

FLIGHT FINDER: NAVIGATING YOUR AIR TRAVEL OPTIONS

Introduction:

“Welcome to our **Flight Finder**, your one-stop solution for exploring and booking the best air travel options! With our user-friendly interface and powerful search features, we make finding the perfect flight simple, fast, and stress-free. Whether you're planning a business trip, a family vacation, or a last-minute getaway, our platform helps you compare airlines, check ticket prices, view schedules, and choose the best route with ease. Explore available flights, select your preferred options, and get ready to travel confidently. Experience smarter and smoother travel planning with our Flight Finder today!”

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 Based Case Study:

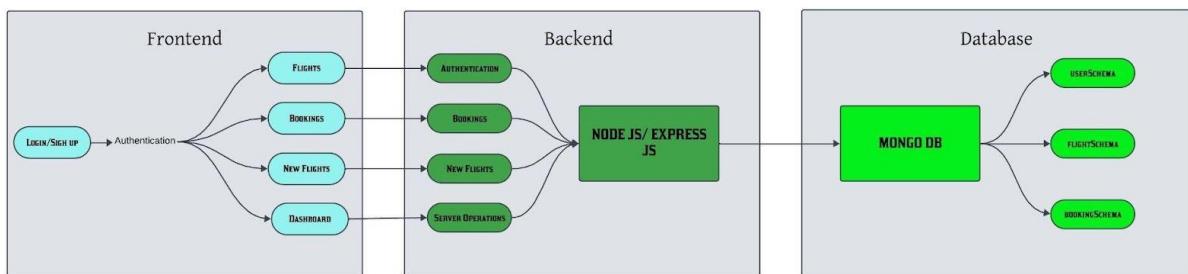
Scenario:

- 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.

Outcome: Through our team's **Flight Finder: Navigating Your Air Travel Options** platform, users like John can effortlessly search, compare, and book flights tailored to their preferences. This solution saves time, reduces the stress of travel planning, and provides a smooth, secure booking experience from start to finish. By offering convenience, flexibility, and real-time information, Flight Finder empowers travelers to focus on their journey and important events ahead, rather than the complexities of booking tickets.

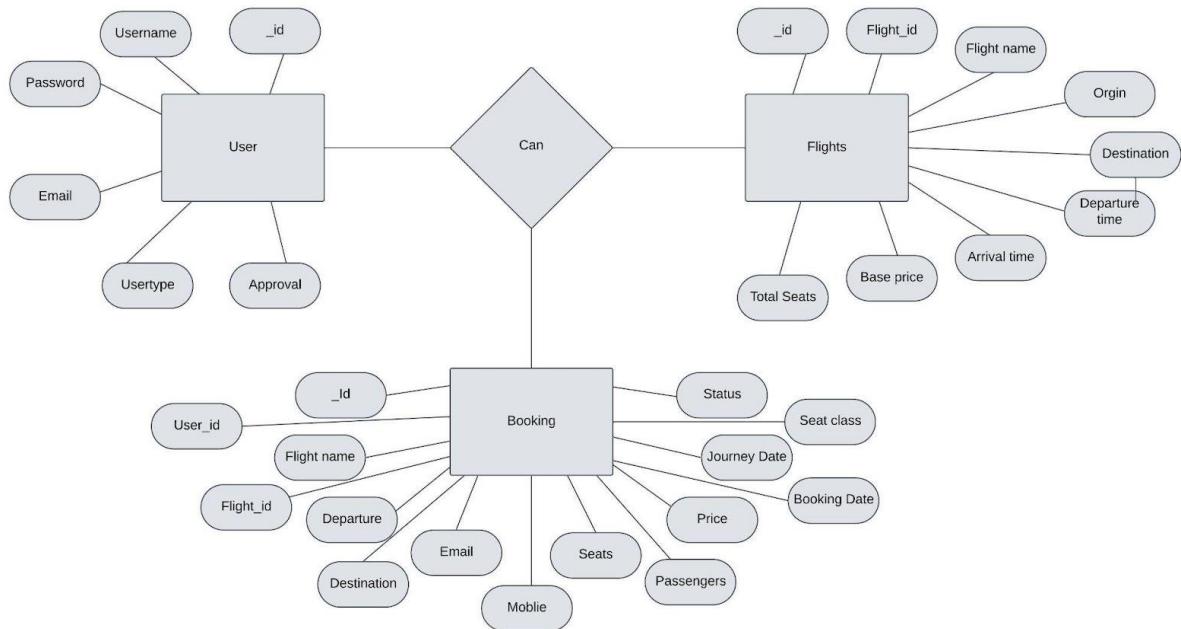
Technical Architecture :-



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.

Key Features of Flight Finder:-

• Flight Search Functionality

Users can enter travel details such as departure city, destination, travel dates, class, and number of passengers. The system retrieves available flights based on the entered information.

• Flight Details Display

The platform displays important flight information including airline name, departure and arrival times, travel duration, and ticket cost. This helps users clearly understand their travel options.

- **Cost Comparison**

Users can compare ticket prices from different airlines to choose the option that best fits their budget.

- **User-Friendly Interface**

The website is designed with a simple and easy-to-use interface, allowing users to navigate, search, and view flight options without confusion.

- **Real-Time Data Retrieval**

The system quickly fetches available flight information based on user input, ensuring that the displayed details are relevant and up-to-date.

- **Efficient Data Management**

The platform stores and manages flight information efficiently, allowing smooth retrieval and display of results without delays.

- **Admin Management**

Administrators can manage flight data and update information to ensure accurate and up-to-date listings.

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://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 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/BMunendra/Flight-Booking-Portal-MERN/tree/main>

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.

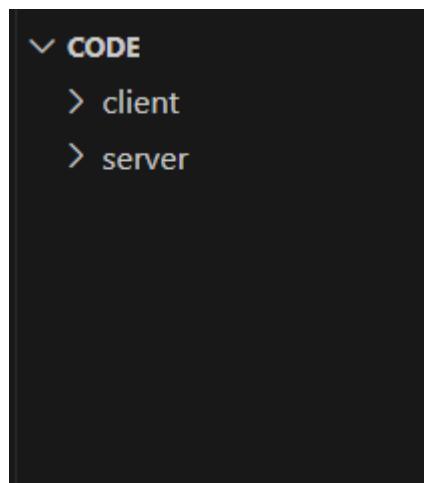
Access the App:

- 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.

CODE

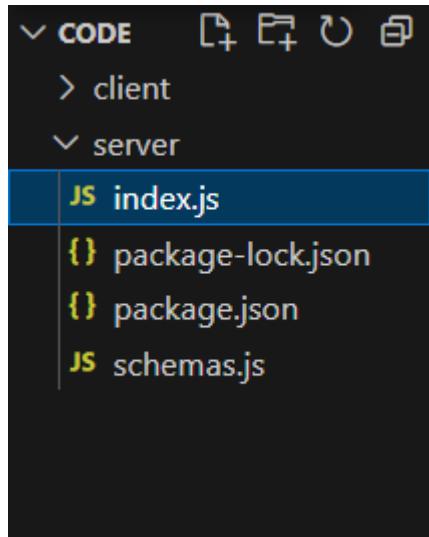
- client
 - public
 - src
 - .gitignore
 - package-lock.json
 - package.json
 - README.md
- server

CODE

- client
 - public
 - src
 - assets
 - components
 - context
 - pages
 - RouteProtectors
 - styles
 - App.css
 - App.js
 - App.test.js
 - index.css
 - index.js
 - logo.svg
 - reportWebVitals.js
 - setupTests.js
 - .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.



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

Let's start with the project development with the help of the given activities.

Milestone 1:

Folder setup:

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

- client
- Server

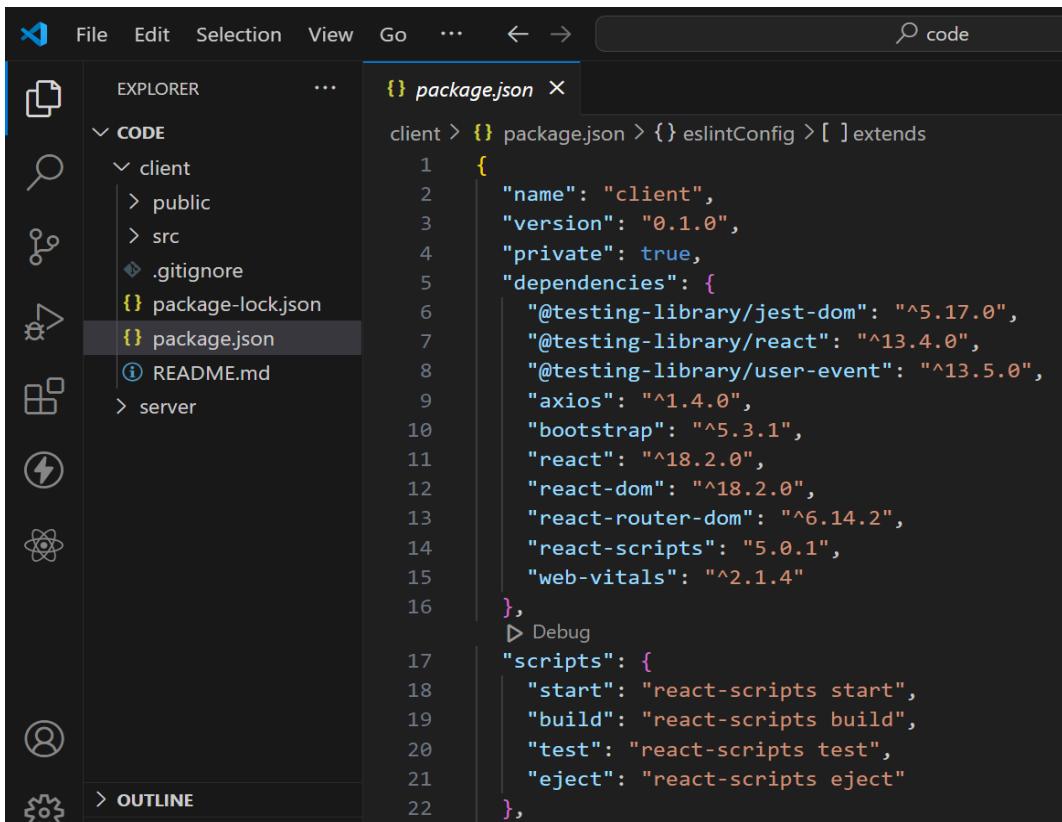
Installation of required tools:

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 is the Explorer sidebar, which displays a file tree with a 'CODE' section containing 'client' (with 'public' and 'src' subfolders), '.gitignore', 'package-lock.json', 'package.json' (which is selected), and 'README.md'. Below 'client' is a 'server' folder. The main editor area on the right shows the contents of the 'package.json' file:

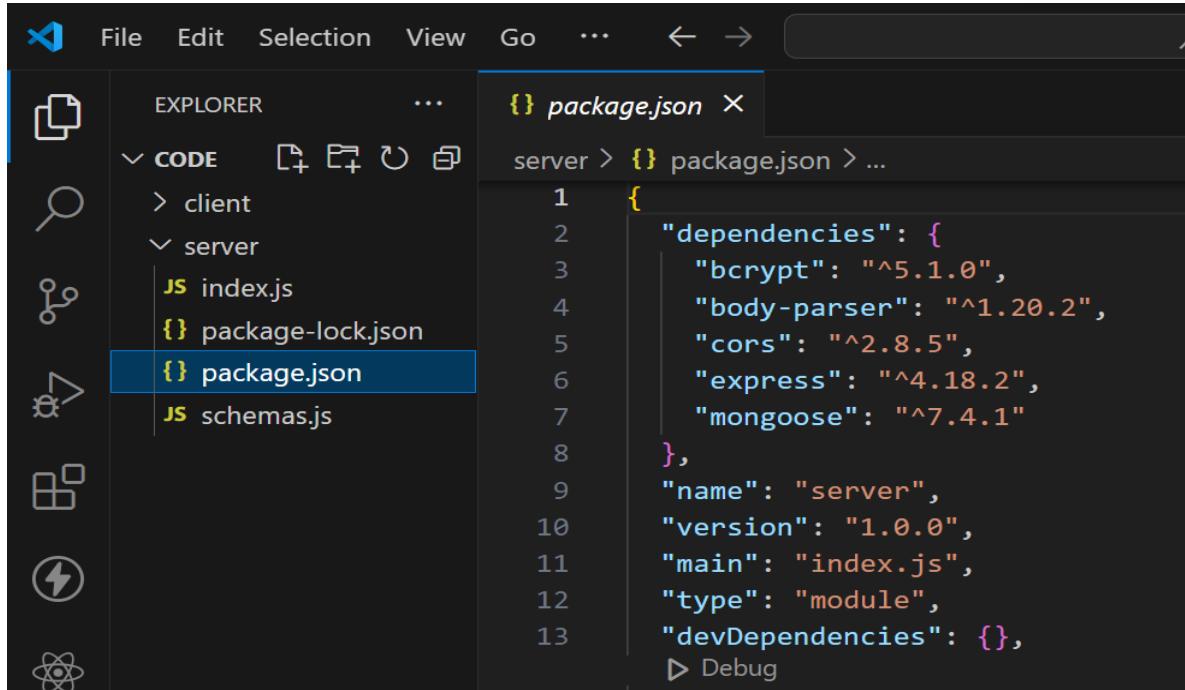
```
1  {
2    "name": "client",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.17.0",
7      "@testing-library/react": "^13.4.0",
8      "@testing-library/user-event": "^13.5.0",
9      "axios": "^1.4.0",
10     "bootstrap": "^5.3.1",
11     "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 'CODE' section containing 'client' and 'server' folders. Inside 'server', there are files: 'index.js', 'package-lock.json', and 'package.json'. 'package.json' is currently selected and highlighted with a blue background. On the right, the main editor pane displays the contents of the 'package.json' file:

```
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": {}  
▷ Debug
```

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.

```
server > js schemas.js > (1) 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},
24   flightId: {type: String}, required: true,
25   departure: {type: String},
26   destination: {type: String},
27   email: {type: String},
28   mobile: {type: String},
29   seats: {type: String},
30   passengers: [
31     {
32       name: { type: String },
33       age: { type: Number }
34     }
35   ],
36   totalPrice: { type: Number },
37   bookingDate: { type: Date, default: Date.now },
38   journeyDate: { type: Date },
39   journeyTime: { type: String },
40   seatClass: { type: String },
41   bookingStatus: {type: String, default: "confirmed"}
42 });
43
44 export const User = mongoose.model('users', userSchema);
45 export const Flight = mongoose.model('Flight', flightSchema);
46 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 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.

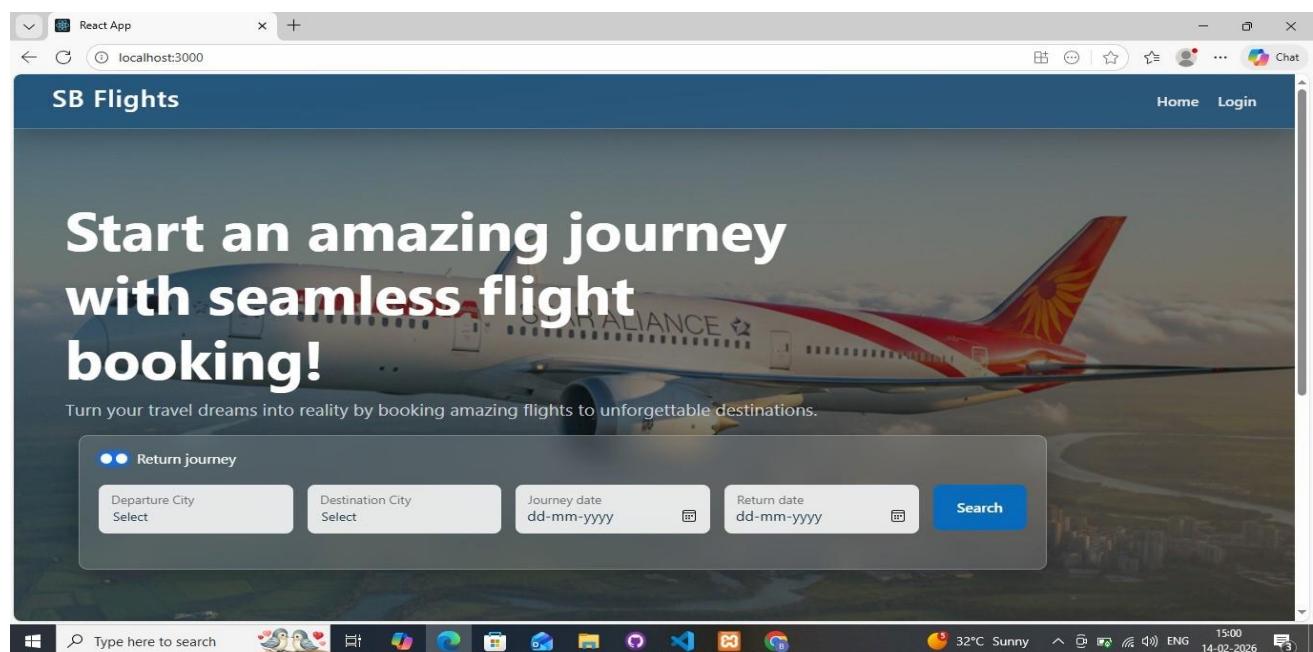
5. Update Flight:

- 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.

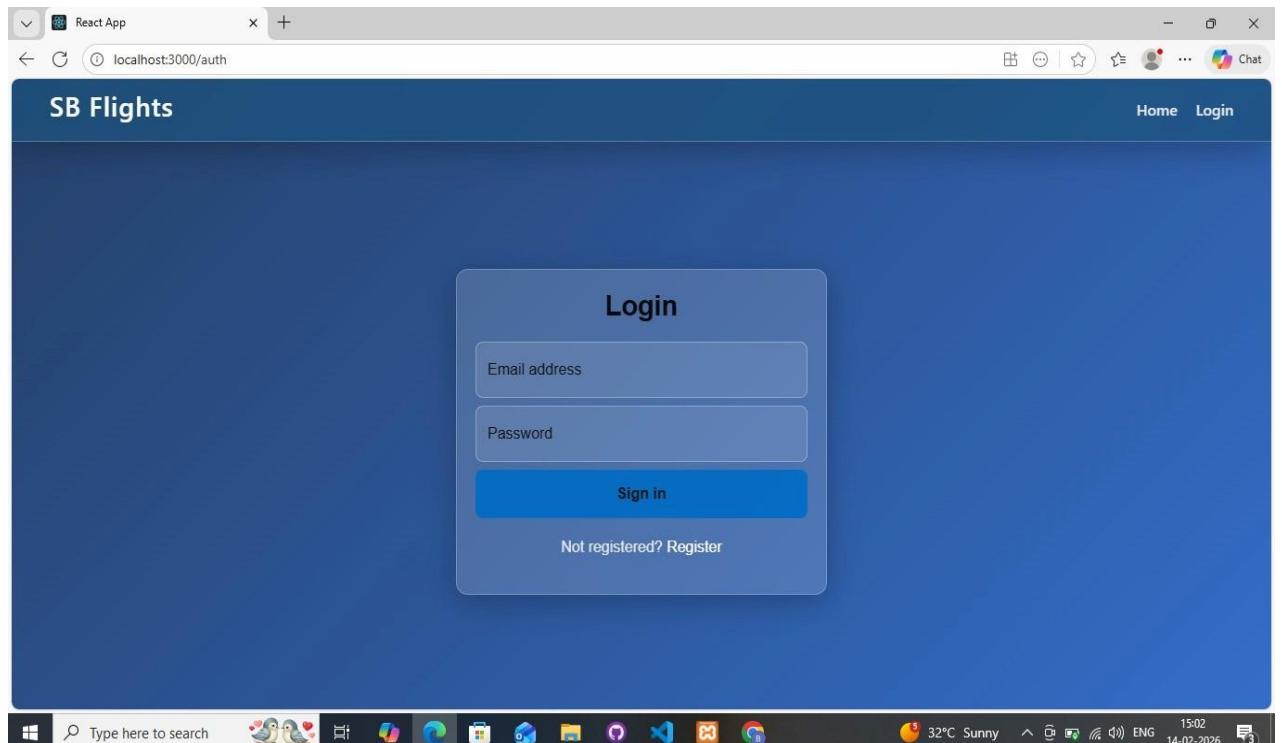
Milestone 5: Project Implementation

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

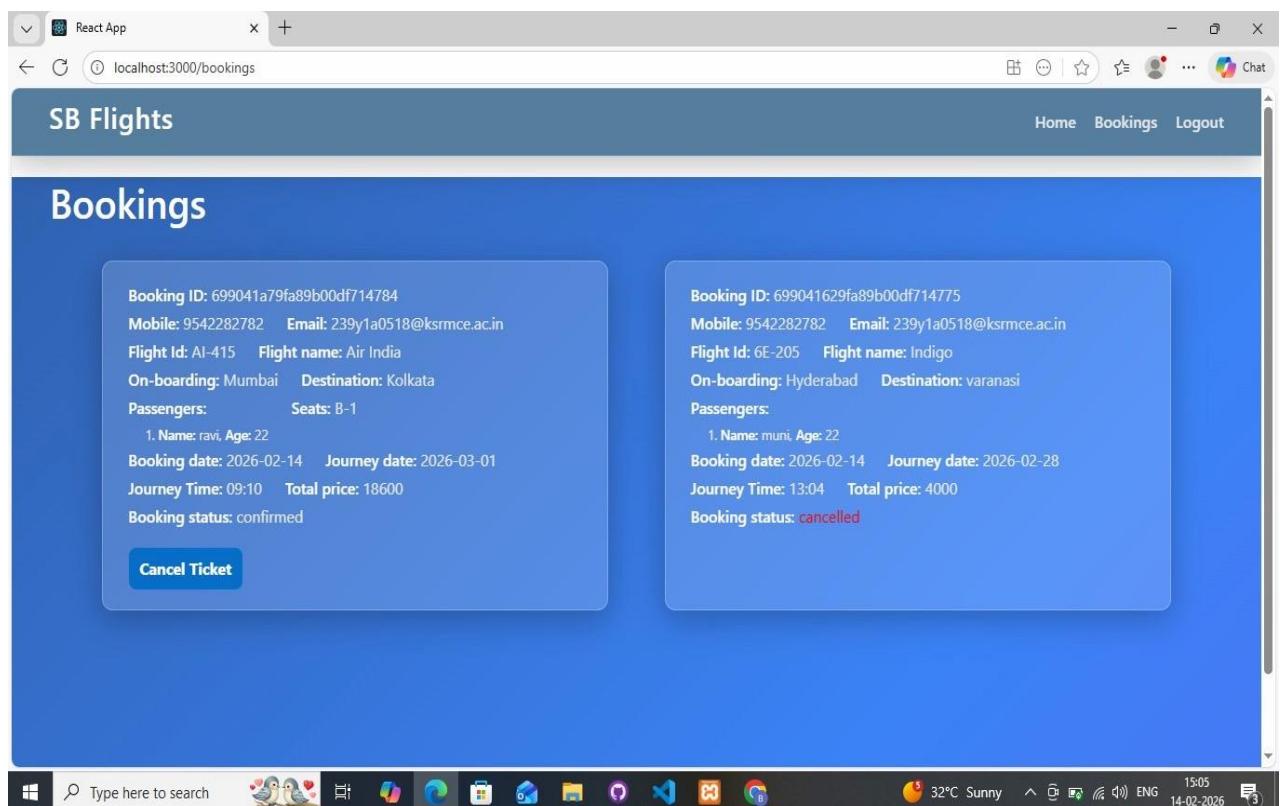
1. Landing page UI:-



2.Authentication:-



3.User bookings:-



4.Admin Dashboard:-

The screenshot shows the SB Flights Admin dashboard. At the top, there are three cards: 'Users' (2), 'Bookings' (3), and 'Flights' (27). Below these is a section titled 'New Operator Applications' containing a single entry: 'Operator name: Indigo' and 'Operator email: 239y1a0515@ksrmce.ac.in'. There are 'Approve' and 'Reject' buttons next to the application details. The bottom of the screen shows a Windows taskbar with various icons and system status.

5.All users:-

The screenshot shows the SB Flights Admin dashboard under the 'All Users' section. It displays two user profiles. The first profile is for a flight operator: 'UserId' 6990407a9fa89b00df714755, 'Username' spice jet, and 'Email' 239y1a0518@ksrmce.ac.in. The second profile is for another user: 'Id' 699040a49fa89b00df714758, 'Flight Name' Indigo, and 'Email' 239y1a0515@ksrmce.ac.in. The bottom of the screen shows a Windows taskbar with various icons and system status.

6.Flight Operator:-

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/flight-admin". The main title is "SB Flights (Operator)". The navigation bar includes links for Home, Bookings, Flights, Add Flight, and Logout. Below the navigation, there are three cards: "Bookings" (1 item, View all), "Flights" (1 item, View all), and "New Flight (new route)" (Add now). The background is blue.

7.All Bookings:-

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/all-bookings". The main title is "SB Flights (Admin)". The navigation bar includes links for Home, Users, Bookings, Flights, and Logout. The main content is titled "Bookings" and displays three booking details in cards:

- Booking ID:** 699041a79fa89b00df714784
Mobile: 9542282782 **Email:** 239y1a0518@ksrmce.ac.in
Flight Id: AI-415 **Flight name:** Air India
On-boarding: Mumbai **Destination:** Kolkata
Passengers: Seats: B-1
1. Name: ravi, Age: 22
Booking date: 2026-02-14 **Journey date:** 2026-03-01
Journey Time: 09:10 **Total price:** 18600
Booking status: confirmed

[Cancel Ticket](#)
- Booking ID:** 699041629fa89b00df714775
Mobile: 9542282782 **Email:** 239y1a0518@ksrmce.ac.in
Flight Id: 6E-205 **Flight name:** Indigo
On-boarding: Hyderabad **Destination:** Varanasi
Passengers:
1. Name: muni, Age: 22
Booking date: 2026-02-14 **Journey date:** 2026-02-28
Journey Time: 13:04 **Total price:** 4000
Booking status: cancelled
- Booking ID:** 699011e349a08658baccd26
Mobile: 9542282782 **Email:** 239y1a0518@ksrmce.ac.in
Flight Id: 09878767898 **Flight name:** Indigo
On-boarding: Indore **Destination:** Jaipur

The bottom of the screen shows a Windows taskbar with various icons and system status.

8.New Flight:-

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/new-flight". The page has a dark blue header with the text "SB Flights (Operator)" and navigation links for "Home", "Bookings", "Flights", "Add Flight", and "Logout". Below the header is a large blue rectangular area containing a form titled "Add new Flight". The form fields are as follows:

- Flight Name: Indigo
- Flight Id: (empty input field)
- Departure City: Select
- Departure Time: (empty input field)
- Destination City: Select
- Arrival time: (empty input field)
- Total seats: 0
- Base price: 0

At the bottom of the form is a blue button labeled "Add now".

The demo of the app is available at:

<https://drive.google.com/file/d/1ZMsAakMCa2ZPTecU79QkmrAxkk9Rl2z-/view?usp=drivesdk>

*** Happy Hacking!! ***