

# Flight Booking APP

## Flight Finder: Navigate your Air travel option

Team Members:

Here List team members and their roles:

1. *S. Srinija (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. *M. Subhash (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.

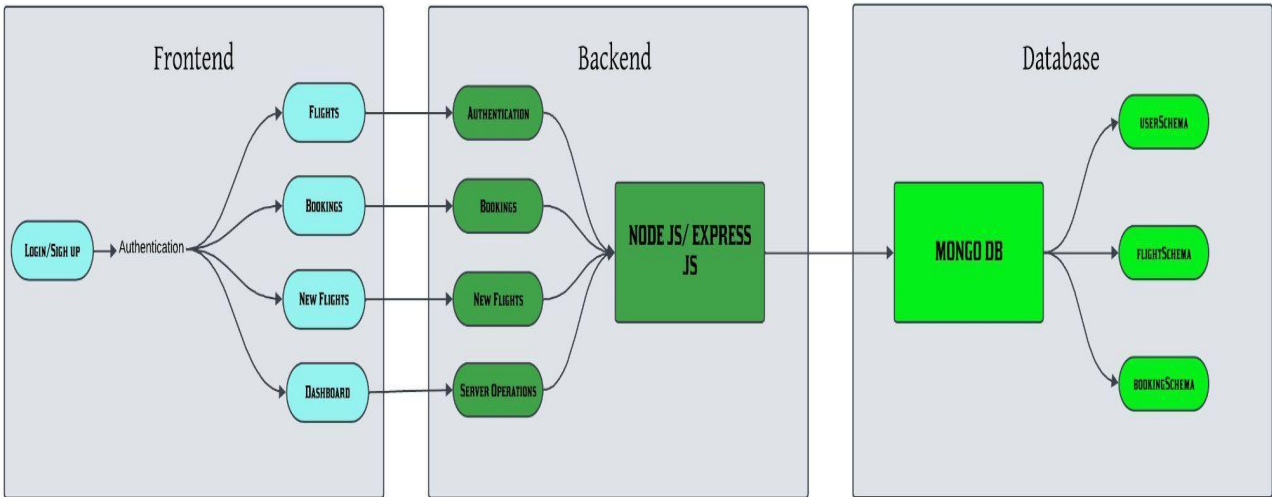
This scenario highlights how Flight Finder streamlines the entire travel process—offering users convenience, customization, and real-time support throughout their journey.

## TECHNICAL ARCHITECTURE

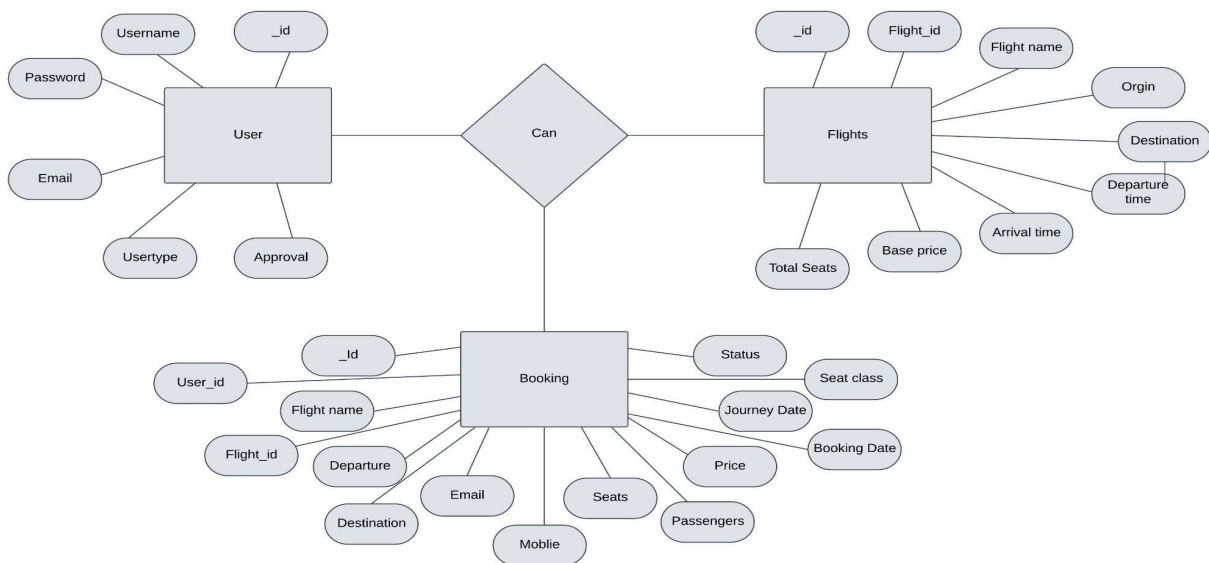
In the architecture of the Flight Finder application:

- 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/sudabathulasiri/FlightFinder>

**Running The FlightFinder App:**

1. Clone the repository: git clone <https://github.com/sudabathulasiri/FlightFinder.git>

*cd "project file" (or cd FlightFinder if that's your folder name)*

2. Set up the backend:

*cd Backend  
npm install*

3. Set up environment variables: Create a .env file inside the Backend folder with your MongoDB URI, JWT secret, and email credentials.
4. Start the backend server:

*npm run dev*

The backend will run on <http://localhost:5000>.

5. Open the frontend: Open **app.html** (for users) or **admin-login.html**(for admins) from the Frontend folder in your browser. Or use a local server (like VS Code Live Server) to serve the Frontend folder.

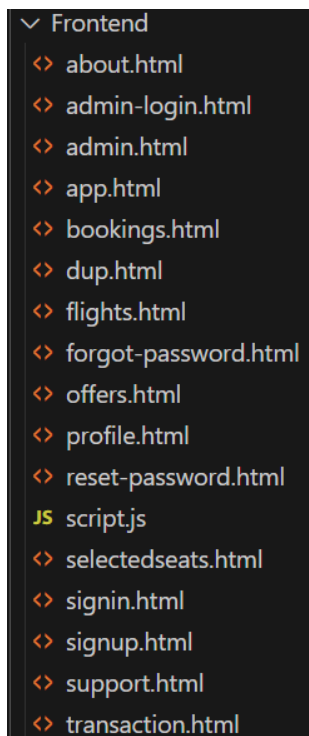
You have now set up the FlightFinder app on your local machine. You can proceed with further customization, development, and testing as needed.

## PROJECT STRUCTURE:

Inside the FlightFinder app directory, you will find the following main folders:

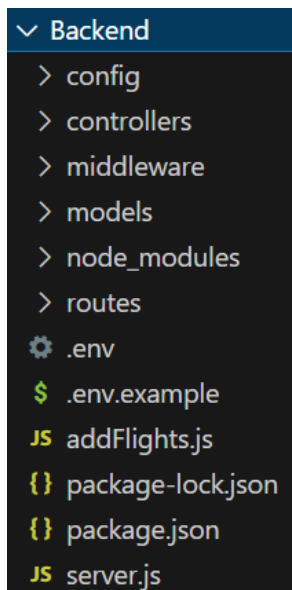
- **Frontend directory:** This folder contains all the client-side code (HTML, CSS, JavaScript) for the user interface.

Example structure:



- **Backend directory:** This folder contains all the server-side code (Node.js, Express.js, MongoDB) and API logic.

Example structure:



## 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:**

Create two main folders:

  - Frontend (for all HTML, CSS, JS, and assets)



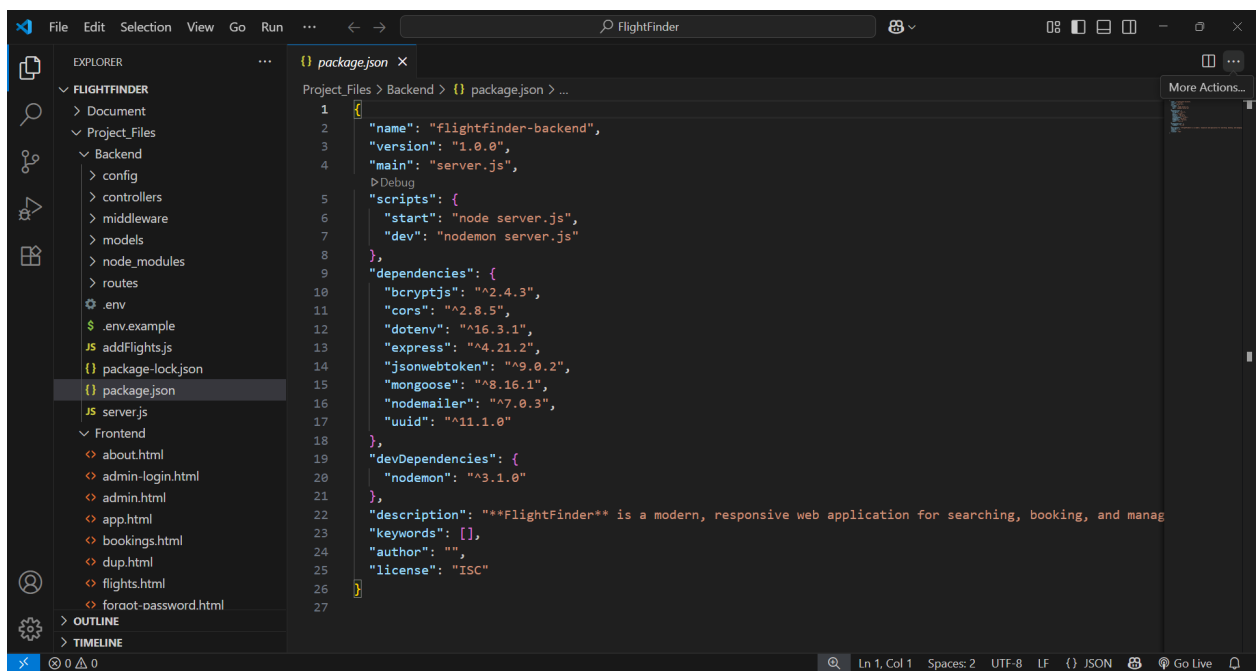
- Backend (for all server-side code and configuration)
- **Installation of required tools:**

**Frontend:** No installation needed for vanilla HTML/CSS/JS. (Optional: Use VS Code Live Server for local development.)

**Backend:** Open the Backend folder in your terminal and install the required libraries:

*npm install express mongoose dotenv bcryptjs cors nodemailer*

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 editor with the 'package.json' file open in the 'Backend' folder. The file contains the following JSON structure:

```
1 {
2   "name": "flightfinder-backend",
3   "version": "1.0.0",
4   "main": "server.js",
5   "scripts": {
6     "start": "node server.js",
7     "dev": "nodemon server.js"
8   },
9   "dependencies": {
10    "bcryptjs": "^2.4.3",
11    "cors": "^2.8.5",
12    "dotenv": "^16.3.1",
13    "express": "^4.21.2",
14    "jsonwebtoken": "^9.0.2",
15    "mongoose": "^8.16.1",
16    "nodemailer": "^7.0.3",
17    "uuid": "^11.1.0"
18  },
19  "devDependencies": {
20    "nodemon": "^3.1.0"
21  },
22  "description": "***FlightFinder** is a modern, responsive web application for searching, booking, and managing flights.",
23  "keywords": [],
24  "author": "",
25  "license": "ISC"
26 }
```

## Milestone 2: Backend Development:

1. **Database Configuration:** Set up a MongoDB database locally or using MongoDB Atlas. Define collections for users, flights, bookings, etc.
2. **Create Express.js Server:** Set up an Express.js server in server.js to handle HTTP requests and serve API endpoints. Configure middleware such as express.json() for parsing request bodies and cors for cross-origin requests.
3. **Define API Routes:** Create separate route files for different functionalities: flights, users, bookings, authentication. Implement route handlers using Express.js to interact with the database.

4. **Implement Data Models:** Define Mongoose schemas for users, flights, bookings, etc. Create corresponding Mongoose models and implement CRUD operations.
5. **Seeding Sample Data (Optional):** To quickly populate your database with sample flight data for testing, use a seed script (e.g., addFlights.js). Run: `node addFlights.js` This will insert sample flights into your MongoDB database for testing.
6. **User Authentication:** Create routes and middleware for user registration, login, and logout. Use JWT for authentication and protect routes that require login.
7. **Handle Flights and Bookings:** Create routes and controllers to handle flight listings and bookings. Implement booking functionality, including seat selection, validation, and payment simulation.
8. **Admin Functionality:** Implement routes and controllers for admin-specific actions (adding flights, managing bookings, etc.). Add authentication and authorization checks to ensure only admins can access these routes.
9. **Error Handling:** Implement error handling middleware to catch and handle errors during API requests. Return appropriate error responses with relevant messages and HTTP status codes.

This structure and flow will help you organize, develop, and maintain your FlightFinder app efficiently

### Milestone 3: Database development

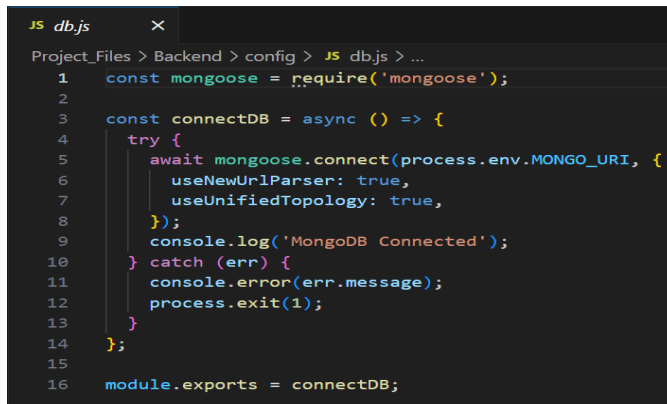
- **Configure schema:**

Define Mongoose schemas for your MongoDB collections (users, flights, bookings, etc.). Use your ER diagram as a reference to create these schemas. Example schemas:

- User: name, email, password, role, etc.
- Flight: airline, flight number, from, to, departure, arrival, seats, price, etc.
- Booking: user, flight, seats, booking date, status, etc.

- **Connect Database to Backend:**

Ensure your backend connects to MongoDB before handling any requests. Example connection code (in Backend/config/db.js):



```

JS db.js
Project_Files > Backend > config > JS db.js > ...
1  const mongoose = require('mongoose');
2
3  const connectDB = async () => {
4    try {
5      await mongoose.connect(process.env.MONGO_URI, {
6        useNewUrlParser: true,
7        useUnifiedTopology: true,
8      });
9      console.log('MongoDB Connected');
10     } catch (err) {
11       console.error(err.message);
12       process.exit(1);
13     }
14   };
15
16   module.exports = connectDB;

```

Call connectDB() in your server.js before starting the server.

## Milestone 4: Frontend development.

### 1. Login/Register:

- Create HTML pages (signin.html, signup.html) with forms for user registration and login.
- Use JavaScript to handle form submissions and send data to the backend via fetch POST requests.
- On successful login, store the JWT token in localStorage and redirect users/admins to their respective home or dashboard pages.

### 2. Flight Search and Booking (User):

- In app.html, provide a search form for users to enter departure city, destination, date, class, and number of passengers.
- On form submission, use JavaScript to redirect to flights.html with the search parameters in the URL.
- In flights.html, use JavaScript to fetch available flights from the backend based on the search criteria and display them.
- Each flight listing includes a "Book" button that allows users to select seats and proceed with booking.

### 3. Managing User Bookings:

- In mybookings.html (or bookings.html), display the user's current and past bookings by fetching data from the backend.
- Provide an option to cancel bookings, which sends a request to the backend to update the booking status.

### 4. Admin Features:

- In admin.html, provide forms for admins to add new flights, update existing flights, and manage bookings.
- Use JavaScript to send requests to the backend for adding, updating, or deleting flights.

- Display all flights, bookings, and users in the admin dashboard for management purposes.

#### 5. Profile and Support:

- In profile.html, allow users to view and update their profile information.
- In support.html, provide a contact form or FAQ section for user support.

#### 6. Password Reset:

- In forgot-password.html, allow users to enter their email to receive a reset code.
- In reset-password.html, allow users to enter the code and set a new password.

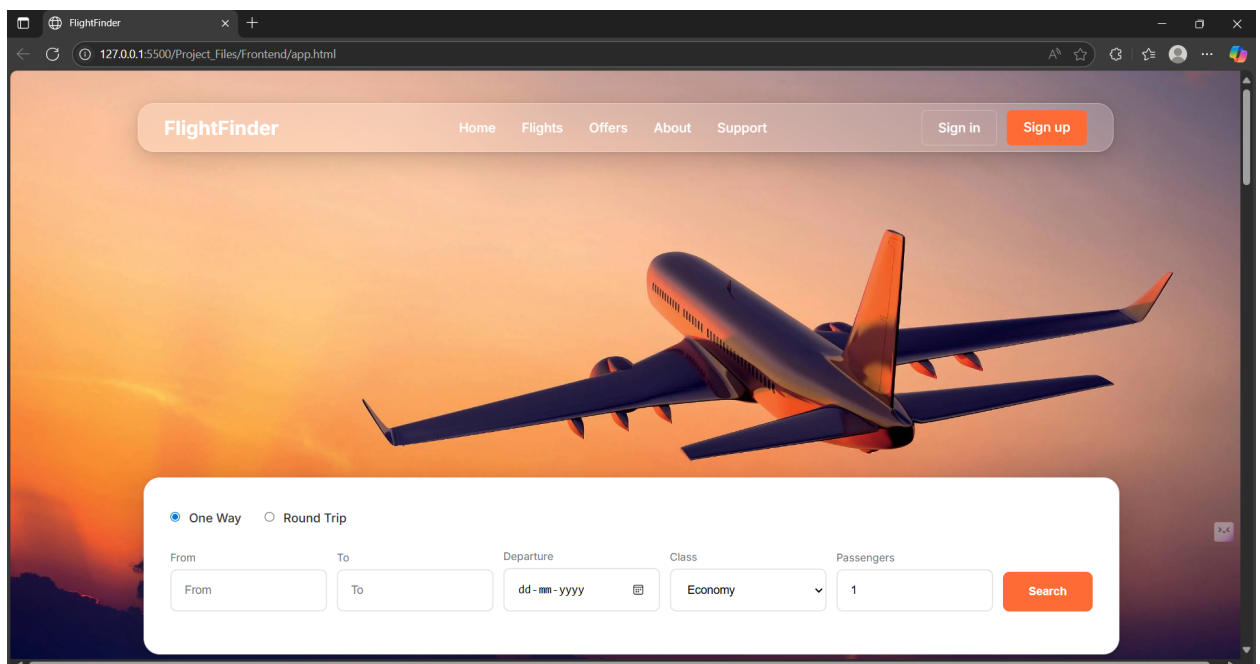
#### 7. Navigation and User Experience:

- Use JavaScript to dynamically update navigation links and show/hide options based on whether the user is logged in.
- Use multiple HTML pages for different features, and update the DOM with JavaScript as needed.

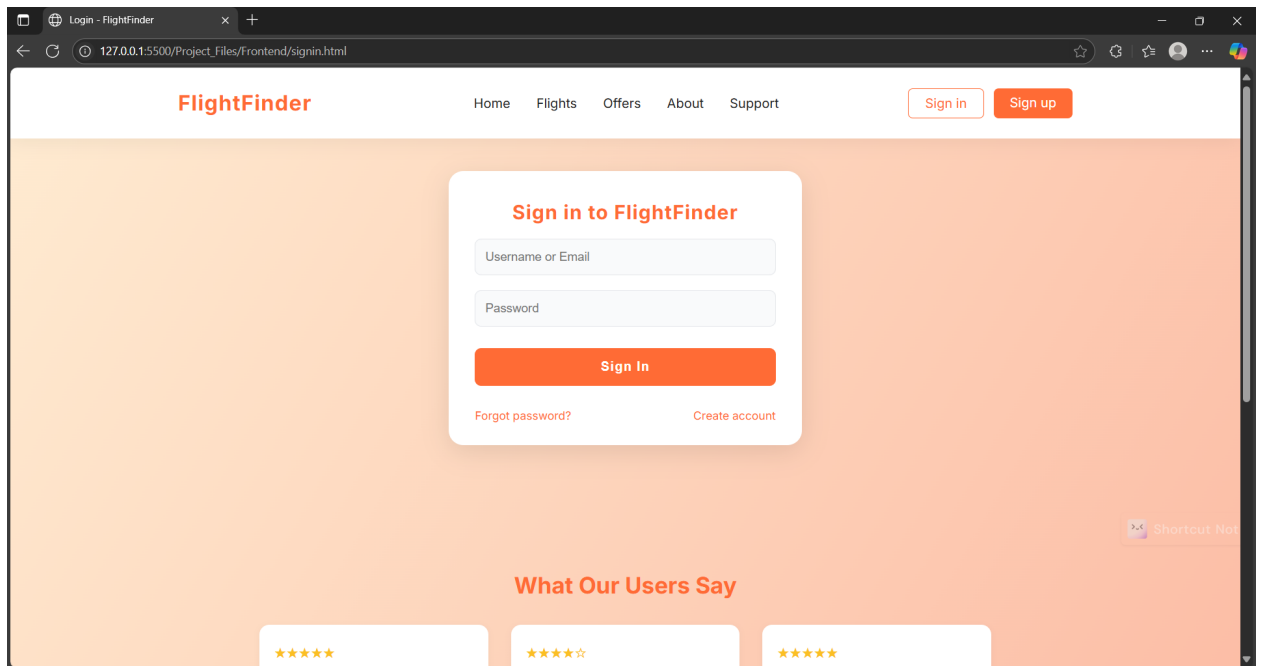
### Milestone 5: Project Implementation.

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

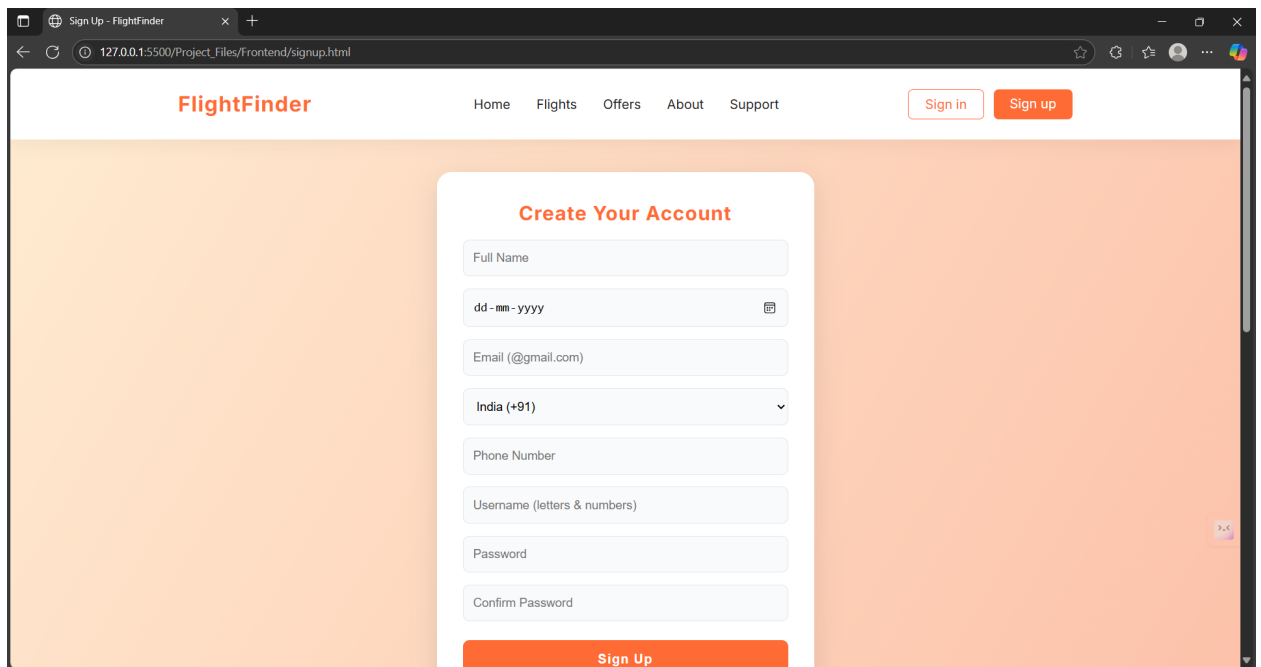
- Landing page UI



- Sign in and Sign up



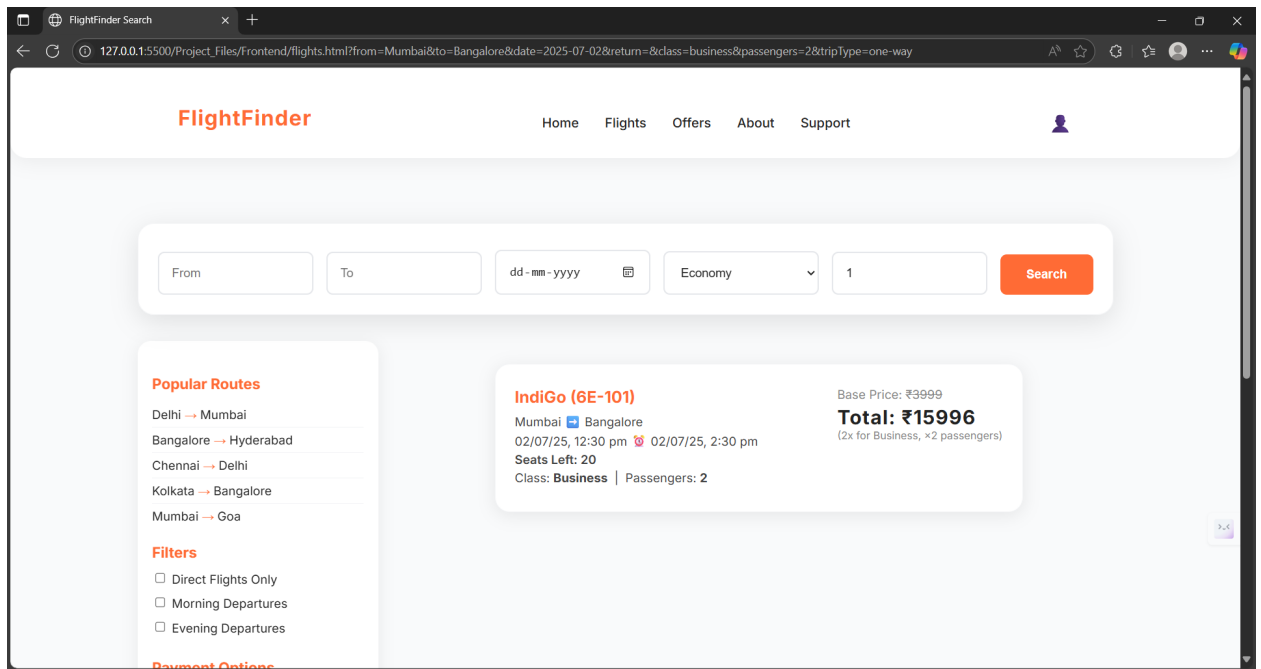
The screenshot shows the 'Sign in to FlightFinder' page. The browser address bar displays '127.0.0.1:5500/Project\_Files/Frontend/signin.html'. The page features a navigation bar with 'Home', 'Flights', 'Offers', 'About', and 'Support' links, along with 'Sign in' and 'Sign up' buttons. The main content area has a light orange background. A white card in the center contains the title 'Sign in to FlightFinder', input fields for 'Username or Email' and 'Password', a 'Sign in' button, and links for 'Forgot password?' and 'Create account'. At the bottom, a section titled 'What Our Users Say' shows three star ratings: '★★★★★', '★★★★☆', and '★★★★★'. A 'Shortcut Not' notification is visible in the bottom right corner.



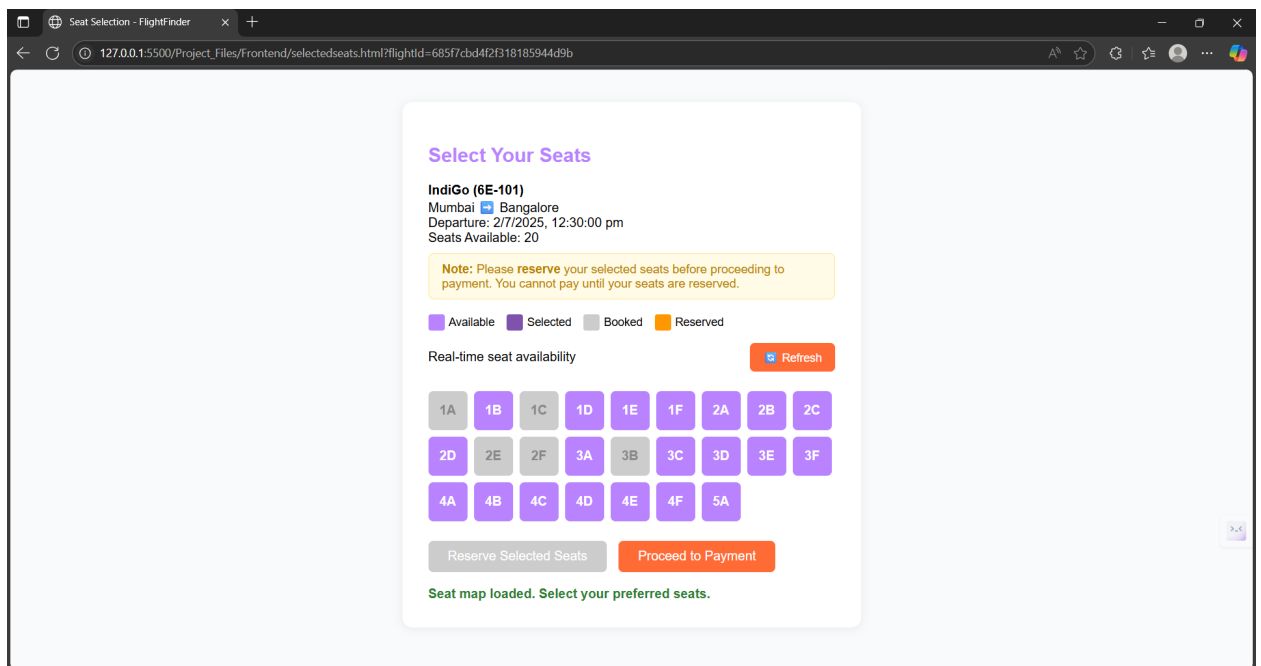
The screenshot shows the 'Create Your Account' page. The browser address bar displays '127.0.0.1:5500/Project\_Files/Frontend/signup.html'. The page layout is consistent with the sign-in page. The main content area has a light orange background. A white card in the center contains the title 'Create Your Account' and several input fields: 'Full Name', 'dd-mm-yyyy' (with a calendar icon), 'Email (@gmail.com)', 'India (+91)' (with a dropdown arrow), 'Phone Number', 'Username (letters & numbers)', 'Password', and 'Confirm Password'. A 'Sign Up' button is at the bottom of the card. The navigation bar and 'What Our Users Say' section are also present.

**Note :** When the admin is signing in the page will be redirected to admin page.

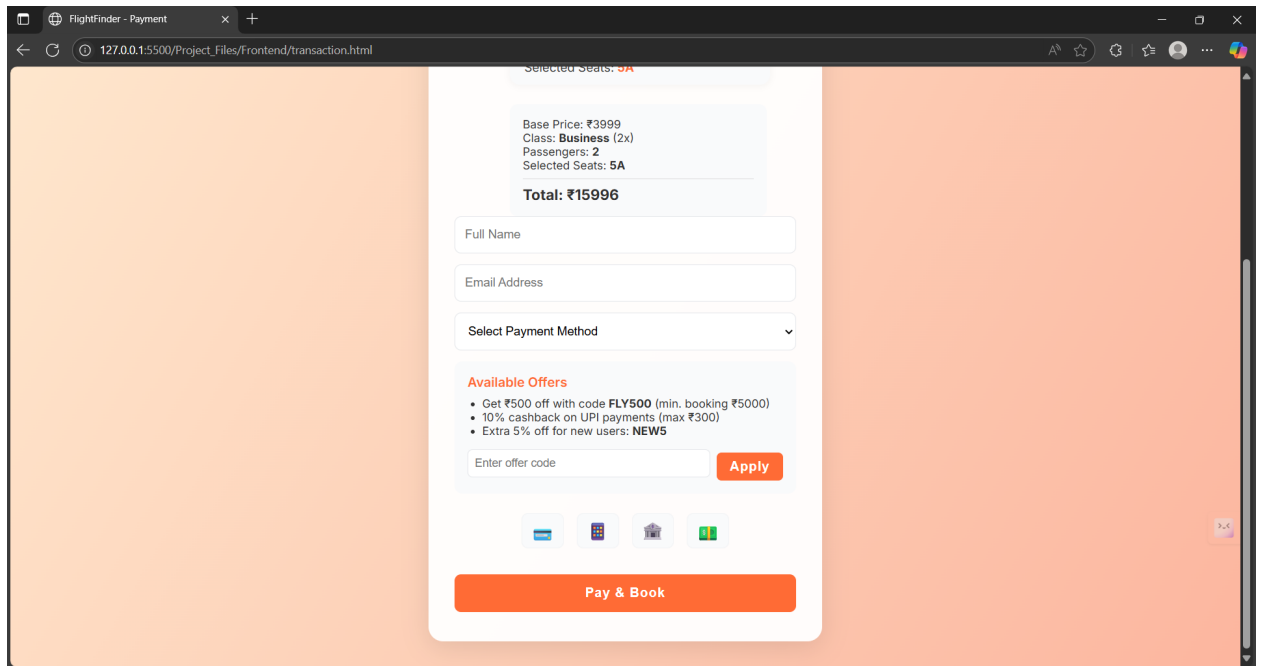
- Listing of Files



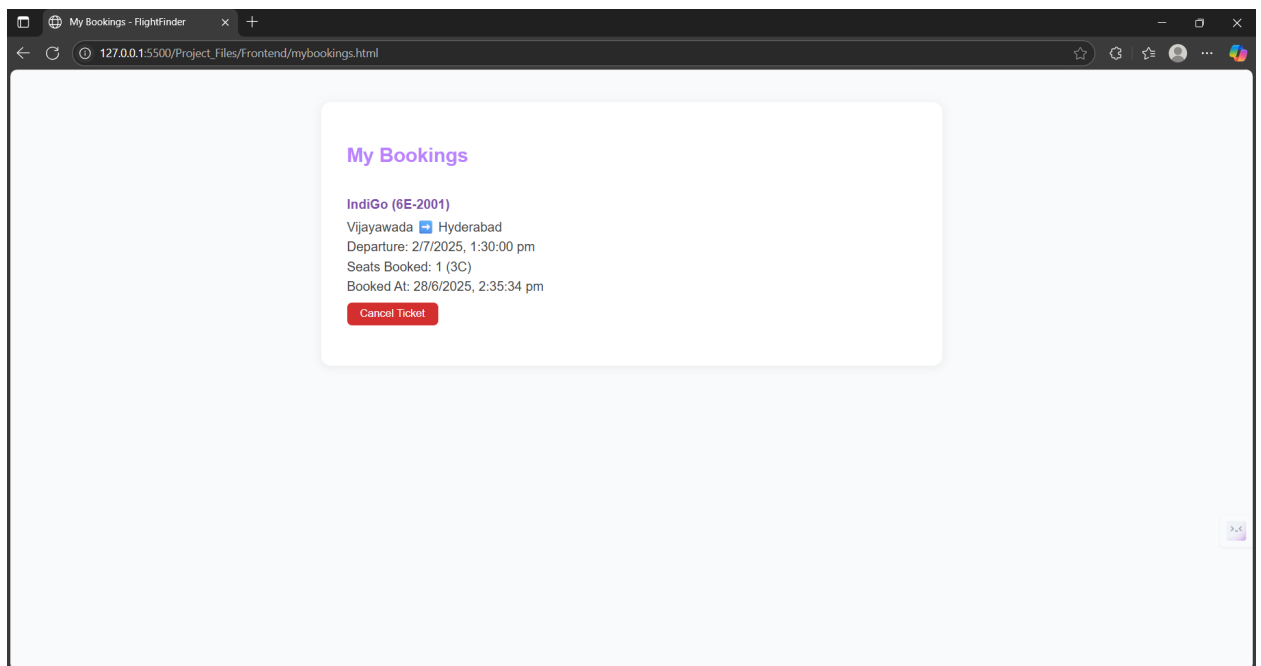
- **Seat Selection**



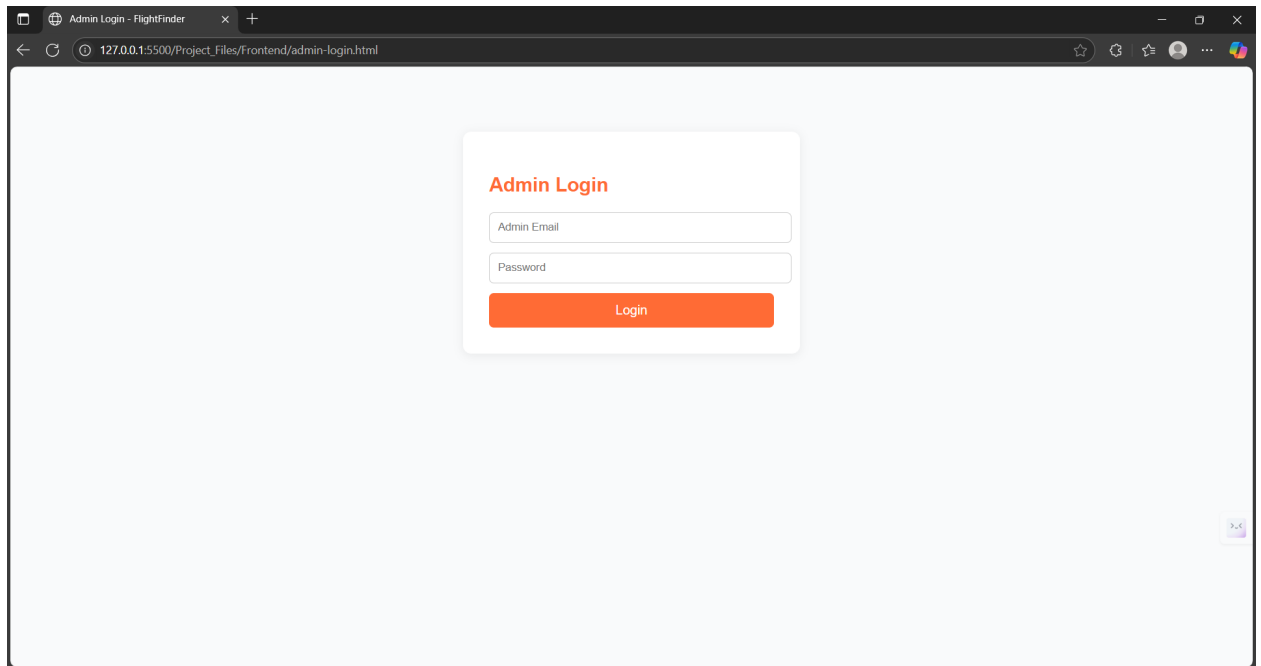
- **Payment and Booking**



- **Bookings Page**



- **Admin-login**



**Note:** we can login as admin either from admin-login.html or signin.html. When we want to register as a new admin, just signup/signin first and then go to mongodb shell and type the prompt(you can see the full version in README.md).

- **Admin Panel**

