package Manager)
* Installation:

1. Clone the repository: git clone [https://github.com/Surendar-13/Flight-Booking-App]

2. Navigate to the project folder: cd Flight-Booking-App

3. Install dependencies:

* Frontend: cd client && npm install
* Backend: cd server && npm install

4. Set up environment variables:

1. Frontend: Add API URLs to .env
2. Backend: Configure MongoDB URI and API keys in .env

Folder Structure

* Client (React Frontend):
  + src/components: Reusable React components like Header, Footer, FlightSearch, and FlightResult
  + src/pages: Pages like Home, Login, and Bookings
  + src/assets: Images, icons, and CSS files
* Server (Node.js Backend):
* routes/: API endpoints for authentication, flight search, and booking
* models/: MongoDB schemas for flights and users
* controllers/: Logic for handling requests and responses

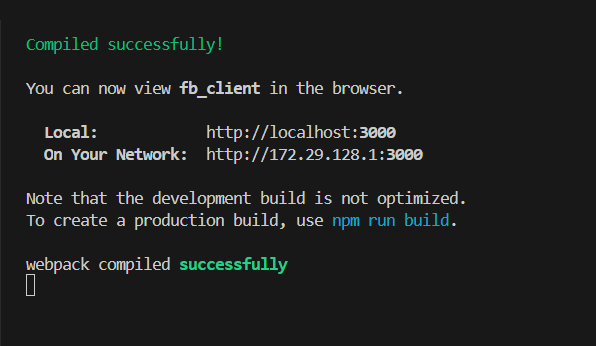
A screenshot of a computer program

Description automatically generated

Running the Application

* Frontend:

Run npm start in the client directory.



* Backend:

Run npm start in the server directory.

A screen shot of a computer

Description automatically generated

API Documentation

1. GET /api/flights:
   * Description: Fetch available flights based on source, destination, and date.
   * Request Parameters**:**
     + source (string, required): The departure city (e.g., "Delhi").
     + destination (string, required): The arrival city (e.g., "Mumbai").
     + date (string, required): The travel date in YYYY-MM-DD format.
   * Sample Request URL:

GET http://localhost:5001/api/flights/search

* + Response:
    - 200 OK:

{

"flights": [

{

"flightId": "12345",

"flightNo": "SK123",

"airline": "Sky Airlines",

"departure": "10:00",

"arrival": "12:00",

"duration": "2h",

"seatsAvailable": 50,

"price": 5000

}

] }

* + 404 Not Found:

{

"error": "No flights found for the given criteria."

}

1. GET /api/seats/:flightId
   * Description: Fetch seat availability for a specific flight
   * Request Parameters:

* flightId (string, required): The unique ID of the flight.

Sample Request URL:

* GET /api/seats/12345
* Response:
  + 200 OK:

{

"flightId": "12345",

"seats": [

{ "seatNo": "1A", "available": true },

{ "seatNo": "1B", "available": false }

]

}

* 404 Not Found:

{

"error": "Flight not found."

}

1. POST /api/book

* Description: Book selected seats for a specific flight.
* Request Body:

{

"userId": "98765",

"flightId": "12345",

"seats": ["1A", "1B"]

}

* Response:
  + 201 Created:

{

"message": "Booking successful!",

"bookingId": "booking123"

}

* + 400 Bad Request:

{

"error": "Seat 1B is already booked."

}

1. POST /api/login

* **Description:** Authenticate the user and return a JWT token.
* Request Body:

{

"email": "user@example.com",

"password": "password123"

}

* Response:
  + 200 OK:

{

"message": "Login successful.",

"token": "jwt.token.string"

}

* + 401 Unauthorized:

{

"error": "Invalid email or password."

}

1. POST /api/register

* Description: Register a new user account.
* Request Body:

{

"name": "John Doe",

"email": "john.doe@example.com",

"password": "password123"

}

* Response:
  + 201 Created:

{

"message": "Registration successful!",

"userId": "user123"

}

* + 400 Bad Request:

{

"error": "Email already exists."

}

1. POST /api/logout

* Description: Log the user out by invalidating the JWT token.
* Request Headers:
  + Authorization (string, required): The Bearer token.
* Response:
  + 200 OK:

{

"message": "Logout successful."

}

* + 401 Unauthorized:

{

"error": "Invalid or expired token."

}

1. GET /api/bookings/:userId

* Description: Retrieve all bookings for a specific user.
* Request Parameters:
  + userId (string, required): The ID of the user.
* Sample Request URL:
* GET /api/bookings/98765
* Response
  + 200 OK:

{

"userId": "98765",

"bookings": [

{

"bookingId": "booking123",

"flightNo": "SK123",

"seats": ["1A", "1B"],

"date": "2024-12-01",

"price": 10000

}

]

}

* + 404 Not Found:

{

"error": "No bookings found for this user."

}

1. DELETE /api/bookings/:bookingId

* Description: Cancel a specific booking.
* Request Parameters:
  + bookingId (string, required): The ID of the booking.
* Sample Request URL:
* DELETE /api/bookings/booking123
* Response:
  + 200 OK:

{

"message": "Booking canceled successfully."

}

* + 404 Not Found:

{

"error": "Booking not found."

}

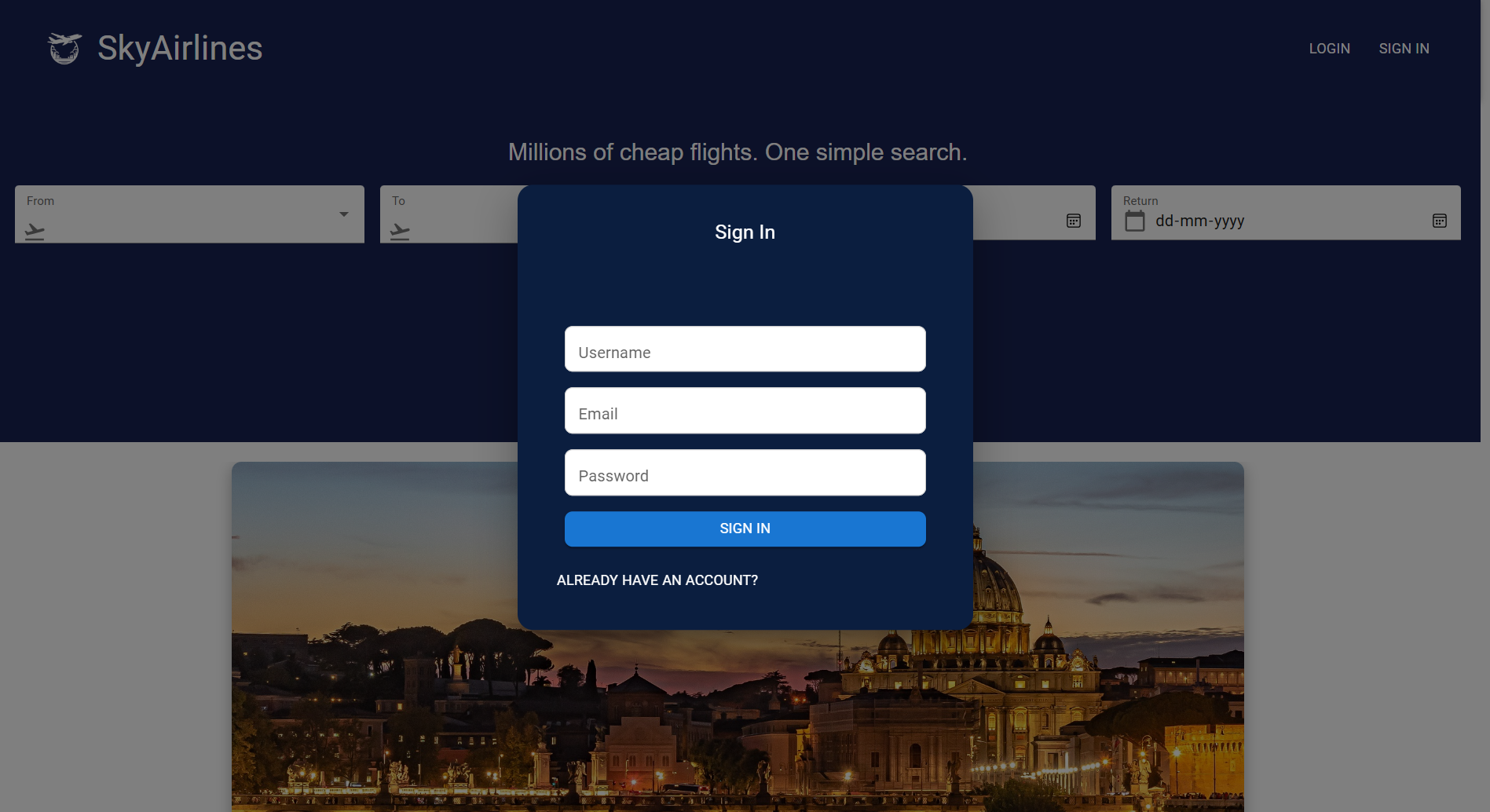
Error Handling Format

* 400 Bad Request: Returned when request parameters are missing or invalid.
* 401 Unauthorized: Returned for invalid or missing authentication tokens.
* 404 Not Found: Returned when the requested resource does not exist.
* 500 Internal Server Error: Generic error for unexpected server issues.

Authentication

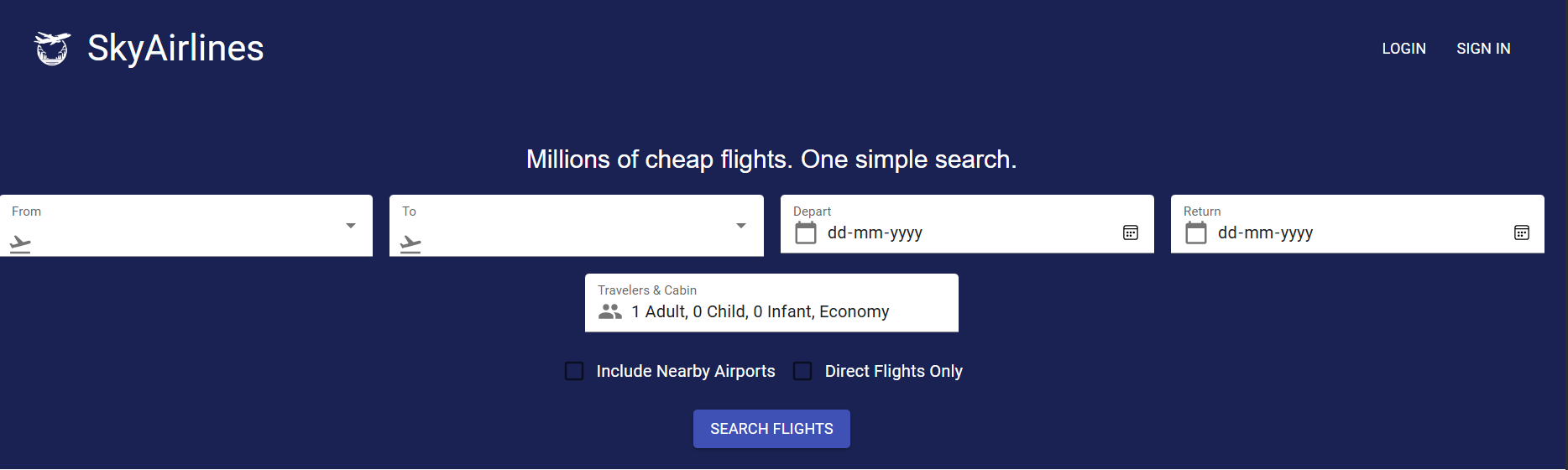
Methodology:

* + Implemented JWT-based authentication.
  + Tokens are generated at login and stored in secure cookies.



User Interface

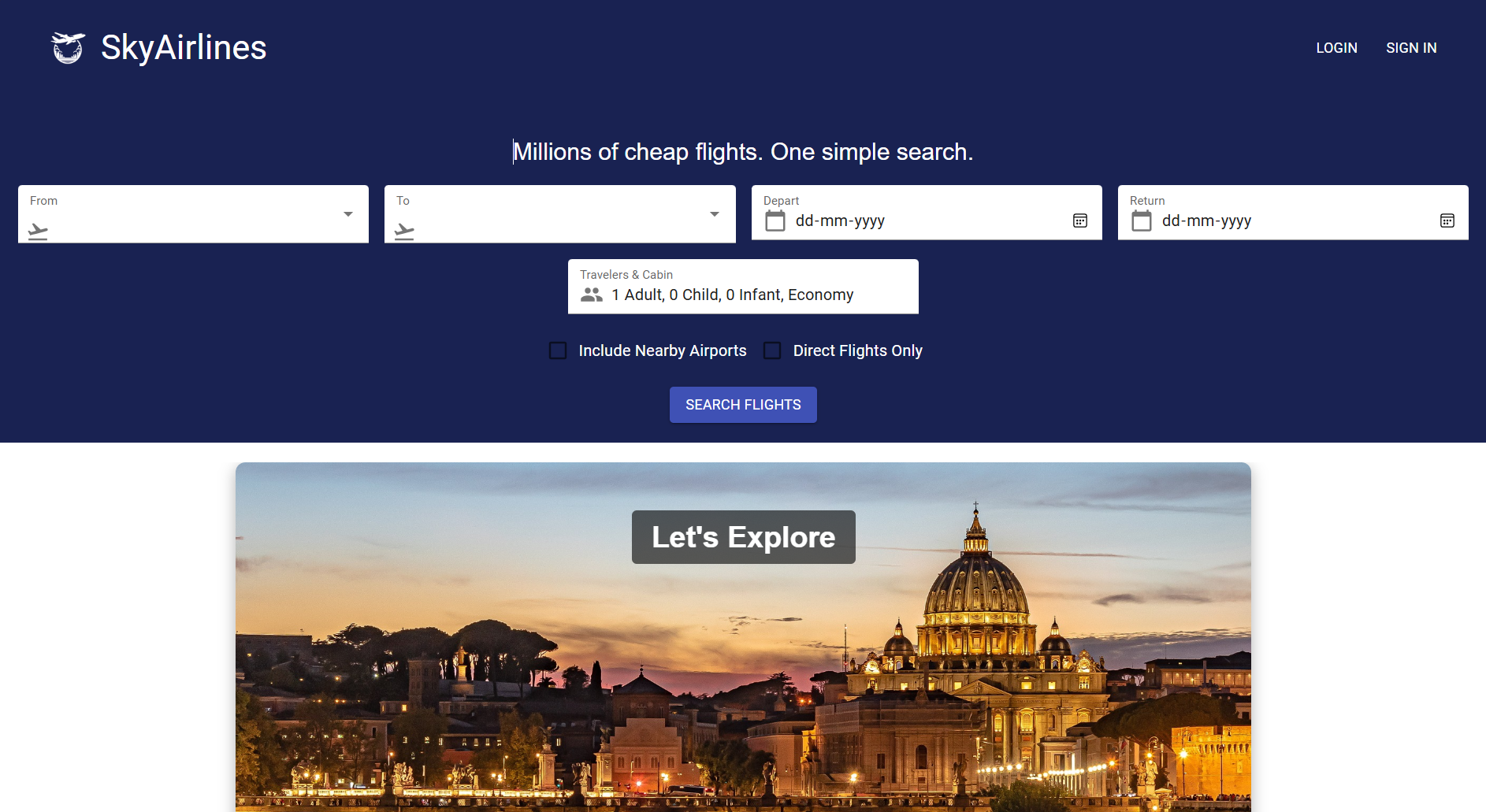
* Highlights:
  + Intuitive flight search bar
  + Interactive seat selection with conditional rendering of the "Book Now" button
  + Mobile-friendly design with responsive element



Results and Screenshots

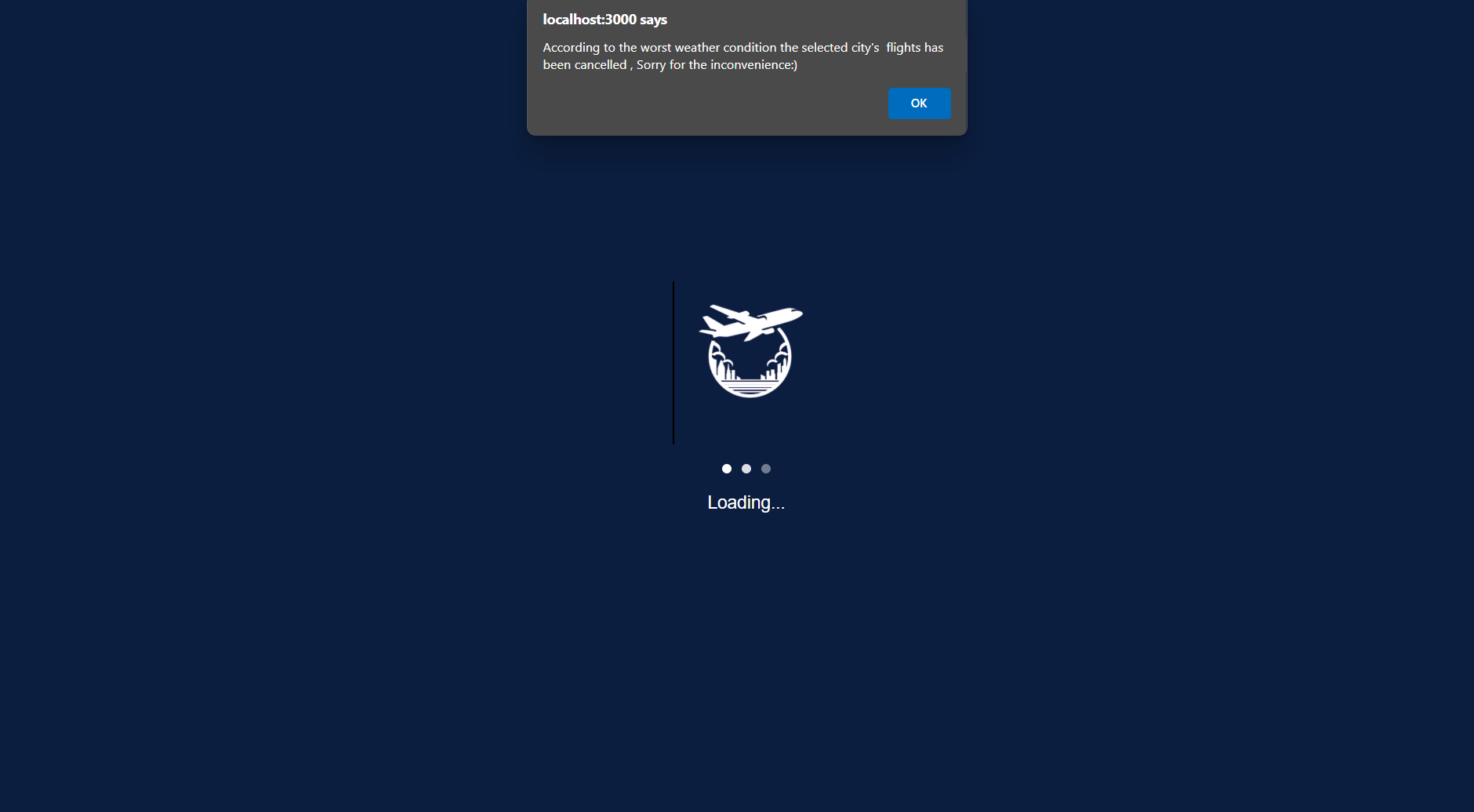
* Homepage:

Displays flight search options.



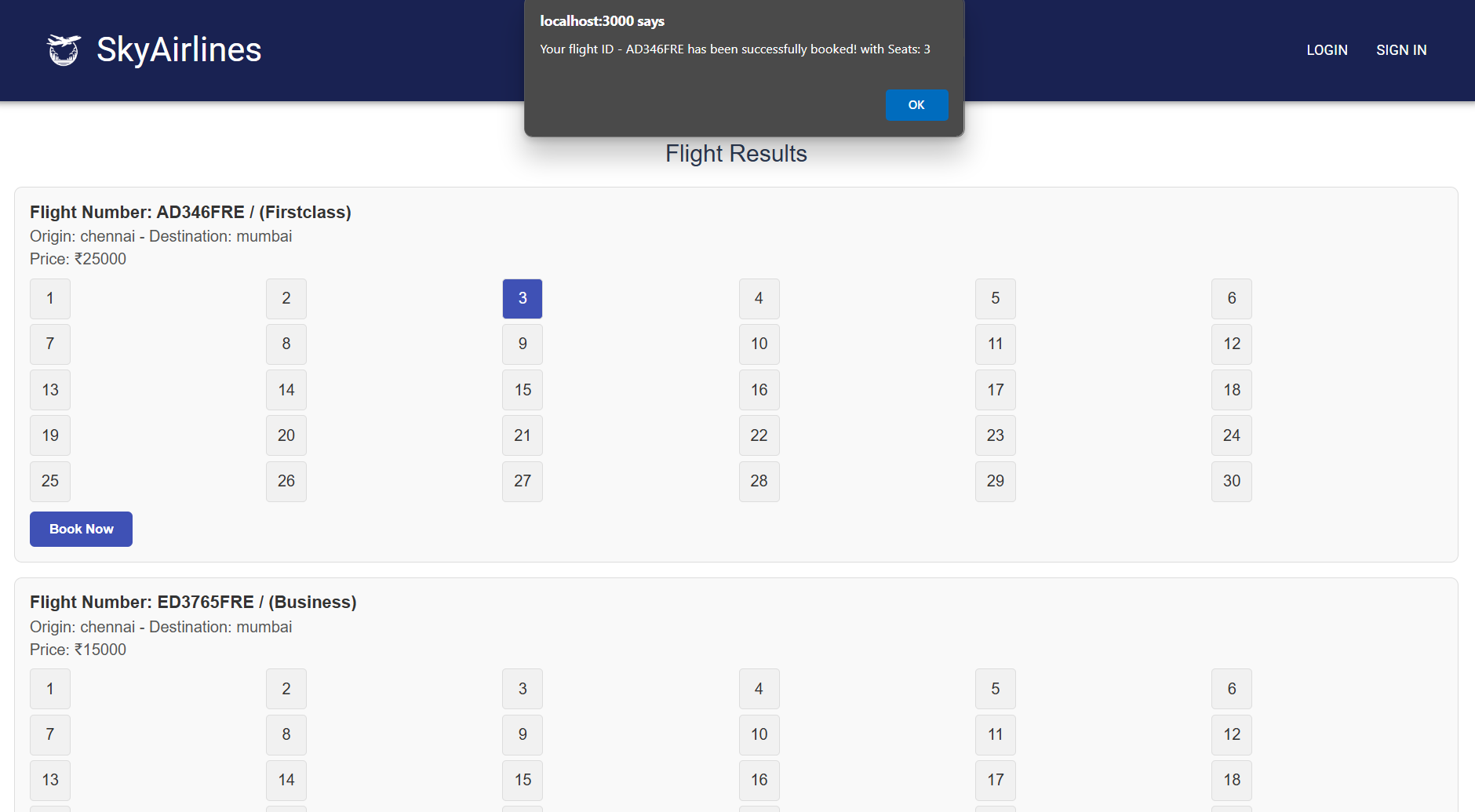
* Search Results:

Shows available flights with filters for sorting.



* Seat Selection:

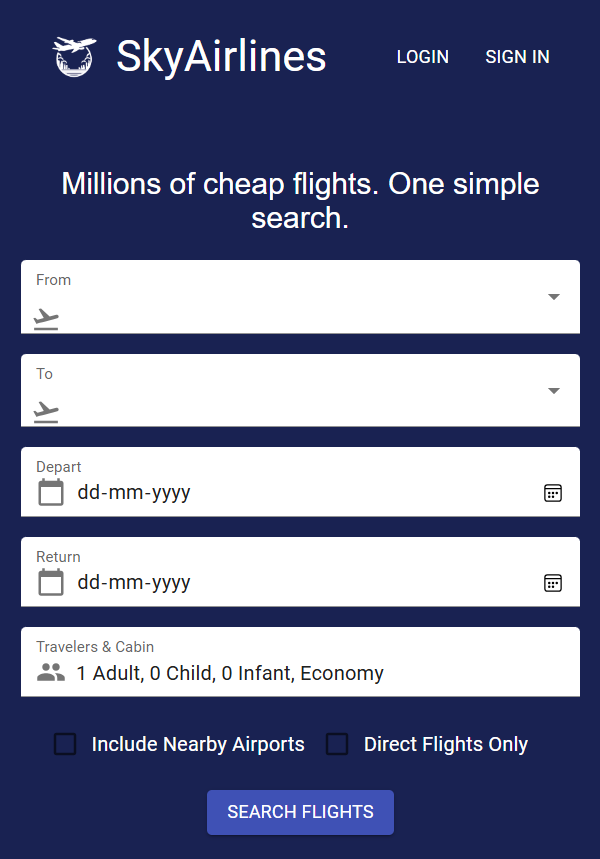
Interactive graphical seat map.



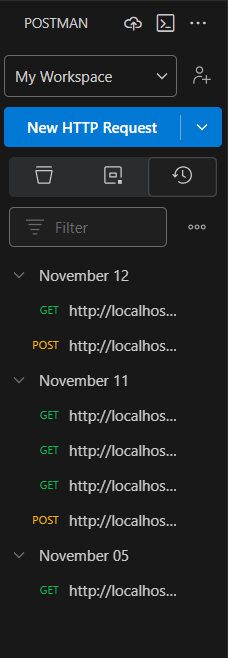
* Design Philosophy:
* Mobile-first design for accessibility.
* Colors and typography selected for clarity and aesthetics.

Testing and Validation

* Frontend Testing:
  + Manually validated functionality on Chrome and mobile browsers.



* Backend Testing:
  + Used Postman to test API responses for various scenarios.



* Database Validation:
  + Verified MongoDB collections to ensure data integrity.
* Tools:
  + Jest for unit testing React components
  + Postman for API testing
* Strategy:
  + Unit testing for critical components (e.g., FlightResult)
  + API endpoint testing for correctness and performance

Demo Link 🔗

https://drive.google.com/file/d/1\_-7DHOBosKSYRsnB2vEWqxCiw214SbHh/view?pli=1

Known Issues

* Example Issue: Sometimes the seat selection interface takes longer to load under heavy traffic.
* Workaround: Optimizing the rendering logic for the FlightResult component is under progress.

Conclusion

The project demonstrates the practical application of the MERN stack for building a functional web application. Key takeaways include gaining proficiency in API integration, MongoDB, and responsive design.



A plane flying in the sky

Description automatically generated

Future Enhancements

* Payment gateway integration for booking confirmation.
* User Authentication via Social platforms like (Google, Facebook, etc…)
* Advanced search filters (e.g., airline, price).
* Progressive Web App (PWA) version for offline use.

References

* React.js Official Documentation
* MongoDB Atlas Documentation
* Material-UI Library
* Tutorials on RESTful API Design
* Skyscanner Website

MongoDB Completion Certificate

A certificate of completion with a signature

Description automatically generated

THANK YOU