Full Stack Development with MERN

Topic: Flight Ticket Booking Application

1. Introduction

- **Project Title**: Flight Ticket Booking Application.
- **Team Members**

Sangeetha R.K	: Team Lead and Coordinator
Roshni N	: Frontend Developer
Smruthilaya Hariharan	: Backend Developer
Levin R Macedo	: Database Manager & API Intergrator

2. Project Overview

Purpose: The Flight System Client project is a React.js-based frontend for managing flight booking and related tasks. Its goal is to create an intuitive and user-friendly interface for interacting with a flight management system.

Key Features:

- Flight Search and Booking: Users can search for available flights and book them.
- User Authentication: Includes login and signup functionalities.
- Dynamic UI: Responsive and interactive components enhance user experience.
- API Integration: Communicates with a backend server for real-time updates.

3. Architecture:

Frontend (React.js):

- Component-based Design: Uses reusable React components to build a modular and scalable UI.

- State Management: Likely uses React's Context API or third-party libraries like Redux for managing state.
- Routing: React Router is used for navigation between different pages or views.
- API Integration: Connects to the backend via REST APIs to fetch and update data dynamically.

Backend (Node.js and Express.js):

- REST API: Implements a robust backend to handle user requests and business logic.
- Middleware: Utilizes middleware for authentication, request parsing, and error handling.
- Data Validation: Ensures secure and validated interactions between the frontend and database.

Database (MongoDB):

- Schema Design: The database includes schemas for users, flights, and bookings.
- Relationships: Likely implements references or embedded documents for linking user data to bookings or flights.
- CRUD Operations: Provides seamless integration for creating, reading, updating, and deleting records via Mongoose.

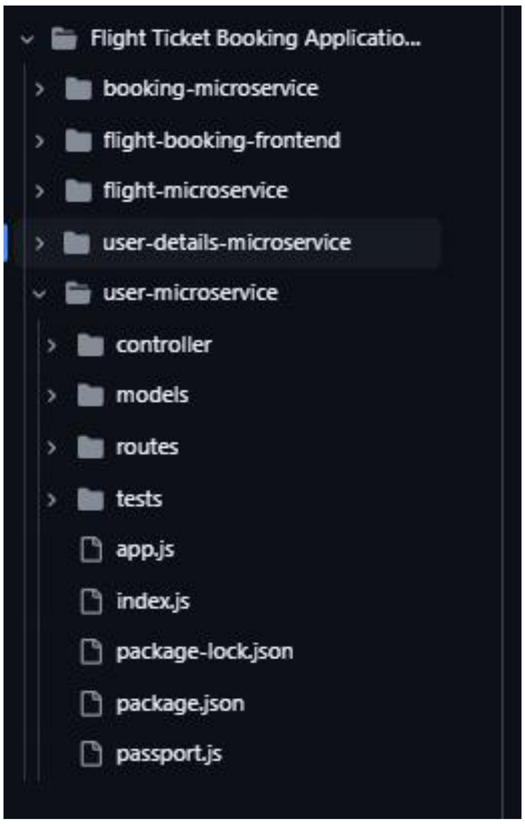
4. Setup Instructions

- **Prerequisites:**
    - Install Node.js and npm
    - Install Mongo DB or get access to a Mongo DB cluster
    - Install Postman to test backend code and have VS code to edit and run the project

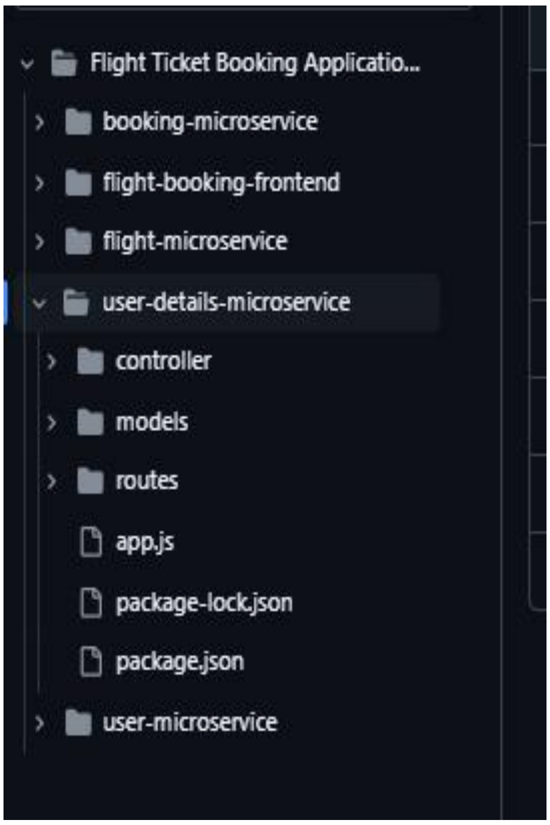
- **Installation:**

- **Clone the Repository**
  - Open a terminal or command prompt and run the below commands:
    - git clone https://github.com/Sangeetha-44/NM-MERN-Project.git
    - cd NM-MERN-Project
- **Install Dependencies**
  - Navigate to the **server** directory to install backend dependencies:
    - cd server
    - npm install
  - Navigate to the **client** directory to install frontend dependencies:
    - cd ../client
    - npm install
- **Set Up Environment Variables**
  - Create an .env file in the **server** directory and add the required environment variables such as:
    - MONGO_URI for your MongoDB connection string.
    - PORT for the backend server .
    - Add all other variables mentioned in the project documentation
- **4. Start the Application**
  - Start the backend server:
    - cd server
    - npm start
  - Start the frontend server:
    - cd ../client
    - npm start

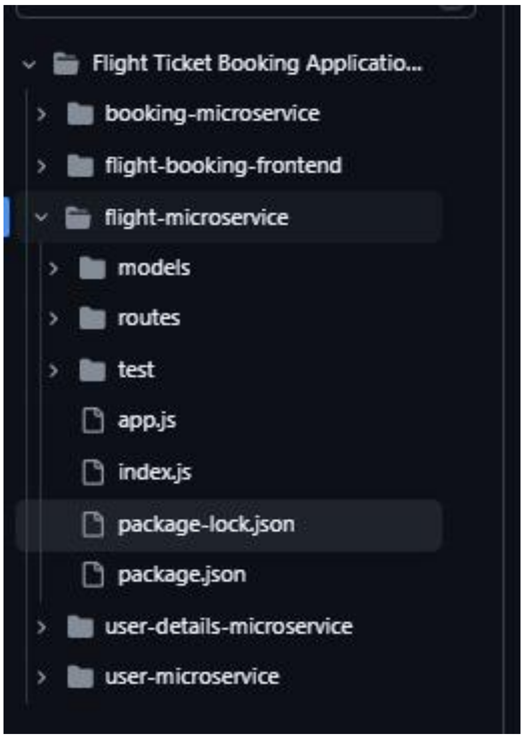
5. Folder Structure



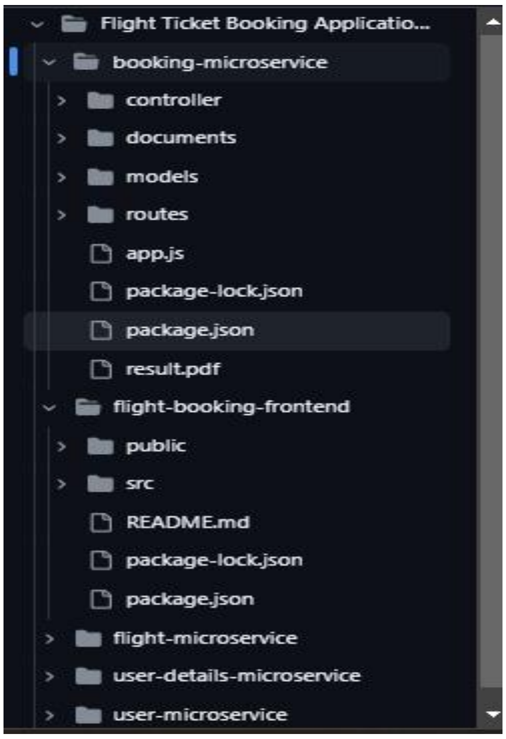
```
Flight Ticket Booking Applicatio...
  booking-microservice
  flight-booking-frontend
  flight-microservice
  user-details-microservice
  user-microservice
    controller
    models
    routes
    tests
    app.js
    index.js
    package-lock.json
    package.json
    passport.js
```



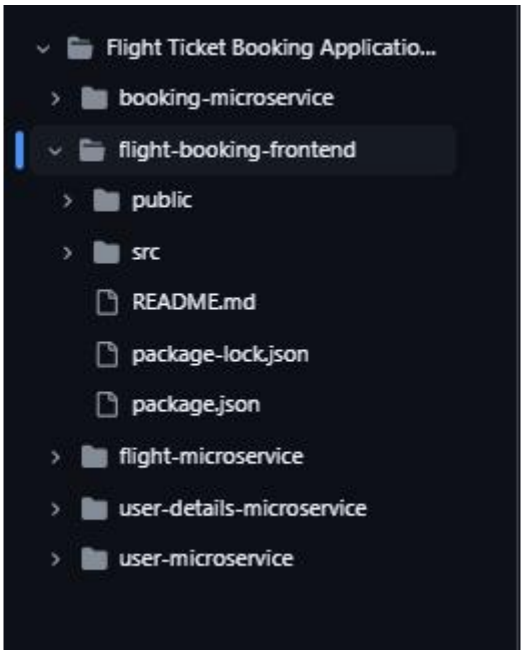
```
Flight Ticket Booking Applicatio...
  booking-microservice
  flight-booking-frontend
  flight-microservice
  user-details-microservice
    controller
    models
    routes
    app.js
    package-lock.json
    package.json
  user-microservice
```



```
Flight Ticket Booking Applicatio...
  booking-microservice
  flight-booking-frontend
  flight-microservice
    models
    routes
    test
    app.js
    index.js
    package-lock.json
    package.json
  user-details-microservice
  user-microservice
```



```
Flight Ticket Booking Applicatio...
  booking-microservice
    controller
    documents
    models
    routes
    app.js
    package-lock.json
    package.json
    result.pdf
  flight-booking-frontend
    public
    src
    README.md
    package-lock.json
    package.json
  flight-microservice
  user-details-microservice
  user-microservice
```



```
Flight Ticket Booking Applicatio...
  booking-microservice
  flight-booking-frontend
    public
    src
    README.md
    package-lock.json
    package.json
  flight-microservice
  user-details-microservice
  user-microservice
```

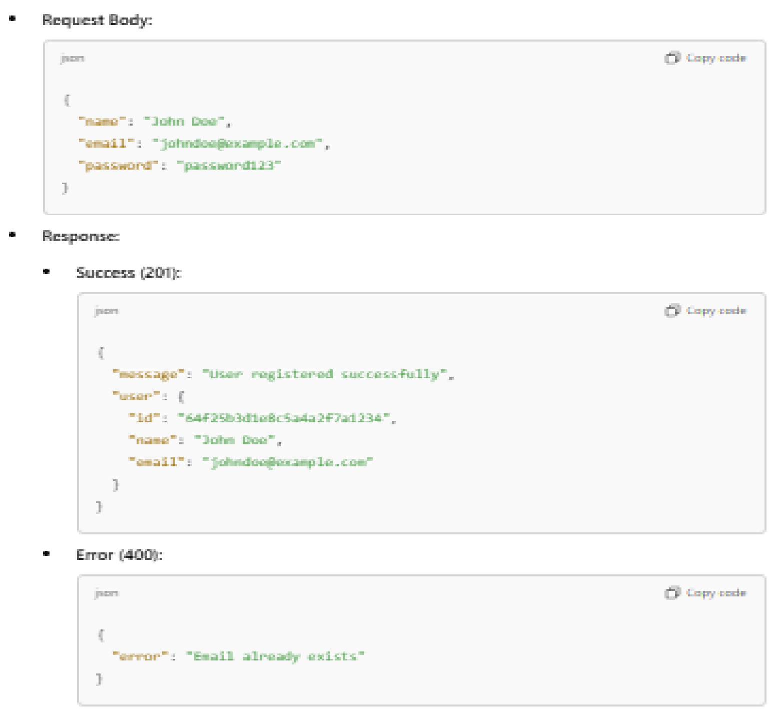
6. Running the Application

The commands to start the frontend and backend servers locally.

- **Frontend:**
  - cd client
  - npm install
  - npm start
- **Backend:**
  - cd server
  - mongod
  - npm install
  - npm start

7. API Documentation

- The endpoints are as follows
  - Registering a new user



- Request Body:

json

```json
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "password": "password123"
}
```

- Response:
  - Success (201):

json

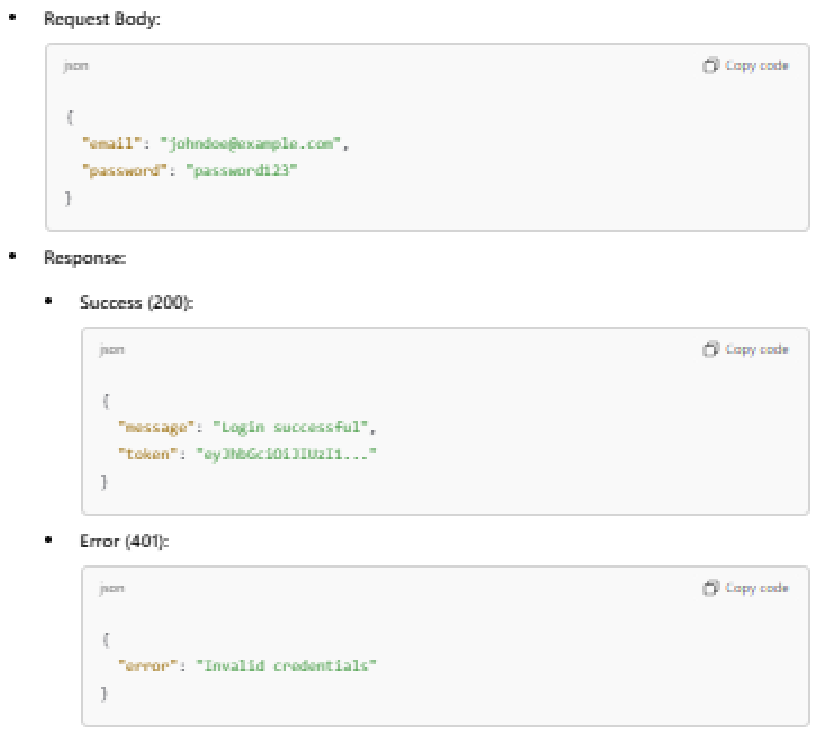
```json
{
  "message": "User registered successfully",
  "user": {
    "id": "64f25b3d1e8c5a4a2f7a1234",
    "name": "John Doe",
    "email": "johndoe@example.com"
  }
}
```

  - Error (400):

json

```json
{
  "error": "Email already exists"
}
```

- User login



- Request Body:

json

```json
{
  "email": "johndoe@example.com",
  "password": "password123"
}
```

- Response:
  - Success (200):

json

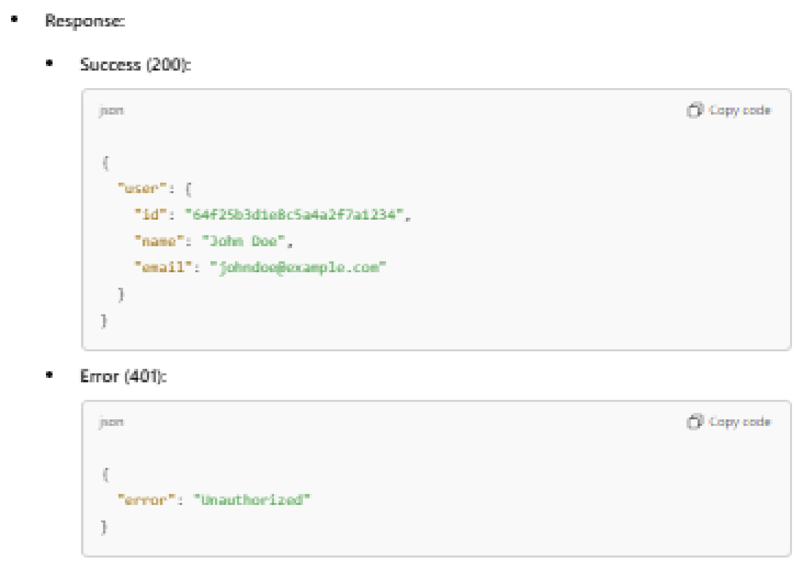
```json
{
  "message": "Login successful",
  "token": "eyJhbGciOiJIUzI1..."
}
```

  - Error (401):

json

```json
{
  "error": "Invalid credentials"
}
```

Get User Profile



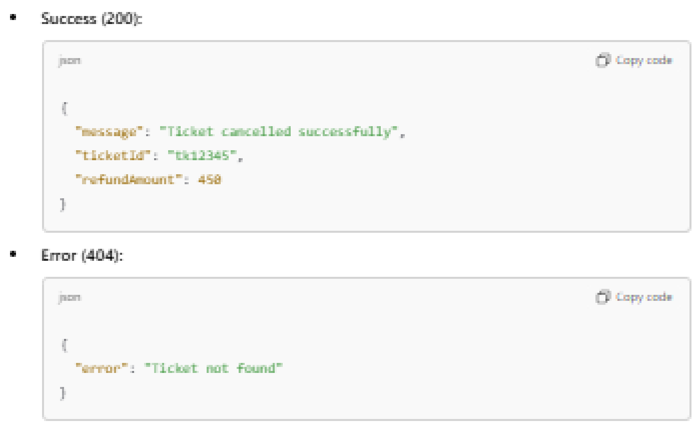
- Response:
  - Success (200):

```json
{
  "user": {
    "id": "64f25b3d1e8c5a4a2f7a1234",
    "name": "John Doe",
    "email": "johndoe@example.com"
  }
}
```

  - Error (401):

```json
{
  "error": "Unauthorized"
}
```

Booking Cancellation



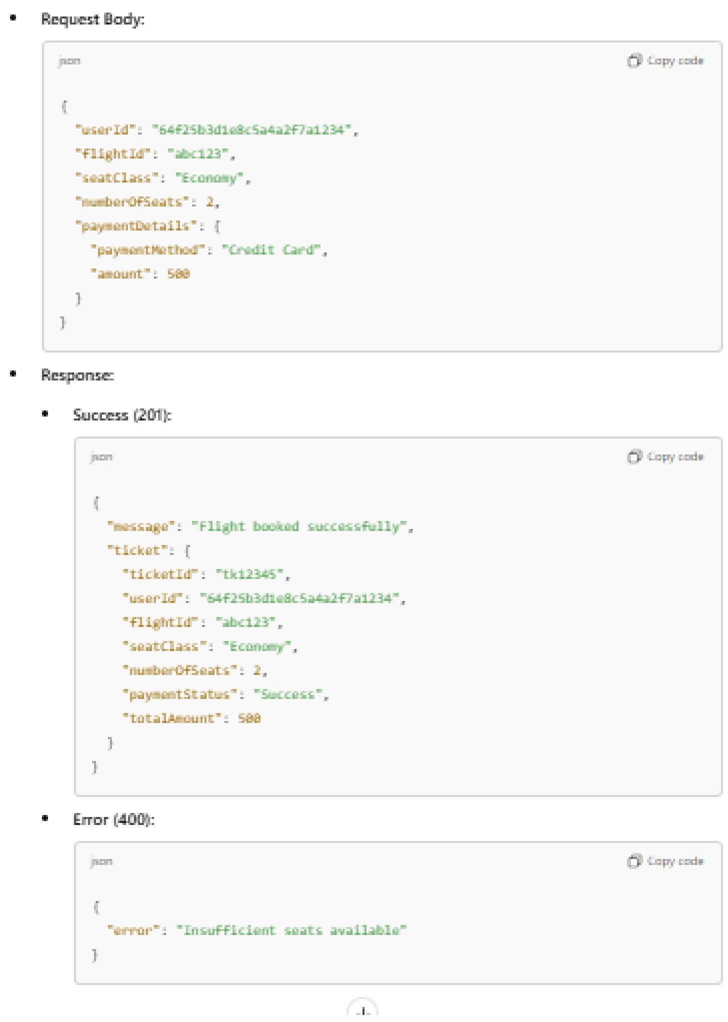
- Success (200):

```json
{
  "message": "Ticket cancelled successfully",
  "ticketId": "tk12345",
  "refundAmount": 450
}
```

- Error (404):

```json
{
  "error": "Ticket not found"
}
```

- Booking a ticket



- Request Body:

```json
{
  "userId": "64f25b3d1e8c5a4a2f7a1234",
  "flightId": "abc123",
  "seatClass": "Economy",
  "numberOfSeats": 2,
  "paymentDetails": {
    "paymentMethod": "Credit Card",
    "amount": 500
  }
}
```

- Response:
  - Success (201):

```json
{
  "message": "Flight booked successfully",
  "ticket": {
    "ticketId": "tk12345",
    "userId": "64f25b3d1e8c5a4a2f7a1234",
    "flightId": "abc123",
    "seatClass": "Economy",
    "numberOfSeats": 2,
    "paymentStatus": "Success",
    "totalAmount": 500
  }
}
```

  - Error (400):

```json
{
  "error": "Insufficient seats available"
}
```

8. Authentication

- **User Credentials Verification**:
  - The backend API verifies user credentials (email and password) sent via a POST /users/login request.
  - Passwords are compared using a hashing library to ensure security.
- **Token Generation**:
  - Upon successful login, the server generates a **JSON Web Token (JWT)** using a secret key.
  - The token contains encoded information, such as the user ID and expiration time
- **Authorization: Middleware for Token Validation**:
  - A middleware function (e.g., auth.js) intercepts protected routes.
  - It checks for a valid JWT in the Authorization header or cookies:
  - The token is decoded using a library like **jsonwebtoken**.

9. User Interface

- Provide screenshots or GIFs showcasing different UI features.

10. Testing

- Tested the UI by hosting locally. React Test Library assisted for DOM instructions.
- Used Postman to test the endpoints. Ensured the backend server is running, and the necessary authentication tokens are included in the request headers.
- Ensured data validation rules are enforced via Mongoose schemas.

11. Screenshots or Demo

The below link is a brief demo of the project.

https://drive.google.com/file/d/1AvU0CjfDN5YhbAeDkghUK6RH9G7c8gBp/view?usp=sharing

12. Known Issues

- **Performance Bottlenecks**
  - **Problem:** Fetching large datasets (e.g., stock data) slows down the application.
  - **Impact:** Delays in page loads and API responses.
  - **Potential Solution:** Implement pagination or infinite scrolling for large data sets.
- **MongoDB Connection Issues**
  - **Problem:** The backend occasionally fails to reconnect to MongoDB after a network interruption.
  - **Impact:** Downtime for the backend server.
  - **Temporary Fix:** Use a retry mechanism in the MongoDB connection configuration.

13. Future Enhancements

- **Advanced User Authentication**
  - Implement **OAuth2.0** for social media logins (Google, Facebook).
  - Add **2FA (Two-Factor Authentication)** for enhanced security.
- **Enhanced UI/UX**

    - Improve the mobile responsiveness of the application.
    - Add themes (dark mode, light mode).
    - Use charts and graphs (e.g., with **Chart.js** or **D3.js**) for visualizing stock trends.
- **Analytics and Reporting**
    - Implement user activity tracking to gather insights on popular features.
    - Allow users to generate custom stock reports.
- **Cloud Deployment Enhancements**
    - Move the project to a cloud provider like **AWS**, **Azure**, or **Google Cloud**.
    - Set up **CI/CD pipelines** for automated deployment and scaling.
- **AI/ML Features**
    - Predict stock prices using **machine learning models**.
    - Offer personalized stock recommendations based on user activity.