

# **PROJECT REPORT**

**Flight finder: Navigating your air travel options.**

**Team Name – MS Flights**

## **TEAM MEMBERS:**

**MADHU HASITHA KONDETI - 23MH1A0514**

**PYDIMALLA UDAYKUMAR -24P35A0546**

## TABLE OF CONTENT

SNO	Content	Page no
1.	Table of content	0-1
2.	Introduction	2-3
3.	Technical architecture	3-5
4.	Prerequisites	6-8
5.	Project structure	9-11
6.	Project flow	12-16
7.	Project implementation	17-26
8.	Challenge faced	27-28
9.	Future scope	29-30
10.	Conclusion	31
11.	References	32

# **Flight finder: Navigating your air travel options.**

## **INTRODUCTION**

In today's fast-paced world, travel has become an essential part of our lives, whether it's for business, leisure, or personal reasons. With the advent of technology, booking flights has become more accessible and convenient than ever before, thanks to flight booking apps. A flight booking app is a mobile application that allows users to search, compare, and book flights easily from the comfort of their smartphones or tablets. The primary purpose of a flight booking app is to streamline the process of planning and booking air travel. These apps provide users with a range of features and functionalities that simplify the entire journey, from searching for flights to managing bookings and receiving travel updates.

### **Description:**

This Flight Booking APP is the ultimate digital platform designed to revolutionize the way you book flight tickets. With this app your flight travel experience will be elevated to new heights of convenience and efficiency. Our user-friendly web app empowers travelers to effortlessly discover, explore, and reserve flight tickets based on their unique preferences. Whether you're a frequent commuter or an occasional traveler, finding the perfect flight journey has never been easier.

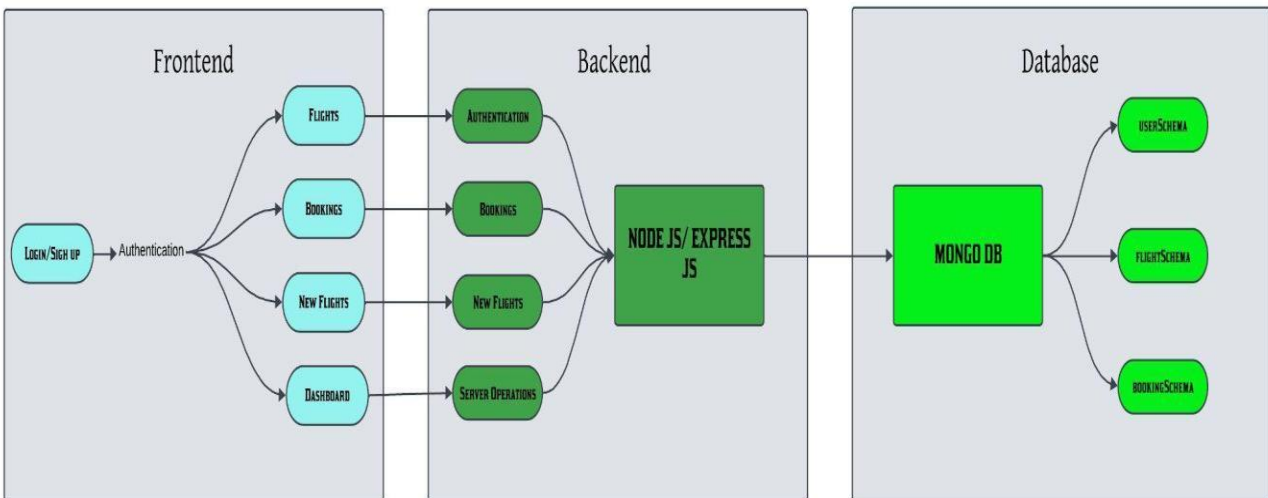
This successful flight booking app combines a user-friendly interface, efficient search and booking capabilities, personalized features, robust security measures, reliable performance, and continuous improvement based on user feedback.

### **Scenario**

- John, a frequent traveler and business professional, needs to book a flight for an upcoming conference in Paris. He prefers using a flight booking app for its convenience and features.
- John opens the flight booking app on his smartphone and enters his travel details for Departure as New York City, Destination as Paris, Date of Departure on April 10th and return on April 15th and Class as Business class, Number of passengers as 1

- The app quickly retrieves available flight options based on John's preferences. He sees a range of choices from different airlines, including direct flights and those with layovers. The results show details such as price, airline, duration, and departure times.
- Using the app's filters, John narrows down the options to show only direct flights with convenient departure times. He also selects his preferred airline based on past experiences and loyalty programs.
- After choosing a flight, John proceeds to select his seat in the business class cabin. The app provides a seat map with available seats highlighted, allowing John to pick a window seat with extra legroom.
- John securely enters his payment information using the app's integrated payment gateway. The app processes the payment and generates a booking confirmation with his e-ticket and itinerary details.
- This scenario demonstrates how a flight booking app streamlines the entire travel process for users like John, offering convenience, customization, and real-time assistance throughout their journey.

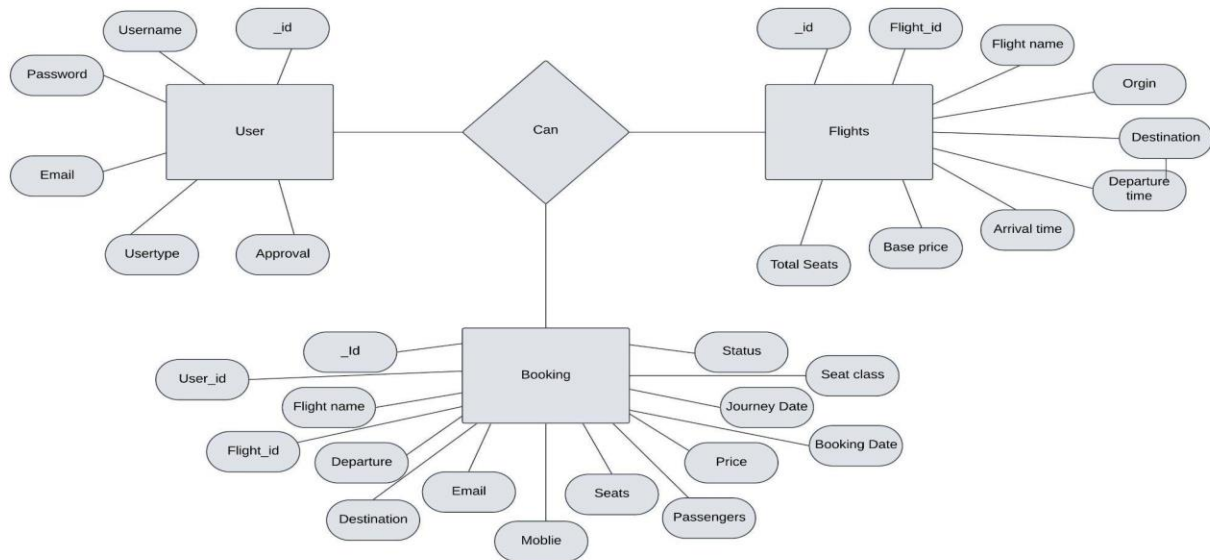
# TECHNICAL ARCHITECTURE



In this architecture diagram:

- The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Flight Search, and Booking.
- The backend is represented by the "Backend" section, consisting of API endpoints for Users, Flights, Admin and Bookings. It also includes Admin Authentication and an Admin Dashboard.
- The Database section represents the database that stores collections for Users, Flights, and Flight Bookings.

## ER DIAGRAM



The flight booking ER-diagram represents the entities and relationships involved in a flight booking system. It illustrates how users, bookings, flights, passengers, and payments are interconnected. Here is a breakdown of the entities and their relationships:

**USER:** Represents the individuals or entities who book flights. A customer can place multiple bookings and make multiple payments.

**BOOKING:** Represents a specific flight booking made by a customer. A booking includes a particular flight details and passenger information. A customer can have multiple bookings.

**FLIGHT:** Represents a flight that is available for booking. Here, the details of flight will be provided and the users can book them as much as the available seats.

**ADMIN:** Admin is responsible for all the backend activities. Admin manages all the bookings, adds new flights etc.

## PREREQUISITES:

To develop a full-stack flight booking app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

**Node.js and npm:** Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

**MongoDB:** Set up a MongoDB database to store hotel and booking information. Install MongoDB locally using a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

**Express.js:** Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command:  
`npm install express`

**React.js:** React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

**HTML, CSS, and JavaScript:** Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

**Database Connectivity:** Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

**Front-end Framework:** Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

**Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at:  
<https://git-scm.com/downloads>

**Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

**To Connect the Database with Node JS go through the below provided link:**

- <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb>

**To run the existing Flight Booking App project downloaded from GitHub:**

Follow below steps:

**Clone the repository:**

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the e-commerce app.
- Execute the following command to clone the repository:

**Git clone:** [Flightfinder - Navigating Your Air Travel Options](#)

**Install Dependencies:**

- Navigate into the cloned repository directory:  
**cd Flight-Booking-App-MERN**
- Install the required dependencies by running the following command:  
**npm install**

**Start the Development Server:**

- To start the development server, execute the following command:  
**npm run dev or npm run start**
- The e-commerce app will be accessible at <http://localhost:3000> by default. You can change the port configuration in the .env file if needed.

**Access the App:**

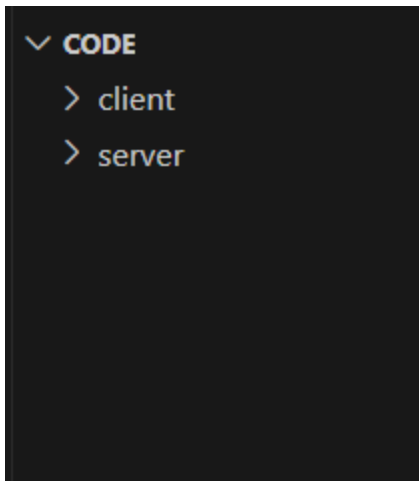
- Open your web browser and navigate to <http://localhost:3000>
- You should see the flight booking app's homepage, indicating that the installation and the setup was successful.



You have successfully installed and set up the flight booking app on your local machine. You can now proceed with further customization, development, and testing as needed.

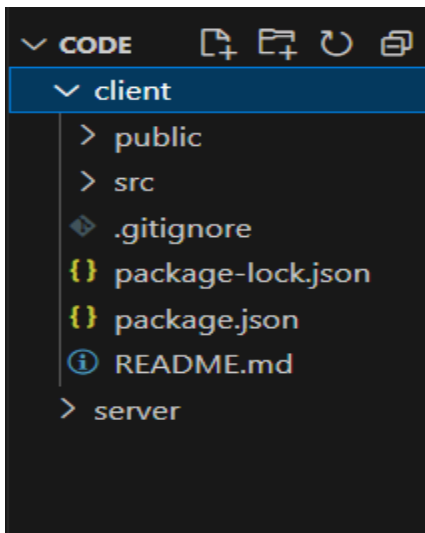
## PROJECT STRUCTURE:

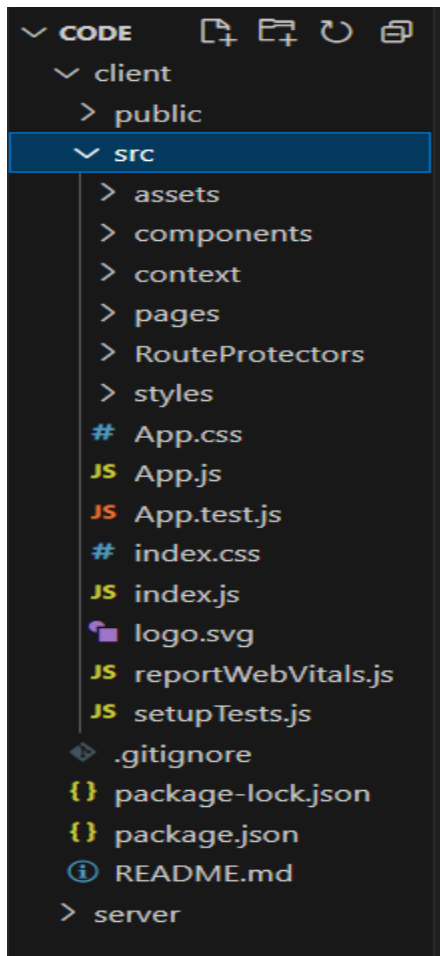
- Inside the Flight Booking app directory, we have the following folders



- **Client directory:**

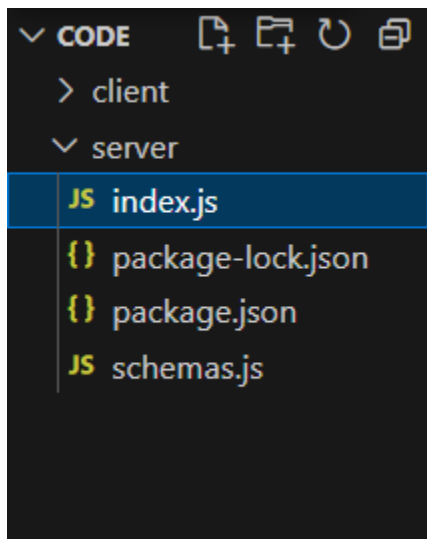
The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api's.





- **Server directory:**

The below directory structure represents the directories and files in the server folder (back end) where, node js, express is and MongoDB are used along with Api.



## **APPLICATION FLOW:**

- **USER:**
  - Create their account.
  - Search for his destination.
  - Search for flights as per his time convenience.
  - Book a flight with a particular seat.
  - Make his payment.
  - And also cancel bookings.
- **ADMIN**
  - Manages all bookings.
  - Adds new flights and services.
  - Monitor User activity.

# PROJECT FLOW:

## Milestone 1: Project setup and configuration.

### Folder setup:

To start the project from scratch, firstly create frontend and backend folders to install essential libraries and write code.

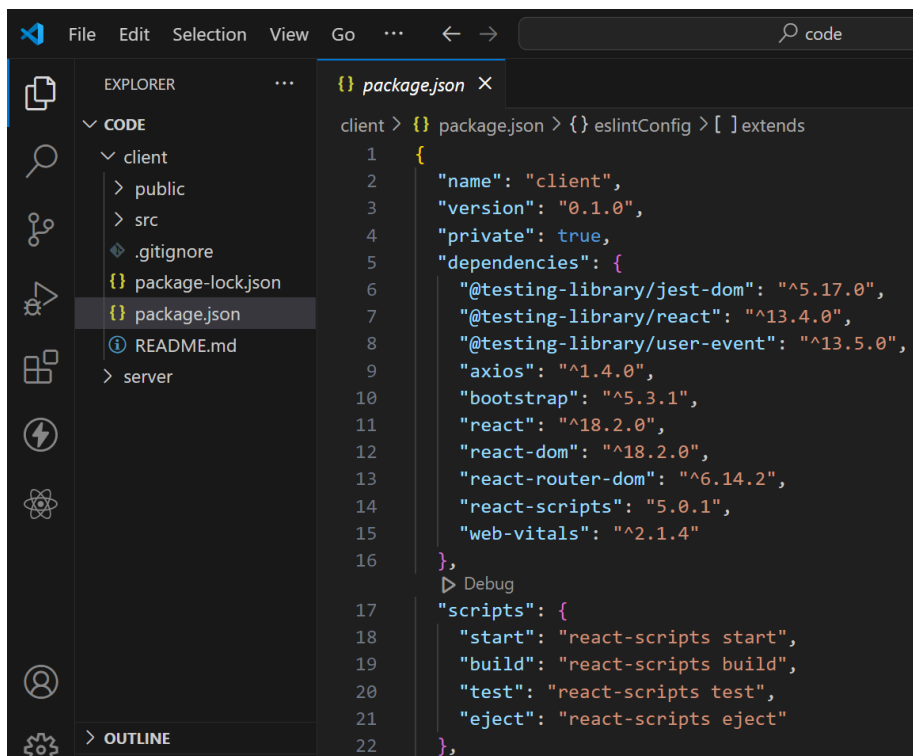
- client
- Server

### Installation of required tools:

Now, open the frontend folder to install all the necessary tools we use.  
For frontend, we use:

- React Js
- Bootstrap
- Axios

After installing all the required libraries, we'll be seeing the package.json file similar to the one below.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under the 'CODE' section, with the 'client' folder expanded. Inside 'client', files like 'public', 'src', '.gitignore', 'package-lock.json', 'package.json', and 'README.md' are visible. The 'server' folder is also listed. The main editor area shows the 'package.json' file for the 'client' folder. The file content is as follows:

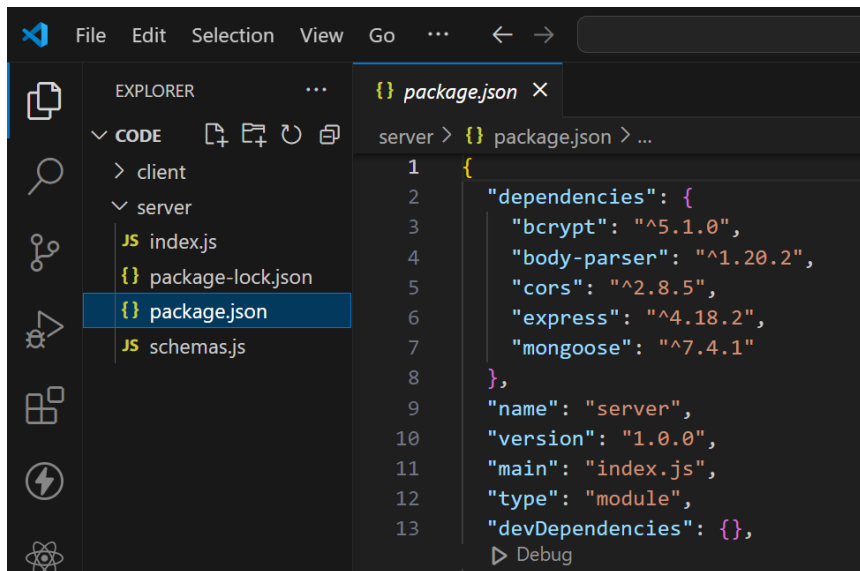
```
client > {} package.json > {} eslintConfig > [ ] extends
1  {
2    "name": "client",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.17.0",
7      "@testing-library/react": "^13.4.0",
8      "@testing-library/user-event": "^13.5.0",
9      "axios": "^1.4.0",
10     "bootstrap": "^5.3.1",
11     "react": "^18.2.0",
12     "react-dom": "^18.2.0",
13     "react-router-dom": "^6.14.2",
14     "react-scripts": "5.0.1",
15     "web-vitals": "^2.1.4"
16   },
17   "scripts": {
18     "start": "react-scripts start",
19     "build": "react-scripts build",
20     "test": "react-scripts test",
21     "eject": "react-scripts eject"
22   },
```

Now, open the backend folder to install all the necessary tools that we use in the backend.

For backend, we use:

- bcrypt
- body-parser
- cors
- express
- mongoose

After installing all the required libraries, we'll be seeing the package. Json file similar to the one below.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure with a 'server' folder containing 'index.js', 'package-lock.json', 'package.json', and 'schemas.js'. The 'package.json' file is selected and open in the main editor. The file content is as follows:

```
1 {
2   "dependencies": {
3     "bcrypt": "^5.1.0",
4     "body-parser": "^1.20.2",
5     "cors": "^2.8.5",
6     "express": "^4.18.2",
7     "mongoose": "^7.4.1"
8   },
9   "name": "server",
10  "version": "1.0.0",
11  "main": "index.js",
12  "type": "module",
13  "devDependencies": {},
14 }
```

## Milestone 2: Backend Development:

### 1. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for flights, users, bookings, and other relevant data.

### 2. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

### 3. Define API Routes:

- Create separate route files for different API functionalities such as flights, users, bookings, and authentication.
- Define the necessary routes for listing flights, handling user registration and login managing bookings, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

#### **4. Implement Data Models:**

- Define Mongoose schemas for the different data entities like flights, users, and bookings.
- Create corresponding Mongoose models to interact with the MongoDB database. Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

#### **5. User Authentication:**

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

#### **6. Handle new Flights and Bookings:**

- Create routes and controllers to handle new flight listings, including fetching flight data from the database and sending it as a response.
- Implement booking functionality by creating routes and controllers to handle booking requests, including validation and database updates.

#### **7. Admin Functionality:**

- Implement routes and controllers specific to admin functionalities such as adding flights, managing user bookings, etc.
- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

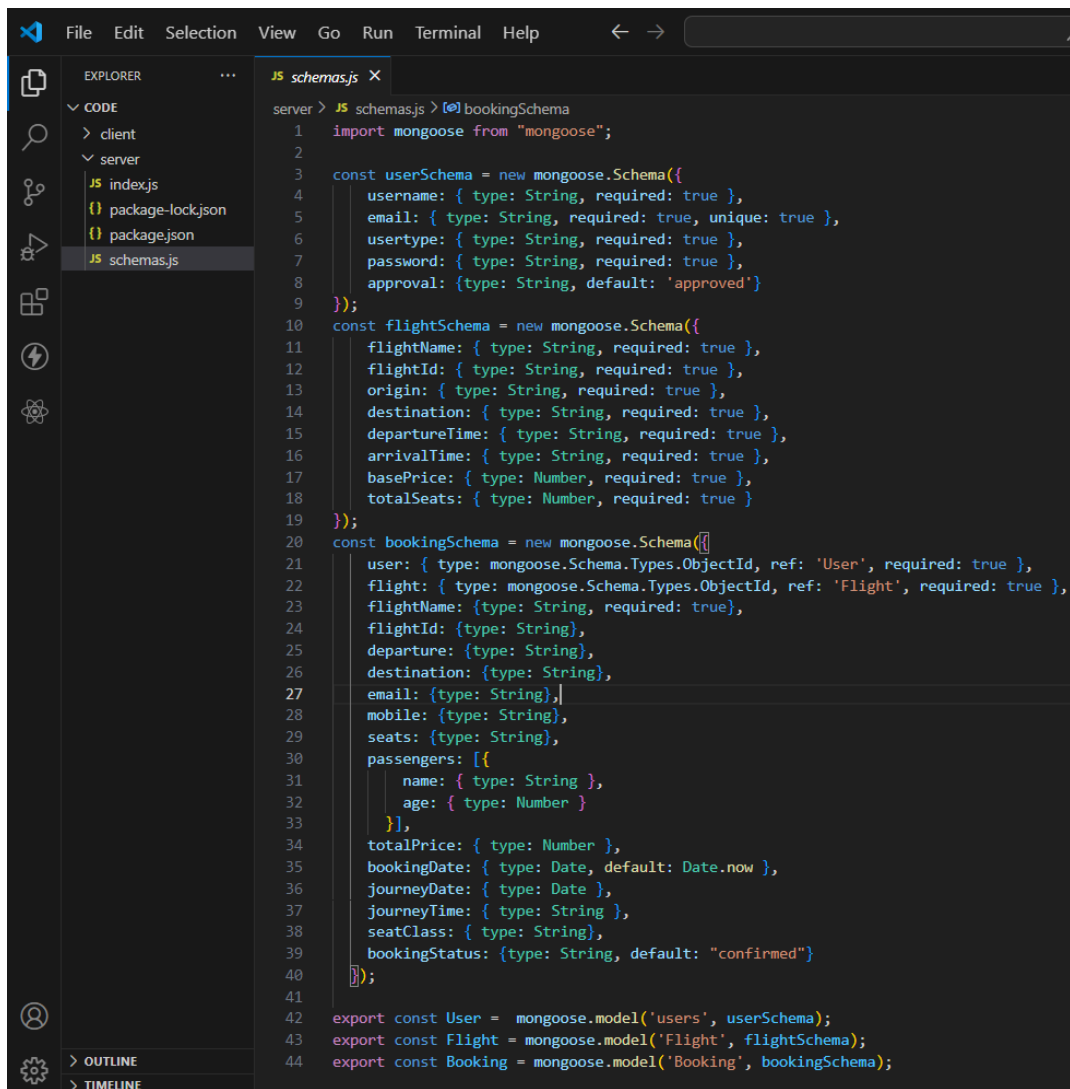
#### **8. Error Handling:**

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

### **Milestone 3: Database development**

- **Configure schema**

Firstly, configure the Schemas for MongoDB database, to store the data in such a pattern. Use the data from the ER diagrams to create the schemas. The Schemas for this application look alike to the one provided below.



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'server' folder is expanded, showing files like 'index.js', 'package-lock.json', 'package.json', and 'schemas.js'. The 'schemas.js' file is open in the editor. The code defines three Mongoose schemas: 'userSchema', 'flightSchema', and 'bookingSchema'. 'userSchema' has fields for username, email, password, and approval. 'flightSchema' has fields for flightName, flightId, origin, destination, departureTime, arrivalTime, basePrice, and totalSeats. 'bookingSchema' has fields for user, flight, flightName, flightId, departure, destination, email, mobile, seats, passengers (an array of objects with name and age), totalPrice, bookingDate, journeyDate, journeyTime, seatClass, and bookingStatus. At the bottom, the code exports the models for User, Flight, and Booking.

```
server > JS schemas.js > bookingSchema
1  import mongoose from "mongoose";
2
3  const userSchema = new mongoose.Schema({
4    username: { type: String, required: true },
5    email: { type: String, required: true, unique: true },
6    usertype: { type: String, required: true },
7    password: { type: String, required: true },
8    approval: { type: String, default: 'approved' }
9  });
10 const flightSchema = new mongoose.Schema({
11   flightName: { type: String, required: true },
12   flightId: { type: String, required: true },
13   origin: { type: String, required: true },
14   destination: { type: String, required: true },
15   departureTime: { type: String, required: true },
16   arrivalTime: { type: String, required: true },
17   basePrice: { type: Number, required: true },
18   totalSeats: { type: Number, required: true }
19 });
20 const bookingSchema = new mongoose.Schema({
21   user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
22   flight: { type: mongoose.Schema.Types.ObjectId, ref: 'Flight', required: true },
23   flightName: { type: String, required: true },
24   flightId: { type: String },
25   departure: { type: String },
26   destination: { type: String },
27   email: { type: String },
28   mobile: { type: String },
29   seats: { type: String },
30   passengers: [{
31     name: { type: String },
32     age: { type: Number }
33   }],
34   totalPrice: { type: Number },
35   bookingDate: { type: Date, default: Date.now },
36   journeyDate: { type: Date },
37   journeyTime: { type: String },
38   seatClass: { type: String },
39   bookingStatus: { type: String, default: "confirmed" }
40 });
41
42 export const User = mongoose.model('users', userSchema);
43 export const Flight = mongoose.model('Flight', flightSchema);
44 export const Booking = mongoose.model('Booking', bookingSchema);
```

- **Connect database to backend**

Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.



The screenshot shows a code snippet for connecting to a MongoDB database and starting a server. It sets a default port of 6001, connects to the database using 'process.env.MONGO\_URL', and then listens on the specified port. It also includes error handling for the database connection.

```
const PORT = process.env.PORT || 6001;
mongoose.connect(process.env.MONGO_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(()=>{

  server.listen(PORT, ()=>{
    console.log("Running @ ${PORT}");
  });

}).catch((err)=>{
  console.log("Error: ", err);
})
```



## **Milestone 4: Frontend development.**

### **1. Login/Register**

- Create a Component which contains a form for taking the username and password.
- If the given inputs match the data of user or admin or flight operator then navigate it to their respective home page

### **2. Flight Booking (User):**

- In the frontend, we implemented all the booking code in a modal. Initially, we need to implement flight searching feature with inputs of Departure city, Destination, etc.,
- Flight Searching code: With the given inputs, we need to fetch the available flights. With each flight, we add a button to book the flight, which redirects to the flight-Booking page.

### **3. Fetching user bookings:**

- In the bookings page, along with displaying the past bookings, we will also provide an option to cancel that booking.

### **4. Add new flight(Admin):**

- Now, in the admin dashboard, we provide functionality to add new flights.
- We create a html form with required inputs for the new flight and then send an HTTP request to the server to add it to the database.

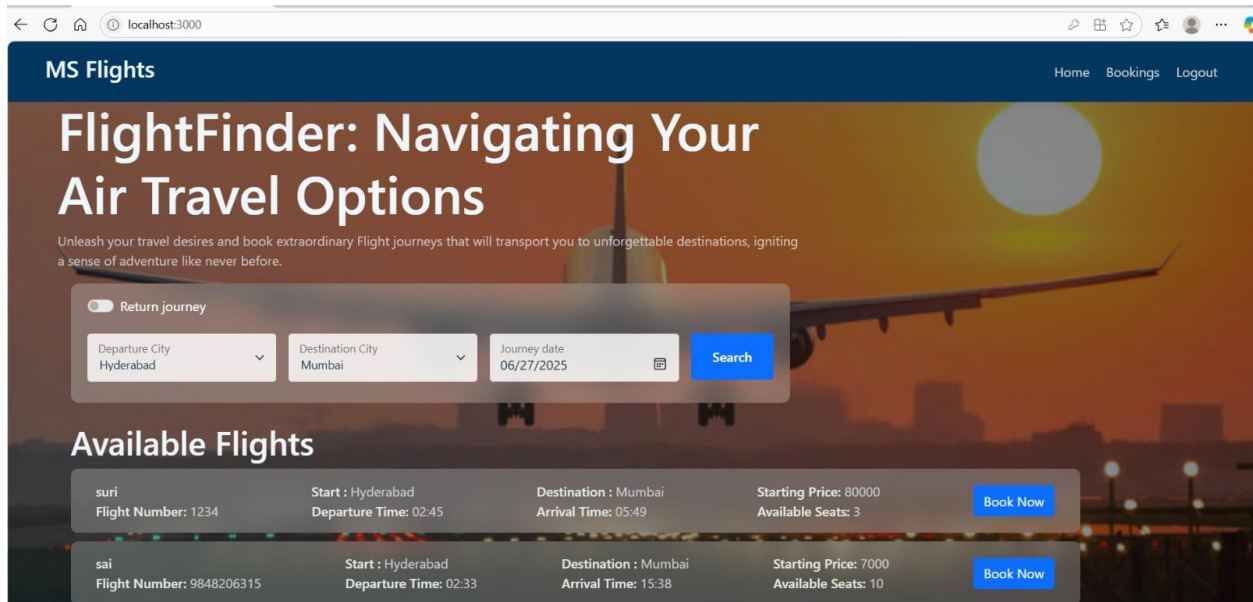
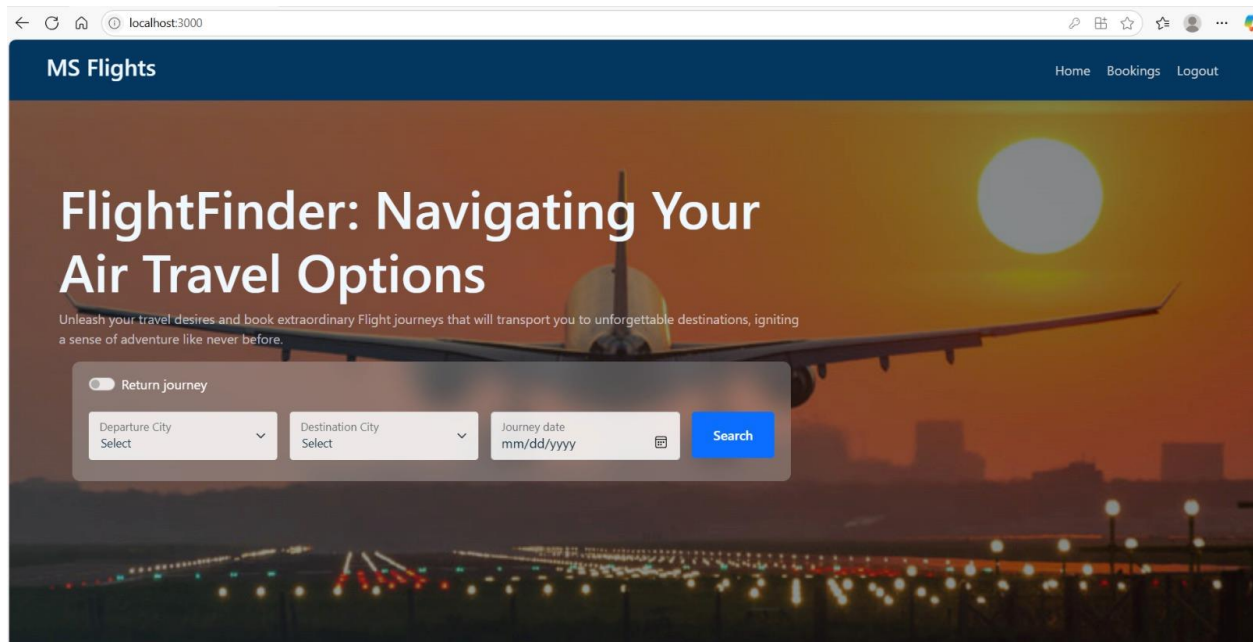
### **5. Update Flight:**

- Here, in the admin dashboard, we will update the flight details in case if we want to make any edits to it
- Along with this, implement additional features to view all flights, bookings, and users in the admin dashboard.

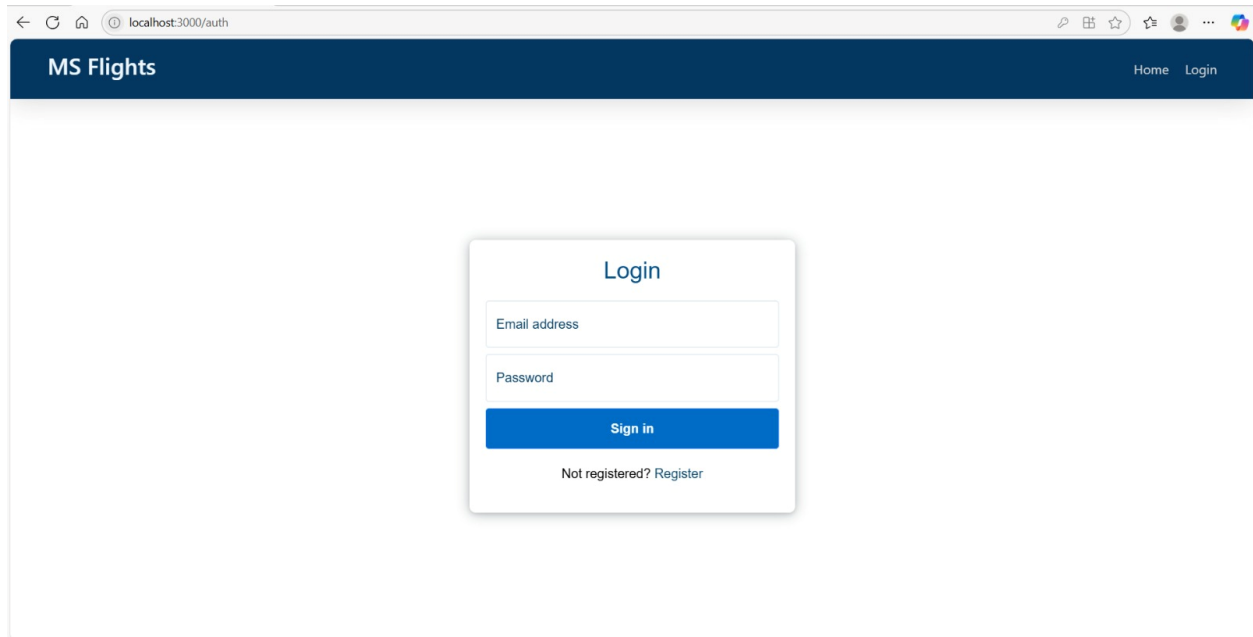
## Milestone 5: Project Implementation.

Finally, after finishing coding the projects we run the whole project to test it's working process and look for bugs. Now, let's have a final look at the working of our video conference application

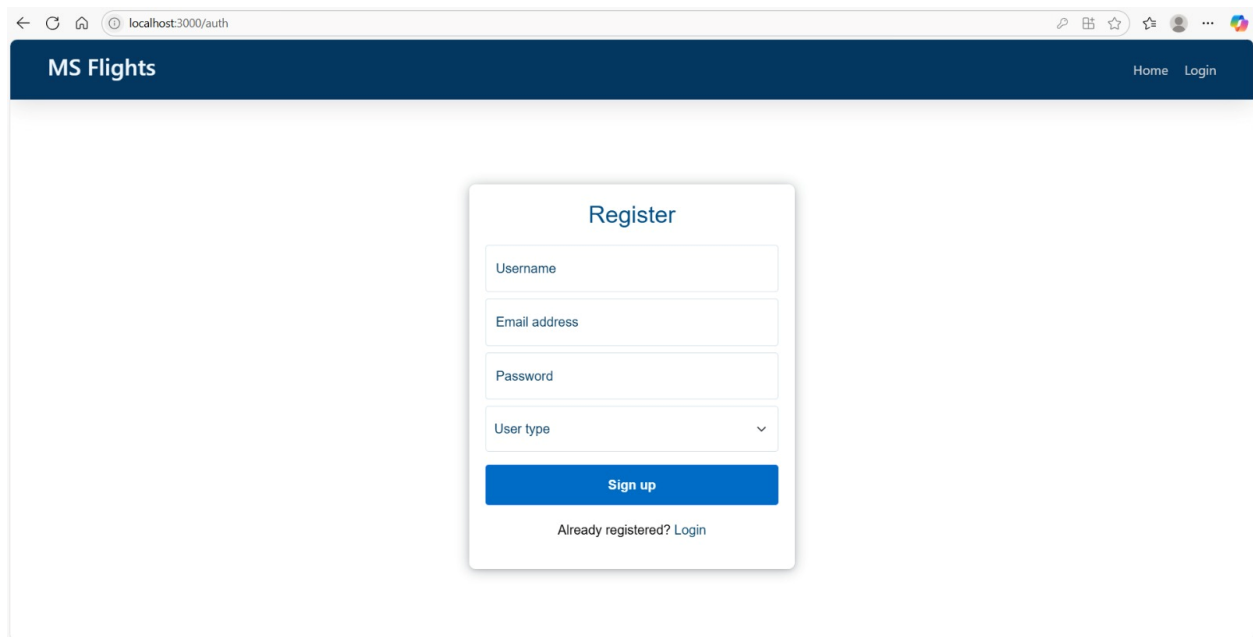
- **Landing page UI**



- **Authentication**



A screenshot of a web browser showing the login page of 'MS Flights'. The browser's address bar displays 'localhost:3000/auth'. The page has a dark blue header with 'MS Flights' on the left and 'Home Login' on the right. In the center, there is a white login form with a blue border. The form is titled 'Login' and contains two input fields: 'Email address' and 'Password'. Below these fields is a blue button labeled 'Sign in'. At the bottom of the form, there is a link that says 'Not registered? Register'.



A screenshot of a web browser showing the register page of 'MS Flights'. The browser's address bar displays 'localhost:3000/auth'. The page has a dark blue header with 'MS Flights' on the left and 'Home Login' on the right. In the center, there is a white register form with a blue border. The form is titled 'Register' and contains four input fields: 'Username', 'Email address', 'Password', and 'User type' (which is a dropdown menu). Below these fields is a blue button labeled 'Sign up'. At the bottom of the form, there is a link that says 'Already registered? Login'.

- **User bookings**

localhost:3000/book-flight/685c3c75021af6813846e5f0

MS Flights

Home Bookings Logout

### Book ticket

Flight Name: suri Flight No: 1234

Base price: 80000

Email Mobile

No of passengers Journey date Seat Class

0 06/27/2025 Select

Total price: 0

Book now

localhost:3000/book-flight/685c3c75021af6813846e5f0

MS Flights

Home Bookings Logout

### Book ticket

Flight Name: suri Flight No: 1234

Base price: 80000

Email madhu99@gmail.com Mobile 6303632112

No of passengers Journey date Seat Class

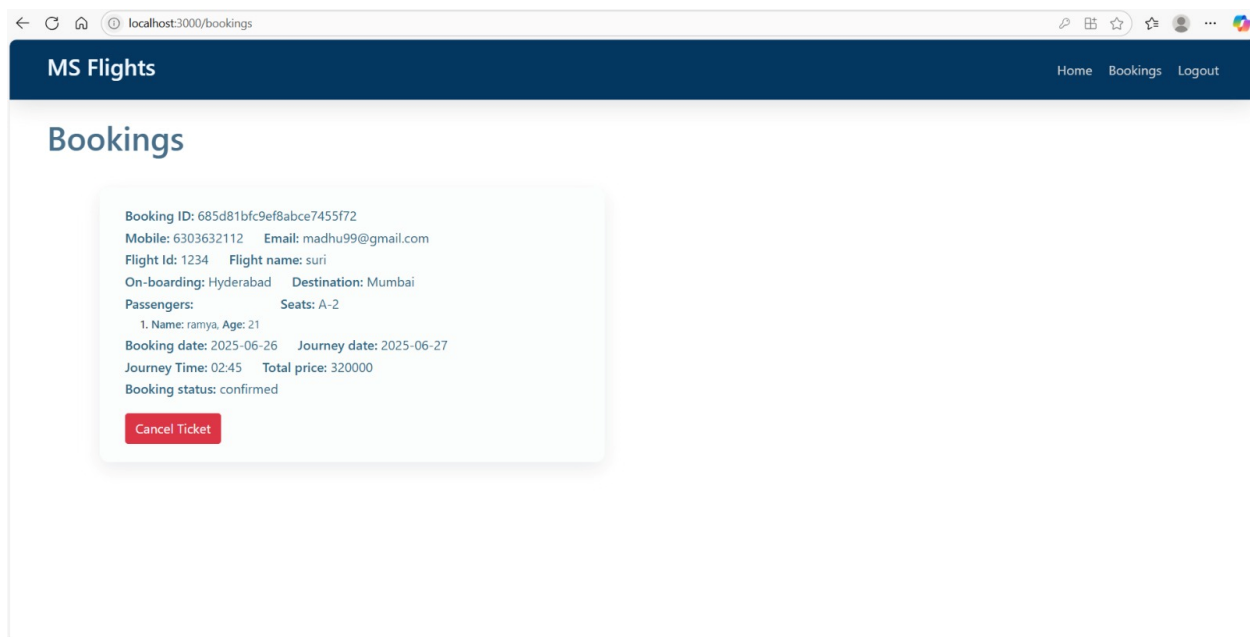
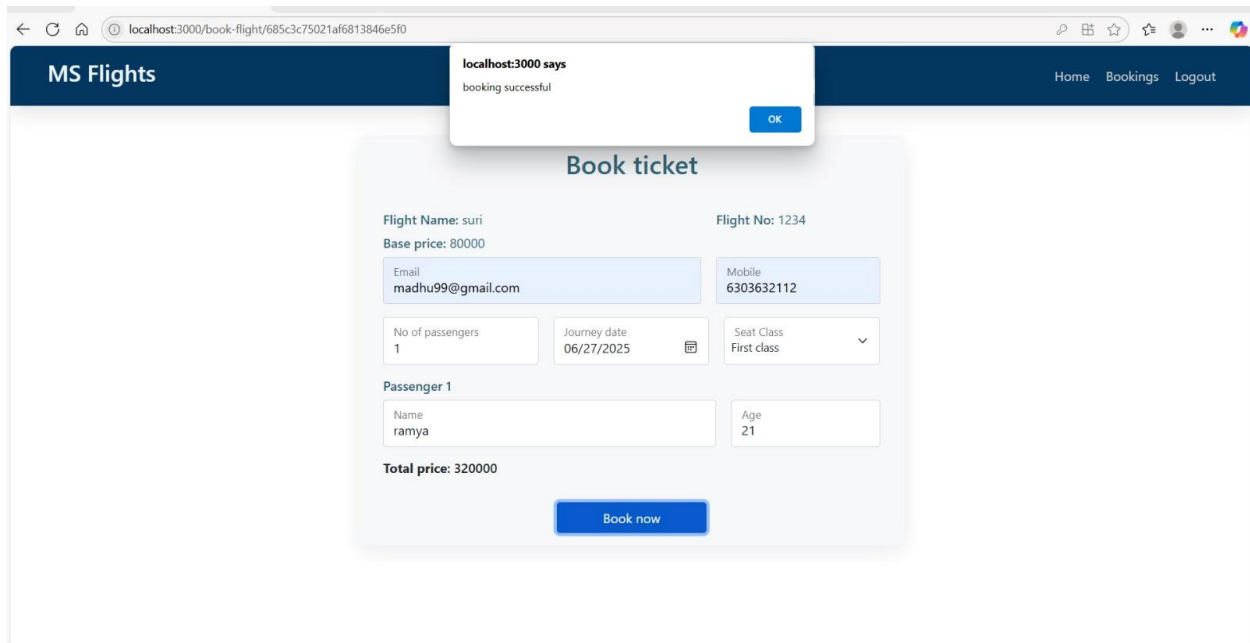
1 06/27/2025 First class

Passenger 1

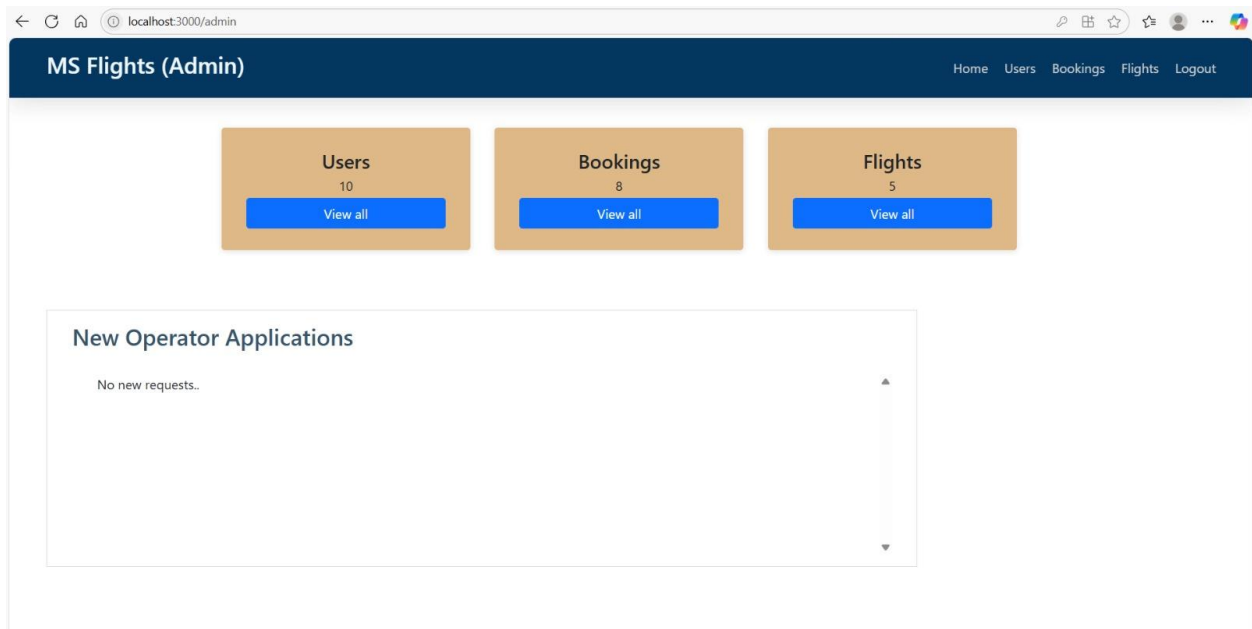
Name ramya Age 21

Total price: 320000

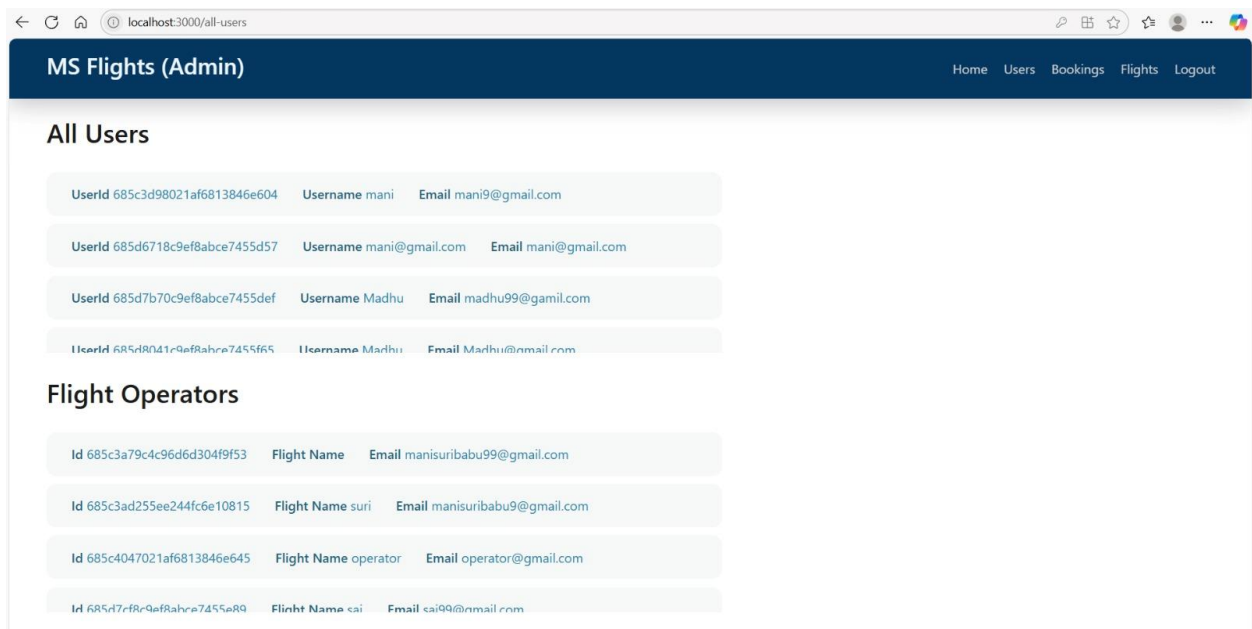
Book now



- Admin Dashboard



- All users



localhost:3000/all-users

## MS Flights (Admin)

Home Users Bookings Flights Logout

### All Users

Userid 685c3d98021af6813846e604	Username mani	Email mani9@gmail.com
Userid 685d6718c9ef8abce7455d57	Username mani@gmail.com	Email mani@gmail.com
Userid 685d7b70c9ef8abce7455def	Username Madhu	Email madhu99@gmail.com
Userid 685d80d41c9ef8abce7455f65	Username Madhu	Email Madhu@gmail.com

### Flight Operators

Id 685c3a79c4c96d6d304f9f53	Flight Name	Email manisuribabu99@gmail.com
Id 685c3ad255ee244fc6e10815	Flight Name suri	Email manisuribabu9@gmail.com
Id 685c4047021af6813846e645	Flight Name operator	Email operator@gmail.com
Id 685d7c7f8c9ef8abce7455e0a	Flight Name sai	Email sai99@gmail.com

- All Bookings

localhost:3000/all-bookings

## MS Flights (Admin)

Home Users Bookings Flights Logout

### Bookings

Booking ID: 685d81bfc9ef8abce7455f72  
Mobile: 6303632112 Email: madhu99@gmail.com  
Flight Id: 1234 Flight name: suri  
On-boarding: Hyderabad Destination: Mumbai  
Passengers: Seats: A-2  
1. Name: ramya, Age: 21  
Booking date: 2025-06-26 Journey date: 2025-06-27  
Journey Time: 02:45 Total price: 320000  
Booking status: confirmed

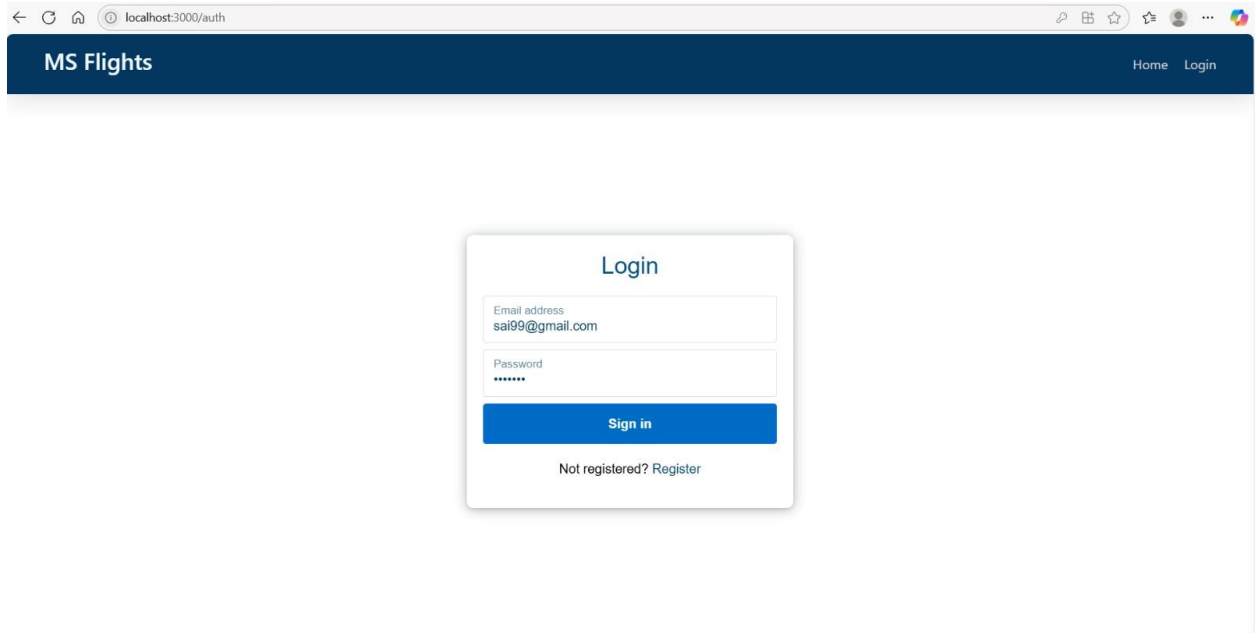
Cancel Ticket

Booking ID: 685d7c22c9ef8abce7455e0a  
Mobile: 6303632112 Email: madhu99@gmail.com  
Flight Id: 1234 Flight name: suri  
On-boarding: Hyderabad Destination: Mumbai  
Passengers:  
1. Name: mani, Age: 26  
2. Name: sai, Age: 27  
Booking date: 2025-06-26 Journey date: 2025-06-27  
Journey Time: 02:45 Total price: 480000  
Booking status: cancelled

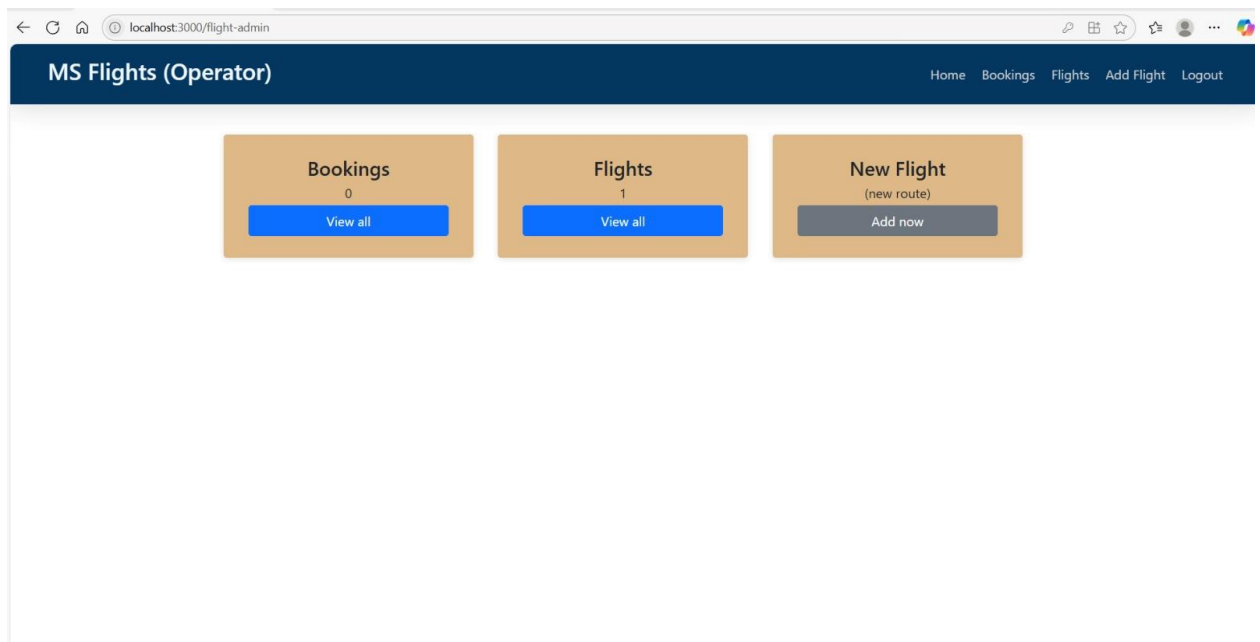
Booking ID: 685d7bdc9ef8abce7455df5  
Mobile: 6303632112 Email: madhu99@gmail.com  
Flight Id: 1234 Flight name: suri  
On-boarding: Hyderabad Destination: Mumbai  
Passengers: Seats: A-1  
1. Name: Madhu, Age: 21  
Booking date: 2025-06-26 Journey date: 2025-06-27

Booking ID: 685d67eec9ef8abce7455d5d  
Mobile: 9848206315 Email: mani@gmail.com  
Flight Id: 1234 Flight name: suri  
On-boarding: Hyderabad Destination: Mumbai  
Passengers: Seats: P-1  
1. Name: manikanta, Age: 30  
Booking date: 2025-06-26 Journey date: 2025-06-27

- **Flight Operator**

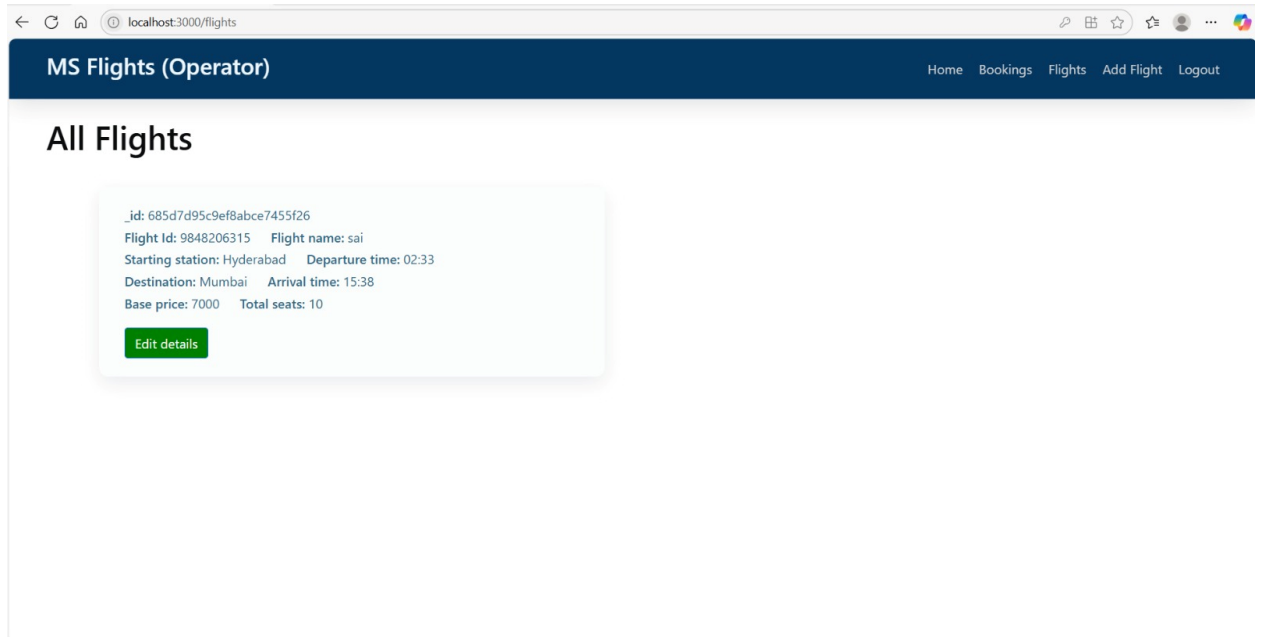


The screenshot shows a web browser at localhost:3000/auth. The page has a dark blue header with "MS Flights" on the left and "Home Login" on the right. In the center is a white login box with the title "Login". It contains two input fields: "Email address" with the value "sai99@gmail.com" and "Password" with masked characters "\*\*\*\*\*". Below these is a blue "Sign in" button. At the bottom of the box is a link "Not registered? Register".



The screenshot shows a web browser at localhost:3000/flight-admin. The page has a dark blue header with "MS Flights (Operator)" on the left and navigation links "Home Bookings Flights Add Flight Logout" on the right. The main content area features three orange cards. The first card is titled "Bookings" with a count of "0" and a blue "View all" button. The second card is titled "Flights" with a count of "1" and a blue "View all" button. The third card is titled "New Flight" with the subtitle "(new route)" and a grey "Add now" button.





- **New Flight**

MS Flights (Operator)

Home Bookings Flights Add Flight Logout

### Add new Flight

Flight Name  
sai

Flight Id

Departure City  
Select

Departure Time  
--:-- --

Destination City  
Select

Arrival time  
--:-- --

Total seats  
0

Base price  
0

Add now

localhost:3000/new-flight

MS Flights (Operator) Home Bookings Flights Add Flight Logout

### Add new Flight

Flight Name sai	Flight Id 09705328488
Departure City Hyderabad	Departure Time 04:02 AM
Destination City Mumbai	Arrival time 03:59 PM
Total seats 3	Base price 5000

Add now

localhost:3000/new-flight

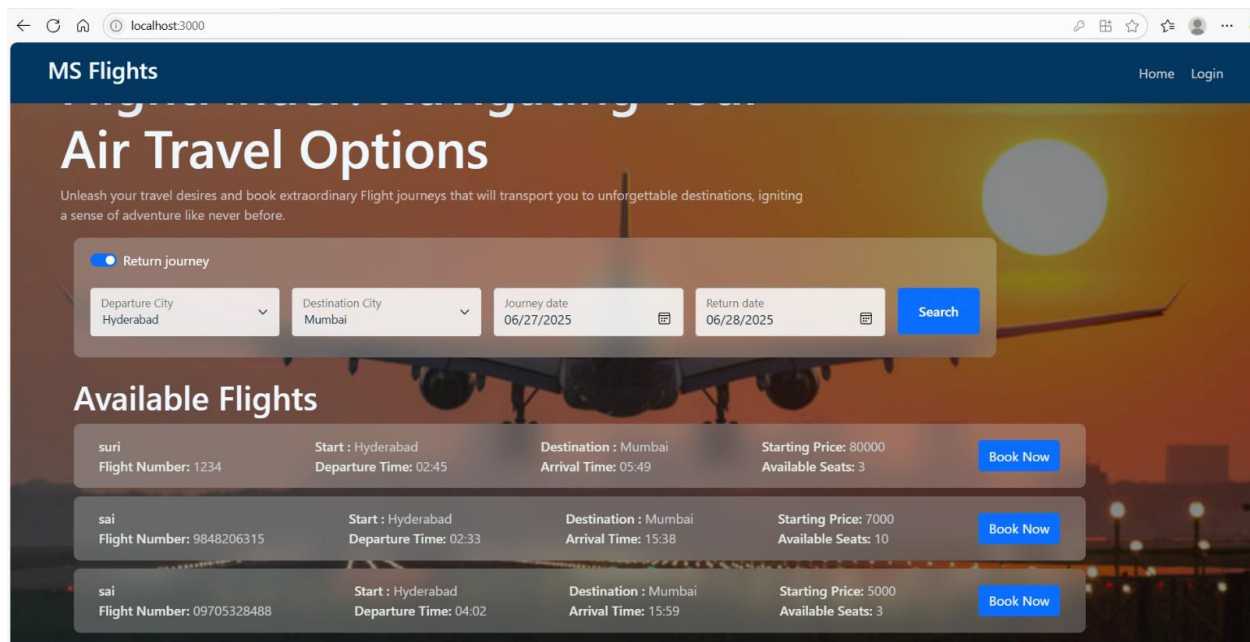
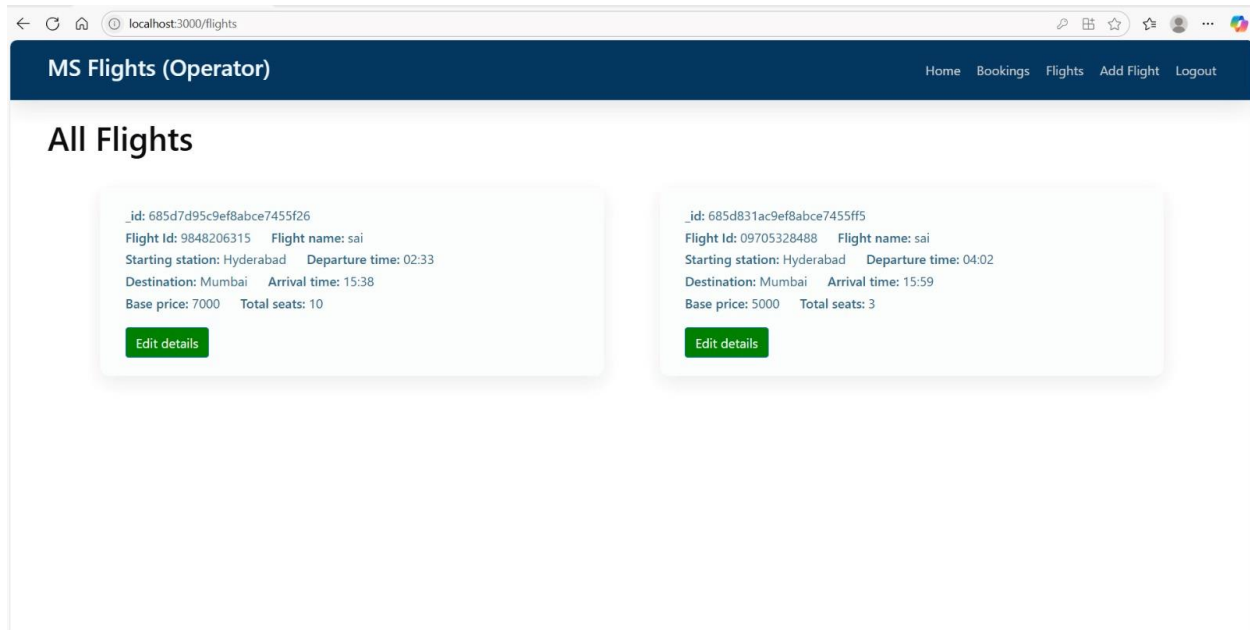
MS Flights (Operator) Home Bookings Flights Add Flight Logout

localhost:3000 says  
Flight added successfully!!  
OK

### Add new Flight

Flight Name sai	Flight Id 09705328488
Departure City Hyderabad	Departure Time 04:02 AM
Destination City Mumbai	Arrival time 03:59 PM
Total seats 3	Base price 5000

Add now



# Challenge Faced

## 1. Integration Across the MERN Stack

Coordinating React (frontend), Express/Node, and MongoDB backend often leads to mismatched data formats or API miscommunication. Debugging whether an issue originates in the UI call, Express routing, or DB layer can be time-consuming

## 2. Authentication & Security

Implementing secure user login, JWT/session handling, role-based access (especially for admin panels), token expiry, and protection against XSS/CSRF requires careful planning. Beginners frequently struggle with securely storing tokens and guarding sensitive routes.

## 3. API Integration (Flights & Payments)

Connecting flight data APIs or payment gateways introduces challenges around authentication, rate limits, error handling, and transforming inconsistent or complex external data into your app's schema.

## 4. Performance, Caching & Scalability

As flight records grow, MongoDB performance can degrade without proper indexing, caching, or pagination. Server-side throttling and Node.js clustering may be needed to maintain responsiveness.

## 5. Error Handling & Monitoring

Centralizing error handling in Express (middleware), using React error-boundary components, and consistent logging (e.g., via Morgan/Winston/Sentry) is essential—but often overlooked in smaller projects ([n-school.com][2]).

## 6. User Experience & UI Design

Complex UIs—flight search, filters, seat maps, booking forms—are prone to confusion and poor usability. Without iterative design feedback, the interface may feel cluttered or unintuitive.

## 7. Complex State Management

Managing global state in React (e.g., search filters, flight selections, user profiles) can become messy without Redux, Context API, or custom hooks—leading to bugs and data inconsistencies.

## 8. Cross-Cutting Concerns Handling

Multilingual support, multiple currencies, time zones, date formatting, and storing user preferences adds complexity in ensuring a smooth, globally functional experience.

## **9. Testing, Deployment & CI/CD**

Setting up automated unit, integration, and end-to-end tests across all stack layers is challenging. Deploying and maintaining runtime consistency (e.g., CORS, environment variables, build pipelines) also adds overhead.

## **10. Offline & Connectivity Handling**

Users may lose connectivity mid-flow. Without robust offline handling, seat map state and booking progress can be lost, degrading trust and usability.

# Future Scope of Flight Finder

The current version of Flight Finder provides a solid foundation for flight booking with an intuitive interface, real-time flight listings, and secure booking. To further enhance the user experience and meet evolving traveler expectations, the following future improvements and expansions are proposed:

## **1. Enhanced Personalization with AI**

**Smart Recommendations:** Use machine learning to suggest flights based on user preferences, travel history, and seasonal trends.

**Dynamic Pricing Alerts:** Notify users about price drops or better deals using predictive price modeling.

**Custom Travel Itineraries:** Suggest accommodations, local transportation, and attractions along with flights.

## **2. Multi-City and Complex Route Support**

Allow users to book multi-leg or round-the-world journeys.

Include open-jaw tickets (e.g., flying into one city and returning from another).

## **3. Real-Time Flight Status and Notifications**

Integrate real-time flight tracking APIs to provide updates on delays, gate changes, and cancellations.

Push notifications or SMS/email alerts for changes to bookings or reminders.

## **4. Loyalty and Reward Integration**

Partner with major airlines and loyalty programs.

Enable users to earn and redeem points or miles through the app.

## **5. In-App Check-In and Boarding Passes**

Allow users to check in directly through the app and access digital boarding passes.

Integrate with airline systems for a smoother pre-flight experience.

## **6. Multilingual and Multi-Currency Support**

Support for global users by adding multiple languages and real-time currency conversion during checkout.

## **7. Chatbot and Virtual Travel Assistant**

Implement an AI-powered chatbot for 24/7 support to answer FAQs, guide users, or handle issues like rescheduling.

Provide travel tips, visa information, and local customs for international trips.

## **8. Eco-Friendly Flight Options**

Highlight flights with lower carbon emissions.

Partner with carbon offset programs to give users an option to contribute during checkout.

## **9. Admin Panel Enhancements**

Advanced analytics for bookings, revenue trends, and user engagement.

Real-time management of flight data and promotional campaigns.

## **10. Mobile App Development**

Expand to dedicated Android and iOS apps with offline features such as ticket storage and itinerary access.

Integrate biometric login (FaceID, fingerprint) for faster access and better security.

## **11. Third-Party Integrations**

Integrate with travel insurance providers, ride-hailing apps (e.g., Uber, Lyft), and hotel booking platforms for an all-in-one travel planning ecosystem.

## **12. Blockchain for Ticket Verification**

Explore blockchain technology to prevent fraud and enhance transparency in ticket bookings and transactions.

By implementing these future enhancements, Flight Finder can evolve into a comprehensive travel companion app, capable of addressing the full spectrum of modern traveler needs while maintaining a high standard of convenience, security, and innovation.

## CONCLUSION

The **Flight Finder** mini-project—built with the MERN (MongoDB, Express, React, Node.js) stack—successfully delivers a streamlined, full-stack flight booking experience. From secure user authentication and dynamic flight searches to intuitive seat selection and payment handling, the app encapsulates real-world travel app functionality.

Key accomplishments in the project include:

- **End-to-End Workflow:** Users can search, filter, select, and book flights, covering the full booking lifecycle.
- **Componentized Architecture:** React components and modular backend APIs facilitate maintainability and future expansion.
- **Security & Standards:** JWT-based authentication, protected routes, and secure payment-handling illustrate adherence to best practices.
- **Data Handling:** MongoDB collections for users, flights, and bookings ensure efficient management and scalability of core data.

Alongside these strengths, the project helped identify important development lessons: ensuring seamless state management across components, integrating third-party APIs gracefully, and addressing performance tuning and error handling in a real-world context. Flight Finder establishes a solid foundation for next-level enhancements such as live flight status updates, multi-city itineraries, loyalty programs, multilingual support, and mobile responsiveness. With further iteration, it has the potential to transform into a production-grade travel platform.

In summary, this project not only reinforces modern web development skills but also simulates the demanding architectural and UX considerations faced by real-world travel services—making it a valuable milestone in your development portfolio.



## REFERENCES

The following online resources, tools, and platforms were extensively used during the research, development, testing, and documentation of this project:

1. **MDN Web Docs (Mozilla Developer Network)**  
<https://developer.mozilla.org>  
For foundational understanding of HTML, CSS, JavaScript, and DOM manipulation.
2. **React.js Official Documentation**  
<https://react.dev>  
For learning about component structure, hooks, state management, and routing.
3. **Node.js & Express.js Documentation**  
<https://nodejs.org/en/docs>  
<https://expressjs.com>  
For backend logic, server configuration, and RESTful API development.
4. **MongoDB Documentation**  
<https://www.mongodb.com/docs>  
For database schema design, CRUD operations, and Mongoose integration.
5. **Bootstrap Documentation**  
<https://getbootstrap.com>  
Used for building responsive layouts and UI components.
6. **Material UI Documentation**  
<https://mui.com>  
For implementing polished, accessible, and consistent design elements.
7. **Postman**  
<https://www.postman.com>  
Used for testing API endpoints and analyzing backend responses.
8. **GitHub**  
<https://github.com>  
For version control and collaborative development.
9. **YouTube Channels**
  - Traversy Media
  - The Net Ninja
  - Programming with MoshFor practical tutorials, full-stack project walkthroughs, and advanced tips.
10. **GeeksforGeeks**  
<https://www.geeksforgeeks.org>  
Used for reference on algorithms, backend logic, and JavaScript topics.

**ChatGPT by OpenAI**

<https://chat.openai.com>