Full Stack Development with MERN

1. Introduction

Project Title: Flight Booking app using MERN
Team Members:

- J.SAKTHIVEL– Full stack Developer
- T.DILLI GANESH – Full-stack Developer
- T.SAMJOCOB– Front-end Developer
- B.DINAKARA PANDIAN– Database

2. Project Overview

A flight booking app using MongoDB serves to efficiently manage and scale large amounts of dynamic, real-time data. Key purposes include:

1. **Flexibility:** MongoDB's schema-less design accommodates evolving flight and booking data.
2. **Scalability:** It can scale horizontally to handle high traffic during peak times.
3. **High Availability:** Replication ensures uptime and data redundancy.
4. **Real-time Updates:** Handles live flight status, availability, and booking updates.
5. **Geospatial Queries:** Supports location-based features like finding nearby airports.
6. **Performance:** Optimized for fast, complex queries and aggregations on large datasets.
7. **Cost-Effective:** Suitable for startups with its open-source nature and scalability options.

Features:

- User registration and login with JWT authentication.
- **Flight management** (create, update, delete flight schedules).
- **User dashboard** for booking status and history tracking.
- **Admin dashboard** for flight and booking management.

3. Architecture

- **Frontend:** Built using React, the frontend provides a dynamic user interface for the flight booking system. It includes components for user authentication, flight search, and booking management, using React Router for seamless navigation.

- **Backend:** The backend is built with Node.js and Express.js. It handles requests, manages authentication, processes bookings, and connects with MongoDB for data storage. Controllers manage flight data and user bookings.

- **Database:** MongoDB stores information about users, flights, and bookings. It is structured with collections for each major feature (e.g., users, flights, bookings) to enable quick retrieval and updates.

4. Setup Instructions

Prerequisites:

- Node.js v14+
- MongoDB v4+
- (Optional) npm or yarn for package management

Installation:

1. Clone the repository: git clone <repository-url>
2. Navigate to both the backend and frontend directories and install dependencies:
   For Backend: cd backend
   npm install
   For Frontend: cd ../frontend
   npm install
3. Set up environment variables by creating a .env file in the backend directory with database connection strings, JWT secret, etc.

5. Folder Structure

Client:

The frontend directory contains:

- src/components - Contains reusable React components.
- src/pages - Different pages (e.g., FlightList, Login, Register).
- src/utils - Utility functions and API calls.
- src/styles - CSS files for styling.

Server:

The backend directory is organized as follows:

- config - Database connection configuration.
- controllers - Functions to handle business logic.
- routers - Defines routes for various API endpoints.
- middlewares - Middleware for authentication and error handling.
- schemas - MongoDB schemas and models.

6. Running the Application

Frontend:

Start the frontend server: npm start

Backend:

Start the backend server: npm start

7. API Documentation

The backend exposes several endpoints, including:

- **POST** /api/components/login.jsx- User login with JWT authentication.
- **POST** /api/components/register.jsx - User registration.
- **GET** /pages/FlightBookings.jsx- Fetch available flights based on search criteria.

Example Response:

```
{
  "status": "success",
  "data": {
```

```
  "flights": [
    {
      "id": "flight-id",
      "airline": "Airline Name",
      "departure": "Departure Location",
      "destination": "Destination Location",
      "departure-Time": "YYYY-MM-DDTHH:MM:SS",
      "arrival-Time": "YYYY-MM-DDTHH:MM:SS",
      "price": "Flight Price"
    }
  ]
 }
}}
```

8. Authentication

Authentication is handled using JWT tokens. After login, users receive a token stored in local storage. Protected routes require a valid token for access, which is verified using middleware.

9. User Interface

The interface includes:

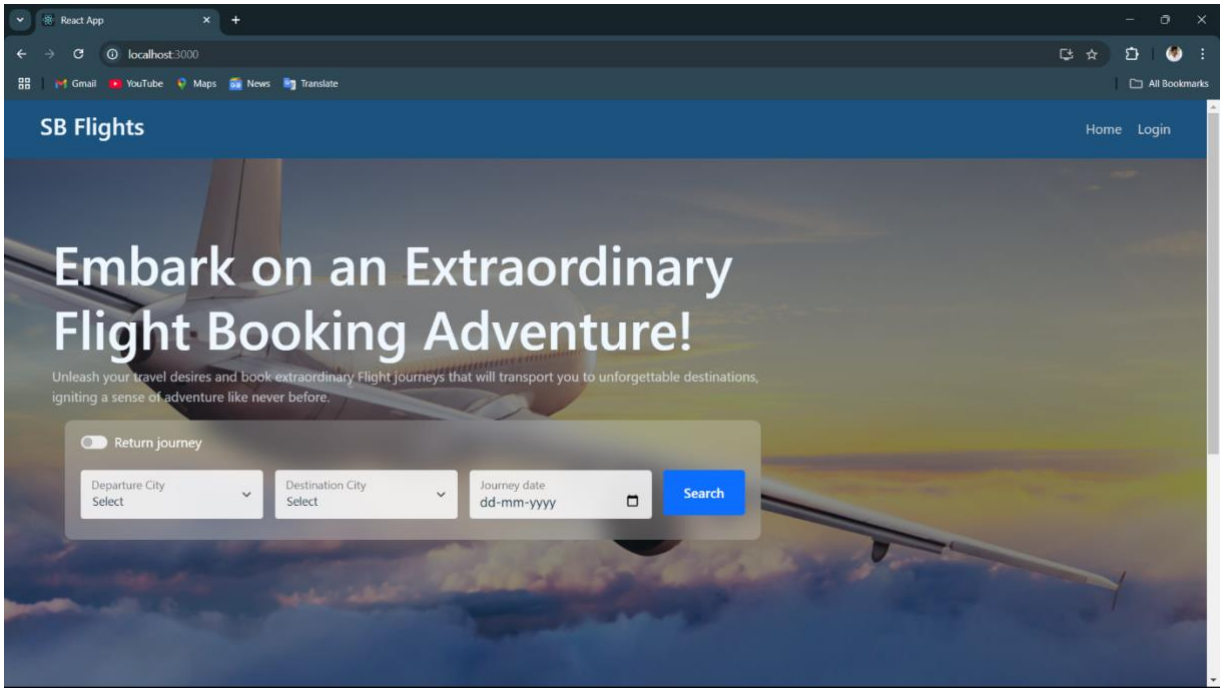
- Login and registration form for user authentication.
- Flight search page displaying available flights based on user preferences.
- Dashboard page where users can view their booking history and monitor current booking status.

10. Testing

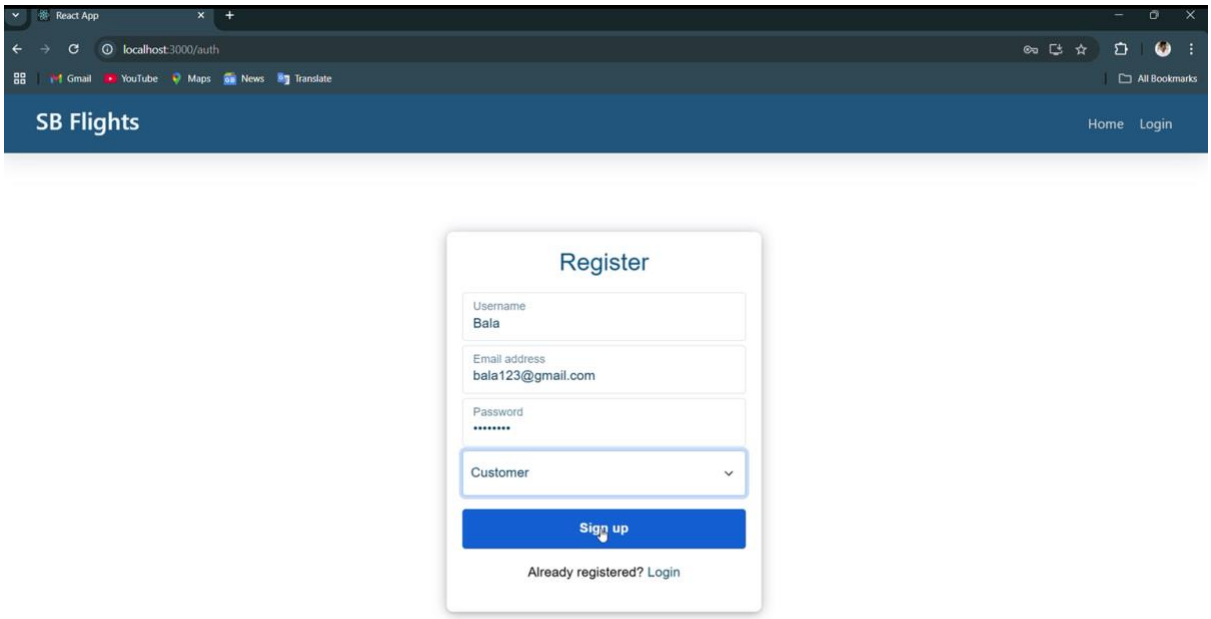
Testing is performed using tools like Jest and Postman for API testing. Unit tests cover component functionality and API response validation.

11. Screenshots

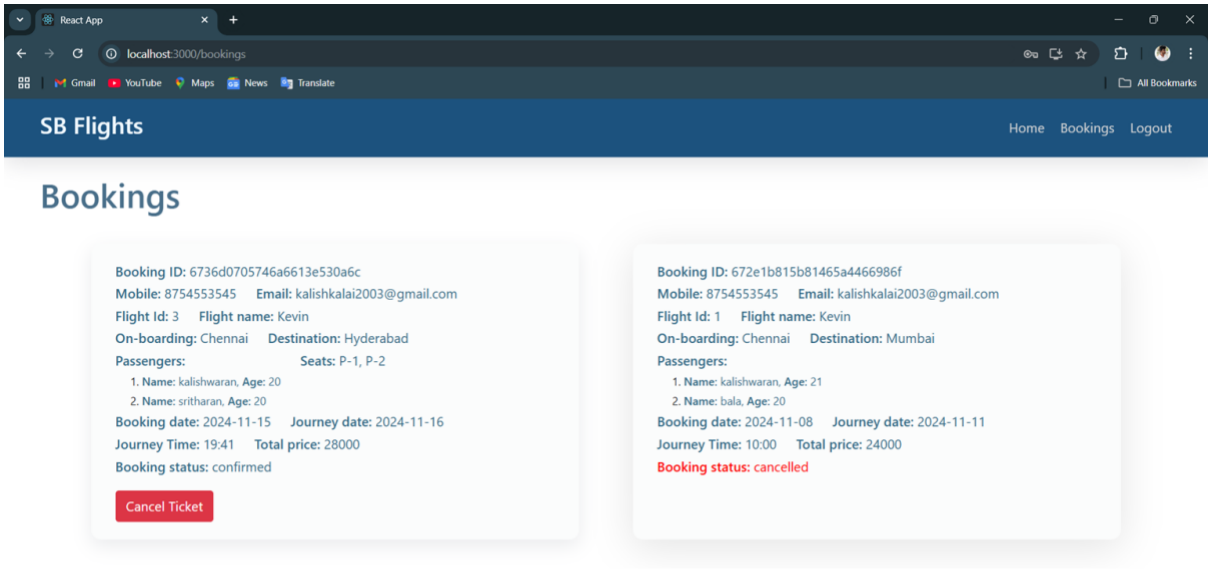
Landing page



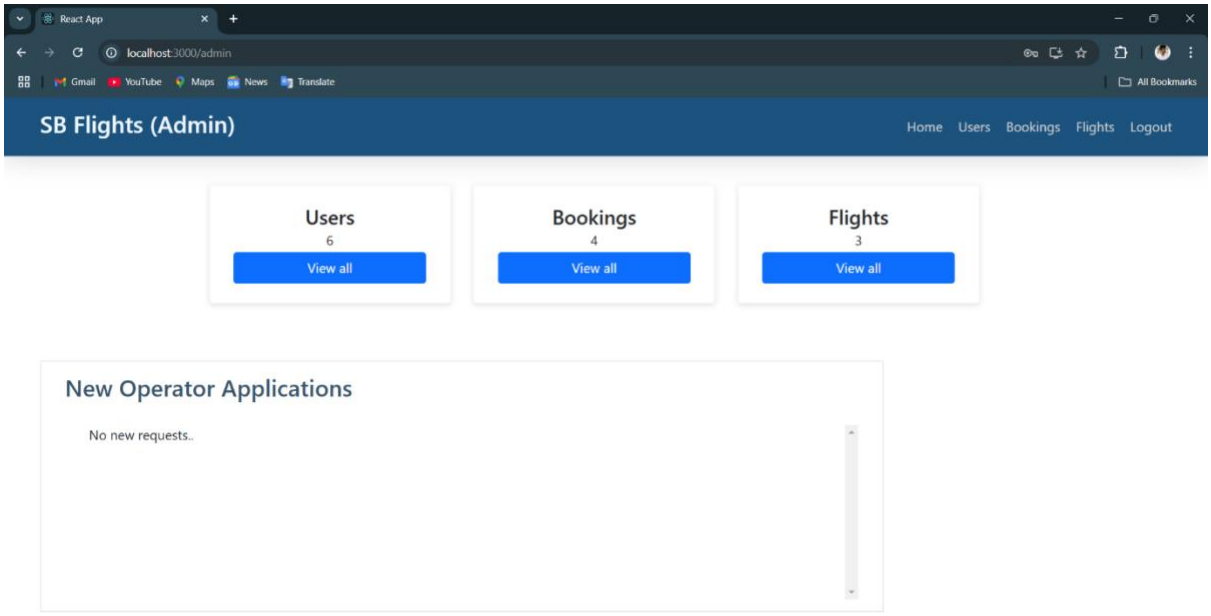
Login Form and Authentication



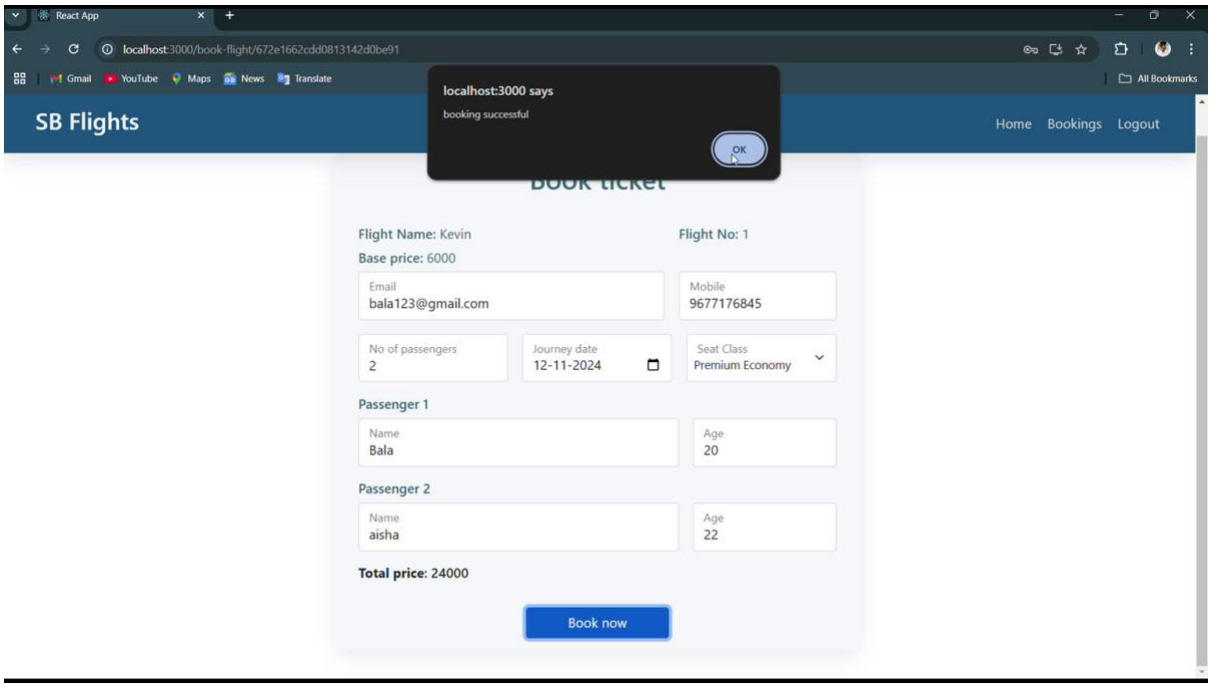
User Bookings



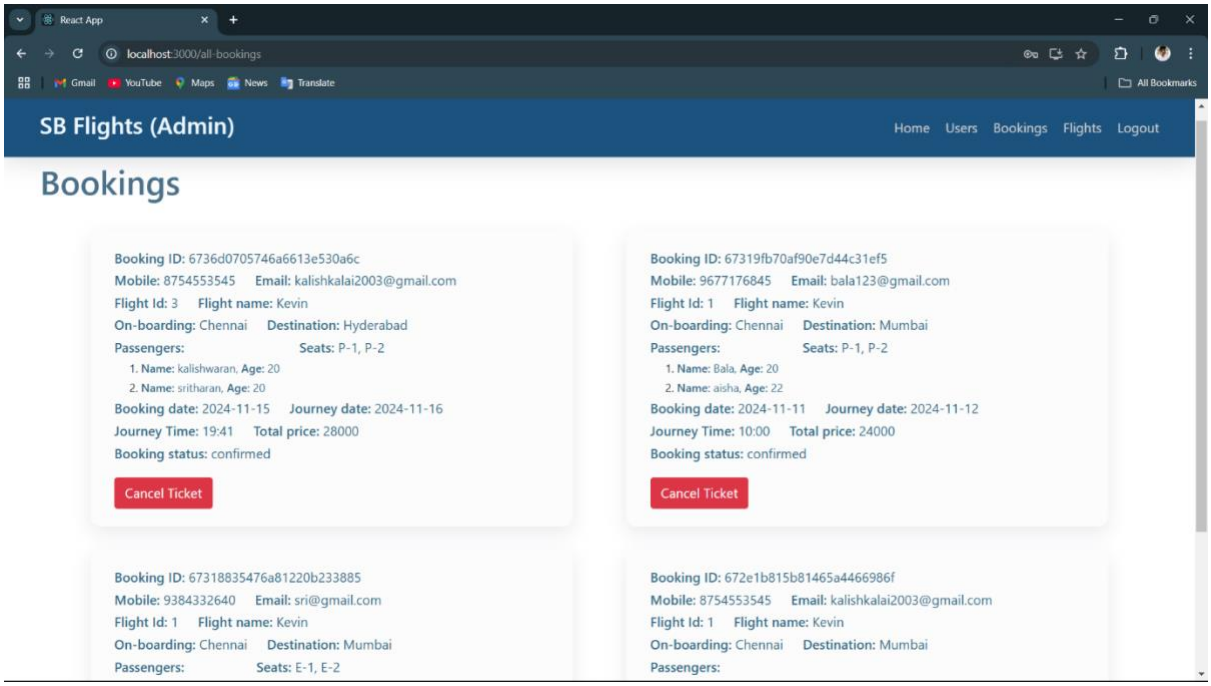
Admin Dashboard



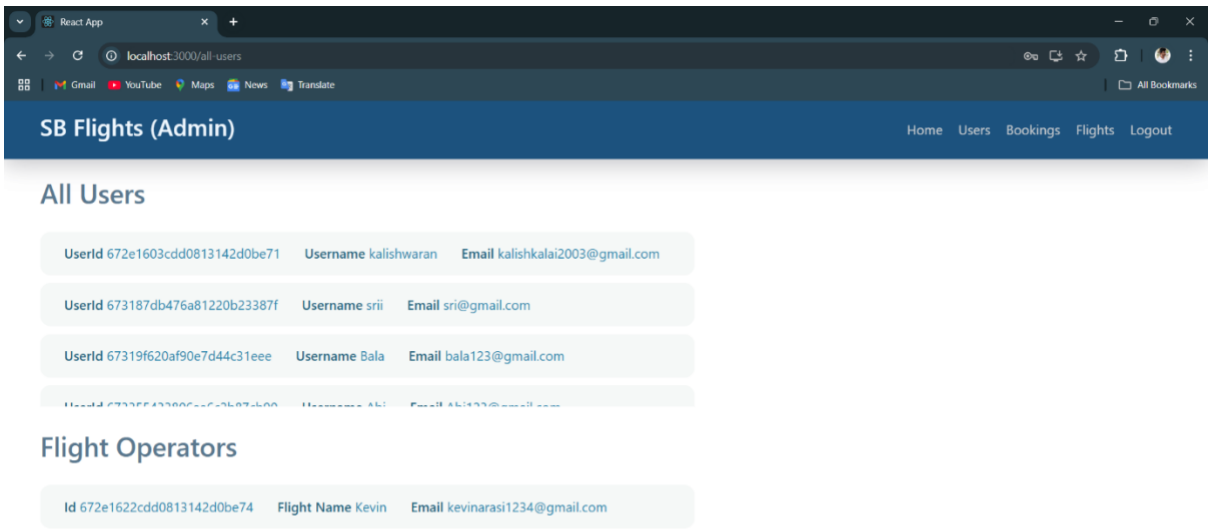
Flight Ticket Booking



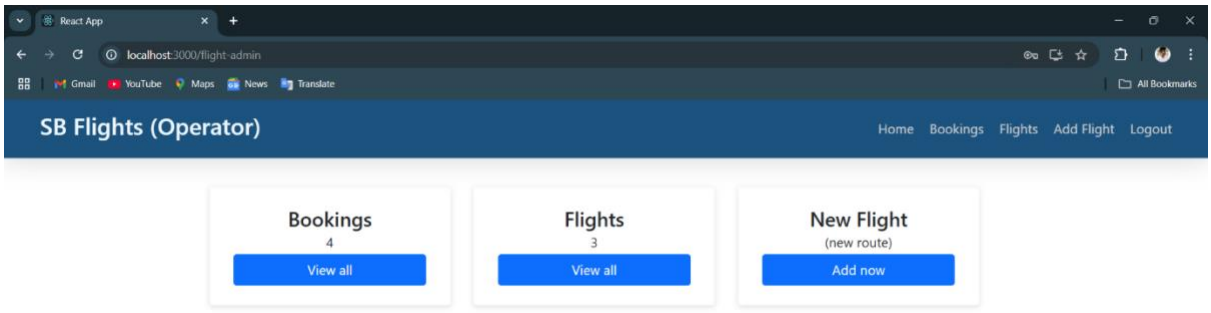
All Flight Bookings



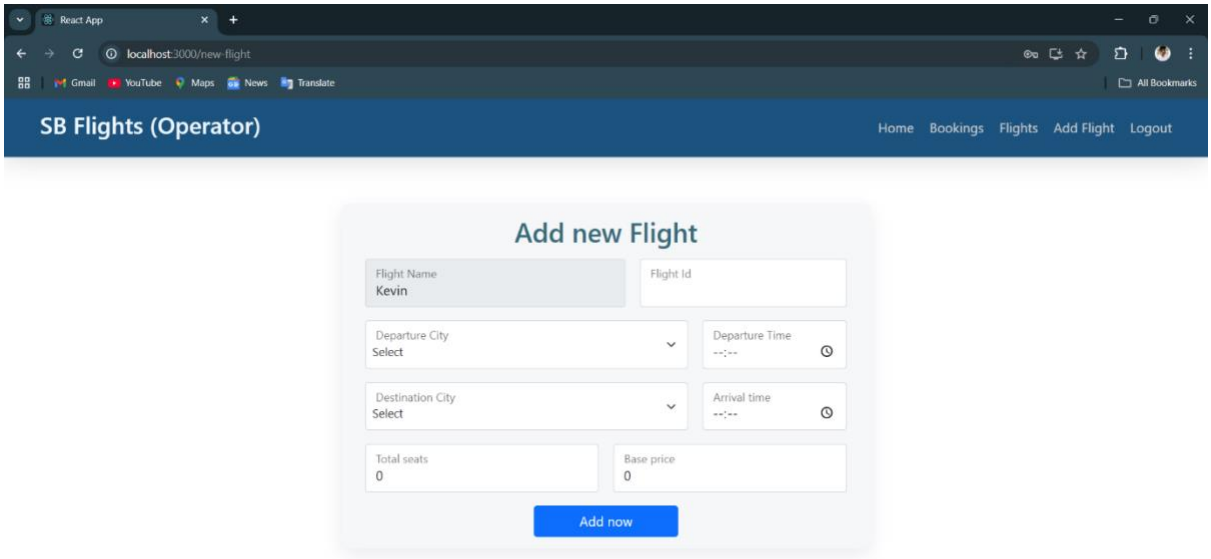
All Users



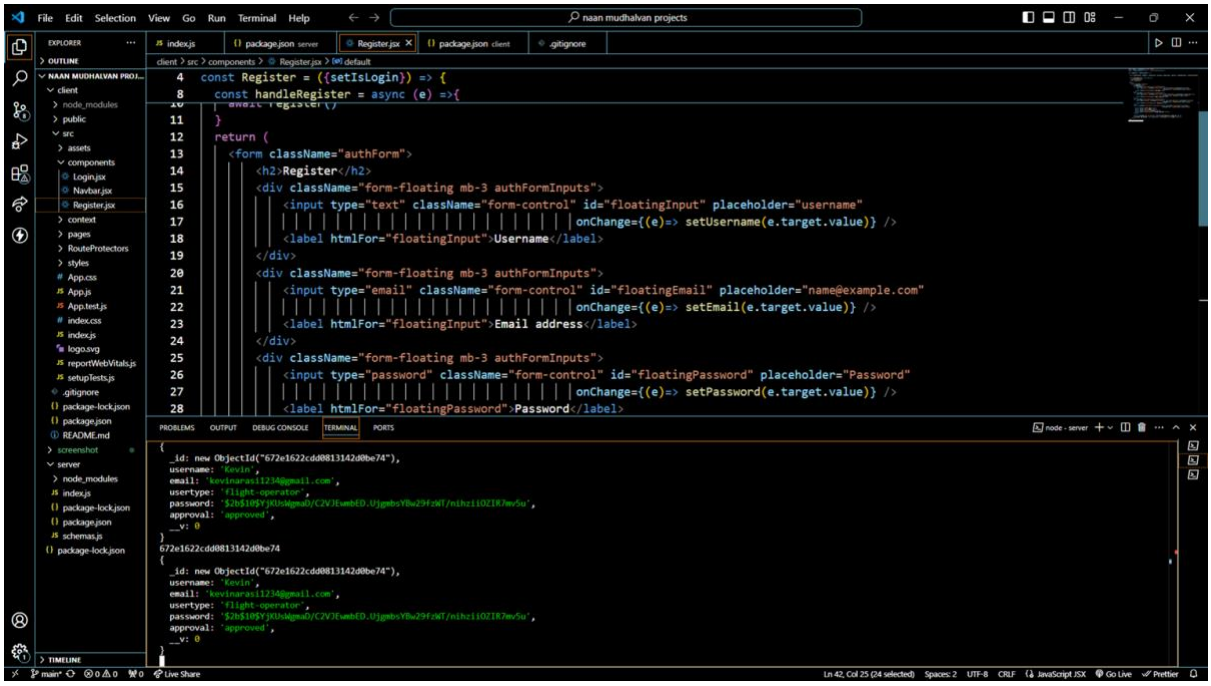
Flight operator



New Flights



Code



.

12. Known Issues

- **Large Transactions**: MongoDB may struggle with large or multi-step transactions. Solution: Use sharding and partition data properly.
- **Backup and Recovery**: Large data sets can slow backup processes. Solution: Use managed services like MongoDB Atlas for optimized backups.

13. Future Enhancements

- Add support for filtering flights based on additional criteria like layovers and baggage options.
- Implement seat selection functionality during booking.
- Enhance user experience with real-time flight status updates.