

FLIGHT BOOKING APP USING MERN

Introduction:

The Flight Booking App is a comprehensive platform designed to simplify the process of booking flights and managing travel arrangements. This system allows users to search for flights, book tickets, and manage their reservations, all within an intuitive interface. With features like flight browsing, booking management, and secure payment options, the app serves travelers, airline staff, and administrators efficiently.

Travelers can search for flights based on destination, date, and airline preferences to find the best options for their journeys. Once a flight is chosen, users can book tickets, manage their travel schedules, and receive notifications and reminders. Airline staff benefit from a dedicated dashboard to manage bookings, update schedules, and communicate with passengers, while administrators ensure smooth operation, regulatory compliance, and resolve any issues that arise.

The Flight Booking App is built on a robust technical foundation, utilizing a client-server model with a MERN stack (MongoDB, Express.js, React, and Node.js). This architecture supports a seamless, efficient, and secure flight booking experience, meeting the growing demand for accessible and well-organized travel services.

Key Features

User Registration & Profile Creation:

- **Secure Sign-Up** using email and password authentication.
- **Profile Creation** that securely stores personal and travel information for seamless booking.

Flight Browsing & Filtering:

- **Search and Filter** flights based on destination, departure/arrival times, and airline preferences.
- **Real-Time Availability Updates** allow users to view only available flights and seating options, reducing booking conflicts.

Top Deals Section:

Display Exclusive Flight Deals and Discounts on popular routes, providing users with curated options for budget-friendly travel.

Highlight Limited-Time Offers and Seasonal Promotions to encourage bookings with urgency and attract frequent travelers.

Enable Easy Comparison of Deals across different airlines and routes, allowing users to find the best prices without extensive searching.

Booking & Reservation Management:

- **User-Friendly Booking Interface** enables users to select flights, choose dates, times, and seat preferences, and upload necessary travel documents (e.g., ID proof).
- **Automated Confirmation Messages and Reminders** sent via email or SMS help ensure that travelers don't miss flights.

Admin Dashboard:

- **Admin Management of Flights, Users, and Operators** to maintain smooth operations, including handling cancellations, refunds, and customer support.
- **Verification and Approval** of new operator registrations to ensure only approved airlines are listed on the platform.

Operator Dashboard:

- **Operators Can Manage Flight Schedules, View Bookings, and Update** flight statuses (e.g., on-time, delayed, or canceled).
- **Secure Access to Passenger Manifests** with options to add notes and flight-specific instructions.

Return Flight Booking:

- **Easy Return Booking Option** for round-trip travelers, allowing them to select return flights with the same filtering and scheduling options for a convenient experience.

Description:

The Flight Booking App is a user-centric platform designed to simplify the flight booking process. This app connects travelers and airline operators through an intuitive digital interface, enabling users to search, filter, and book flights based on destination, departure time, and real-time availability.

For travelers, the app offers secure registration, profile creation, and document upload, with automated notifications and reminders to ensure a smooth travel experience. Operators have access to a dedicated dashboard to manage flight schedules, confirm bookings, and view passenger details, while administrators manage operator registrations, monitor system performance, and oversee compliance for a seamless experience.

Built using Bootstrap and Material UI for a sleek, responsive frontend, the app also uses Axios for seamless backend communication, with Express.js and MongoDB managing server logic and data storage. Moment.js facilitates precise scheduling, and security libraries like bcrypt ensure the secure handling of user data.

With features that enhance accessibility, communication, and efficiency, the Flight Booking App meets the growing demand for accessible and well-organized travel services, providing travelers with convenient, reliable booking options and helping operators manage their schedules effectively.

Scenario-Based Case Study:

1. **User Registration:** Sarah, a frequent traveler, downloads and opens the Flight Booking App. She registers as a new user by providing her email address and creating a secure password. Once registered, Sarah is welcomed to the app and can log in to access its features.
2. **Browsing Flights:** After logging in, Sarah is directed to a dashboard showcasing a list of available flights. The app provides various filters for her to search for flights based on destination, date, and preferred airline. She filters the options to find a direct flight to New York City on her selected travel date.
3. **Booking a Flight:** Sarah selects her chosen flight and clicks the "Book Now" button. A booking form appears, prompting her to select her preferred seating, confirm the travel details, and upload necessary travel documents, such as her ID proof. Once the form is completed, Sarah submits the booking request and immediately receives a confirmation that her booking is in process.
4. **Booking Confirmation:** The airline operator reviews the booking request and confirms Sarah's reservation. The status of her flight changes to "Confirmed," and Sarah receives a notification with the flight details—date, time, and terminal information—via both email and SMS.
5. **Booking Management:** As the departure date approaches, Sarah can access her booking details through the app's dashboard. Here, she can manage her reservation, including the option to reschedule or cancel the booking if needed. She can also contact customer support for any assistance related to her flight.
6. **Admin Approval (Background Process):** In the background, the app's admin team is reviewing new airline operator registrations. A new airline, Sky High Airlines, has applied to join the platform. After verifying the legitimacy of the operator, the admin approves it, ensuring that only trusted airlines are listed on the platform, in compliance with travel regulations.
7. **Platform Governance:** The admin monitors the platform's overall operation, addressing issues, handling disputes, and implementing system improvements.

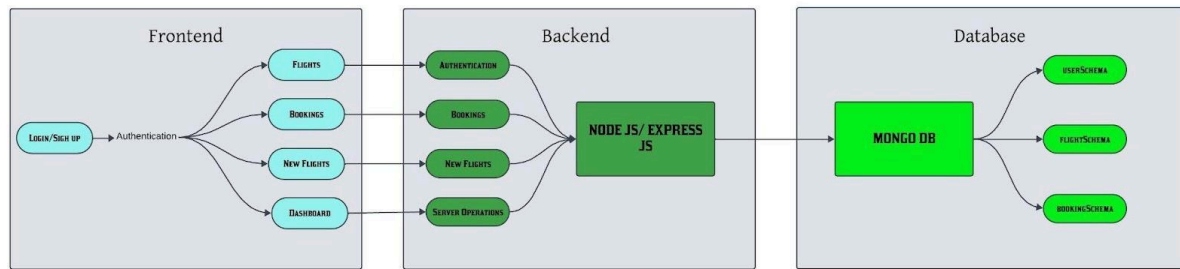
Ensuring the app's compliance with privacy regulations and terms of service is also a key responsibility, creating a safe and reliable experience for all users.

8. **Operator's Flight Management:** On the day of the flight, the airline operator logs into their dashboard and reviews all scheduled flights. They confirm Sarah's seat and monitor other reservations, updating statuses for any changes such as delays or cancellations to keep travelers informed.
9. **Flight Check-In and Boarding:** On the day of departure, Sarah arrives at the airport and checks in for her flight through the app or at the airport counter. She receives her boarding pass and completes the boarding process smoothly, thanks to the app's real-time updates and notifications.
10. **Post-Flight Follow-Up:** After completing her trip, Sarah receives a follow-up notification from the app with options to provide feedback on her experience. She also has access to her past bookings and travel history within the app, which can be useful for future planning.

Technical Architecture:

The Flight Booking App is built on a modern client-server model for efficient performance and scalability. The frontend utilizes Bootstrap and Material UI to create a responsive user interface, with Axios managing smooth API communication between client and server. The backend is powered by Express.js, providing robust server-side logic, while MongoDB handles scalable data storage for user profiles, flight information, and booking details.

For security, JWT is used for session management, and bcrypt for password hashing, ensuring secure authentication. Moment.js handles date and time functionalities for precise scheduling. The admin interface oversees operator registration, platform governance, and compliance, while Role-based Access Control (RBAC) is implemented to manage user permissions. Scalability is supported by MongoDB, and performance is optimized through load balancing and caching techniques.



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.

FRONTEND TECHNOLOGIES :

React.js: Core framework for component-based UI, managing routes (React Router) for user, admin, and operator home pages.

Bootstrap & Material UI: Provide responsive, stylish design elements (forms, buttons, tables) across components like booking modals, dashboards, and data tables.

Axios: Manages HTTP requests to backend for actions like login, flight search, booking, and admin updates.

Form Handling & Validation: React Hook Form and validation ensure proper input handling for login, registration, and booking.

State Management: Context API or Redux to track global states (auth, booking data) and manage access based on roles (user, admin, operator).

Modals & Data Tables: React Modals for booking, and Material UI data tables for admin views, making actions like editing and filtering easy.

CSS Modules: Custom styling to create a unique look while keeping styles isolated.

Notifications: Instant feedback via React Toastify for actions like bookings, cancellations, and errors.

Moment.js: Manages date and time functions for scheduling and flight times.

BACKEND TECHNOLOGIES :

1. **Database (MongoDB):** Set up MongoDB locally or with MongoDB Atlas; create collections for **flights**, **users**, **bookings**.

2. **Express.js Server:** Configure an Express.js server with **body-parser** and **CORS** middleware for request handling.
3. **API Routes:** Define route files for **flights, users, bookings, and authentication** with endpoints for listing flights, managing users, and handling bookings.
4. **Data Models (Mongoose):** Create Mongoose schemas for **flights, users, bookings** with CRUD operations.
5. **Authentication:** Set up routes for **registration, login, logout**, and secure routes with authentication middleware.
6. **Flight and Booking Management:** Add routes/controllers for **flight listings and booking functionality**, including validation.
7. **Admin Controls:** Implement admin routes for **adding flights and managing bookings** with authorization checks.
8. **Error Handling:** Add middleware to handle API errors and return clear error messages with status codes.

DATABASE AND AUTHENTICATION:

- **MongoDB:** A NoSQL database used for storing flight data, user profiles, bookings, and other relevant information, supporting efficient queries and scalability.
- **JWT (JSON Web Tokens):** Used for secure, stateless authentication, ensuring users (passengers, admins, and flight operators) stay logged in without server-side session storage.
- **Bcrypt:** A library for hashing passwords, ensuring secure storage of user credentials in the database.

Admin Panel & Governance :

- **Admin Interface:** Enables admins to manage flight listings, approve user registrations, and oversee platform settings and day-to-day operations.
- **Role-based Access Control (RBAC):** Ensures different roles (users, admin, operators) have appropriate access to platform features, maintaining data privacy and security.

Scalability and Performance :

- **MongoDB:** Scales horizontally, handling increased data storage and high user traffic as the app grows.
- **Load Balancing:** Distributes traffic evenly across servers to maintain performance, especially during peak times.

- **Caching:** Stores frequently accessed data temporarily to reduce database load, enhancing response times and user experience.

Time Management & Scheduling :

- **Moment.js:** Handles date, time operations, and time zone management for accurate flight scheduling.

Security Features :

- **HTTPS:** Ensures secure data transmission using SSL/TLS encryption.
- **Data Encryption:** Encrypts sensitive user data during transit and at rest for privacy.

Notifications & Reminders :

- **Email/SMS Integration:** Sends timely flight booking, cancellation, and reminder notifications via email or SMS.

ER DIAGRAM:

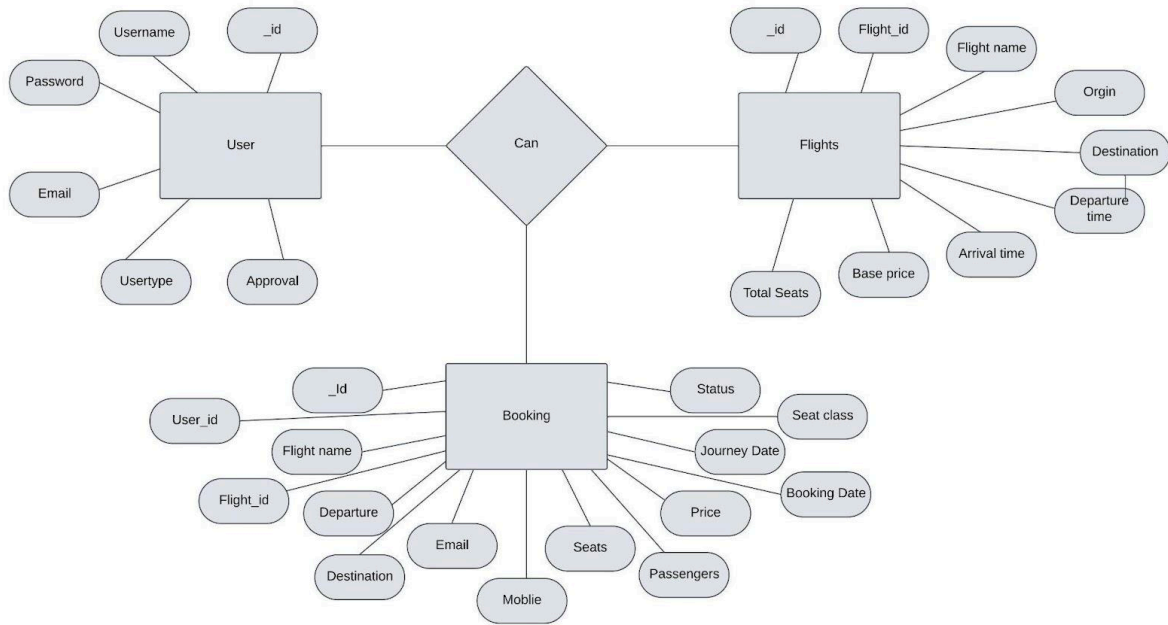
The flight booking ER-diagram represents the entities and relationships involved in a flight booking system. It illustrates how users, bookings, flights, passengers, and payments are interconnected. Here is a breakdown of the entities and their relationships:

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.



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 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: [Karthikg2628/Flight-Booking-app](https://github.com/Karthikg2628/Flight-Booking-app)

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.

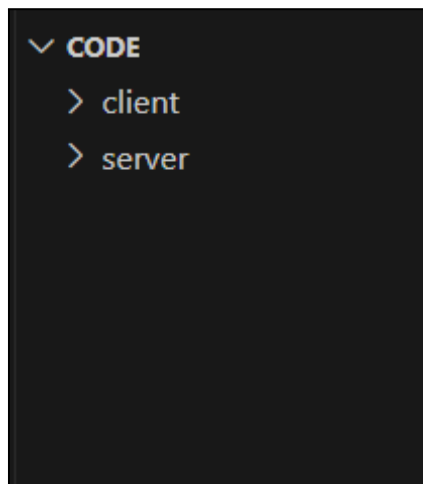
Access the App:

- 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.

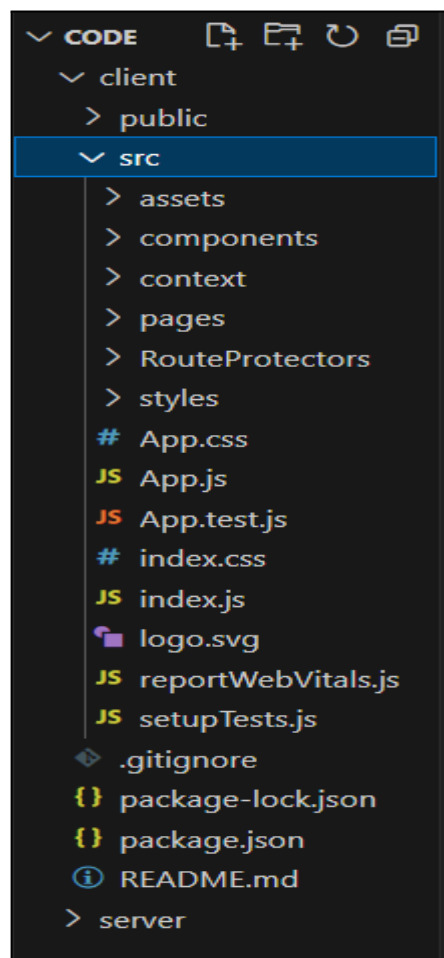
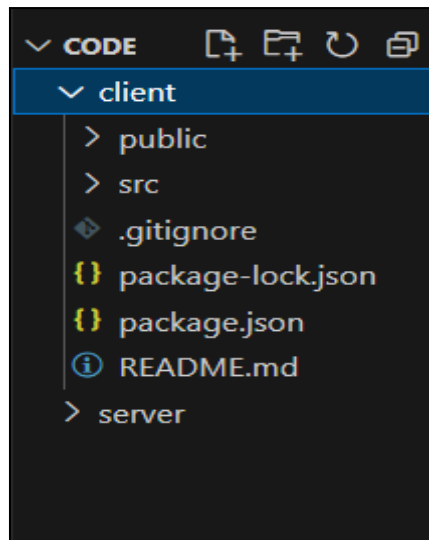
You have successfully installed and set up the flight booking app on your local machine. You can now proceed with further customization, development, and testing as needed.

PROJECT STRUCTURE:

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

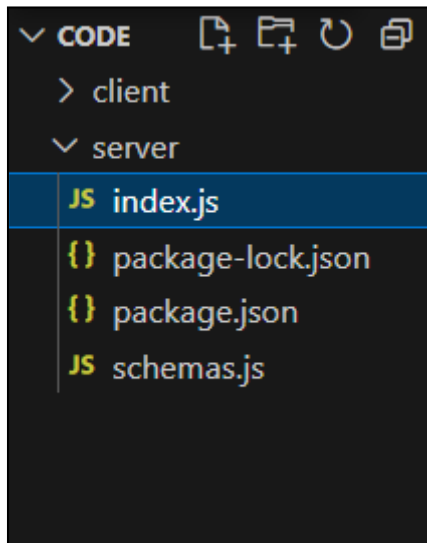


- **Client directory:**
The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api's.



- **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.

SETUP & CONFIGURATION:

Folder setup:

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

- client
- Server

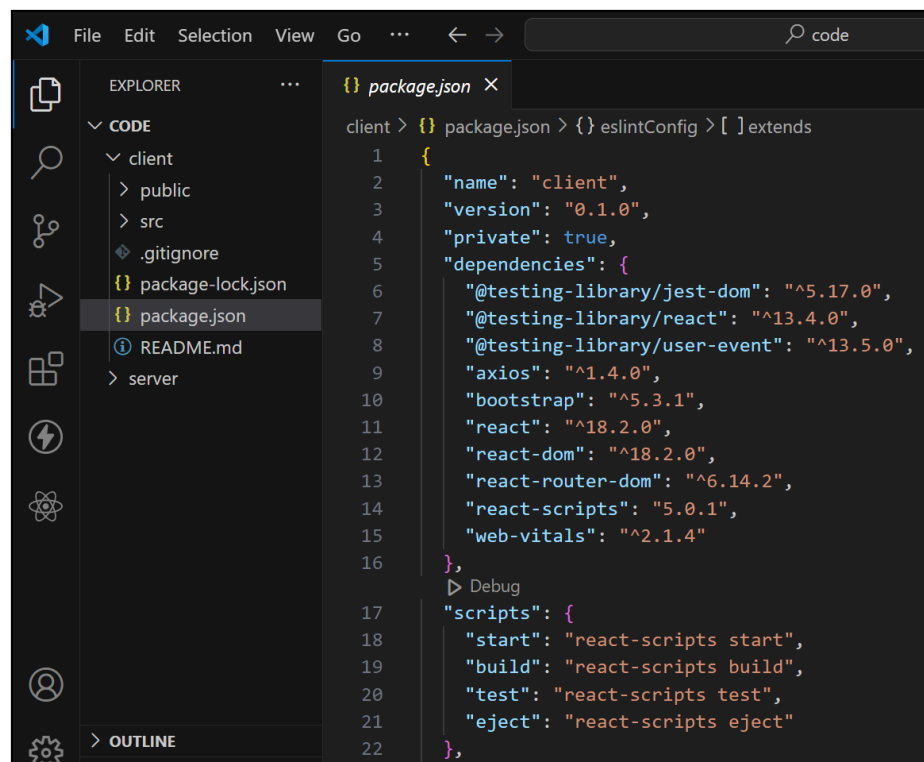
Installation of required tools:

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

For frontend, we use:

- React Js
- Bootstrap
- Axios

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



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure with the 'client' folder expanded, showing files like 'public', 'src', '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The main editor area shows the 'package.json' file for the 'client' folder. The file contains the following JSON content:

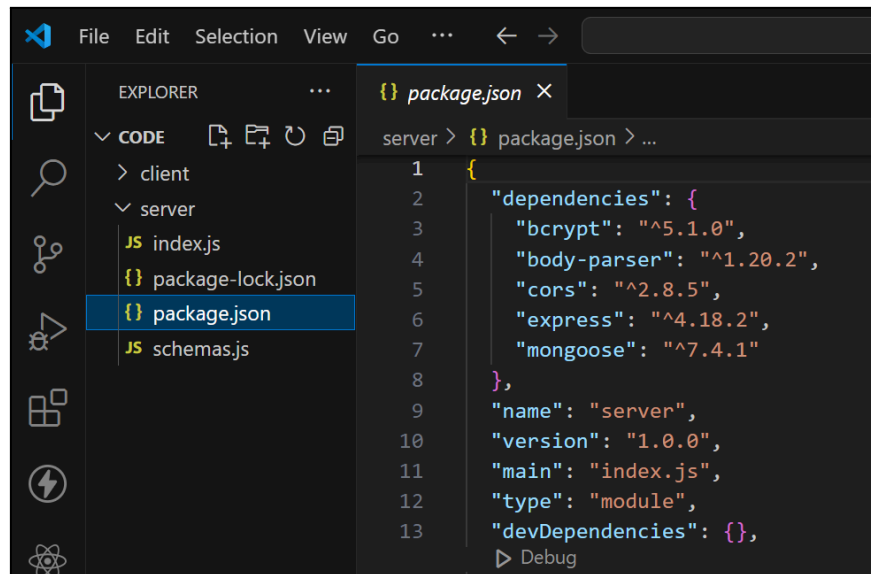
```
1 {
2   "name": "client",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@testing-library/jest-dom": "^5.17.0",
7     "@testing-library/react": "^13.4.0",
8     "@testing-library/user-event": "^13.5.0",
9     "axios": "^1.4.0",
10    "bootstrap": "^5.3.1",
11    "react":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

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

For backend, we use:

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

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



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure with a 'server' directory containing 'index.js', 'package-lock.json', 'package.json', and 'schemas.js'. The 'package.json' file is selected and highlighted. The main editor area displays the content of 'package.json' with the following JSON structure:

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

Frontend Development Configuration:

1. **Login/Register:**
 - Create a form for username and password.
 - Redirect users, admins, or flight operators to their respective dashboards after successful login.
2. **Flight Booking (User):**
 - Implement a flight search feature with inputs for departure city, destination, etc.
 - Fetch and display available flights, adding a "Book" button to redirect to the booking page.
3. **Fetching User Bookings:**
 - Display past bookings with options to cancel them on the bookings page.
4. **Add New Flight (Admin):**
 - Admin can add new flights via a form, sending an HTTP request to the backend to update the database.
5. **Update Flight (Admin):**
 - Admin can edit flight details and manage flights, bookings, and user data through the dashboard.

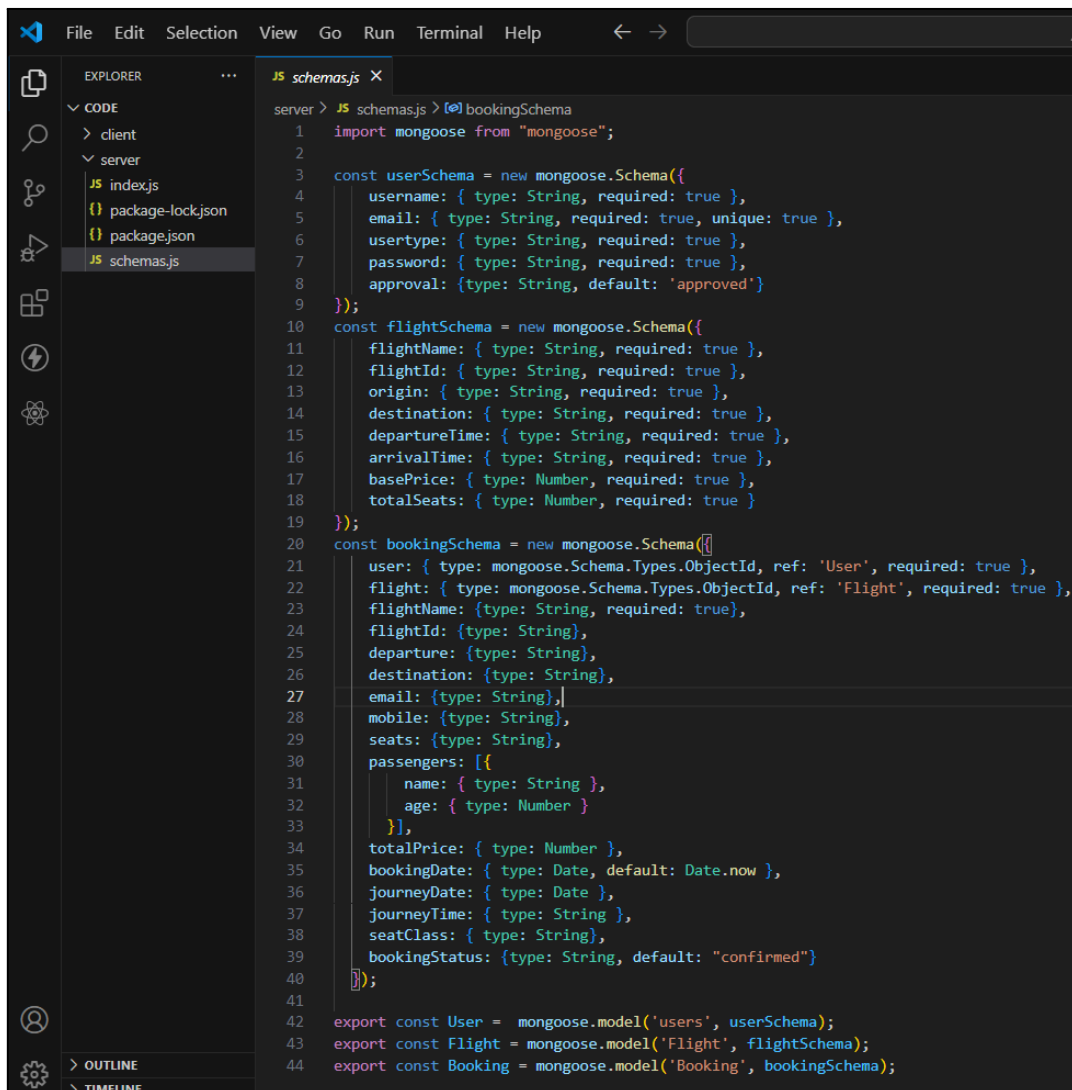
Backend Development Configuration:

1. **Database Configuration:**
 - Set up MongoDB (local or cloud-based) and create collections for flights, users, and bookings.
2. **Express.js Server:**
 - Set up an Express.js server to handle HTTP requests and API endpoints.
 - Configure middleware like body-parser and cors.
3. **API Routes:**
 - Create routes for flights, users, bookings, and authentication.
 - Implement handlers to interact with the database.
4. **Data Models:**
 - Define Mongoose schemas for flights, users, and bookings.
 - Implement CRUD operations for each model.
5. **User Authentication:**
 - Create routes for user registration, login, and logout with authentication middleware.
6. **Flight and Booking Management:**
 - Implement routes for adding flights and handling bookings, including validation and updates.
7. **Admin Functionality:**
 - Set up routes for admins to manage flights, users, and bookings with proper authentication and authorization.
8. **Error Handling:**
 - Implement error handling middleware for API requests and return appropriate error responses.

Database Configuration:

- **Configure schema**

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



The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying a file tree for a project named 'server'. The 'server' folder is expanded, showing files like 'index.js', 'package-lock.json', 'package.json', and 'schemas.js'. The 'schemas.js' file is selected and its content is displayed in the main editor. The code defines two schemas: 'userSchema' and 'flightSchema', and then a 'bookingSchema' that references them. It also shows the mongoose model definitions for 'User', 'Flight', and 'Booking'.

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

- **Connect database to backend**

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

```
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);
})
```

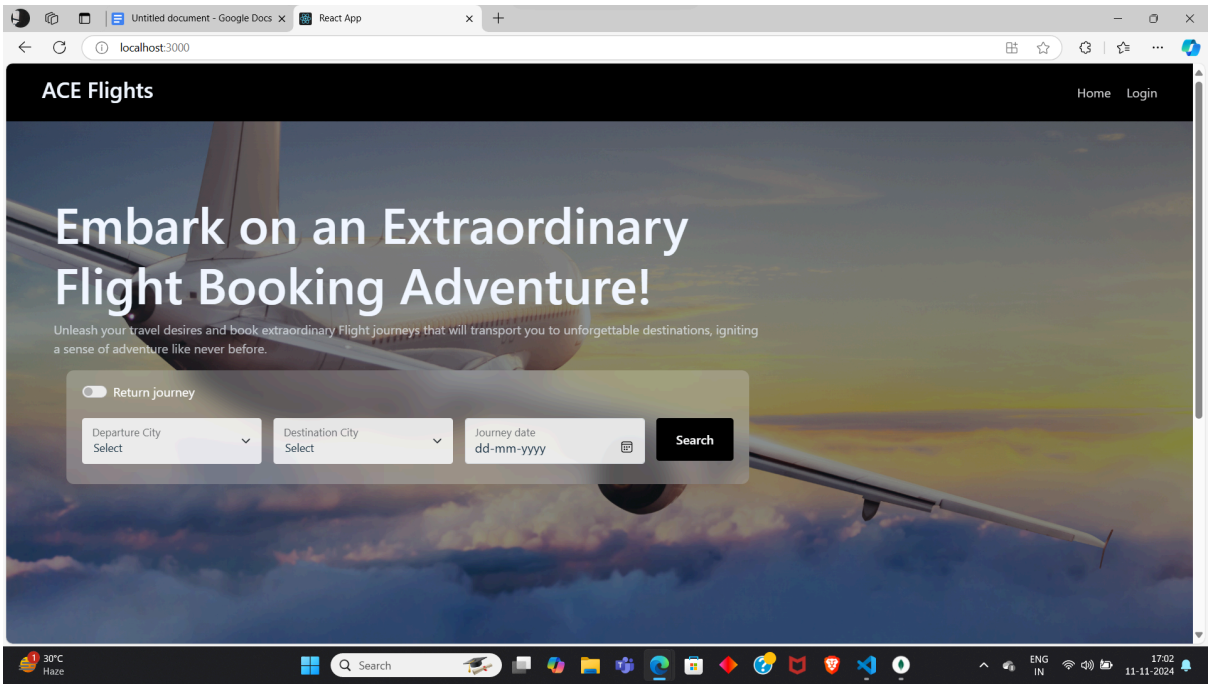

Scenario

- 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.

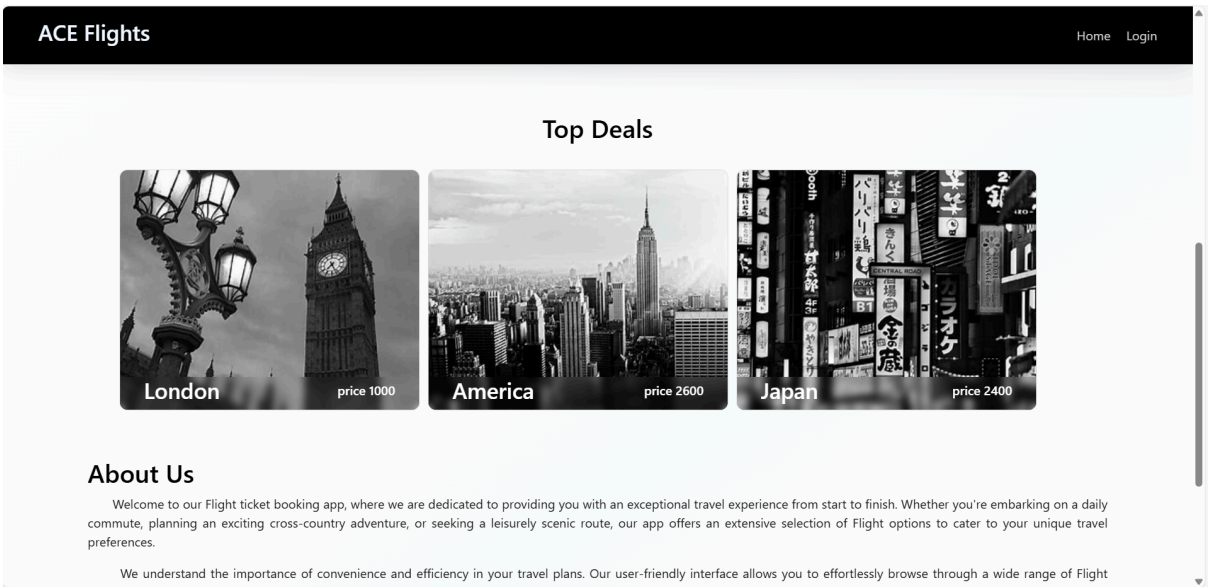
Project Implementation

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



- Top Deals page



- **Authentication**

ACE Flights

[Home](#) [Login](#)

Login

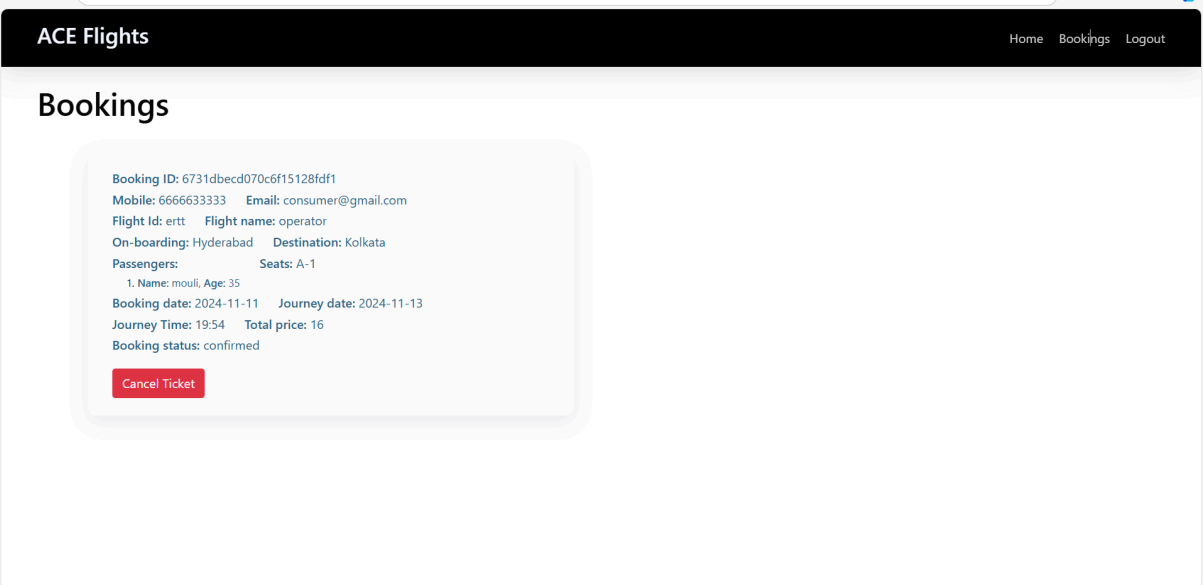
Email address

Password

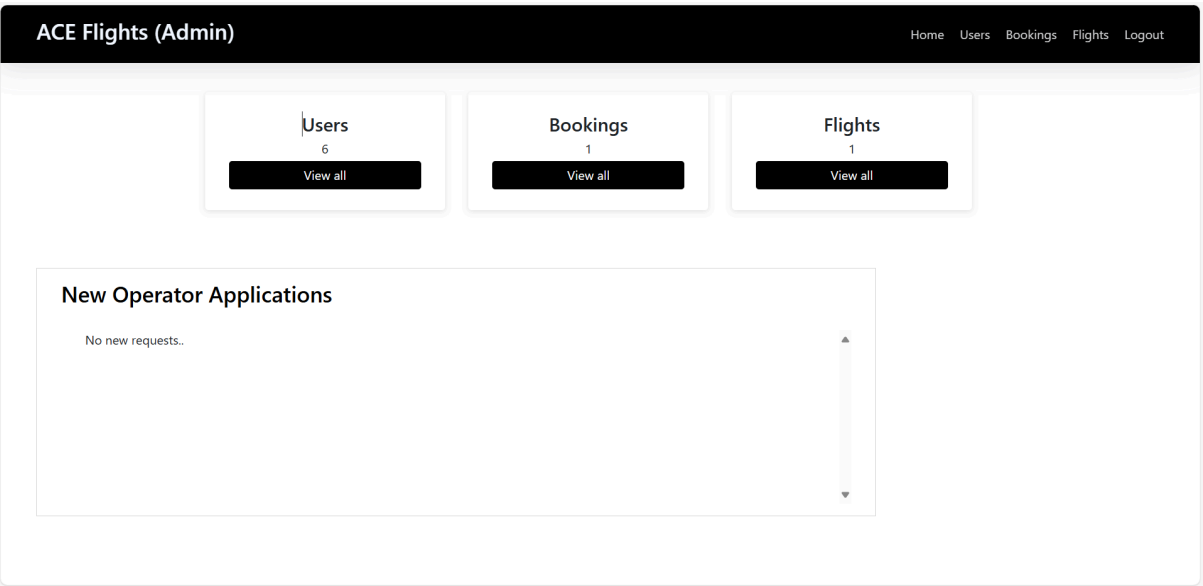
Sign in

Not registered? [Register](#)

- **User bookings**



- **Admin Dashboard**



- **All users**

ACE Flights (Admin)

HomeUsersBookingsFlightsLogout

All Users

Userid673089f2f5cb40df9d2bd62e

Usernamepari

Emailpari@gmail.com

Userid67318e18a311a0eb3c0a968f

Usernameuser 1

Emailmani@gmail.com

Userid6731db50d070c6f15128fde3

Usernamecustomer

Emailcustomer@gmail.com

Flight Operators

Id6731d64f74d4f61e50821f42

Flight Nameoperator

Emailoperator@gmail.com

Id6731e4f7d070c6f15128fe8c

Flight Nameoperator2

Emailoperator2@gmail.com

- Flight Operator

ACE Flights (Operator)

HomeBookingsFlightsAdd FlightLogout

Bookings

1

View all

Flights

1

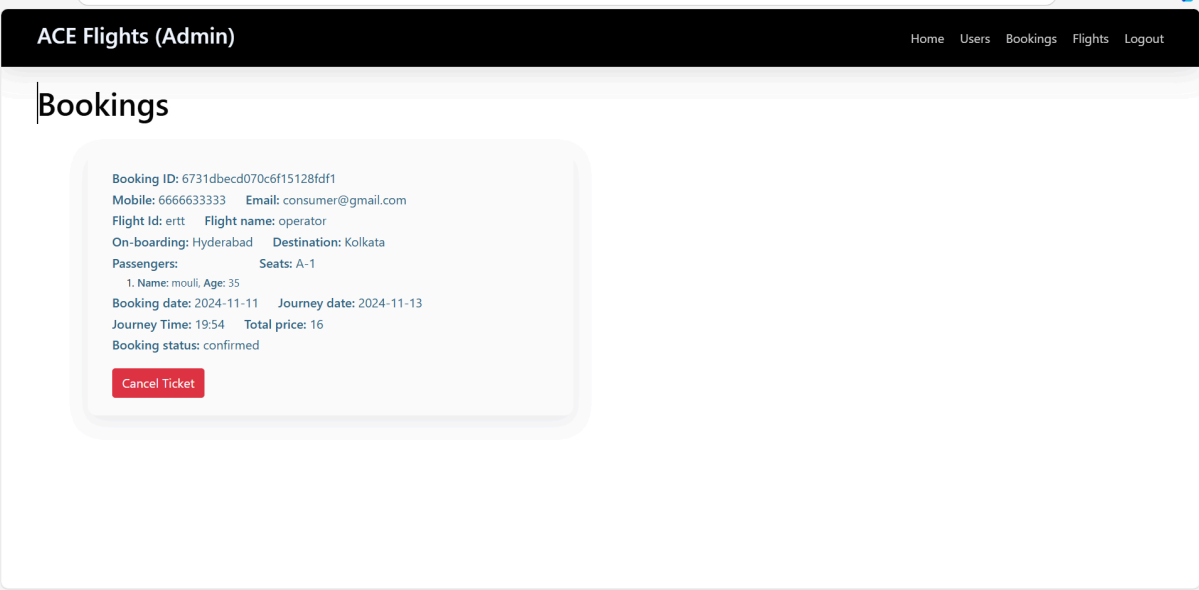
View all

New Flight

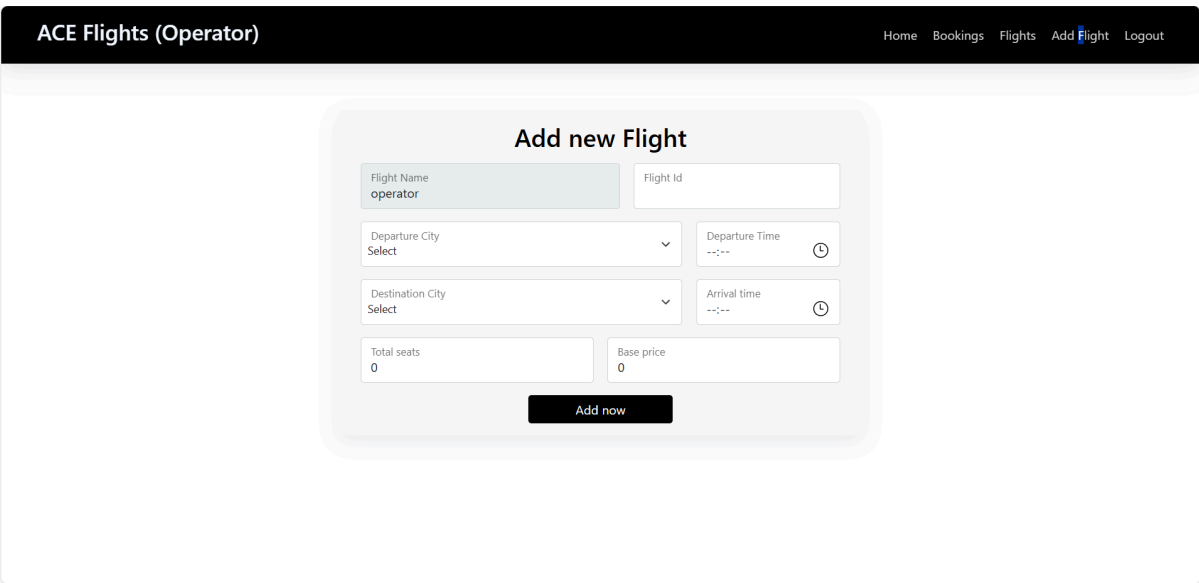
(new route)

Add now

- All Bookings



- **New Flight**



Conclusion

This project outlines the development of a comprehensive flight booking system with user roles for customers and admin. Each milestone—setup, backend, database, frontend, and final implementation—forms a structured foundation for a scalable application. The

integrated functionalities allow for seamless flight booking, user authentication, and role-specific operations. Additionally, a dedicated section for top deals provides users with quick access to promotional flight offers. For further details or to explore the code, please refer to the following resources.

- Project code and assets:

https://drive.google.com/drive/folders/1xny2qMd4TVnszsYlhU7K0VYVBj9DhEp_?usp=sharing

- Demo of the application:

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

Team ID:NM2024TMID15041

Team Members:

- KARTHIK.G
- LOGU .N
- KALAIYARASAN.K
- MANIKANDAN.P
- JENNI.S

