

PROJECT REPORT: FLIGHT FINDER PLATFORM

Table of Contents

1. Introduction
 - 1.1 Project Overview
 - 1.2 Purpose
 - 1.3 Goals and Objectives
2. Project Description
 - 2.1 Features
 - 2.2 Architecture
 - 2.2.1 Frontend
 - 2.2.2 Backend
 - 2.2.3 Database
 - 2.3 System Design
3. Setup Instructions
 - 3.1 Prerequisites
 - 3.2 Installation
4. Folder Structure
 - 4.1 Client (React Frontend)
 - 4.2 Server (Node.js Backend)
5. Running the Application
 - 5.1 Frontend
 - 5.2 Backend
6. API Documentation
 - 6.1 Endpoints
 - 6.2 Data Models
7. Authentication
 - 7.1 Methods
 - 7.2 Security Considerations
8. User Interface
 - 8.1 UI Components
 - 8.2 UX Design
9. Testing
 - 9.1 Testing Strategy
 - 9.2 Test Cases
10. Known Issues
11. Future Enhancements

1. Introduction

1.1 Project Overview

The Flight Booking System is a web application designed to streamline the process of booking flights. It provides users with an interface to search for available flights, select their preferred options, and complete the booking process. This system aims to simplify travel arrangements and provide a convenient platform for users to manage their flight reservations.

1.2 Purpose

The purpose of this system is to offer a user-friendly and efficient platform for managing flight bookings. It aims to simplify the booking process for customers and provide a comprehensive set of features for managing reservations. The system targets individual travelers, travel agencies, and potentially airlines by providing a centralized and easily accessible tool.

1.3 Goals and Objectives

- **Goal:** To develop a robust and scalable flight booking system.
- **Objectives:**
 - Provide an intuitive user interface for flight searches.
 - Enable users to view and manage their booking details.
 - Implement a secure authentication system.
 - Ensure the system is reliable and performs efficiently.
 - Facilitate easy integration with potential third-party services (e.g., payment gateways).

2. Project Description

The Flight Booking System offers a range of features designed to enhance the user experience and streamline the booking process.

2.1 Features

Key features of the system include:

- **Flight Search:** Users can search for flights based on various criteria, such as departure city, arrival city, and date. The search functionality will include filters for time, airline, price range, and number of layovers.
- **Flight Selection:** The system displays a list of available flights that match the search criteria, allowing users to compare options and select their preferred flight. Detailed flight information, including duration, layover details, and aircraft

type, will be shown.

- **Booking Management:** Users can manage their bookings, including viewing booking details, modifying bookings, and canceling reservations. Users will receive booking confirmations and have the ability to print or download their tickets.
- **User Authentication:** The system includes user authentication to secure user data and manage access to booking information. Different user roles (e.g., admin, customer) may have different permissions.

2.2 Architecture

The system follows a modular architecture, separating the frontend, backend, and database components.

2.2.1 Frontend

- The frontend is built using React.js.
- It provides a user-friendly interface for interacting with the system.
- It communicates with the backend API to retrieve flight data and manage bookings.
- The frontend uses components for reusability, state management (e.g., useState, Redux, or Context API), and routing (React Router).

2.2.2 Backend

- The backend is developed using Node.js with Express.js.
- It provides API endpoints for managing flight data, user authentication, and booking information.
- It handles requests from the frontend and interacts with the database.
- The backend implements RESTful API principles, middleware for request handling, and potentially utilizes an ORM or ODM (like Mongoose) for database interactions.

2.2.3 Database

- The system uses MongoDB to store data.
- The database stores information about flights, users, and bookings.
- The database schema includes collections for flights, users, bookings, and potentially other related data like airlines or airports.

2.3 System Design

The system is designed with scalability and maintainability in mind. Key design considerations include:

- **Scalability:** The system should be able to handle a growing number of users and

flight bookings. Load balancing and database sharding might be considered.

- **Maintainability:** The codebase should be well-organized, with clear naming conventions and comments.
- **Security:** Security is a top priority. The system will protect against vulnerabilities like SQL injection and cross-site scripting.
- **Performance:** The system should respond quickly to user requests. Caching and database optimization techniques will be employed.
- **Testability:** The system is designed to be easily testable, with unit, integration, and end-to-end tests.

TRAVEL SUPPORT

PLAN YOUR TRAVEL WITH CONFIDENCE

FIND HELP WITH YOUR BOOKINGS AND TRAVEL PLANS, AND SEE WHAT TO EXPECT ALONG YOUR JOURNEY.

01

TRAVEL REQUIREMENTS FOR DUBAI

STAY INFORMED AND PREPARED FOR YOUR TRIP TO DUBAI WITH ESSENTIAL TRAVEL REQUIREMENTS, ENSURING A SMOOTH AND HASSLE-FREE EXPERIENCE IN THIS VIBRANT AND CAPTIVATING CITY.

02

MULTI-RISK TRAVEL INSURANCE

COMPREHENSIVE PROTECTION FOR YOUR PEACE OF MIND, COVERING A RANGE OF POTENTIAL TRAVEL RISKS AND UNEXPECTED SITUATIONS.

03

TRAVEL REQUIREMENTS BY DESTINATIONS

STAY INFORMED AND PLAN YOUR TRIP WITH EASE, AS WE PROVIDE UP-TO-DATE INFORMATION ON TRAVEL REQUIREMENTS SPECIFIC TO YOUR DESIRED DESTINATIONS.



3. Setup Instructions

3.1 Prerequisites

Before setting up the project, ensure you have the following software dependencies installed:

1. Node.js: Download and install from nodejs.org.
2. npm: Usually comes with Node.js. Verify with `npm -v`.
3. MongoDB: Download and install from mongodb.com. Ensure the MongoDB server is running.
4. Git: Download and install from git-scm.com.
5. Code Editor: A code editor like VSCode, Sublime Text, or Atom.

3.2 Installation

Follow these steps to set up the Flight Booking System on your local machine:

1. **Clone the Repository:**

```
git clone https://github.com/pragathish07/Flight-booking-system.git
cd Flight-booking-system
```

2. **Install Dependencies:**

```
cd server
npm install
cd ../client
npm install
```

3. **Set Up Environment Variables:**

- Create a .env file in both the server and client directories.
- Add the following variables to the .env file in the server directory:
DB_URI=mongodb://localhost:27017/flight-booking
JWT_SECRET=your_jwt_secret_key
PORT=5000
- Add any necessary frontend environment variables in the client .env file. For example: REACT_APP_API_BASE_URL=http://localhost:5000

4. **Start MongoDB Server:**

- Ensure MongoDB is installed and running.
- Start the MongoDB server. This might involve running mongod in your terminal.

5. **Run Backend and Frontend Servers:**

```
cd server
npm start # Start the backend server
cd ../client
npm start # Start the frontend server
```

4. Folder Structure

The project is organized into separate directories for the frontend and backend components.

4.1 Client (React Frontend)

```
client/
├── node_modules/    # Node.js dependencies
├── public/          # Static assets (e.g., index.html, images)
└── src/             # Source code
```

```
| |— components/  # Reusable UI components
| |— pages/      # Main application pages/views
| |— App.js      # Main application component
| |— index.js    # Entry point for the React application
| |— styles/     # CSS stylesheets or other styling files
|— .env          # Environment variables
|— package.json  # Project configuration and dependencies
|— README.md     # Project documentation
```

4.2 Server (Node.js Backend)

server/

```
|— node_modules/  # Node.js dependencies
|— models/        # Data models (schemas for the database)
| |— Flight.js    # Schema for flight data
| |— User.js      # Schema for user data
| |— Booking.js   # Schema for booking data
|— routes/        # API routes
| |— flightRoutes.js # Routes for flight-related operations
| |— userRoutes.js  # Routes for user-related operations
| |— bookingRoutes.js # Routes for booking-related operations
|— config/        # Configuration files
| |— database.js   # Database connection configuration
|— middleware/    # Middleware functions
| |— authMiddleware.js # Middleware for authentication
|— .env           # Environment variables
|— server.js      # Entry point for the Node.js server
|— package.json   # Project configuration and dependencies
|— README.md      # Project documentation
```

5. Running the Application

To start the Flight Booking System locally, follow these steps:

5.1 Frontend

1. Open a terminal and navigate to the client directory:
cd client

2. Start the React development server:
npm start

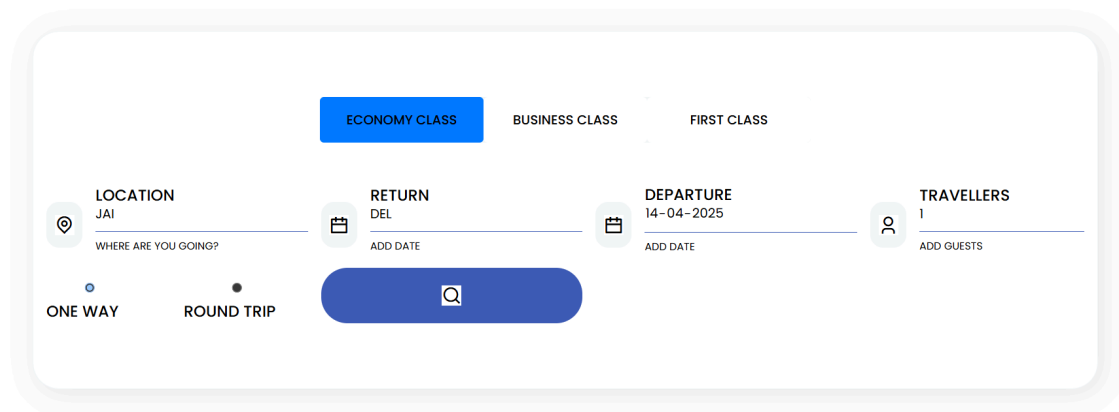
This will typically open the application in your default web browser at <http://localhost:3000>.

FLIGHTS.IO

[REGISTER](#) [LOGIN](#) [HOME](#)

FIND AND BOOK A GREAT EXPERIENCE





The image shows a flight booking form with the following fields and options:

- CLASS:** Three buttons labeled "ECONOMY CLASS" (highlighted in blue), "BUSINESS CLASS", and "FIRST CLASS".
- LOCATION:** A text input field containing "JAI" with a location pin icon and the placeholder text "WHERE ARE YOU GOING?". Below it are two radio buttons: "ONE WAY" (selected) and "ROUND TRIP".
- RETURN:** A text input field containing "DEL" with a calendar icon and the placeholder text "ADD DATE".
- DEPARTURE:** A text input field containing "14-04-2025" with a calendar icon and the placeholder text "ADD DATE".
- TRAVELLERS:** A text input field containing "1" with a person icon and the placeholder text "ADD GUESTS".
- Search:** A large blue button with a magnifying glass icon.

5.2 Backend

1. Open a separate terminal and navigate to the server directory:
`cd server`
2. Start the Node.js backend server:
`npm start`

The server will start listening for requests, typically on port 5000 (or as defined in your .env file). You'll usually see a message in the terminal like "Server running on port 5000".

6. API Documentation

6.1 Endpoints

Endpoint	Method	Description
----------	--------	-------------

/api/flights	GET	Retrieves all flights. the data you provided does not have info about the api documentation. using the given structure, infer the api documentation.
--------------	-----	--

Okay, here's an inferred API documentation based on the code structure you've provided:

6. API Documentation

6.1 Endpoints

This section describes the API endpoints for the Flight Booking System. The API follows RESTful principles where possible.

| Endpoint | Method | Description | Parameters (Request Body) | Example Response

6.2 Data Models

This section describes the structure of the data stored in the database, defining the schema for each entity.

Flight Model

Represents a flight entity.

Field	Type	Description
flightNumber	String	Unique identifier for the flight.
departureCity	String	The city from which the flight departs.
arrivalCity	String	The city where the flight arrives.
departureDate	Date	The date and time of departure.
arrivalDate	Date	The date and time of arrival.
airline	String	The airline operating the flight.
price	Number	The price of the flight.
seatsAvailable	Number	The number of seats currently available on the flight.

User Model

Represents a user entity.

Field	Type	Description
-------	------	-------------

name	String	The name of the user.
email	String	The email address of the user (used for login).
password	String	The password of the user (stored as a hash).
role	String	The role of the user (e.g., "customer", "admin").

Booking Model

Represents a flight booking.

Field	Type	Description
user	ObjectId	Reference to the User model.
flight	ObjectId	Reference to the Flight model.
bookingDate	Date	The date and time when the booking was made.
numberOfPassengers	Number	The number of passengers included in the booking.
totalPrice	Number	The total price of the booking.
status	String	The status of the booking (e.g., "pending", "confirmed", "cancelled").