

Flight Finder

INTRODUCTION

In today's fast-paced world, travel has become an essential part of our lives, whether it's for business, leisure, or personal reasons. With the advent of technology, booking flights has become more accessible and. A Flight Finder is flight booking web application that allows users to search, compare, and book flights easily from the comfort of their smart phones or tablets. The primary purpose of a flight booking app is to streamline the process of planning and booking air travel. These apps provide users with a range of features and functionalities that simplify the entire journey, from searching for flights to managing bookings and receiving travel updates.

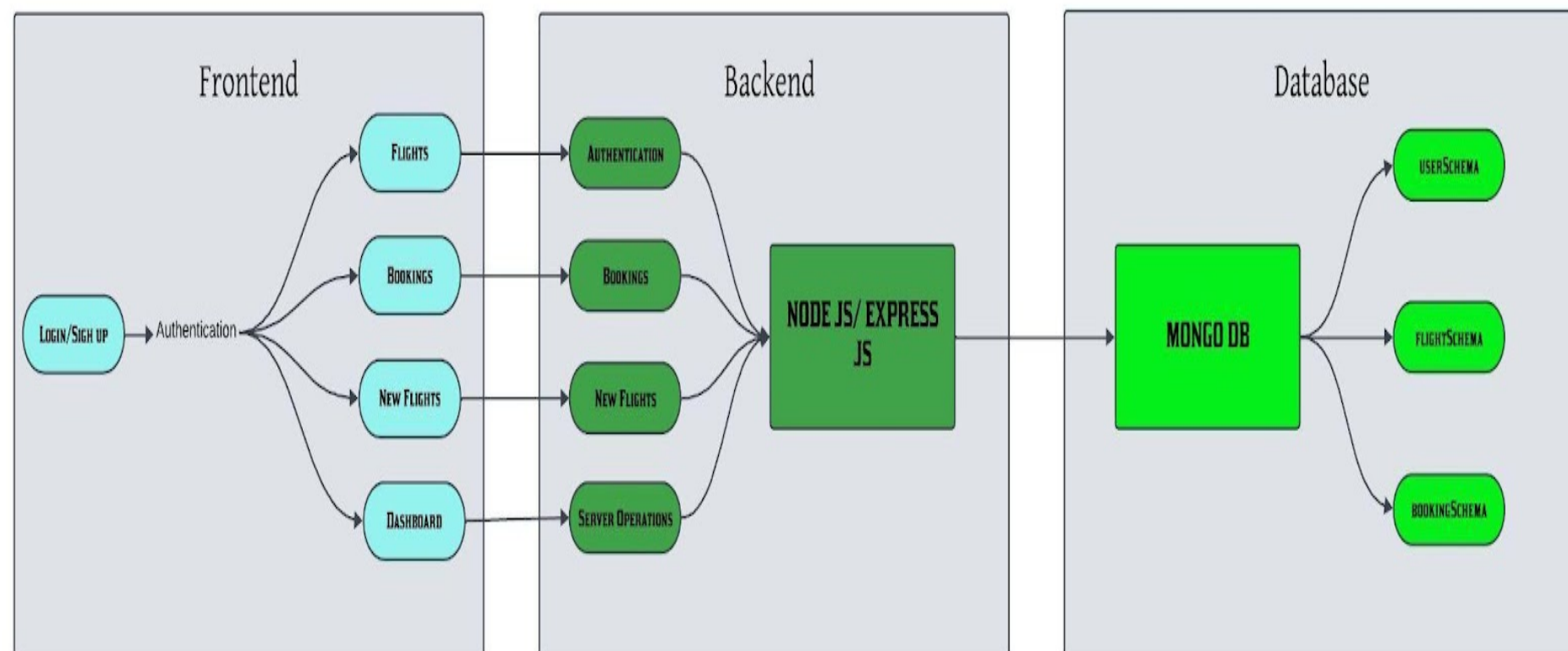
Description

This Flight Finder 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

Surya is a frequent traveler who needs to book a flight to Paris for an upcoming conference. He opens a flight booking app on his smart phone and enters his travel details: departure from New York City, destination Paris, departure date April 10th, return date April 15th, one passenger, and business class. The app quickly shows a list of available flights with important details like price, airline, travel time, and departure times. To make it easier, Surya uses filters to view only direct flights with convenient departure times and selects his preferred airline based on his loyalty program. After choosing a suitable flight, he views the seat map and selects a window seat with extra legroom in the business class cabin. He then enters his payment details securely through the app's payment system. Once the payment is processed, the app confirms the booking and provides him with an e-ticket and full itinerary. This simple and smooth process shows how the flight booking app helps travelers like Surya save time and enjoy a comfortable booking experience.

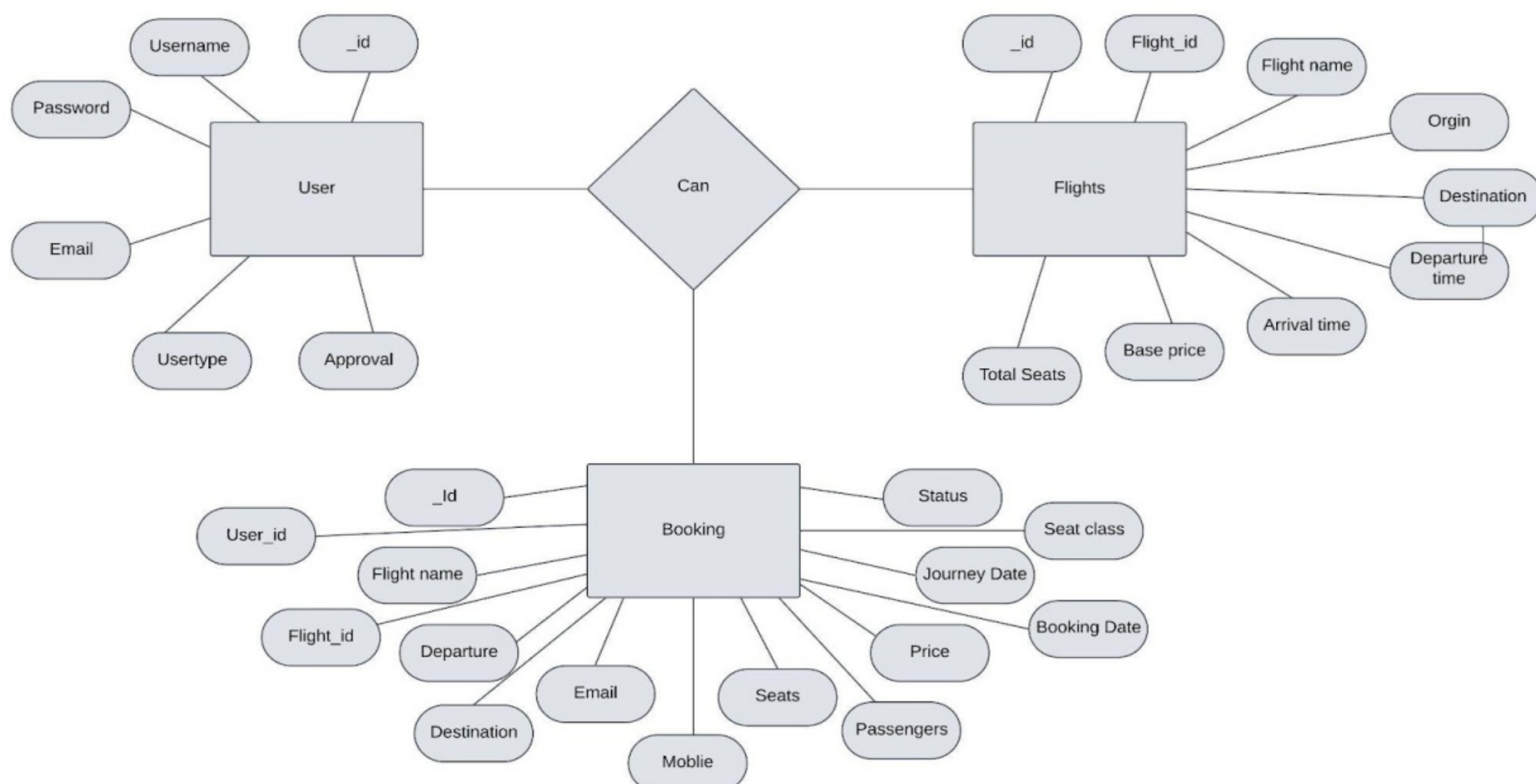
TECHNICAL ARCHITECTURE



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.,

Technologies Used

Frontend (Client Side)

1. **React.js** A JavaScript library for building interactive user interfaces.
Helps create reusable UI components.
2. **HTML** Used to structure the web pages.
3. **CSS** Used for styling and designing the user interface.
4. **JavaScript** Core scripting language used on both frontend and backend.
Adds interactivity to the user interface.

Backend (Server Side)

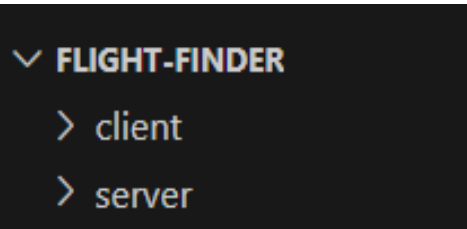
1. **Node.js** A JavaScript runtime for running JS code on the server side.
2. **npm (Node Package Manager)**
Comes with Node.js.
Used to install and manage project dependencies.
3. **Express.js**
A web framework for Node.js.
Handles routing, middleware, and API creation.

Database

1. **MongoDB** A NoSQL database to store user, flight, and booking data.

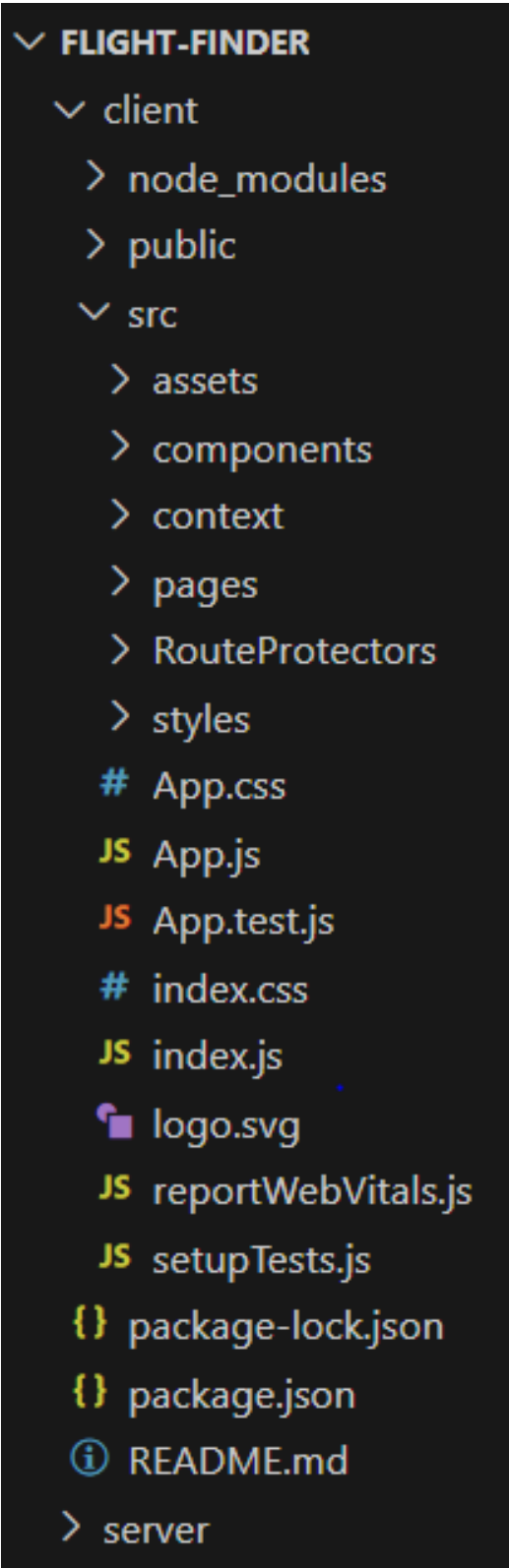
PROJECT STRUCTURE:

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



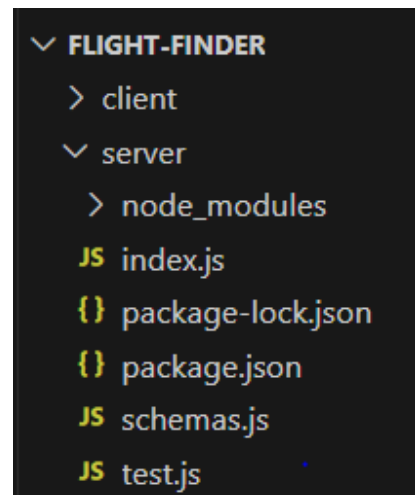
- **Client directory:**

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.
 - And also cancel bookings.
- **ADMIN**
 - Manages all bookings.
 - Adds new flights and services.
 - Monitor User activity.

PROJECT FLOW:

Project setup and configuration

Folder setup:

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

- client
- Server

Installation of required tools:

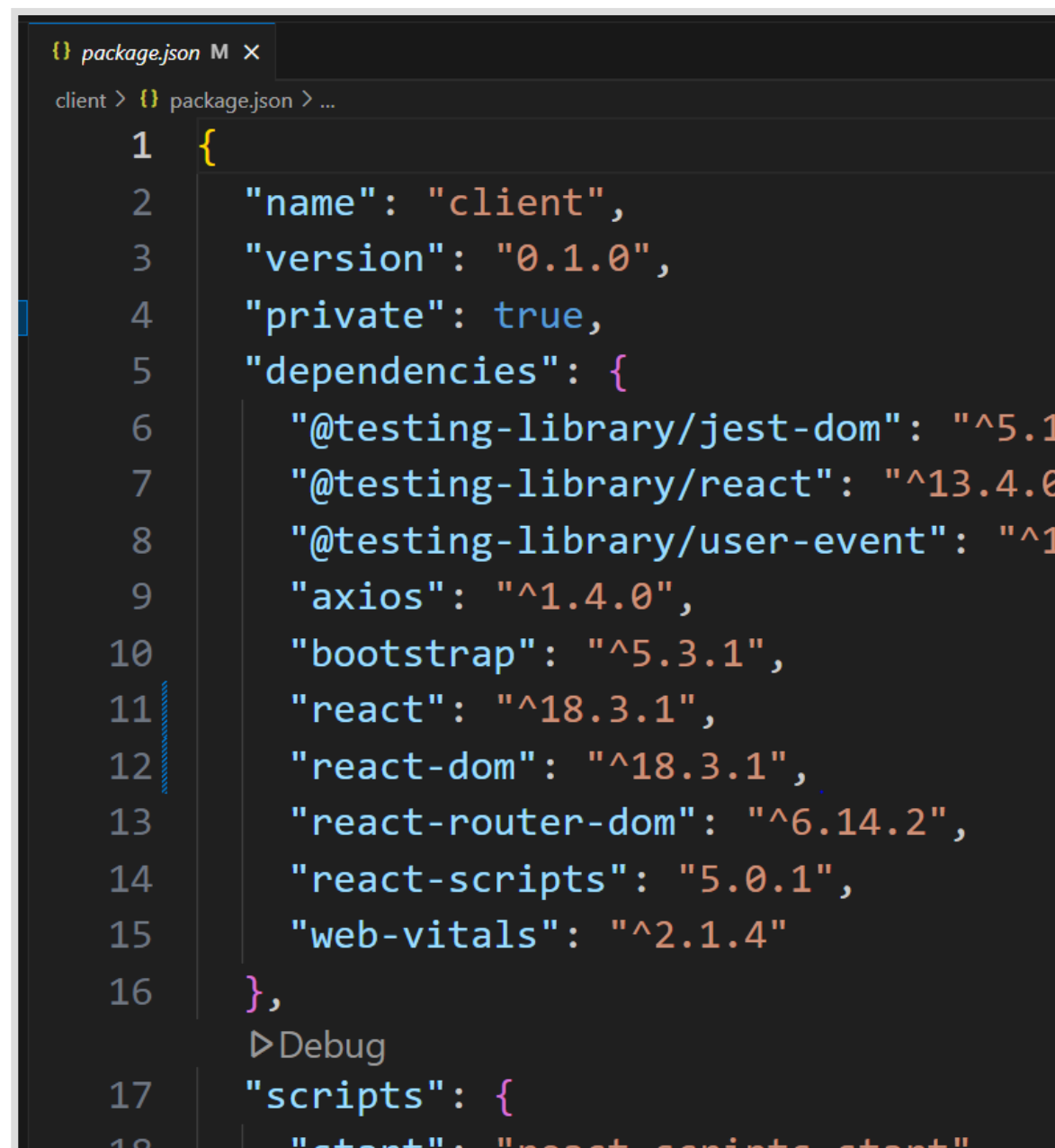
Now, open the frontend folder to install all the necessary tools we use.

For frontend, we use:

- React Js
- Bootstrap

- Axios

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



```
{ package.json M X
client > {} package.json > ...
1  {
2    "name": "client",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.11.1",
7      "@testing-library/react": "^13.4.0",
8      "@testing-library/user-event": "^13.3.0",
9      "axios": "^1.4.0",
10     "bootstrap": "^5.3.1",
11     "react": "^18.3.1",
12     "react-dom": "^18.3.1",
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"
```

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



```

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   "scripts": {
15     "test": "echo \"Error: no test\"
16   },
17   "keywords": [],

```

Backend Development:

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.

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.

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.

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.

User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

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.

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.

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.

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

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.

Fetching user bookings:

- In the bookings page, along with displaying the past bookings, we will also provide an option to cancel that booking.

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.

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.