

Flight Booking APP

Flight Finder: Navigate your Air travel option

Team Members:

Here List team members and their roles:

1.M. Subhash (Full Stack Developer): Combines both frontend and backend responsibilities, ensuring smooth communication between the two. This role also handles bug fixing, feature integration, and overall system performance.

2.S. Madhu Pavan (Frontend Developer): Responsible for designing the user interface using React.js. This role focuses on ensuring a responsive, user-friendly design, as well as integrating the frontend with backend APIs.

3.T. Lakshmi Sravani (Backend Developer): Develops the backend server using Node.js and Express.js, ensuring the creation of secure, scalable RESTful APIs, as well as handling authentication, data processing, and business logic.

4.S. Srinija (Database Administration): Manages the MongoDB database, focusing on schema design, data integrity, and database optimization to ensure efficient data storage and retrieval.

INTRODUCTION

In today's fast-paced world, travel has become an integral part of our lives—be it for business, leisure, or personal reasons. With advancements in technology, booking flights is now more convenient and accessible than ever before, thanks to flight booking applications.

Flight Finder is a modern flight booking app designed to make air travel planning seamless and efficient. It enables users to easily search, compare, and book flights from the comfort of their smartphones or computers. The primary goal of the app is to simplify the entire flight booking experience by providing a user-friendly interface and essential

features—from browsing available flights to managing bookings and receiving real-time travel updates.

DESCRIPTION

Flight Finder is a modern digital platform designed to revolutionize the way users book flight tickets. This web-based application enhances the flight booking experience by offering exceptional convenience, efficiency, and ease of use.

Our intuitive and user-friendly interface empowers traveller's to effortlessly search, explore, and reserve flight tickets tailored to their preferences. Whether you're a frequent flyer or an occasional traveller, Flight Finder makes finding the perfect flight simple and hassle-free.

This powerful flight booking solution brings together an efficient search system, secure booking capabilities, personalized features, robust security, and reliable performance. It continuously evolves through user feedback, ensuring a seamless and satisfying experience for all types of travellers.

SCENARIO

- Arjun, a frequent traveller 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 advanced features.
- He opens the Flight Finder app on his smartphone and enters his travel details:
 - Departure: New York City
 - Destination: Paris
 - Departure Date: April 10
 - Return Date: April 15
 - Class: Business Class
 - Passengers: 1
- The app quickly retrieves available flight options based on Arjun's preferences. It displays a range of choices from various airlines, including direct and connecting flights. Each result includes key details like price, airline name, flight duration, and departure times.
- Using the app's filter options, Arjun refines his search to show only direct flights with convenient departure times. He also selects his preferred airline based on his loyalty membership and past experiences.

- Once he chooses a flight, Arjun selects his business class seat. The app displays a seat map highlighting available options, allowing him to pick a window seat with extra legroom.
- Arjun securely enters his payment details using the app's integrated payment gateway. The app processes the transaction and instantly generates a booking confirmation along with his e-ticket and itinerary.

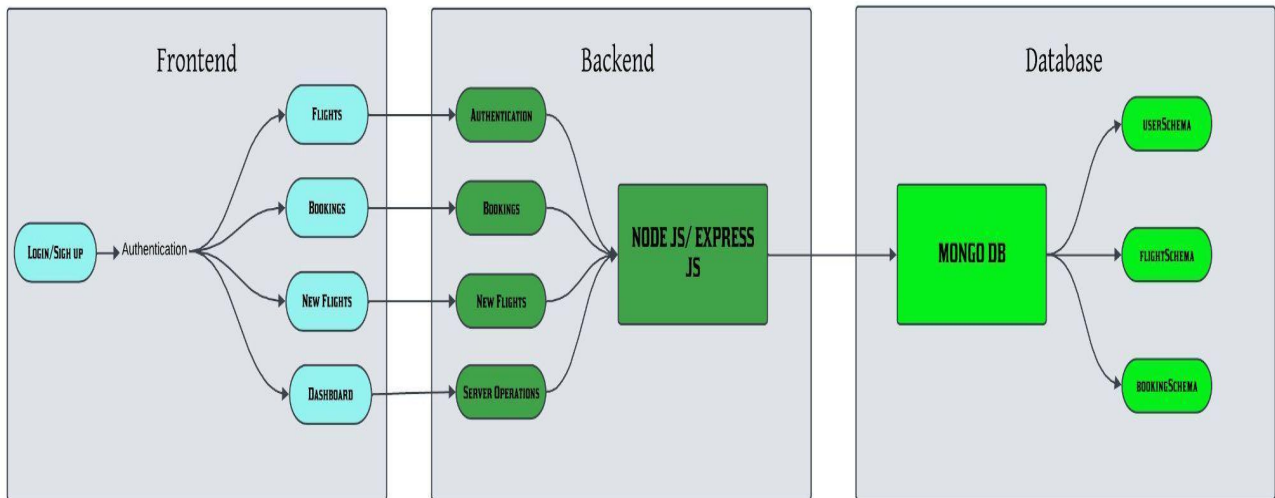
This scenario highlights how Flight Finder streamlines the entire travel process—offering users convenience, customization, and real-time support throughout their journey.

TECHNICAL ARCHITECTURE

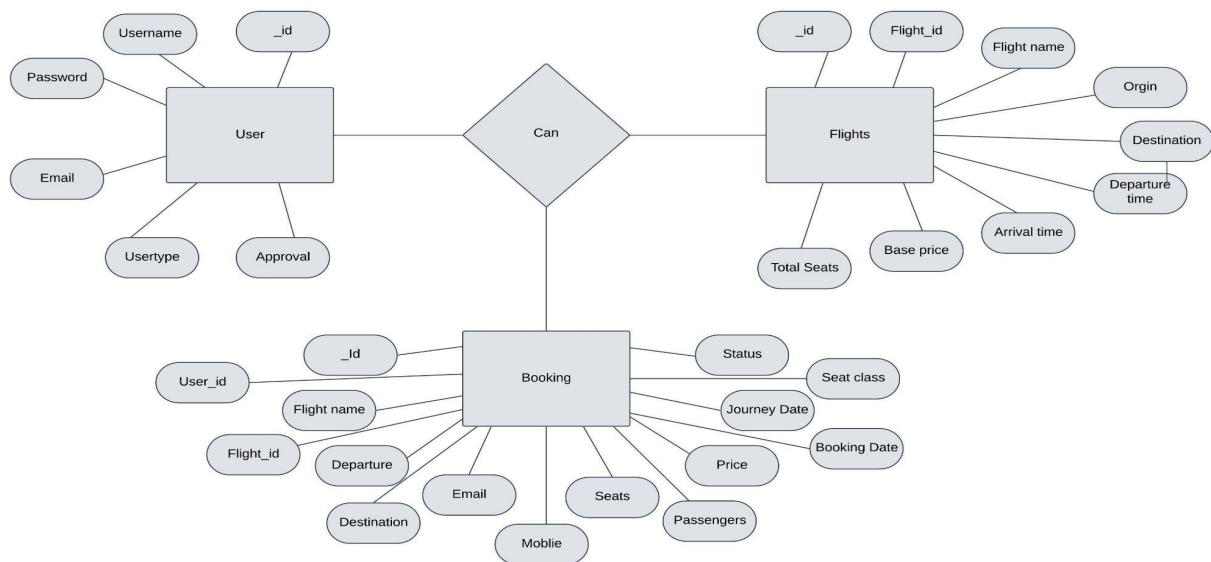
In the architecture of the Flight Finder application:

- The Frontend section represents the user interface, including components such as:
 - User Authentication: Sign up, login, password reset
 - Flight Search: Enter travel details, view available flights
 - Booking: Select flights, choose seats, and complete payment
- The Backend section handles the core logic and services, including:
 - API Endpoints for managing:
 - Users
 - Flights
 - Bookings
 - Admin operations
 - Admin Authentication: Secure access for admin users
 - Admin Dashboard: Interface for managing flights, users, and bookings
- The Database section stores and manages all data, with collections such as:
 - Users: User profiles and credentials
 - Flights: Flight schedules and availability
 - Bookings: Flight reservations and transaction records

This architecture ensures a clear separation of concerns, providing scalability, security, and a smooth user experience across the platform.



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 lightest.

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: https://github.com/71Subhash05/Flight_Finder

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.

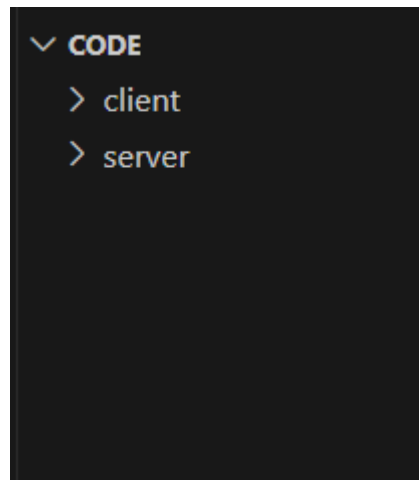
Access the App:

- 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.

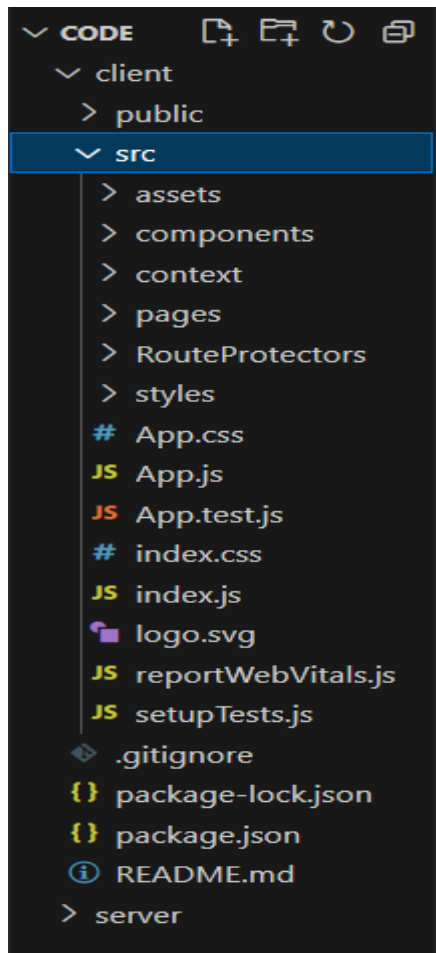
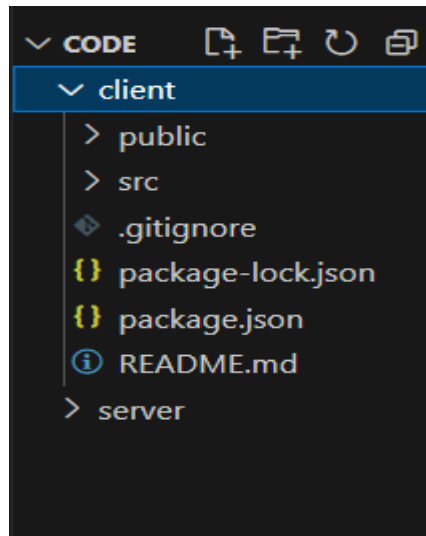
PROJECT STRUCTURE:

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



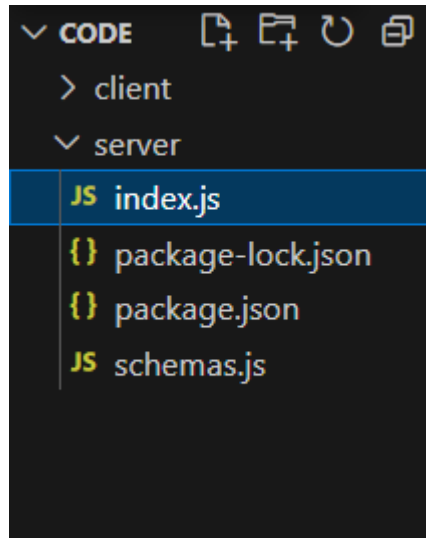
- **Client directory:**

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 js 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

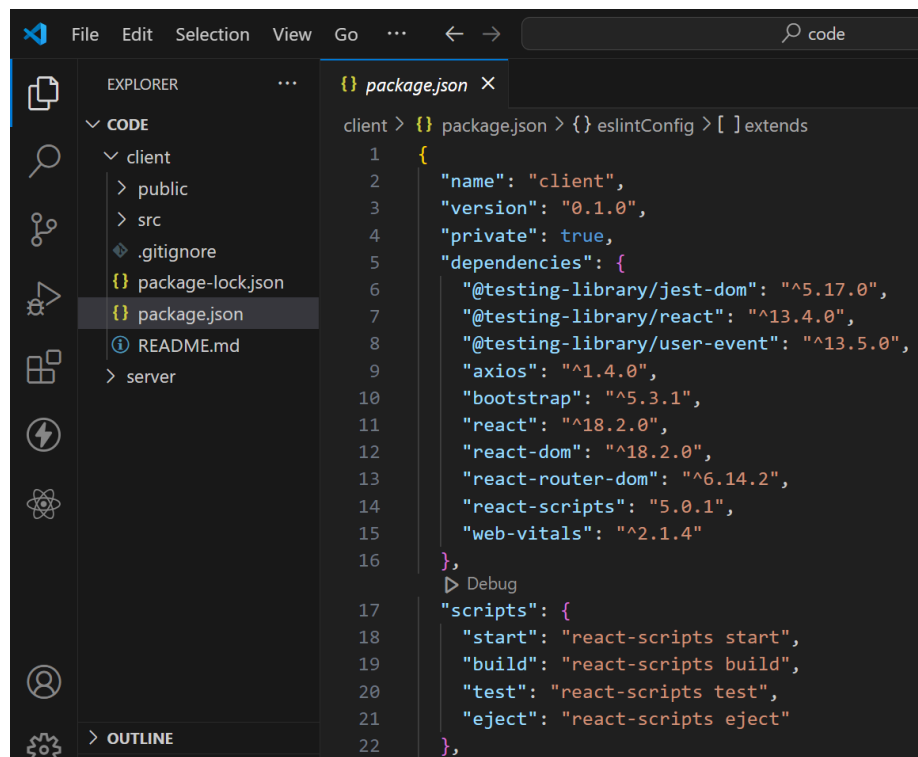
Installation of required tools:

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 with the Explorer sidebar on the left displaying a project structure. The 'client' folder is expanded, showing files like 'public', 'src', '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The 'package.json' file is selected and its content is displayed in the main editor. The code is as follows:

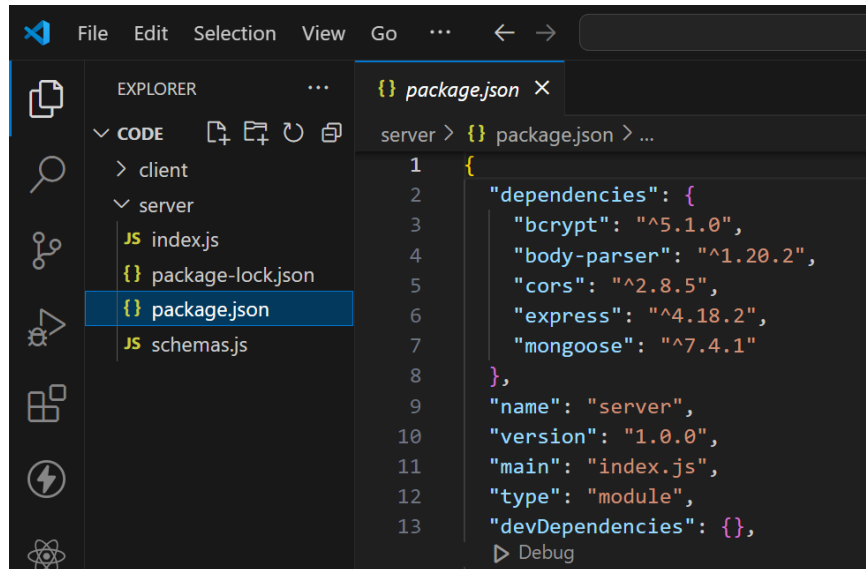
```
client > {} package.json > {} eslintConfig > [ ] extends
1  {
2    "name": "client",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.17.0",
7      "@testing-library/react": "^13.4.0",
8      "@testing-library/user-event": "^13.5.0",
9      "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.



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.

Seeding Sample Data (Optional):

- To quickly populate your database with sample flight data for testing, use the provided seed.js script in the [server](#) folder.
- Open a terminal, navigate to the [server](#) directory, and run:
node seed.js

- This script will insert sample flights into your MongoDB database, making it easier to test the application's features.

Note: Seeding the data is only used for testing purpose. We can add the data simply in the database using the above command without inserting all the flights one by one into the website.

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:

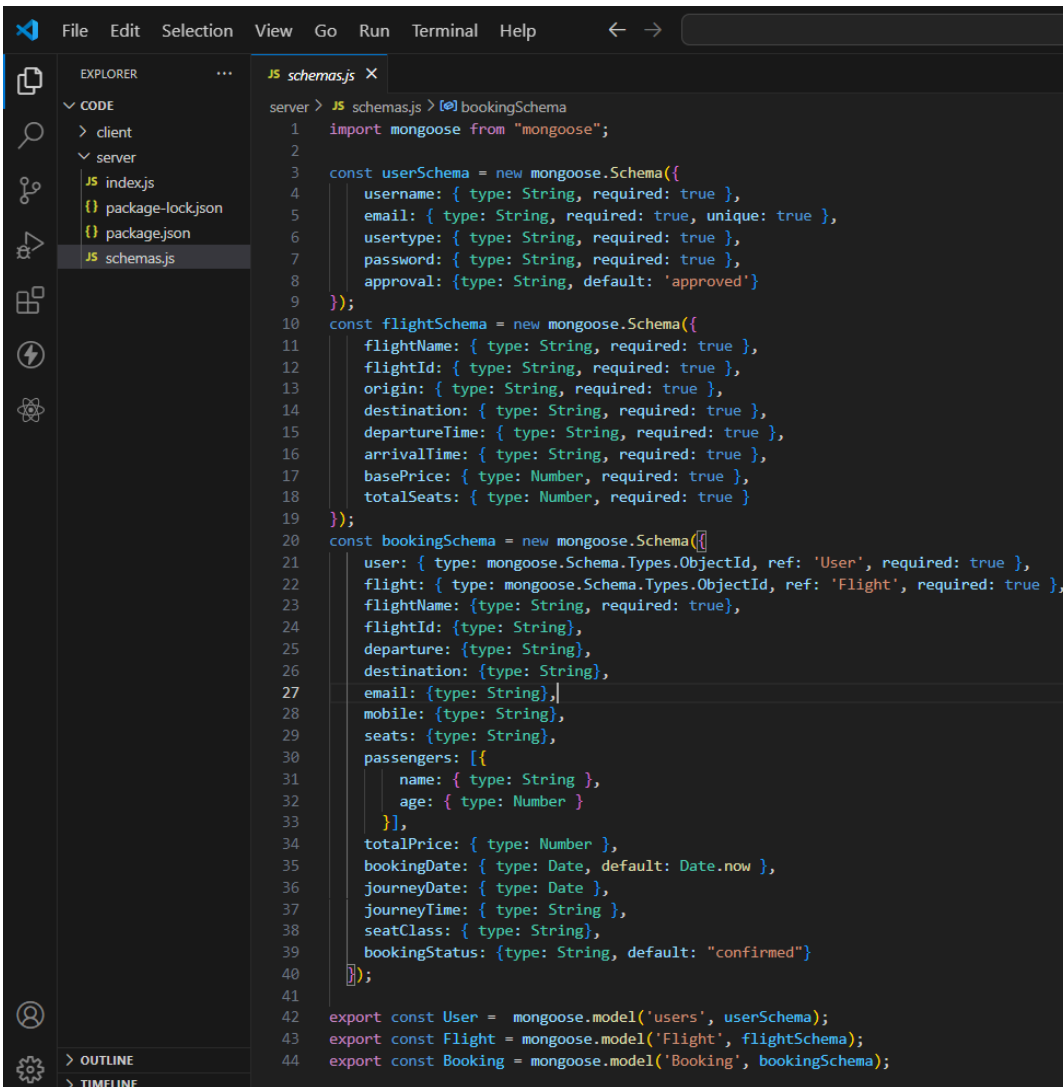
- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

Reference video for backend code:

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 editor interface. The Explorer sidebar on the left shows a project structure with folders 'client' and 'server'. Inside 'server', there are files 'index.js', 'package-lock.json', 'package.json', and 'schemas.js'. The 'schemas.js' file is selected and open in the editor. The code defines three Mongoose schemas: 'userSchema', 'flightSchema', and 'bookingSchema'. 'userSchema' has fields for username, email, userType, password, and approval. 'flightSchema' has fields for flightName, flightId, origin, destination, departureTime, arrivalTime, basePrice, and totalSeats. 'bookingSchema' has fields for user, flight, 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, three models are exported: '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.

```
//
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 httprequest to the server to add it to the database.

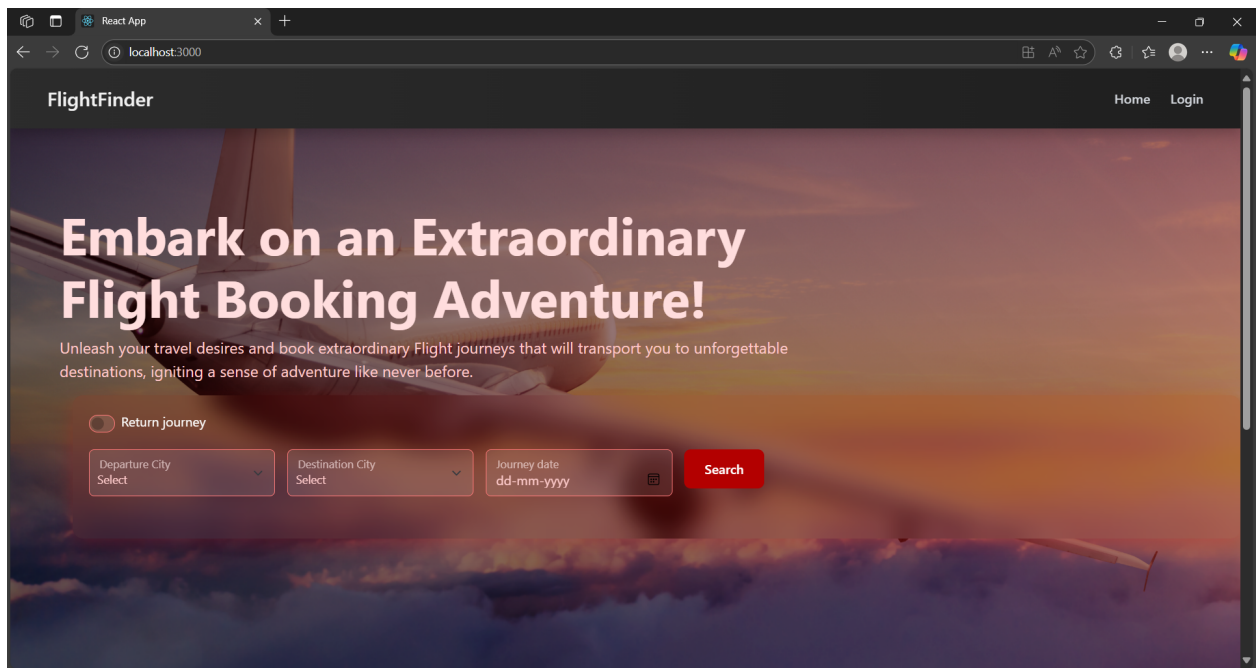
5. Update Flight:

- 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.

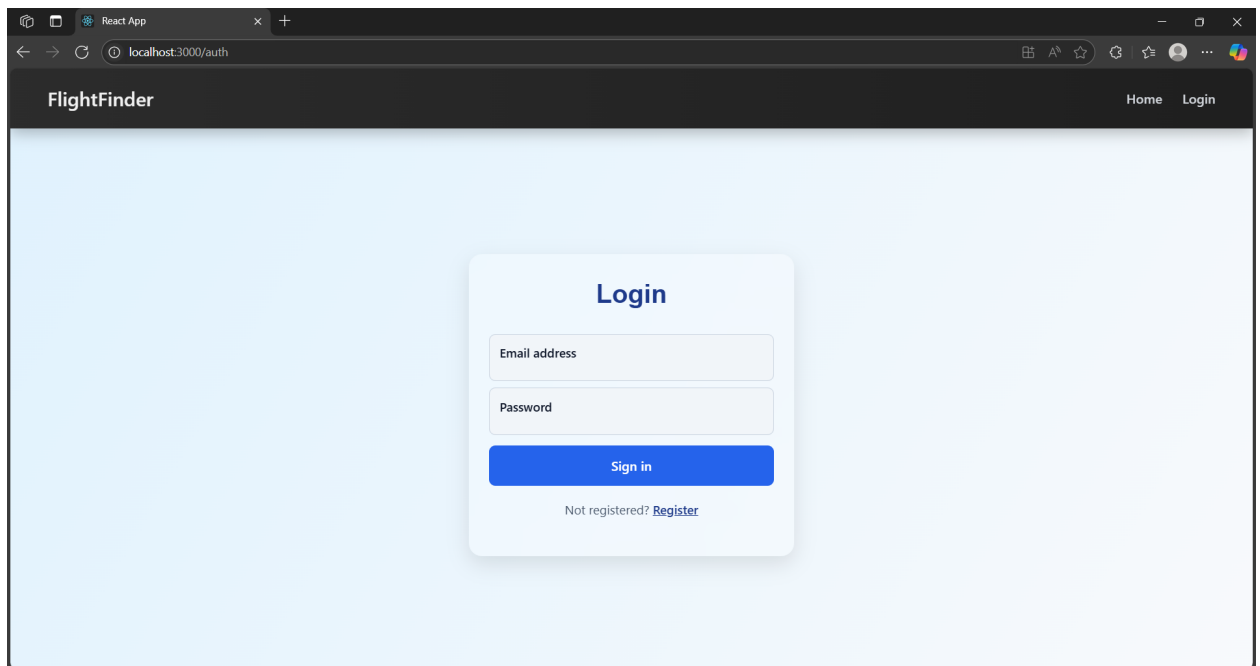
Milestone 5: Project Implementation.

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

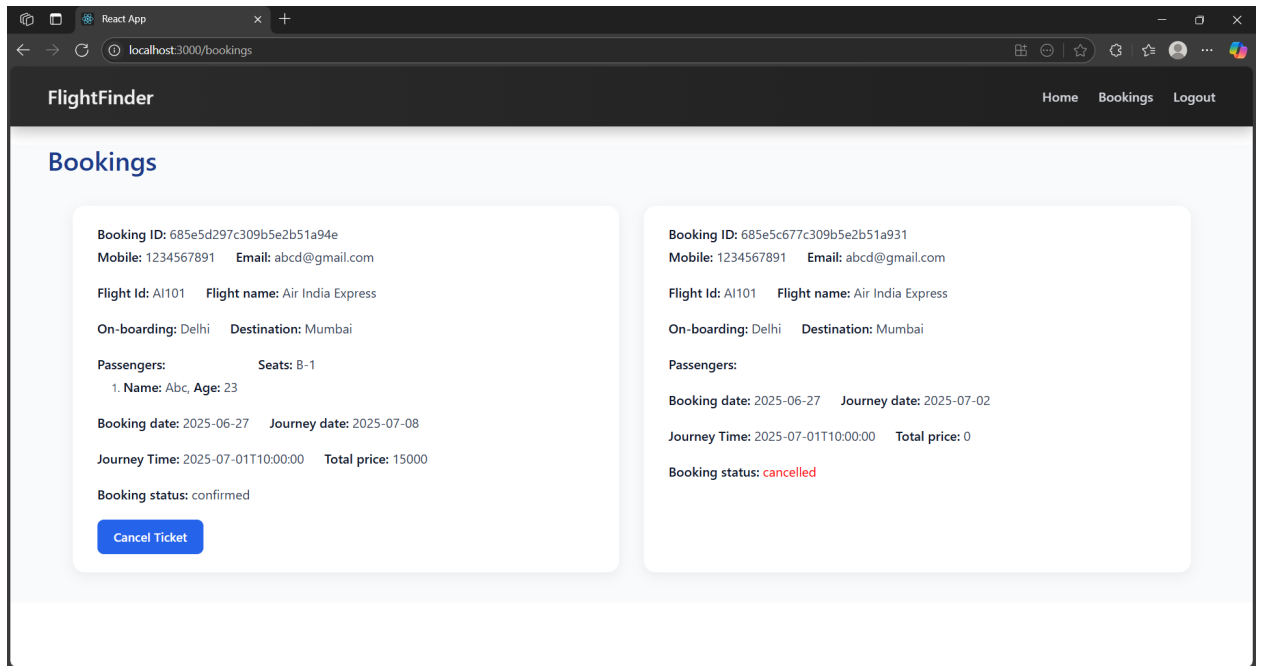
- **Landing page UI**



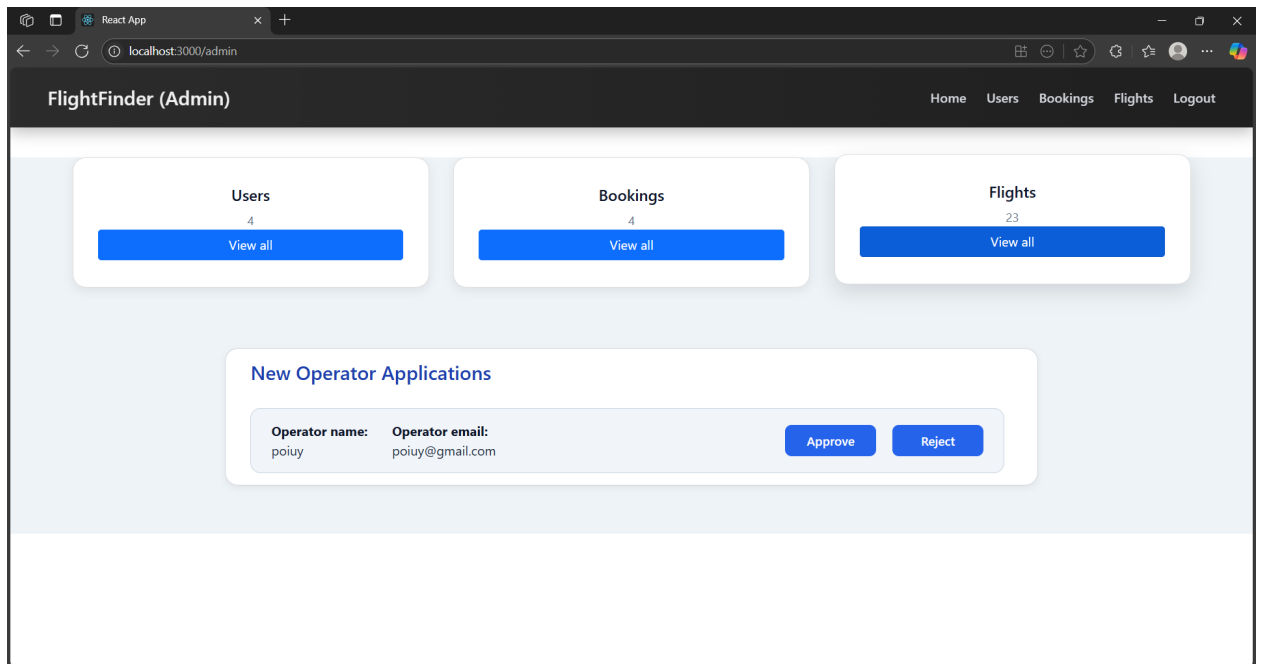
- **Authentication**



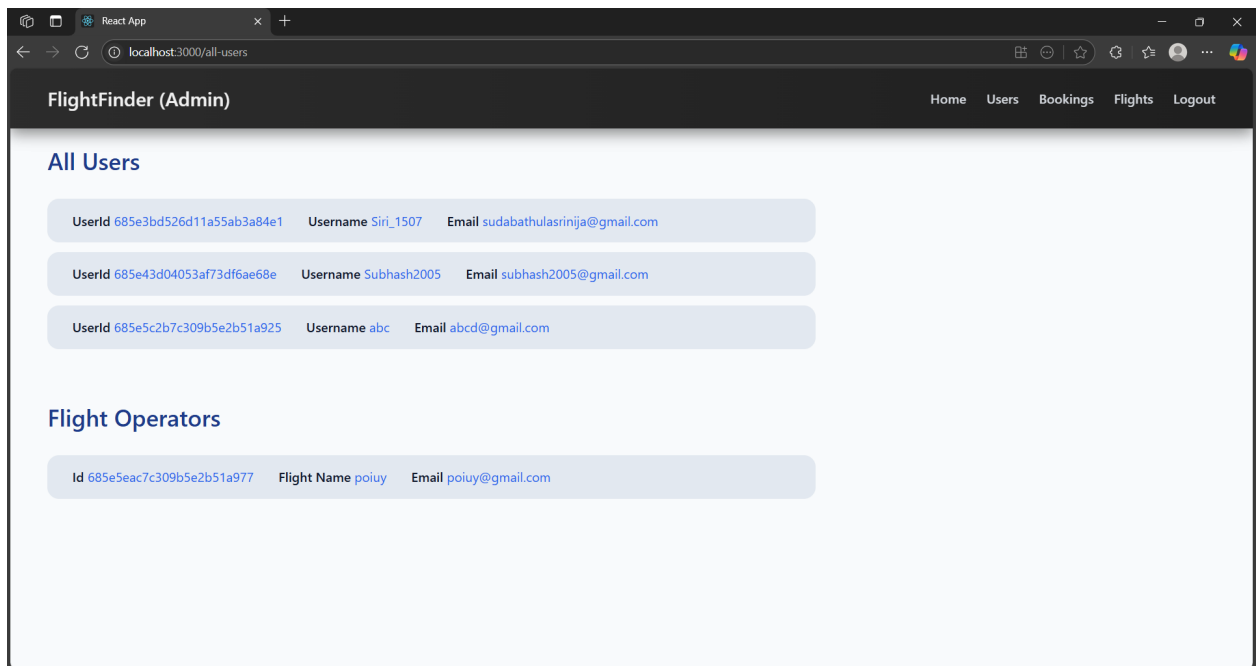
- **User bookings**



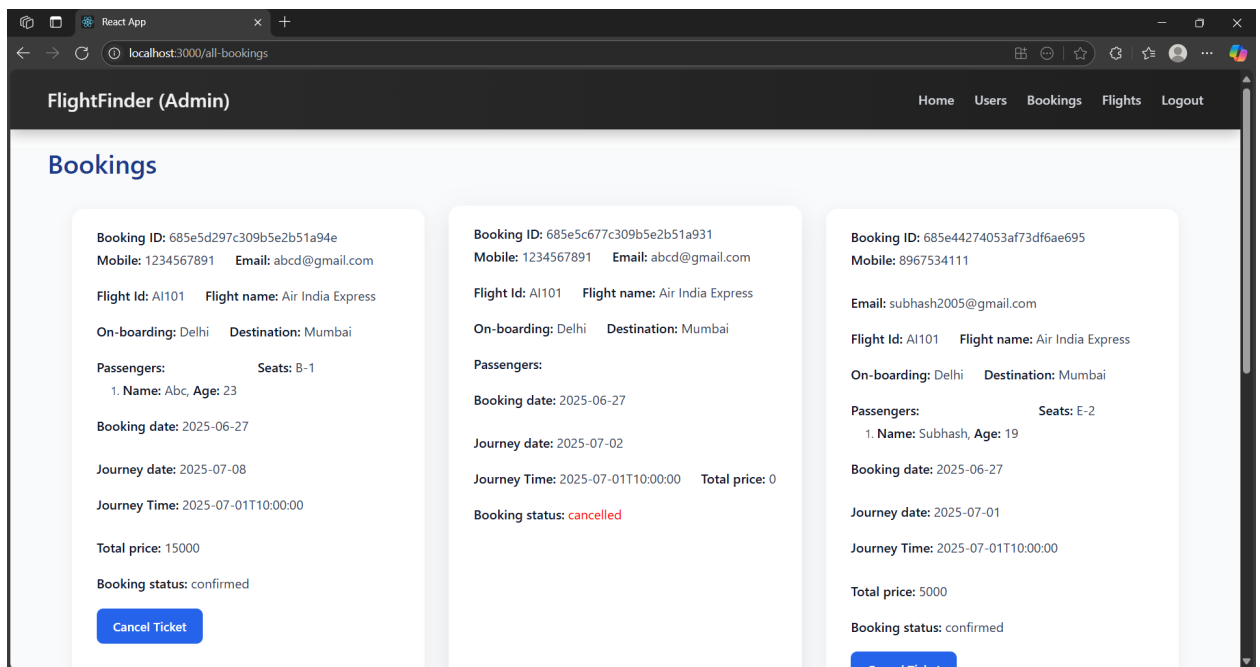
- Admin Dashboard



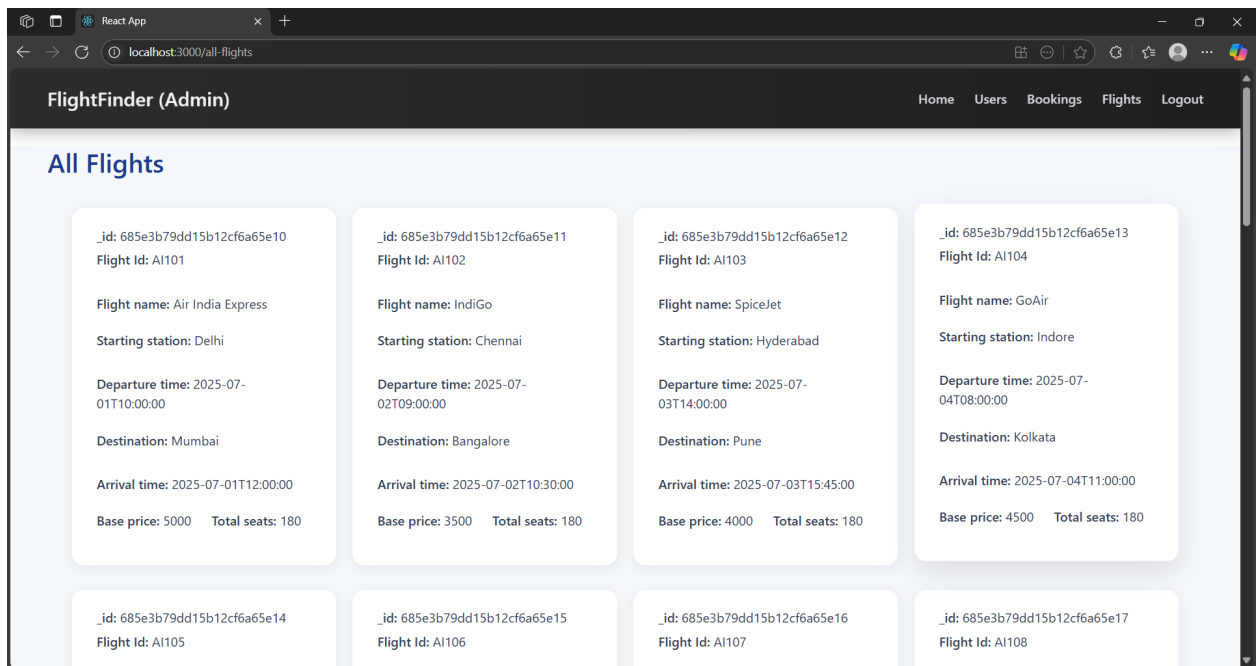
- All users and Flight Operators



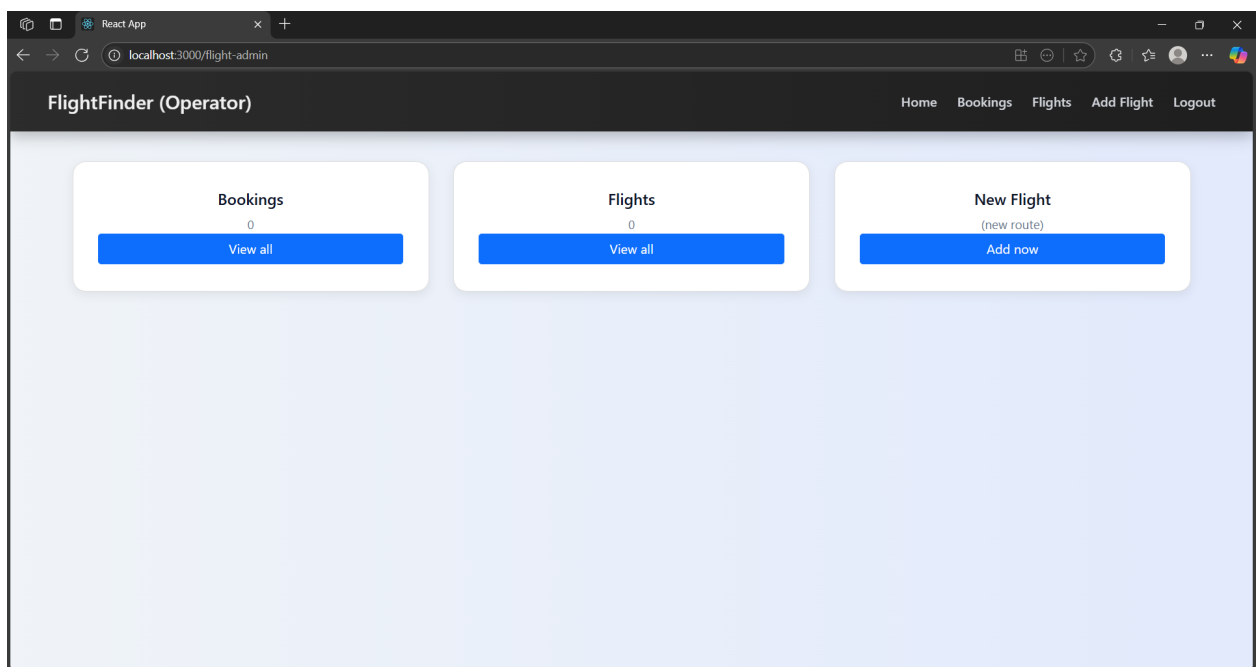
- All Bookings



- All Flights



- **Operator Dashboard (Appears only after the acceptance of Admin)**



- **New Flight**

React App

localhost:3000/new-flight

HomeBookingsFlightsAdd FlightLogout

Add new Flight

Flight Name

poiuy

Flight Id

Departure City

Select

Departure Time

--:--

Destination City

Select

Arrival time

--:--

Total seats

0

Base price

0

Add now