Flight booking App Report

**Introduction**

**Project Title:** Development of Flight booking App **Team Members**

* G.Seshanth [au412721205043]
* M.Venkatakrishnan [au412721205056]
* M.N.Srighandh [au412721205047]
* M.Antony Bharath [au412721205701]

**Project Overview**

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 travellers to effortlessly discover, explore, and reserve flight tickets based on their unique preferences. Whether you're a frequent commuter or an occasional traveller, finding the perfect flight journey has never been easier.

**Objectives**

-To develop a user-friendly and responsive platform for ticket booking.

- To allow users to search for flights based on criteria like source, destination, departure date, return date, and class.

- Admins can add, update, or remove flight details, including schedules, prices, and seat availability.

**Scope**

The project eliminates the need to visit physical travel agencies, making the booking process faster and more efficient. This App provide users with round-the-clock access to book flights, check schedules, and manage bookings.

**Architecture**

**-Frontend:** The React-based frontend ensures a responsive and interactive UI, utilizing React Router for navigation and Redux for state management.

**-Backend:** The backend, powered by Node.js and Express.js, provides RESTful APIs for user authentication, product management, and order processing, stripe for Payment.

**-Database:** MongoDB serves as the database, storing information about users, products, orders, and transactions. The schema is optimized for scalability and quick retrieval of data.

**Prerequisites:**

- Node.js (v14 or later)

-MongoDB (locally or hosted)

-Express.js

-React.js

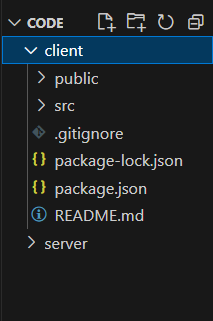
-HTML, CSS, JavaScript

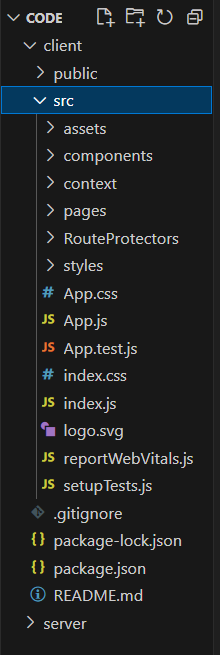
-Git

**Project Structure:**

**-Client directory:(Front end)**

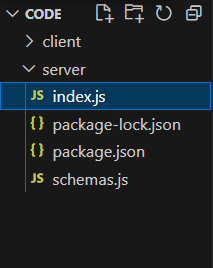
The below directory structure represents the directories and files in the client folder (front end) where, React js is used along with Api’s.





**-Server directory:(Back end)**

The below directory structure represents the directories and files in the server folder (back end) where, Node js, Express js and Mongo Db are used along with Api.



**Running the Application**

Provide commands to start the frontend and backend servers locally.

**-Frontend:** npm start in the client directory.

**-Backend:** npm run dev in the server directory.

**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.

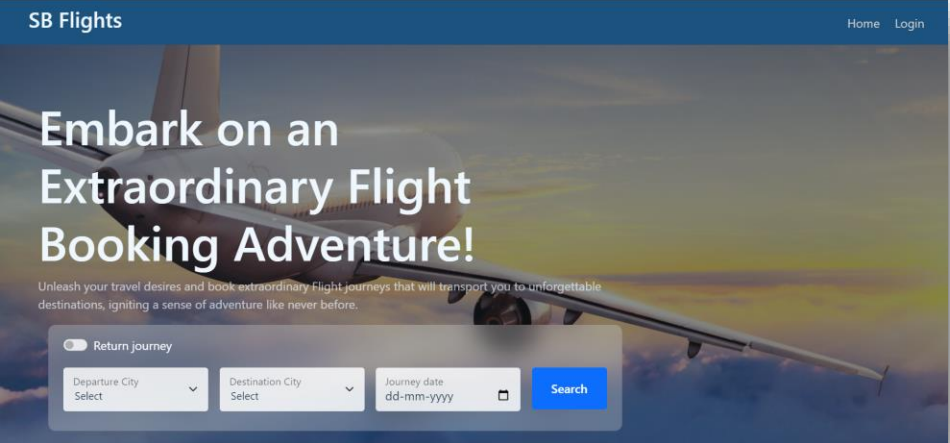
**ADMIN**

-Manages all bookings.

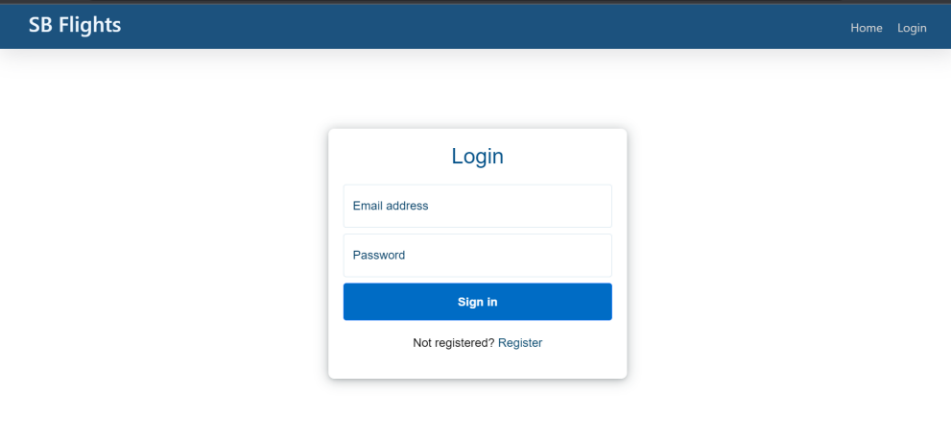
-Adds new flights and services.

-Monitor User activity.

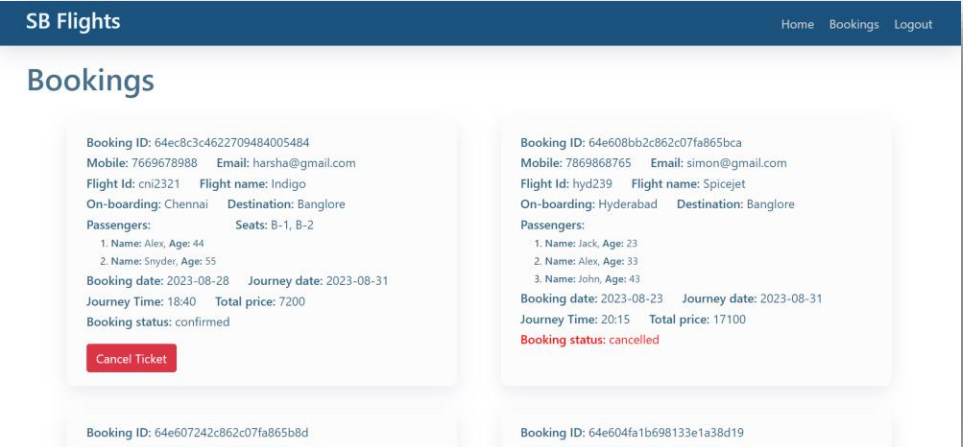
**-Landing page UI**

****

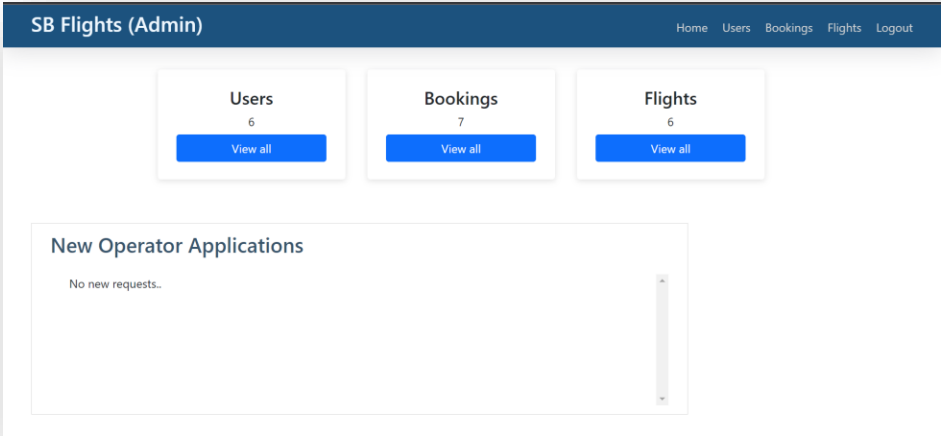
**-Authentication**

****

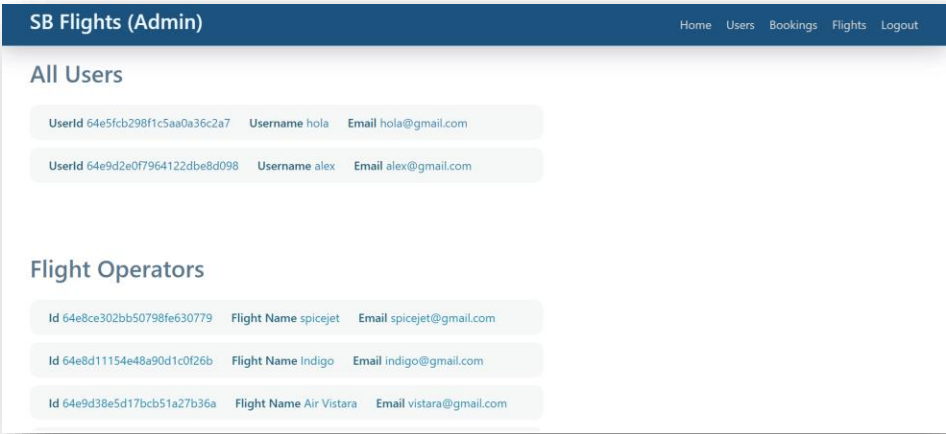
**-User bookings**

****

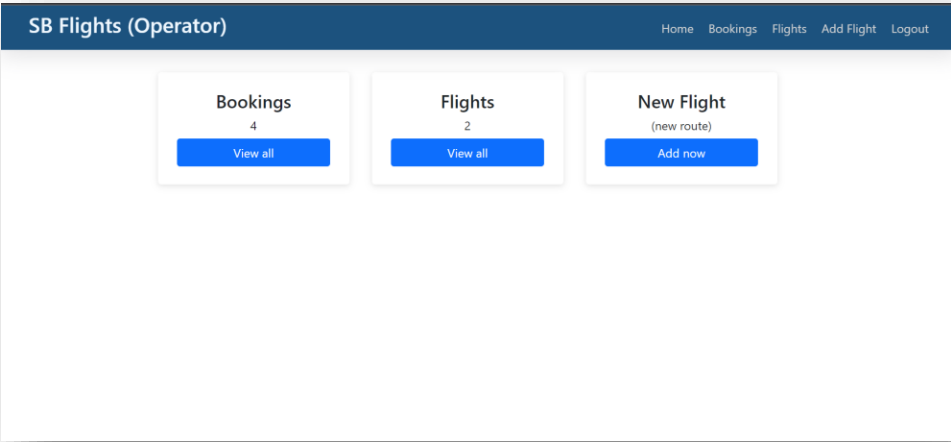
-**Admin Dashboard**

****

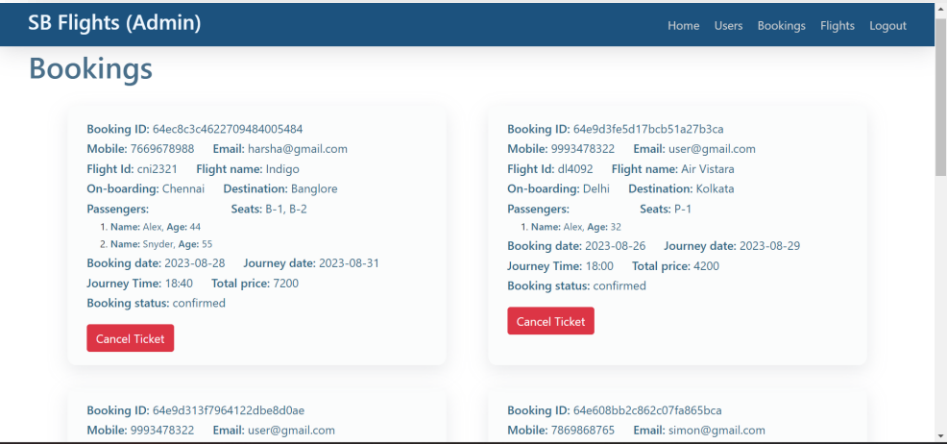
**-All users**

****

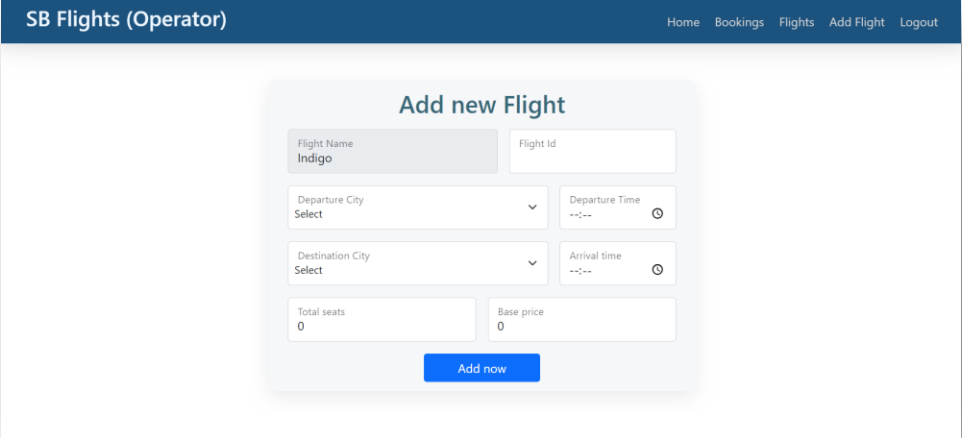
**-Flight Operator**

****

-**All Bookings**

****

-**New Flight**

****

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.

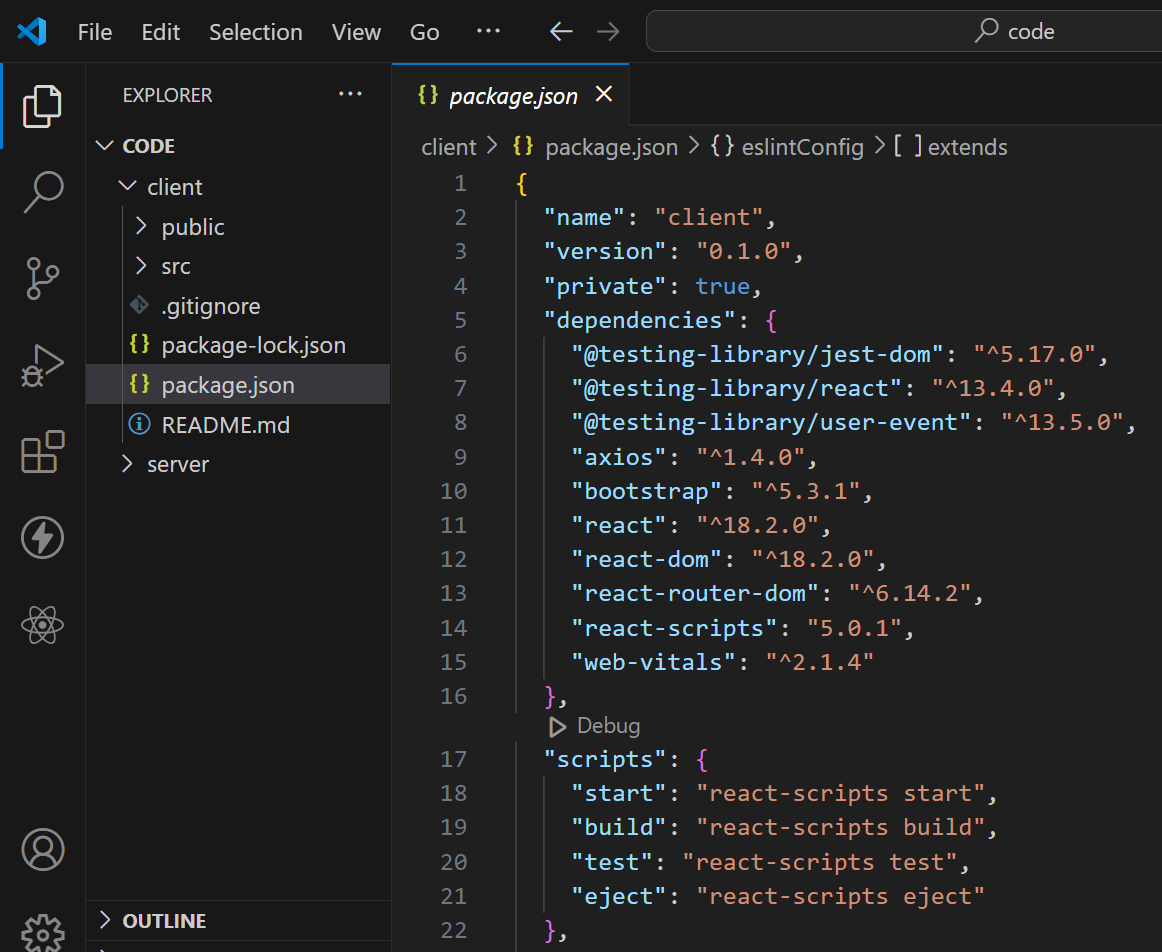
For frontend, we use:

-React Js

-Bootstrap

-Axios

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



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

For backend, we use:

-bcrypt

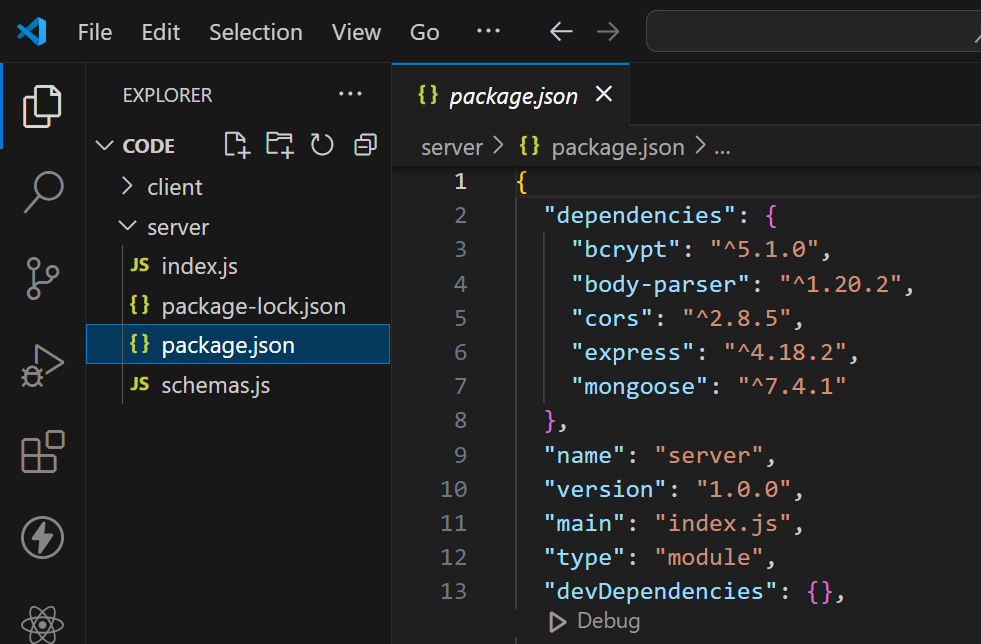
-body-parser

-cors

-express

-mongoose

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

****

**Front-end:**

1. **Login/Register**

-Create a Component which contains a form for taking the username and password.

-If the given inputs match the data of user or admin or flight operator then navigate it to their respective home page

1. **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.

1. **Fetching user bookings:**

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

1. **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 http request to the server to add it to the database.

1. **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.

1. **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.

1. **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.

1. **Error Handling:**

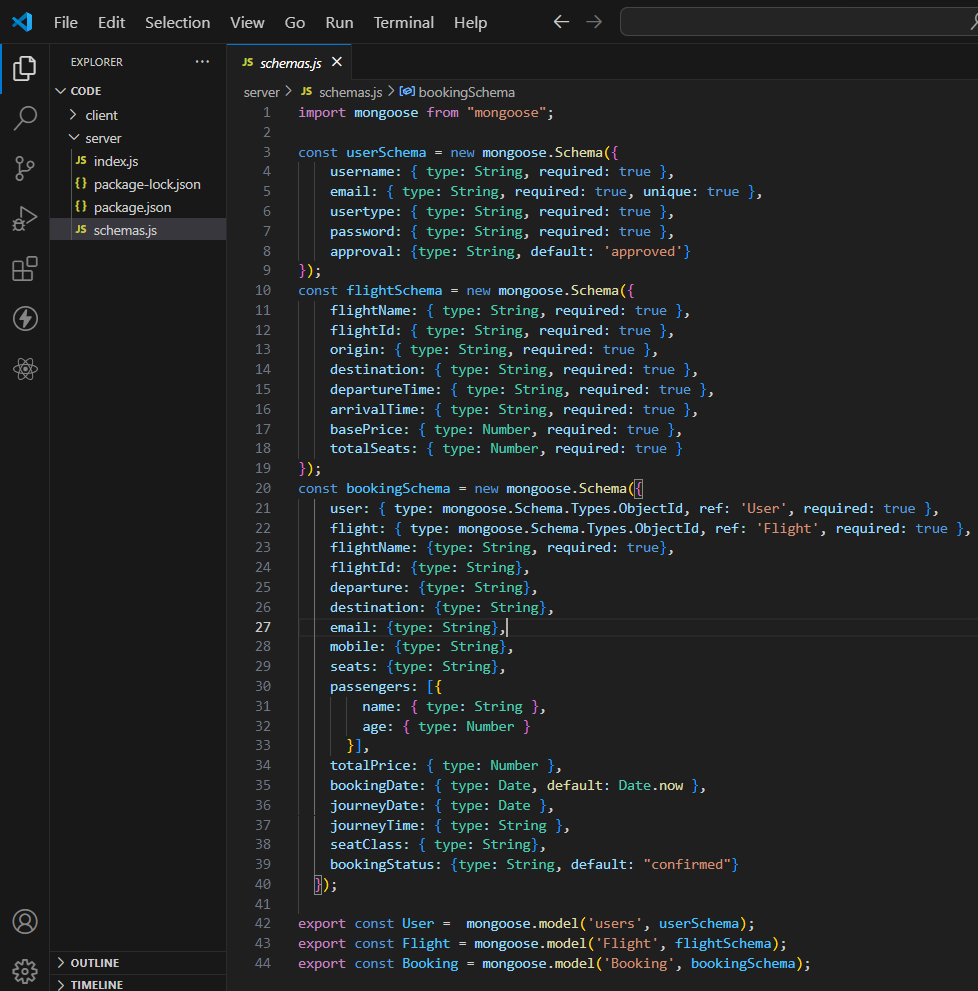
-Implement error handling middleware to catch and handle any errors that occur during the API requests.

-Return appropriate error responses with relevant error messages and HTTP status codes.

Database:

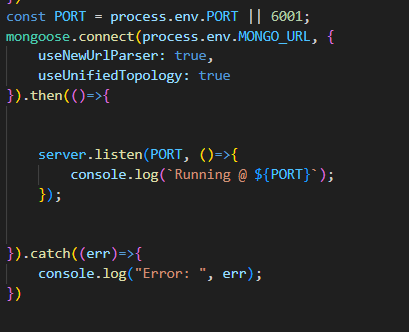
**Configure schema**

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.



Back-end:

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.

1. **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.

1. **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.

1. **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.

1. **User Authentication:**

-Create routes and middleware for user registration, login, and logout.

-Set up authentication middleware to protect routes that require user authentication.