NAAN MUDHALVAN PROJECT

MERN Stack Powered By Mongodb

FLIGHT BOOKING MERN STACK APPLICATION

Submitted by

NM2024TMID01239

ASARUDEEN.A	311421104006
KEERTHANA.T	311421104040
KARTHIKSSELVA.S	311421104038
SRINIVASAN.C	311421104313

COMPUTER SCIENCE AND ENGINEERING

MEENAKSHI COLLEGE OF ENGINEERING

K.K NAGAR WEST, CHENNAI-600078

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	iv
1	INTRODUCTION	3
2	PURPOSE	3
3	FEATURES	3
4	FRONTEND ARCHITECTURE	4
5	BACKEND ARCHITECTURE	12
6	MONGODB AND SCHEMAS	18
7	RUN THE APPLICATION	19
8	API DOCUMENTATION	20
9	USER AUTHENTICATION	20
10	USER INTERFACE	21
11	CONCLUSION	25

INTRODUCTION

The flight booking application provides users with the ability to search, book, and manage flight tickets seamlessly. Built using the MERN stack, the application is designed to deliver a responsive, scalable, and user-friendly interface while ensuring robust backend performance. This document elaborates on the architecture, design choices, features, and the working of the application.

PURPOSE

The purpose of this project is to create a modern flight booking platform that caters to the needs of customers and travel agencies by:

i. Allowing users to search for flights based on criteria like date, destination, and budget.
ii. Supporting secure user authentication for personal accounts.
iii. Enabling online booking, payment integration, and ticket management.
iv. Showcasing the capability of the MERN stack in developing a full-fledged web application.

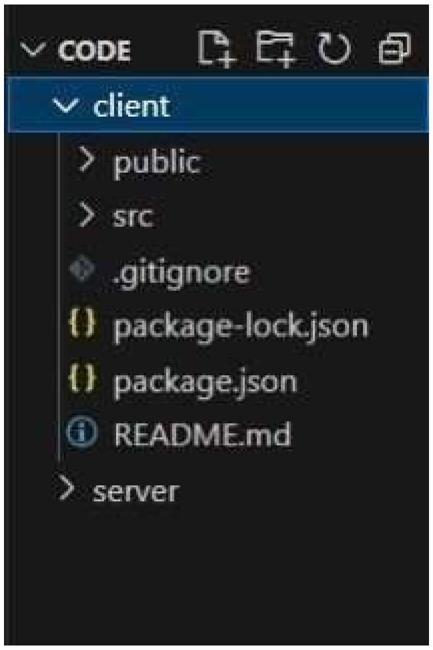
FEATURES

1. User Features:
   - Search and filter flights.
   - Book flights with real-time availability checks.
   - Secure login and registration using JWT.
   - Manage bookings (view, cancel, or reschedule).
   - Payment gateway integration.

2. Admin Features:
   - Manage flight inventory (add, update, delete flights). o Monitor bookings and revenue reports.
   - Manage user accounts.
3. General Features:
   - Mobile-first responsive design. o RESTful API for smooth client-server communication.
   - Real-time updates for bookings and cancellations.

FRONTEND ARCHITECTURE

The frontend is developed in React.js with a focus on modular components and a responsive design framework. Key architectural points include:


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

The image shows a project folder structure typical for a MERN stack application. Here's a detailed breakdown:

1. <u>Folder Overview</u>

i. The project is divided into two main parts:
   a. **<u>Client:</u>** Frontend code using React.
   b. **<u>Server:</u>** Backend code using Node.js and Express (not expanded in the image).

2. <u>Inside the client Folder</u>

The client folder contains the frontend portion of the application. Let's look at its components:

a. <u>public Folder</u>

i. Contains static files like images, fonts, or other assets.
ii. The key file here is:
   i. **<u>index.html:</u>** The entry point for the React application. It acts as the single HTML file for rendering the React components.

b. <u>src Folder</u>

- Houses the source code of the React application, including:
   i. **<u>Components:</u>** Reusable UI pieces like headers, footers, forms, etc.
   ii. **<u>Pages:</u>** Full-page components like login, search results, and booking screens.
   iii. **<u>App.js:</u>** The main file for setting up routes and global configurations.
   iv. **<u>Index.js:</u>** The root JavaScript file that renders the App component into index.html (in public).

c. <u>gitignore</u>

i. Specifies files and directories that Git should ignore (e.g., node_modules, .env).

d. <u>package.json</u>

i. Lists the project dependencies (e.g., React, React Router).
ii. Contains project metadata, scripts (e.g., npm start for starting the frontend).

e. **<u>package-lock.json</u>**

i. Auto-generated file that locks dependency versions for reproducibility.

f. **<u>README.md</u>**

ii. A markdown file for project documentation. Usually includes setup instructions, project goals, and usage notes.

3. **<u>server Folder</u>**

a) This folder (collapsed in the image) contains the backend code, likely structured as:

   i. **<u>Controllers:</u>** Handle business logic for API endpoints.

   ii. **<u>Routes:</u>** Define API endpoints.

   iii. **<u>Models</u>:** Define MongoDB schemas.

   iv. **<u>Middleware</u>:** Includes authentication and error handling.

<u>How to Use This Structure</u>

i. **<u>Setup Dependencies</u>:**

   a. Navigate to both client and server folders and run npm install to install frontend and backend dependencies.

ii. **<u>Run the App</u>:**

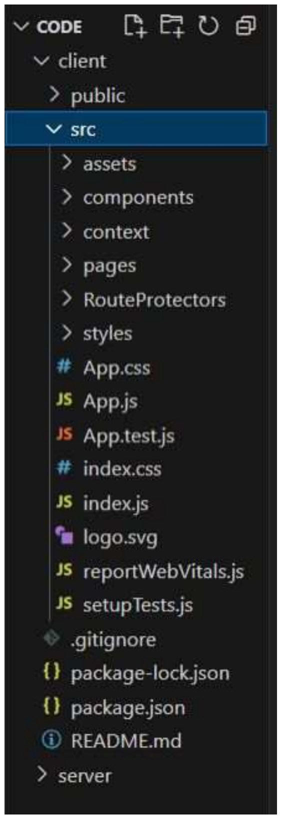
   a. Start the server (backend) from the server folder.

   b. Start the client (frontend) from the client folder.

iii. **<u>Development Workflow</u>:**

   a. Modify files in the src folder (for frontend) and server folder (for backend)

SRC ASSESTS



```
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
```

This image displays the **src folder structure** within the client directory of a MERN stack project. Here's an explanation of each folder/file specifically for a **Flight Booking Web App**:

1. Folder and File Breakdown

a. assets Folder

i. Contains static resources such as images, logos, or other media files used in the React application.

ii. Example:

   i. Flight company logos.

   ii. Background images for the booking portal.

b. <u>components Folder</u>

i. Houses reusable React components that can be shared across the app.
ii. Example Components for a Flight Booking App:
iii. **<u>Header.js:</u>** Navigation bar with links like"Home", "Search Flights", and "Profile".
iv. **<u>Footer.js:</u>** Footer section with contact information.
v. **<u>FlightCard.js</u>:** Displays details of a flight in a card format.
vi. **<u>SearchBar.js:</u>** Allows users to input search criteria like departure, arrival, and date.

c. <u>context Folder</u>

i. Contains files for global state management using React Context API.

<u>Example</u>:

i. **<u>AuthContext.js</u>:** Manages user authentication state (logged-in user data, JWT token).
ii. **<u>FlightContext.js</u>:** Stores flight search results and selected flight data for bookings.

d. <u>pages Folder</u>

i. Stores full-page components for navigation

<u>Example Pages</u>:

i. **<u>Home.js</u>:** Landing page showing popular flights or promotional offers.
ii. **<u>SearchResults.js</u>:** Displays the list of available flights based on search criteria.
iii. **<u>BookingPage.js:</u>** Allows users to complete flight booking.
iv. **Login.js & Register.js:** Handles user authentication.

RouteProtectors Folder

i. Contains components or higher-order functions (HOCs) to restrict access to certain routes.

Example:

ProtectedRoute.js: Ensures only authenticated users can access pages like booking history or admin dashboard.

Example Code:

```
<Route
  path="/bookings"
  element={<ProtectedRoute><BookingPage
/></ProtectedRoute>}
/>
```

f. styles Folder

Holds CSS or SCSS files for styling the application.

Example:

i. **global.css:** Global styling for the app.
ii. **FlightCard.css:** Specific styles for flight display components.

2. Key Files in src

a. App.js

i. The root component of the frontend application. Sets up routes and integrates the layout.

Example:

```
import { BrowserRouter as Router, Route, Routes } from 'react-
router-dom';
import Home from './pages/Home';
import Login from './pages/Login';
import SearchResults from './pages/SearchResults';
function App() {
```

```
return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/search" element={<SearchResults />} />
      </Routes>
 </Router>
  );
   }
  export default App;
```

b. App.test.js

i. Contains test cases for the App component using a testing library like **Jest** or **React Testing Library**.

c. index.js

i. Entry point for rendering the React app into the DOM.
ii. Typically renders the App component and wraps it with global providers (e.g., Context API or Redux).

Example:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
        ReactDOM.render(
          <React.StrictMode>
            <App />
          </React.StrictMode>,
          document.getElementById('root'));
```

d. logo.svg

Placeholder for the application logo (e.g., flight company logo).

E reportWebVitals.js

Measures performance metrics for the application. Optional for production but useful during development.

F setupTests.js

Configures the testing environment for the React app. Used for setting up tools like Jest.

How This Relates to a Flight Booking App

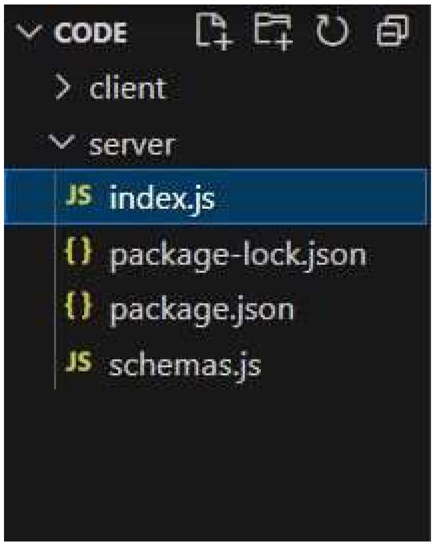
This structure is ideal for organizing a flight booking application:

1. **Frontend Pages (from pages):**
   i. **Login/Register Pages:** Authenticate users with JWT.
   ii. **Search Results Page:** Display flights after users search for routes.
   iii. **Booking Page:** Allow users to confirm their flight selection.
2. **Components:**
   i. **SearchBar.js:** For entering search queries like destination and dates.
   ii. **FlightCard.js:** Displays flight details, prices, and a "Book Now" button.
3. **State Management (from context):**
   i. **AuthContext.js:** Tracks if a user is logged in and stores the JWT token.
   ii. **FlightContext.js:** Manages flight search results and selected bookings.
4. **Protected Routes (from RouteProtectors):**
   i. **ProtectedRoute.js:** Ensures only logged-in users can access certain routes (e.g., /bookings).

5. **Styling (from styles):**
    i. CSS files ensure the app is visually appealing and responsive for all devices.

This structure is clean, modular, and scalable, perfectly suited for a flight booking web app. Let me know if you'd like help building specific components or setting up any of these files!

BACKEND ARCHITECTURE



```
CODE
  > client
  v server
      index.js
      package-lock.json
      package.json
      schemas.js
```

The image displays the **server folder structure** for the backend of

a **Flight Booking Web App** built with the MERN stack. Let's break it down in detail:

1. index.js

This is the entry point for the Node.js backend application. It initializes the server, connects to the database, and sets up middleware and routes.

Key Responsibilities:

i. **Server Initialization:**
   a. Uses **Express.js** to create the server.
   b. Starts listening on a specified port (e.g., http://localhost:5000).

ii. **Database Connection:**
   a. Connects to **MongoDB** using the **Mongoose** library.
   b. Logs a success or error message based on the connection status.

iii. **Middleware:**
   a. Sets up middleware for:
      i. Parsing JSON requests (express.json()).
      ii. Enabling CORS for cross-origin requests.
      iii. Handling authentication (e.g., JWT validation).

iv. **API Routing:**
   a. Defines API endpoints for users, flights, and bookings by importing routes from other files (not shown in this folder).

Example Code (index.js):

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

// App setup
const app = express();
const PORT = 5000;

// Middleware
app.use(cors());
app.use(express.json());
```

```
// Database connection
mongoose.connect('mongodb://localhost:27017/flightBookingApp'
, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch((err) => console.error('Database connection failed:', err));

// Routes
app.use('/api/users', require('./routes/userRoutes'));
app.use('/api/flights', require('./routes/flightRoutes'));
app.use('/api/bookings', require('./routes/bookingRoutes'));

// Start server
app.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
```

2. schemas.js

This file contains **Mongoose schemas and models** for MongoDB collections. Each schema defines the structure of the data stored in the database.

Key Schemas for a Flight Booking App:

1. **User Schema:**
   - Fields: name, email, password, role (user/admin).
   - Purpose: Stores user account information.
2. **Flight Schema:**
   - Fields: flightNumber, departure, destination, price, availableSeats, departureTime, arrivalTime.
   - Purpose: Stores flight details.

3. **Booking Schema:**
    - Fields: userId, flightId, passengers, totalPrice, status.
    - Purpose: Tracks bookings associated with users and flights.

Example Code (schemas.js):

```
const mongoose = require('mongoose');

// User Schema
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, default: 'user' },
});

// Flight Schema
const flightSchema = new mongoose.Schema({
  flightNumber: { type: String, required: true },
  departure: { type: String, required: true },
  destination: { type: String, required: true },
  price: { type: Number, required: true },
  availableSeats: { type: Number, required: true },
  departureTime: { type: Date, required: true },
  arrivalTime: { type: Date, required: true },
});

// Booking Schema
const bookingSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  flightId: { type: mongoose.Schema.Types.ObjectId, ref: 'Flight', required: true },
  passengers: { type: Number, required: true },
  totalPrice: { type: Number, required: true },
  status: { type: String, default: 'confirmed' },
});

// Models
const User = mongoose.model('User', userSchema);
const Flight = mongoose.model('Flight', flightSchema);
const Booking = mongoose.model('Booking', bookingSchema);

module.exports = { User, Flight, Booking };
```

3. <u>package.json</u>

This file contains metadata and dependencies for the backend server.

<u>Key Sections</u>:

i. **<u>Dependencies:</u>**
   a. Common dependencies for a flight booking app:
      i. express: Web framework for creating APIs.
      ii. mongoose: ORM for MongoDB.
      iii. jsonwebtoken: For user authentication.
      iv. bcrypt: For password hashing.
      v. <u>cors: To handle cross-origin requests.</u>

ii. **<u>Scripts:</u>**
   a. Commands for running the server:
      i. npm start: Starts the backend server.
      ii. npm run dev: Runs the server with **nodemon** for auto-restart during development.

Example package.json:

```
{
 "name": "flight-booking-server",
 "version": "1.0.0",
 "main": "index.js",
 "scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js"
 },
 "dependencies": {
  "express": "^4.18.2",
  "mongoose": "^6.5.2",
  "jsonwebtoken": "^9.0.0",
  "bcrypt": "^5.1.0",
  "cors": "^2.8.5"
 },
 "devDependencies": {
  "nodemon": "^2.0.22"
 }
}
```

4. package-lock.json

Auto-generated file that locks dependency versions for reproducible builds.

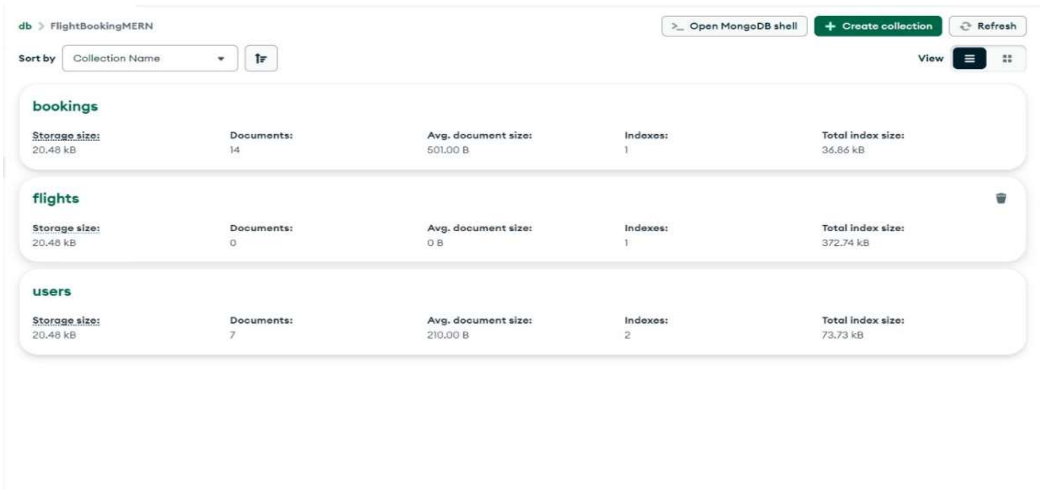
Ensures that the exact same versions of dependencies are installed.

How This Relates to the Flight Booking App

i. **index.js:** Manages the API and server connection to MongoDB.
ii. **schemas.js:** Defines the MongoDB data structure for users, flights, and bookings.
iii. **package.json:** Ensures all required backend tools and libraries are installed.
iv. **Server Functionality Overview:**

a. Users can register/login and make flight bookings.
b. Admins can manage flights (add, update, delete).
c. All actions (e.g., user booking a flight) are linked

MONGODB AND SCHEMA



	Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
bookings	20.48 kB	14	501.00 B	1	36.86 kB
flights	20.48 kB	0	0 B	1	372.74 kB
users	20.48 kB	7	210.00 B	2	73.73 kB

Database: MongoDB

MongoDB stores the application's data in a NoSQL format. Key collections include:

1. **User Collection:**
   i. Fields: userId, name, email, passwordHash, role (user/admin).

    ii. Purpose: Manages user accounts.

2. **Flights Collection:**
    i. Fields: flightId, airline, departure, destination, departureTime, arrivalTime, price, availableSeats.
    ii. Purpose: Stores flight details and availability.

3. **Bookings Collection:**
    i. Fields: bookingId, userId, flightId, passengers, totalPrice, status (confirmed/canceled).
    ii. Purpose: Tracks bookings and associated users.
4. **Payment Collection (optional):**
    i. Fields: paymentId, bookingId, userId, amount, paymentStatus.
    ii. Purpose: Logs payment details.

Schema relations use references (userId, flightId) to maintain data integrity.

RUN THE APPLICATION

1. **Prerequisites:**
    i. Install Node.js, MongoDB, and npm/yarn on your system.
2. **Steps to Run:**
    a. **Backend:**
        i. Navigate to the backend directory.
        ii. Install dependencies: npm install.
        iii. Start the server: npm start.
    b. **Frontend:**
        i. Navigate to the frontend directory.
        ii. Install dependencies: npm install.
        iii. Start the development server: npm start.
    c. **Database:**
        i. Start the MongoDB service.
        ii. Use a database GUI like MongoDB Compass to monitor collections.
3. Access the application in the browser at **http://localhost:3000.**

API DOCUMENTATION

User Authentication:

a. POST /auth/login: Authenticate user credentials.
b. POST /auth/register: Register a new user. Flight Management:
c. GET /flights: Fetch available flights.
d. POST /flights: Add new flight (adminonly).
e. DELETE /flights/:id: Remove a flight (admin-only).

Booking Management:

f. POST /bookings: Create a booking.
g. GET /bookings/:userId: Retrieve user bookings.
h. DELETE /bookings/:id: Cancel a booking. ii. Payment (optional):
i. POST /payments: Process a payment for a booking.

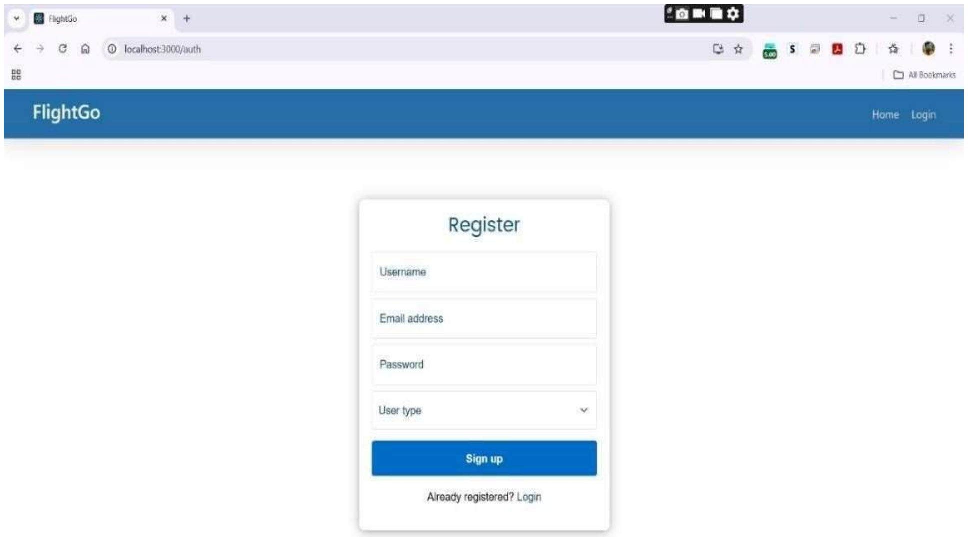
USER AUTHENTICATION

The application implements JWT (JSON Web Token) for secure user authentication:

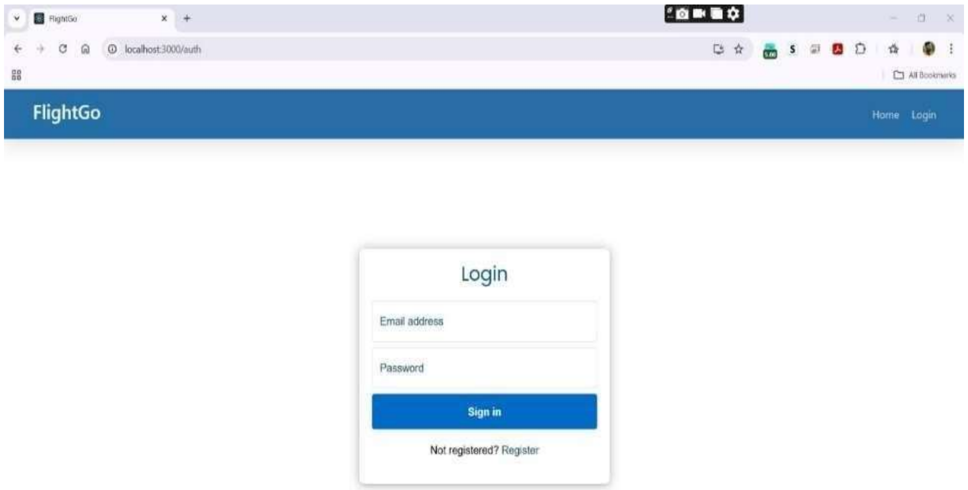
i. Registration: Passwords are hashed using bcrypt before being saved.
ii. Login: On successful login, a JWT is generated and sent to the client.
iii. Protected Routes: Middleware validates the JWT before granting access.

USER INTERFACE:

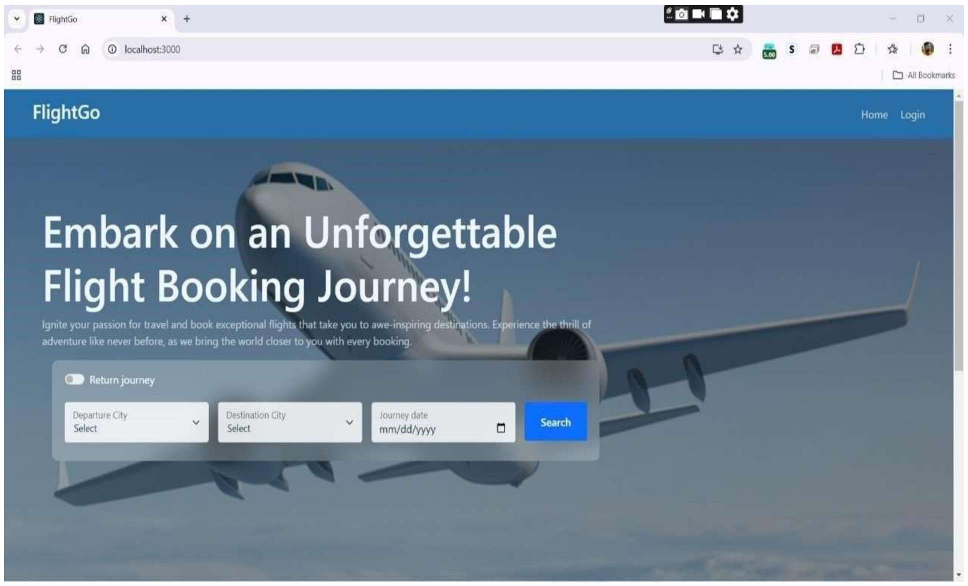
REGISTER INTERFACE



LOGIN PAGE



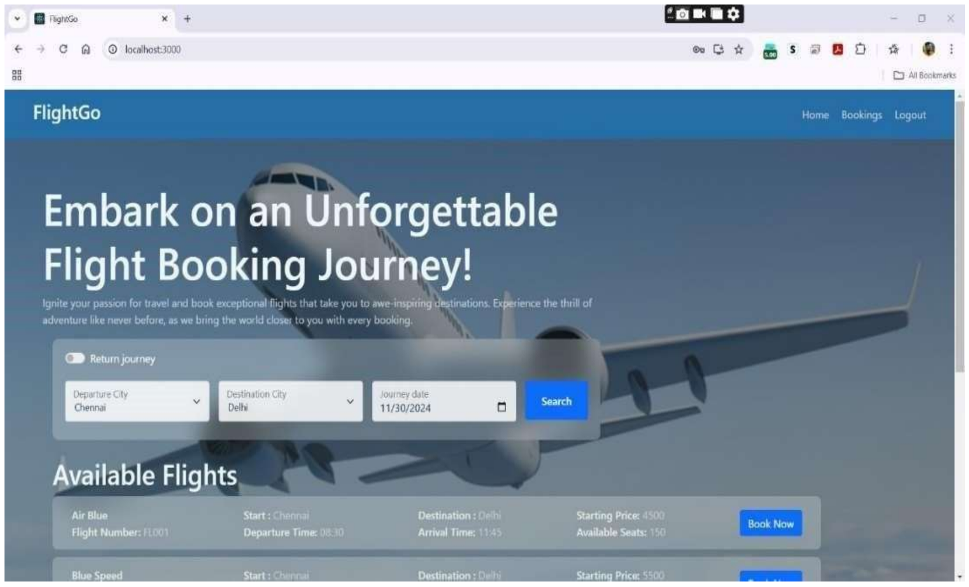
<u>BOOKING INTERFACE:</u>



Embark on an Unforgettable Flight Booking Journey!

Ignite your passion for travel and book exceptional flights that take you to awe-inspiring destinations. Experience the thrill of adventure like never before, as we bring the world closer to you with every booking.

<u>SELECTING DEPARTURE AND DESTINATION:</u>



Embark on an Unforgettable Flight Booking Journey!

Ignite your passion for travel and book exceptional flights that take you to awe-inspiring destinations. Experience the thrill of adventure like never before, as we bring the world closer to you with every booking.

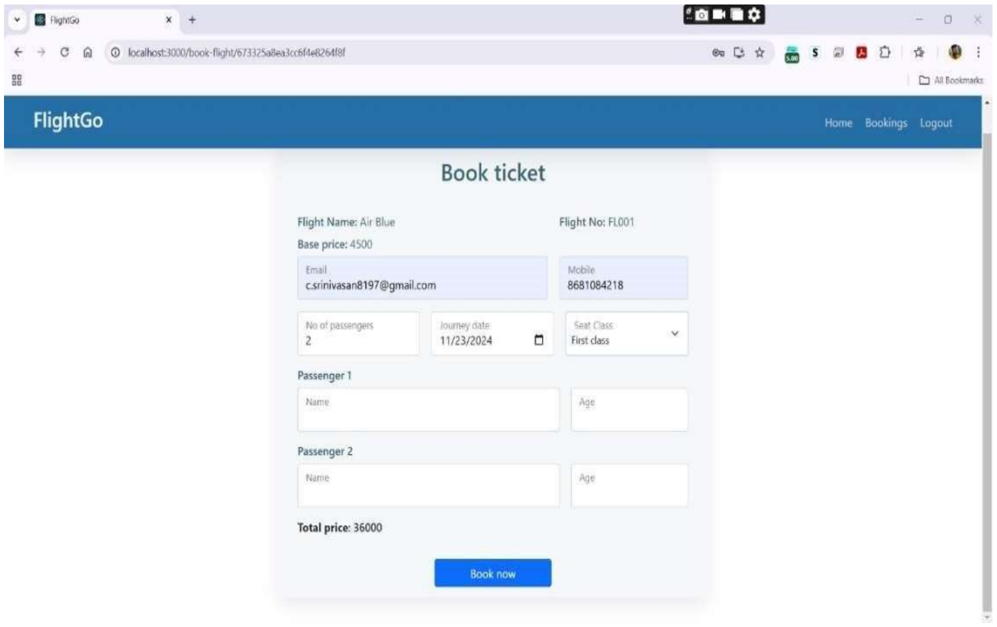
Departure City: Chennai
Destination City: Delhi
Journey date: 11/30/2024

Available Flights

Air Blue
Flight Number: FL001
Start : Chennai
Departure Time: 08:30
Destination : Delhi
Arrival Time: 11:45
Starting Price: 4500
Available Seats: 150

Blue Speed
Start : Chennai
Destination : Delhi
Starting Price: 5500

TICKET BOOKING INTERFACE



Book ticket

Flight Name: Air Blue **Flight No:** FL001

Base price: 4500

Email: c.srinivasan8197@gmail.com

Mobile: 8681084218

No of passengers: 2

Journey date: 11/23/2024

Seat Class: First class

Passenger 1

Name

Age

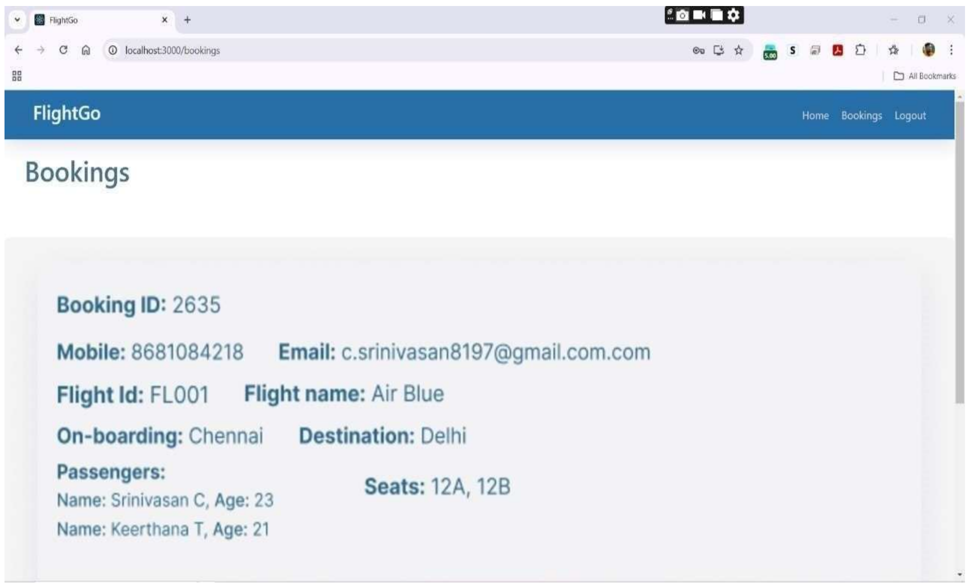
Passenger 2

Name

Age

Total price: 36000

TICKET BOOKING CONFIRMATION



Bookings

Booking ID: 2635

Mobile: 8681084218 **Email:** c.srinivasan8197@gmail.com.com

Flight Id: FL001 **Flight name:** Air Blue

On-boarding: Chennai **Destination:** Delhi

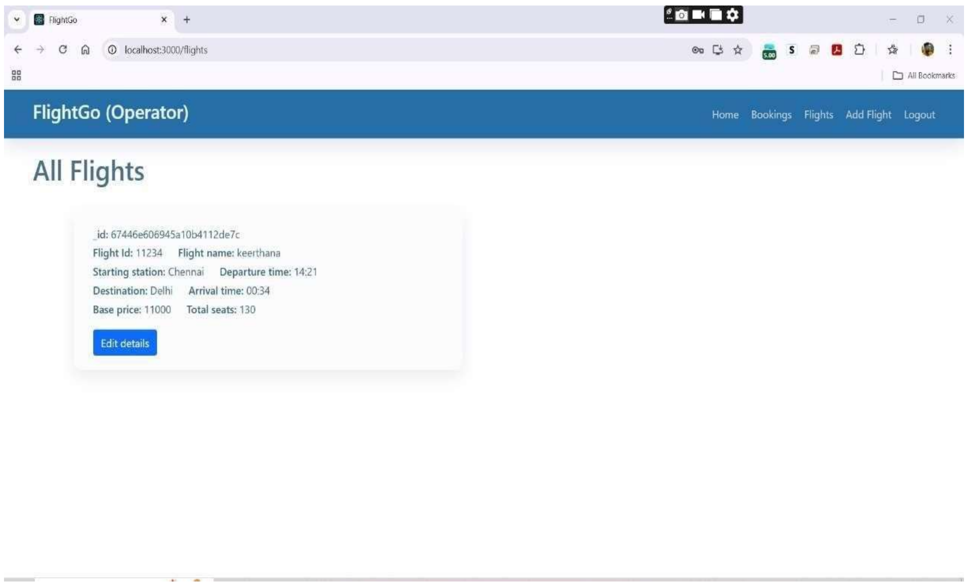
Passengers:

Name: Srinivasan C, **Age:** 23

Name: Keerthana T, **Age:** 21

Seats: 12A, 12B

ADDING FLIGHTS



FlightGo

localhost:3000/flights

All Bookmarks

FlightGo (Operator)

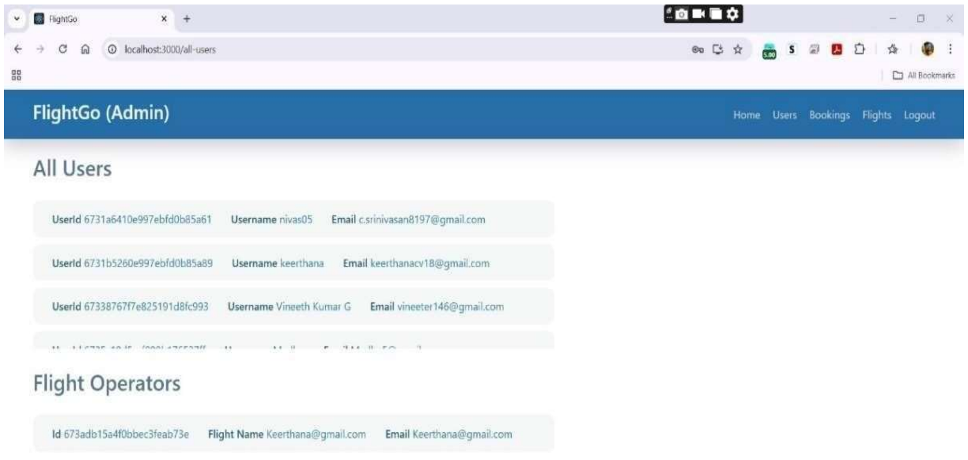
Home Bookings Flights Add Flight Logout

All Flights

_id: 67446e606945a10b4112de7c
Flight Id: 11234 Flight name: keerthana
Starting station: Chennai Departure time: 14:21
Destination: Delhi Arrival time: 00:34
Base price: 11000 Total seats: 130

Edit details

ADMIN PAGE



FlightGo

localhost:3000/all-users

All Bookmarks

FlightGo (Admin)

Home Users Bookings Flights Logout

All Users

UserId 6731a6410e997ebfd0b85a61 Username nivas05 Email c.srinivasan8197@gmail.com

UserId 6731b5260e997ebfd0b85a89 Username keerthana Email keerthanacv18@gmail.com

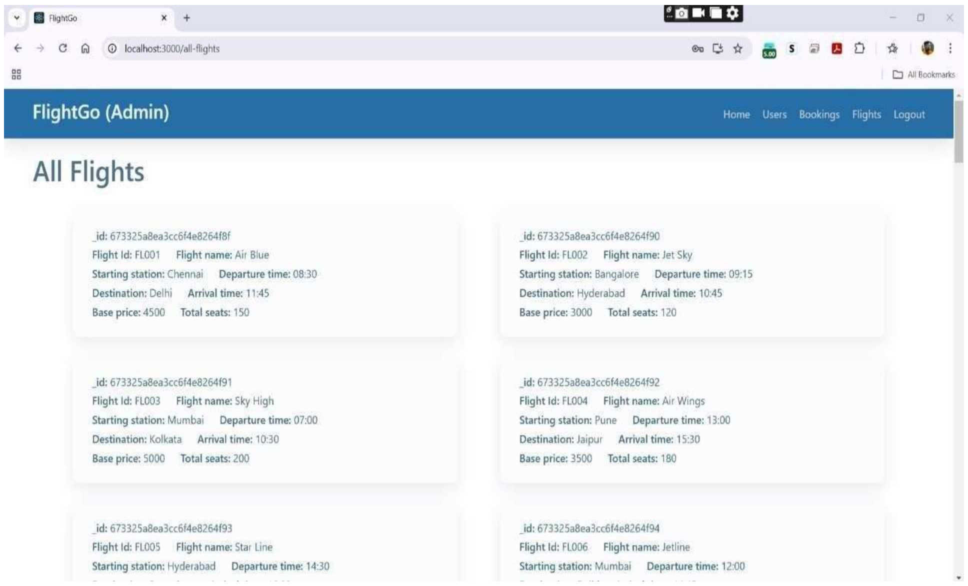
UserId 67338767f7e825191d8fc993 Username Vineeth Kumar G Email vineeter146@gmail.com

[illegible]

Flight Operators

Id 673adb15a4f0bbec3feab73e Flight Name Keerthana@gmail.com Email Keerthana@gmail.com

ALL AVAILABLE FLIGHTS:



All Flights

_id: 673325a8ea3cc6f4e8264f8f
Flight Id: FL001 Flight name: Air Blue
Starting station: Chennai Departure time: 08:30
Destination: Delhi Arrival time: 11:45
Base price: 4500 Total seats: 150

_id: 673325a8ea3cc6f4e8264f90
Flight Id: FL002 Flight name: Jet Sky
Starting station: Bangalore Departure time: 09:15
Destination: Hyderabad Arrival time: 10:45
Base price: 3000 Total seats: 120

_id: 673325a8ea3cc6f4e8264f91
Flight Id: FL003 Flight name: Sky High
Starting station: Mumbai Departure time: 07:00
Destination: Kolkata Arrival time: 10:30
Base price: 5000 Total seats: 200

_id: 673325a8ea3cc6f4e8264f92
Flight Id: FL004 Flight name: Air Wings
Starting station: Pune Departure time: 13:00
Destination: Jaipur Arrival time: 15:30
Base price: 3500 Total seats: 180

_id: 673325a8ea3cc6f4e8264f93
Flight Id: FL005 Flight name: Star Line
Starting station: Hyderabad Departure time: 14:30

_id: 673325a8ea3cc6f4e8264f94
Flight Id: FL006 Flight name: Jetline
Starting station: Mumbai Departure time: 12:00

CONCLUSION

This MERN-based flight booking application demonstrates a complete solution for modern travel platforms, integrating usercentric design, scalable backend systems, and a NoSQL database. The use of React for frontend and MongoDB for the database ensures flexibility and scalability, while Node.js and Express.js provide a reliable API layer. By incorporating authentication, realtime updates, and a modular architecture, the app meets the functional and non-functional requirements of a robust flight booking platform.