Flight Finder Booking App

Project Documentation

Introduction

The Flight Finder Booking App is a full-stack web application designed to allow users to search and book flights conveniently. It supports user authentication, flight listings, bookings, and an admin panel for managing flights. Built using the MERN (MongoDB, Express.js, React.js, Node.js) stack, it is responsive and scalable.

DESCRIPTION

The Flight Finder Booking App simplifies air travel booking with real-time flight data, intuitive UI, and secure payment integration. It connects travelers with available flights and provides admin control over flight schedules and bookings.

Objectives:

1. **Real-Time Flight Listings** – Accurate availability, fares, and route info.
2. **User-Friendly Search Filters** – Flexible search by date, route, airline.
3. **Seamless Booking Experience** – Quick checkout, secure confirmation.
4. **Admin Control Panel** – Manage flights, bookings, users.
5. **Transparent System** – Clear pricing, booking status, and notifications.
6. **24/7 Accessibility** – Responsive UI for anytime, anywhere usage.

Features

Flight Search:

- Filter by **date, source, destination, airline**
- View **non-stop or connecting flights**
- Sort by **price, duration, departure time**

Booking System:

- Select seat type: **Economy / Business**
- Enter passenger details
- **Real-time availability check**
- Get **booking confirmation and PNR**

Booking Management:

- View **booking history**
- **Cancel or reschedule** (if allowed)
- Show **boarding details**

Alerts & Notifications:

- Flight status updates
- Booking confirmation via email
- Delay/reschedule alerts

Admin Panel:

- Add/update/remove flights
- View all user bookings
- Manage airline data and routes

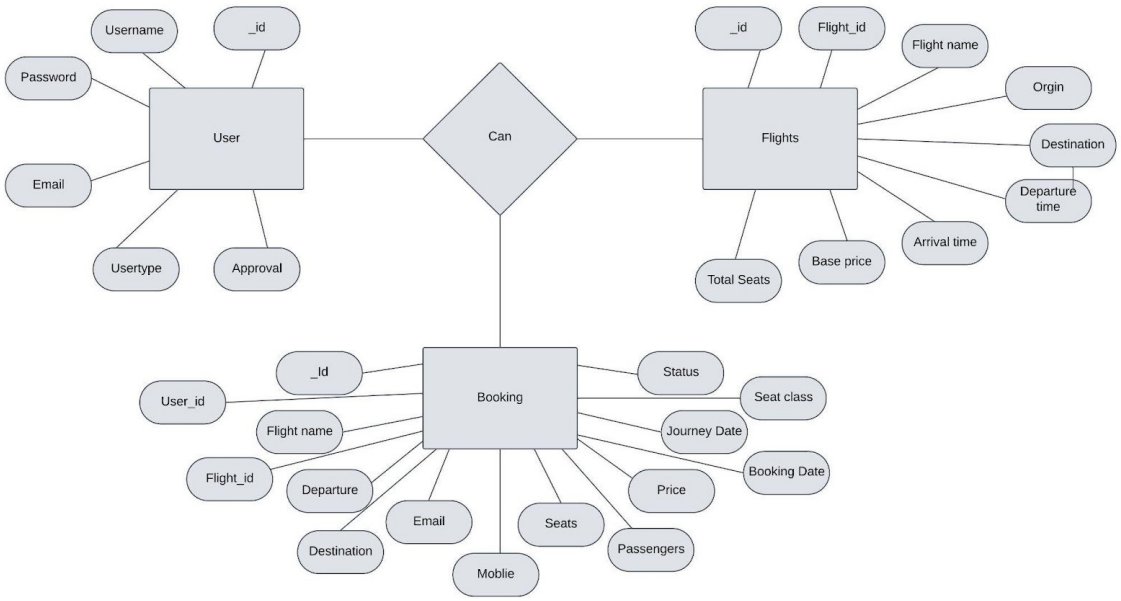
Tech Stack

- Frontend: React.js, styled using TailwindCSS or Bootstrap

- Backend: Node.js with Express.js

- Database: MongoDB
- Authentication: JWT (JSON Web Tokens)

ER DIAGRAM



PRE REQUISITES

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://gitscm.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 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/harsha-vardhan-reddy-07/Flight-Booking-App-MERN

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.

ROLES AND RESPONSIBILITIES

1. Traveler / User

Description: Regular users of the app who search and book flights.

Responsibilities:

- Register & login securely with email and password.
- View all available flights listed on the platform.
- Use filters to search flights by:
    - Source and destination cities
    - Date of travel
    - Airline name
    - Non-stop vs connecting flights
- View detailed flight info:
    - Flight number, departure/arrival time, airline name
    - Seat availability (economy/business), price
- Book tickets by:
    - Selecting desired flight
    - Entering passenger details
    - Choosing seat type
    - Submitting booking
- Get booking confirmation with a unique PNR number.

- View and manage their booking history:
    - See ticket status (Confirmed, Cancelled)
    - Cancel booking (if allowed)
- Receive email alerts for:
    - Booking confirmation
    - Flight rescheduling or cancellation
    - PNR updates

2. **Admin**

Description: Admin controls the entire platform and oversees data integrity.

Responsibilities:

- Login securely to access Admin Dashboard.
- Manage Flights:
    - Add new flights with complete details
    - Update existing flights (e.g., change time, price)
    - Remove outdated flights
- View all bookings made by users.
- View all registered users.
- View list of all flight routes and their statistics (optional).
- Verify or approve airline operators (if that role is added).
- Handle support issues and take user queries (optional).
- Enforce platform rules and ensure data consistency.
- Manage airline details (name, code, logo, etc.)
- Monitor for fake or duplicate bookings.
- Generate analytics (number of bookings per route/date).
- Reset database (optional: only in development environment).

3. **Airline Operator**

Description: Specific airline staff who can manage flights for their own airline only.

Responsibilities:

- Register and wait for Admin approval.
- Once approved:
    - Add flights for their airline
    - Update or cancel their flights
    - Monitor passenger lists

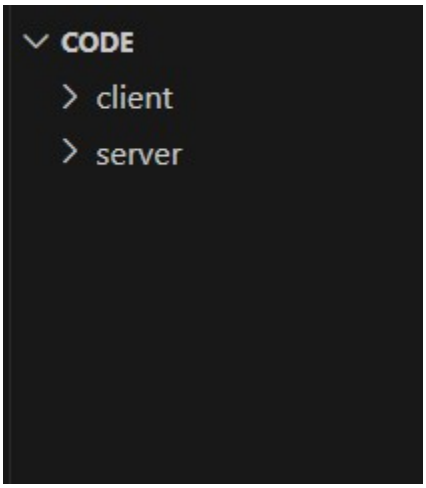
    - Change flight availability status
- Cannot access data from other airlines.
- Can view analytics related to their flights (bookings, routes).
- Useful if you want airlines to directly list flights on your platform like a marketplace.

Project Structure

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

client/ (React frontend)

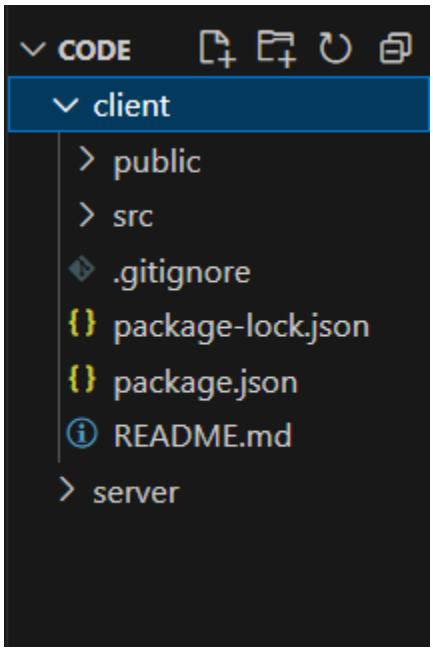
server/ (Node.js backend)

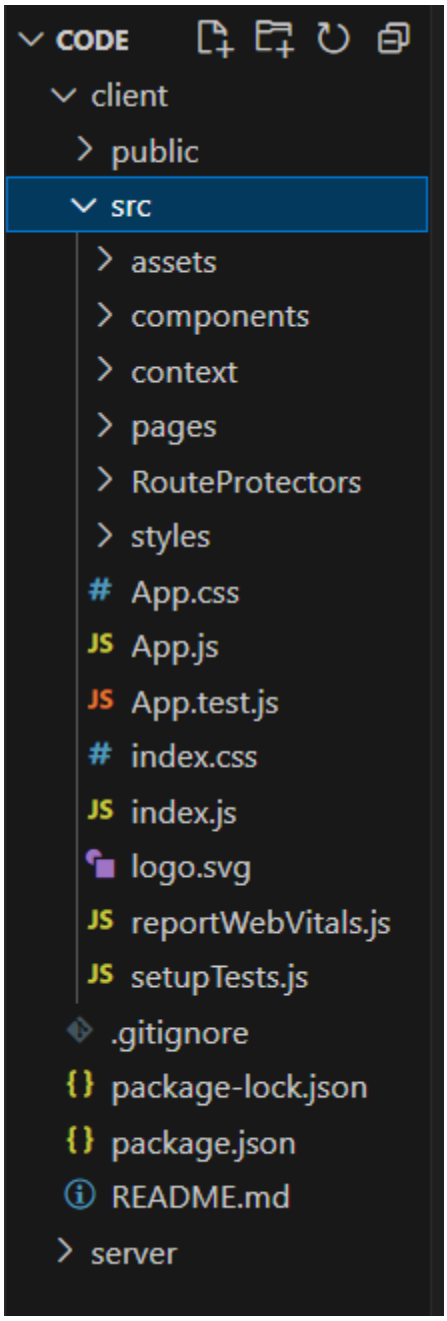


Each part is separated for clarity and maintainability.

Client directory:

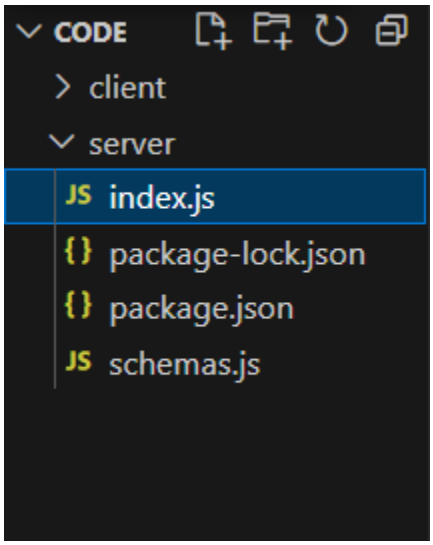
The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api’s.


CODE
client
public
src
.gitignore
package-lock.json
package.json
README.md
server



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.



Installation & Setup

- **Folder setup:**

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

- client
- Server

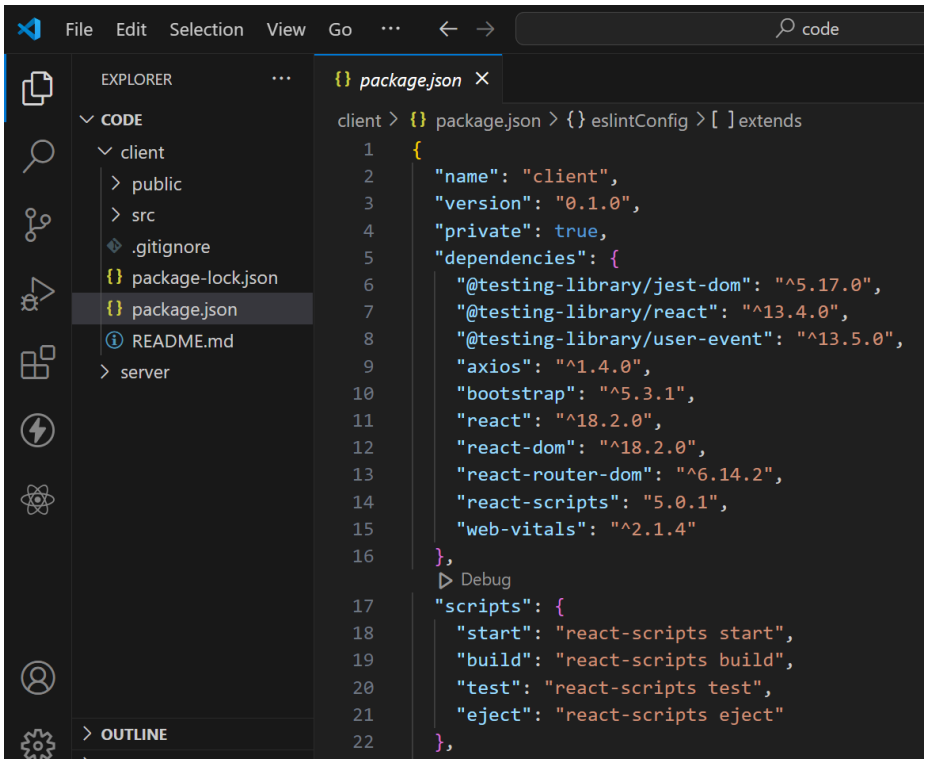
- **Installation of required tools:**

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.

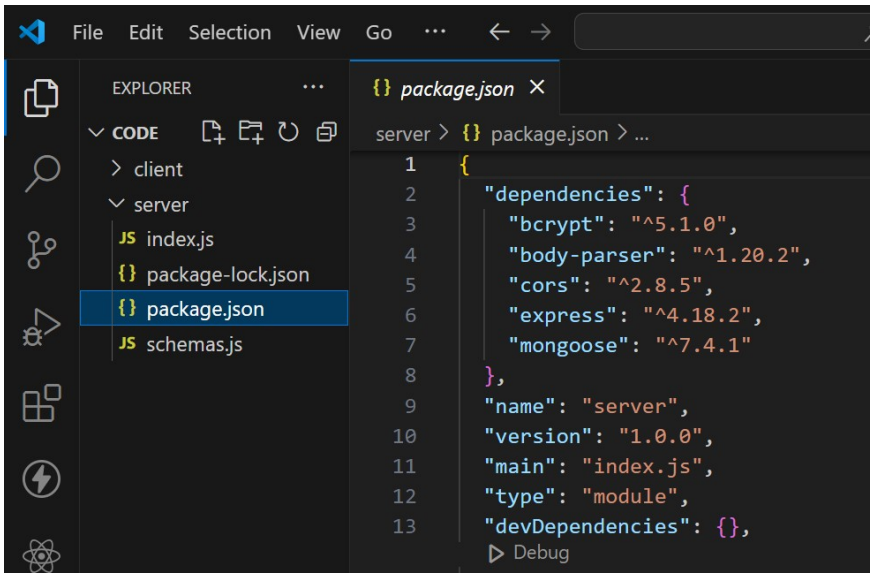


Now, open the backend folder to install all the necessary tools that we use in the backend.

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.



❖ 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.

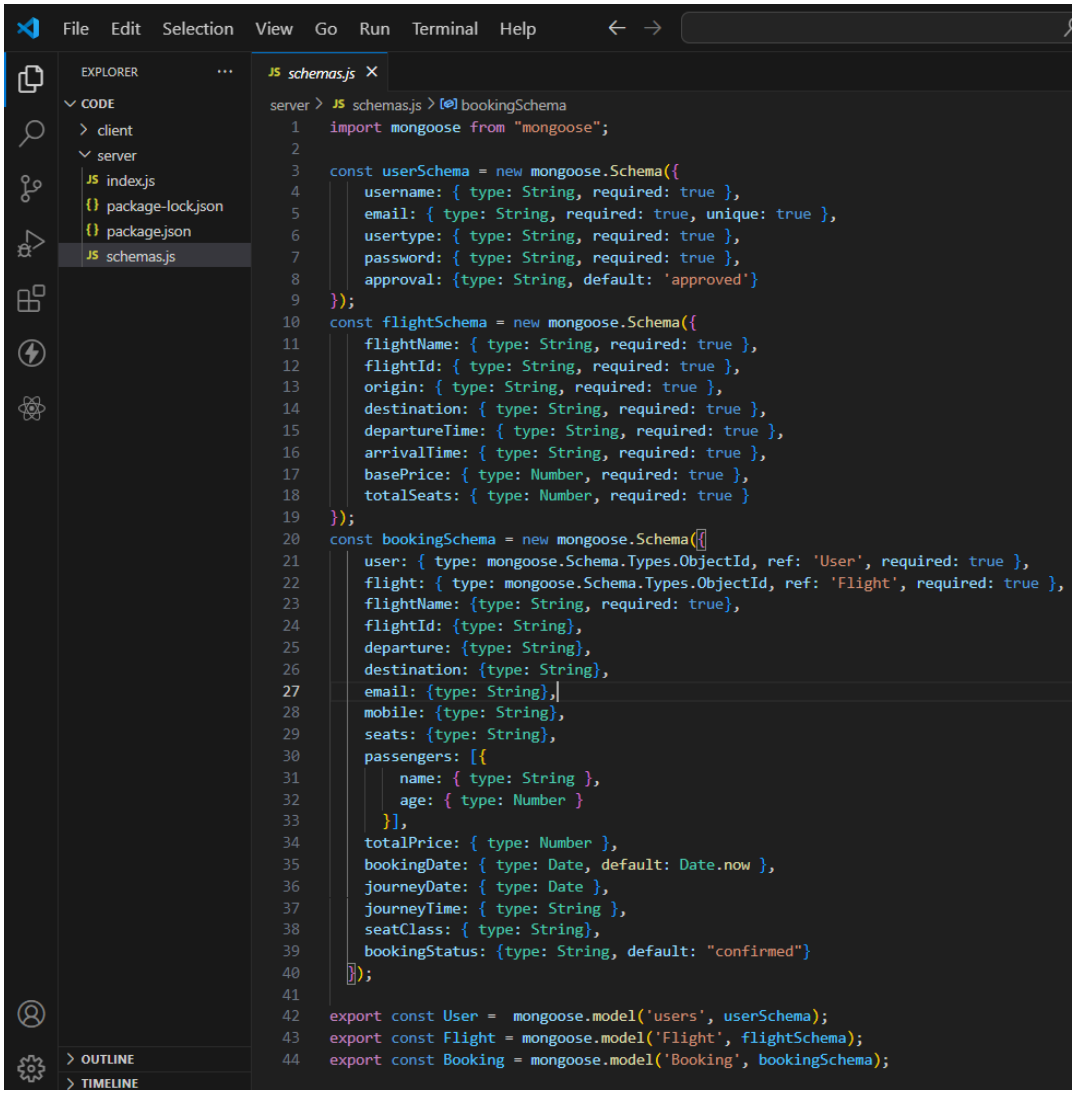
8.Error Handling:

- 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.

❖ **Database development**

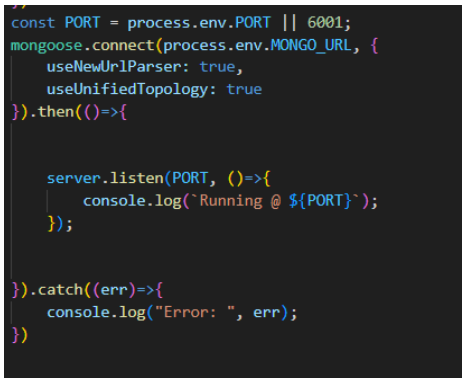
- **Configure schema**

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.



- **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);
})
```

❖ **Frontend development**

1. **Login/Register**

- Create a Component which contains a form for taking the username and password.
- If the given inputs matches 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.

`How It Works

- Users can search for flights based on criteria.
- Available flights are displayed with relevant details.
- Bookings reduce the available seat count.
- JWT tokens are used to manage secure sessions.

Database Models

User:

- name, email, password, role

Flight:

- operator, source, destination, date, time, seatsAvailable, price

Booking:

- userId, flightId, seats, bookedAt

API Endpoints

Auth:

- POST /api/auth/register
- POST /api/auth/login

Flights:

- GET /api/flights
- POST/PUT/DELETE /api/flights (Admin only)

Bookings:

- POST /api/bookings
- GET /api/bookings (User)
- GET /api/bookings/all (Admin)

Admin Access

Admins are identified by the `role` field in the user schema. They can manage all flight data and view all bookings.

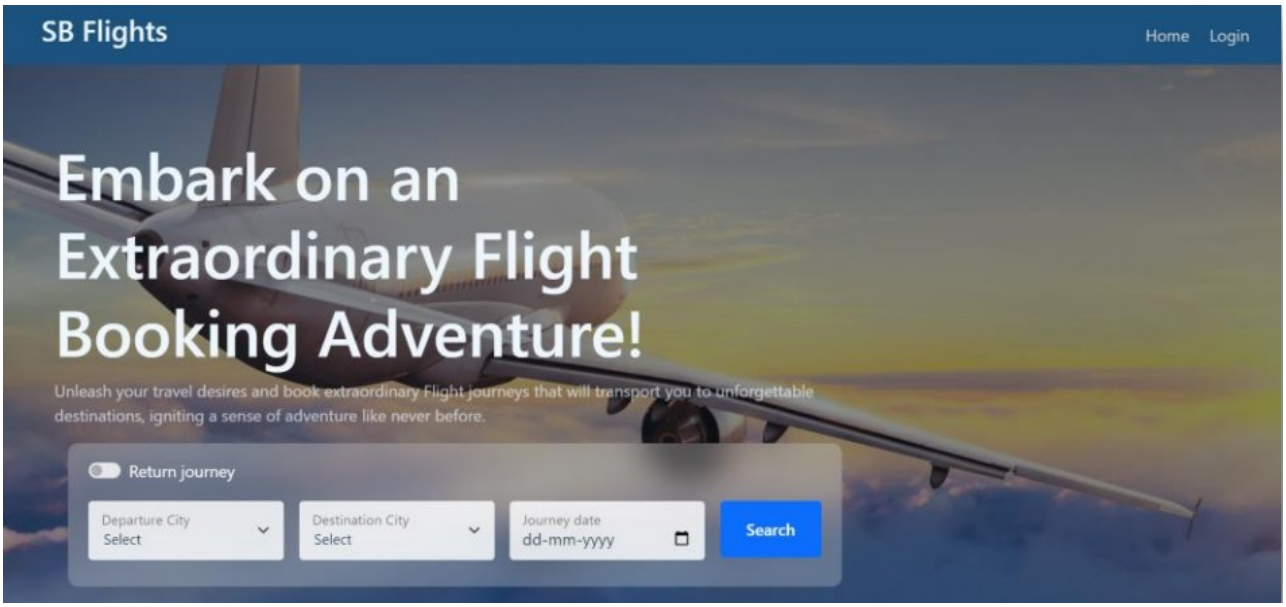
Future Enhancements

- Payment gateway integration (Stripe/PayPal)
- Email notifications for bookings
- Mobile app version
- Better sorting/filtering for flights

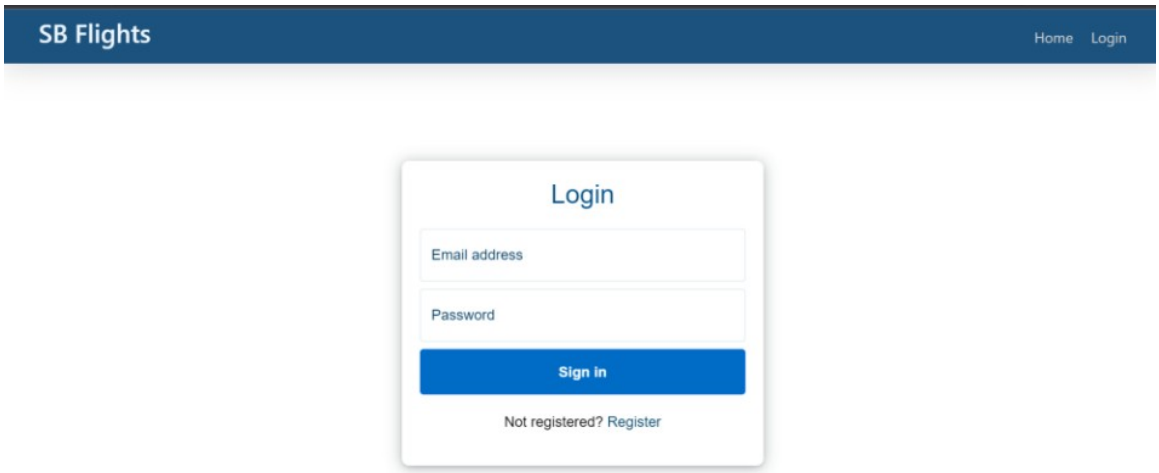
Project Implementation

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

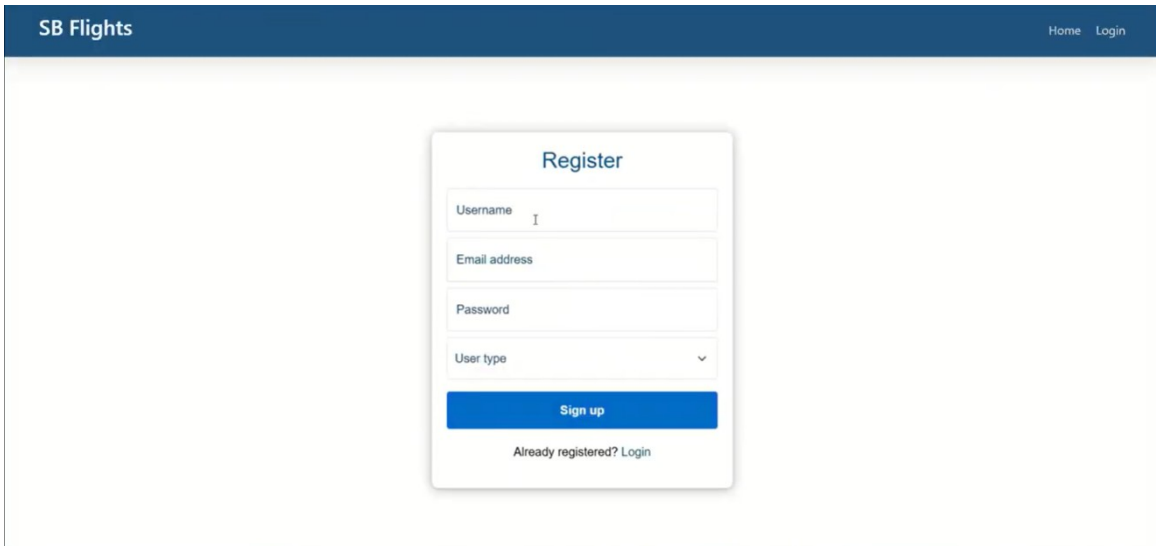
- **Landing page UI**



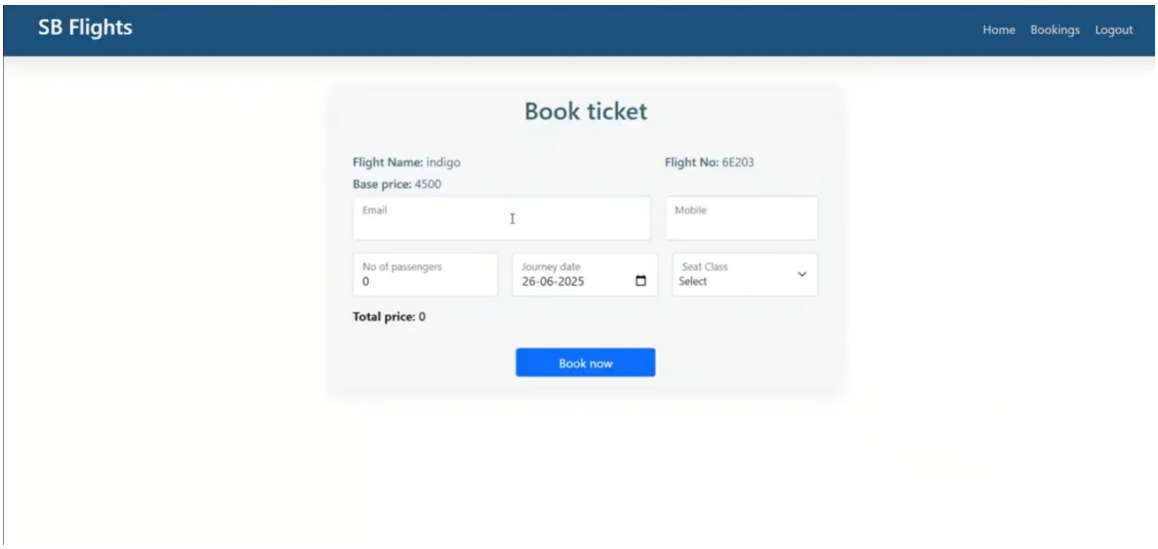
- **Authentication**



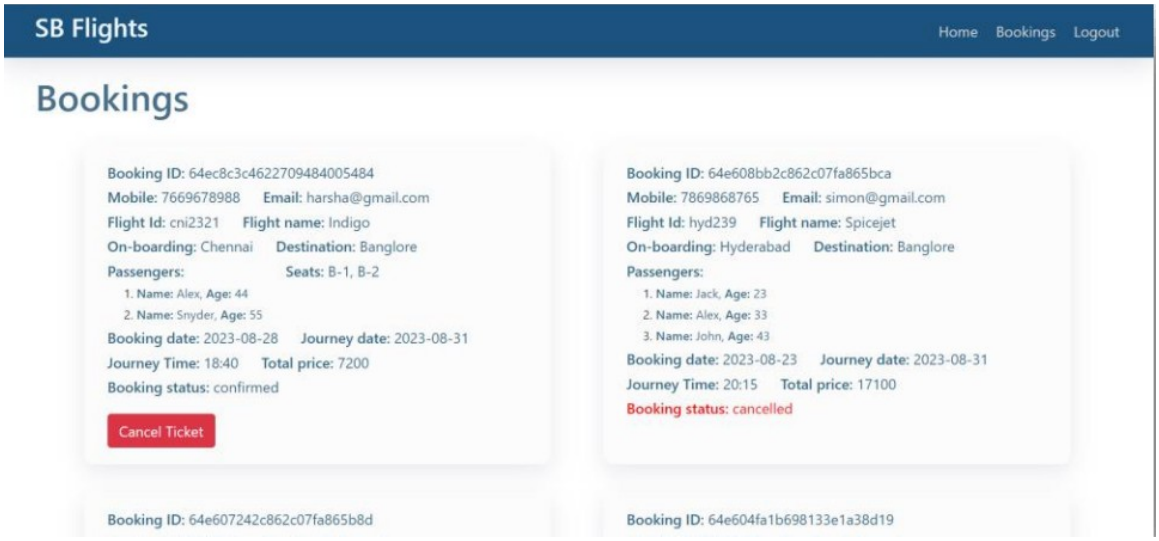
- **Registration**
-



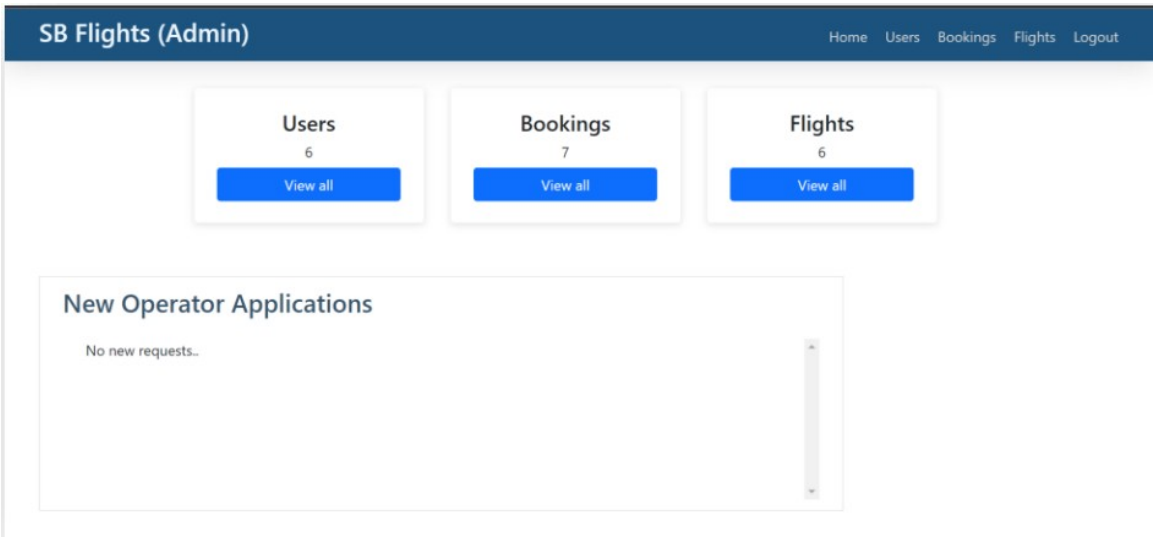
- **Ticket Booking**



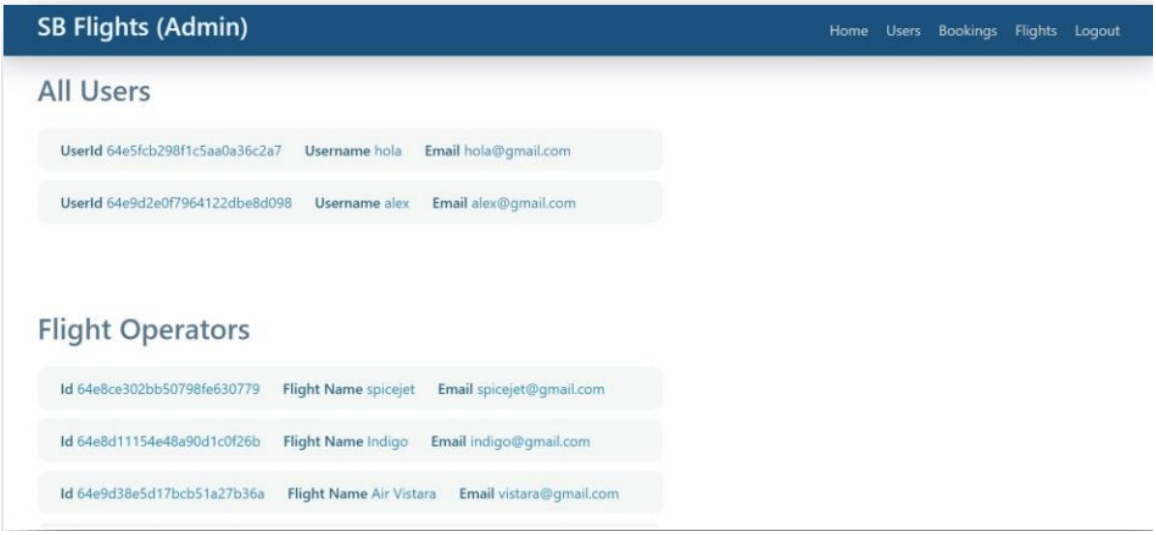
- **User bookings**



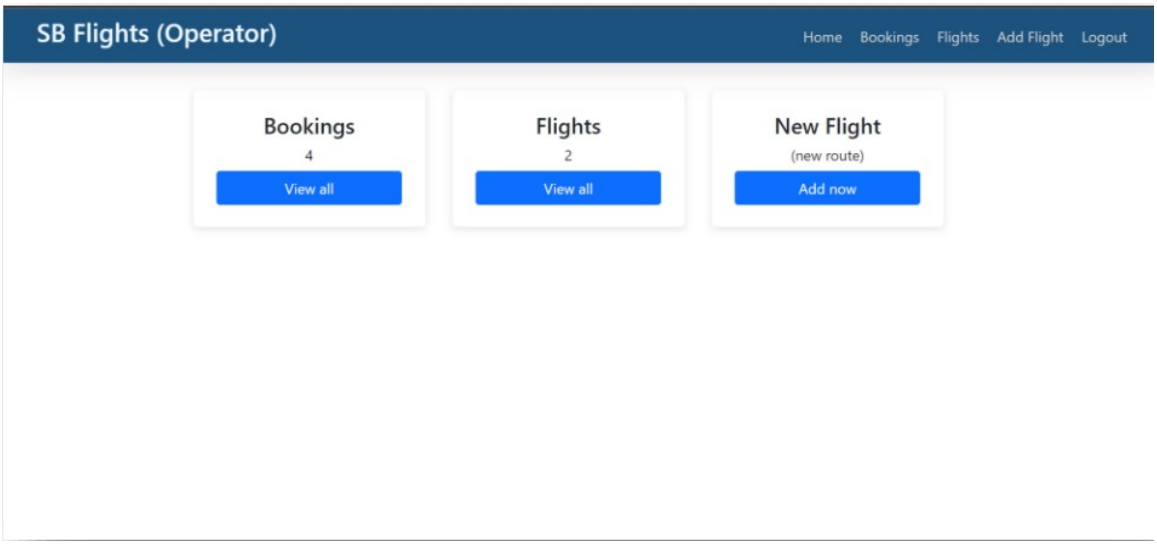
- **Admin Dashboard**



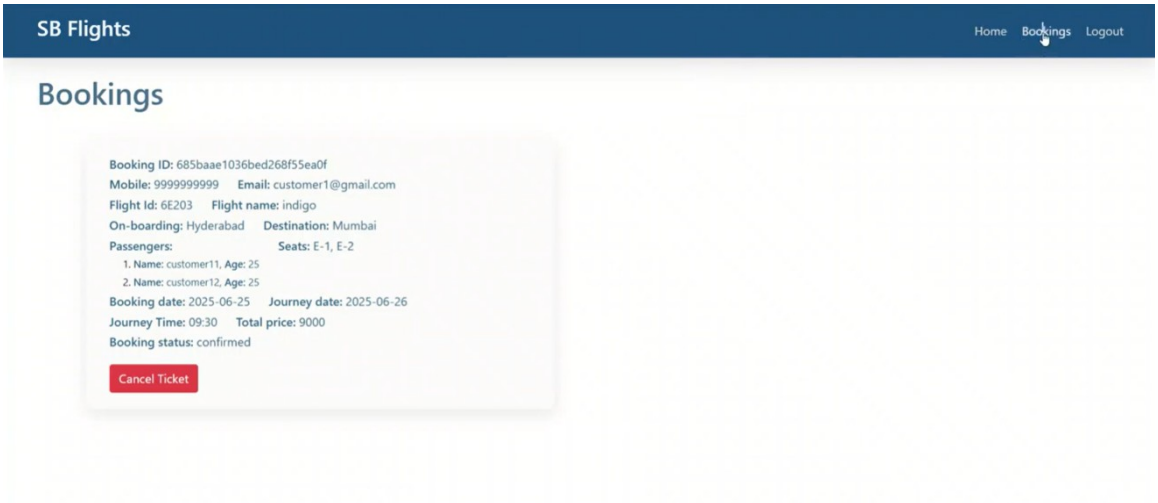
- **All users**



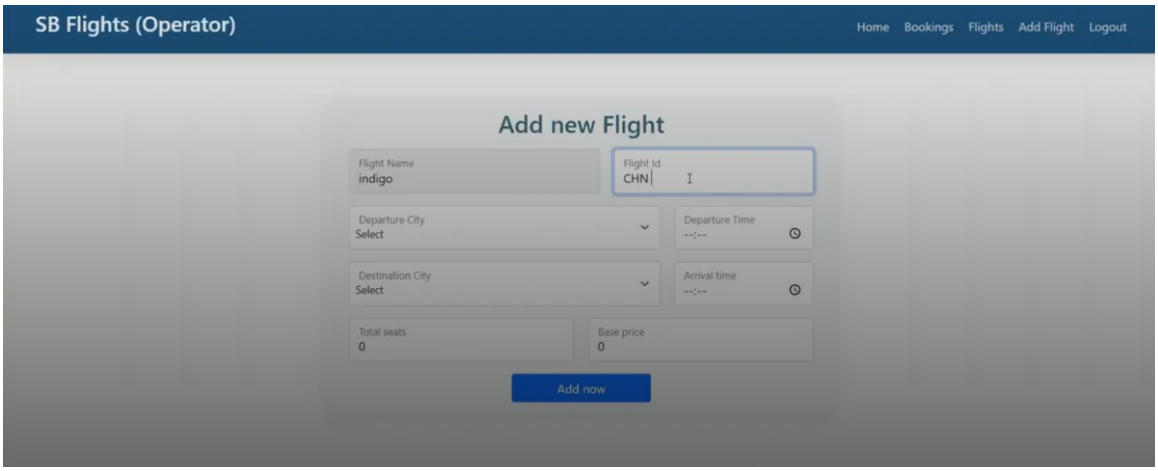
- **Flight Operator**



All Bookings



- **New Flight**



Project demo:

https://github.com/chandini134/Flightfinder-navigating-your-air-travel-options/tree/c28cae3eca70a96a8bd8b09cbee9bd019c559b17/Demo%20video

Code:

backend:

https://github.com/chandini134/Flightfinder-navigating-your-air-travel-options/tree/c28cae3eca70a96a8bd8b09cbee9bd019c559b17/backend

frontend:

https://github.com/chandini134/Flightfinder-navigating-your-air-travel-options/tree/c28cae3eca70a96a8bd8b09cbee9bd019c559b17/frontend

Conclusion

This application provides a complete and scalable solution for flight booking. It can be further enhanced with real-time updates and advanced filtering for better user experience.