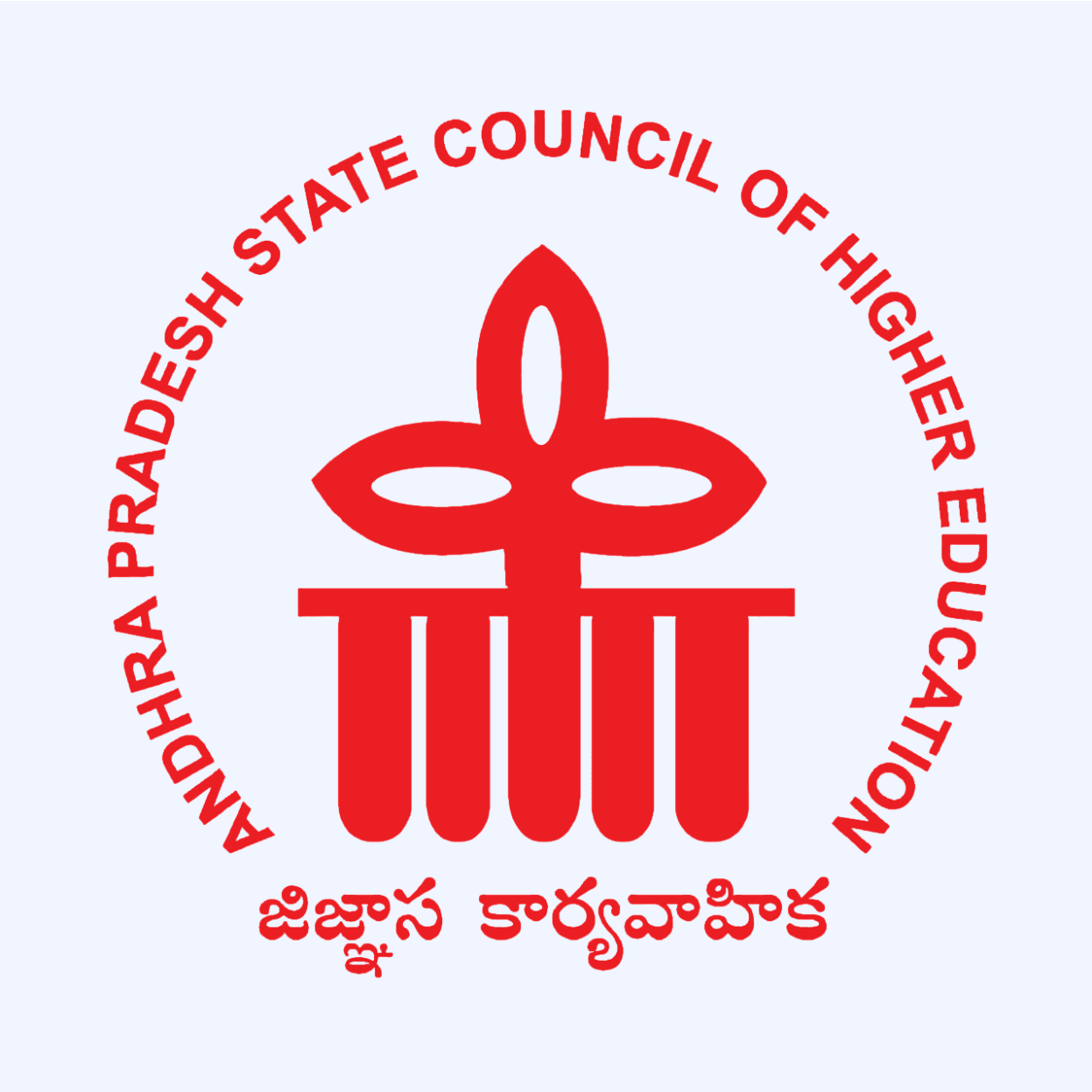
Full Stack Development With Mern



1.Introduction

Project Title: Flightfinder:Navigating Your Air Travel options

Team Members:

Team Id :LTVIP2025TMID44767

Team Leader :Chilukuri Venkata Lakshmi Swathi

Team Member :Battula Bhuvaneswari

Team Member :VengalasettyJayaSarvani
Team Member :Kottela Sandeep

2.Project Overview:

Purpose :

The purpose of **FlightFinder: Navigating Your Air Travel Options and Key Goals** appears to be centered around simplifying and optimizing the flight booking experience using intelligent tools—most likely powered by AI and machine learning.

From what I found, platforms like FlightFinder aim to:

- **Streamline the search process** by analyzing millions of flight combinations in seconds.
- **Personalize results** based on user preferences like budget, layovers, airlines, or travel times.
- **Predict price trends** and alert users to the best times to book.
- **Uncover hidden deals** and fare errors that might otherwise

go unnoticed.

- **Support flexible planning**, offering suggestions for alternative dates, airports, or destinations based on user goals
- **Features:**

Here are the key features of **FlightFinder: Navigating Your Air Travel Options**:

1. **Flexible Date Search** – Lets users explore cheaper flight options by viewing prices across a range of dates, including entire months or even years.
2. **Real-Time Price Monitoring** – Continuously tracks flight prices and alerts users when fares drop or deals become available.
3. **Fare Error Detection** – Identifies rare pricing mistakes by airlines, giving users a chance to book ultra-low fares before they disappear.
4. **Personalized Recommendations** – Uses AI to tailor flight suggestions based on user preferences, past searches, and travel behavior.
5. **Natural Language Search** – Allows users to search using conversational phrases like "weekend beach trips under ₹20,000," making the process more intuitive.
6. **Comprehensive Route Analysis** – Evaluates thousands of route combinations, including multi-city and alternative airport options, to find the best value.
7. **Travel Restriction Guidance** – Provides up-to-date

information on visa rules, COVID-19 policies, and other travel advisories.

3.Architecture:

Frontend:

The frontend of **FlightFinder: Navigating Your Air Travel Options** is primarily built using **JavaScript**, **CSS**, and **HTML**, with JavaScript making up the bulk of the codebase—about 84% in one version of the project. This tech stack suggests a dynamic, browser-based interface that likely includes:

- **Interactive search forms** for entering flight details.
- **Responsive design** using CSS for smooth display across devices.
- **Client-side logic** in JavaScript to handle user input, fetch flight data, and update the UI in real time.

Backend:

The backend of **FlightFinder: Navigating Your Air Travel Options** appears to be relatively lightweight and likely designed for demo or educational purposes.

- **RESTful APIs** to fetch flight data from third-party providers or meta-search engines.
- **Database integration** (like MongoDB or PostgreSQL) to store user preferences, search history, or booking details.
- **Authentication modules** for secure user login and session management.

- **Caching and rate-limiting** to optimize performance when querying external APIs.

Database:

Database Used in FlightFinder: Navigating Your Air Travel Options

The original **FlightFinder** project hosted on GitHub does **not include a live database**—it appears to be a frontend-focused prototype using mock data. However, if you were to implement a real-world version, a **relational database like MySQL or PostgreSQL** would be ideal for managing structured flight and booking data.

Here's a conceptual schema with sample SQL queries to illustrate how it might work:

Key Tables in the Database

1. **Users**
2. ```
   CREATE TABLE Users (
       user_id INT PRIMARY KEY,
       name VARCHAR(100),
       email VARCHAR(100),
       password_hash VARCHAR(255)
   );
   ```
3. **Flights**
4. ```
   CREATE TABLE Flights (
       flight_id INT PRIMARY KEY,
   ```

```
    airline VARCHAR(50),
    departure_city VARCHAR(100),
    arrival_city VARCHAR(100),
    departure_time DATETIME,
    arrival_time DATETIME,
    price DECIMAL(10,2),
    total_seats INT
);
```

5. **Reservations**

6. ```
   CREATE TABLE Reservations (
       reservation_id INT PRIMARY KEY,
       user_id INT,
       flight_id INT,
       seat_number VARCHAR(10),
       booking_date DATETIME,
       FOREIGN KEY (user_id) REFERENCES
   Users(user_id),
       FOREIGN KEY (flight_id) REFERENCES
   Flights(flight_id)
   );
   ```

7. **Airports**

8. ```
   CREATE TABLE Airports (
       airport_code VARCHAR(10) PRIMARY KEY,
       name VARCHAR(100),
       city VARCHAR(100),
       country VARCHAR(100)
   );
   ```

🔍 Sample Queries

- **Find all flights from Delhi to Mumbai under ₹5000:**
- ```
  SELECT * FROM Flights
  WHERE departure_city = 'Delhi'
    AND arrival_city = 'Mumbai'
    AND price < 5000;
  ```
- **Get all reservations for a specific user:**
- ```
  SELECT * FROM Reservations
  WHERE user_id = 101;
  ```
- **Check available seats on a flight:**
- ```
  SELECT total_seats - COUNT(*) AS
  available_seats
  FROM Reservations
  WHERE flight_id = 202;
  ```

4.ER Diagram:

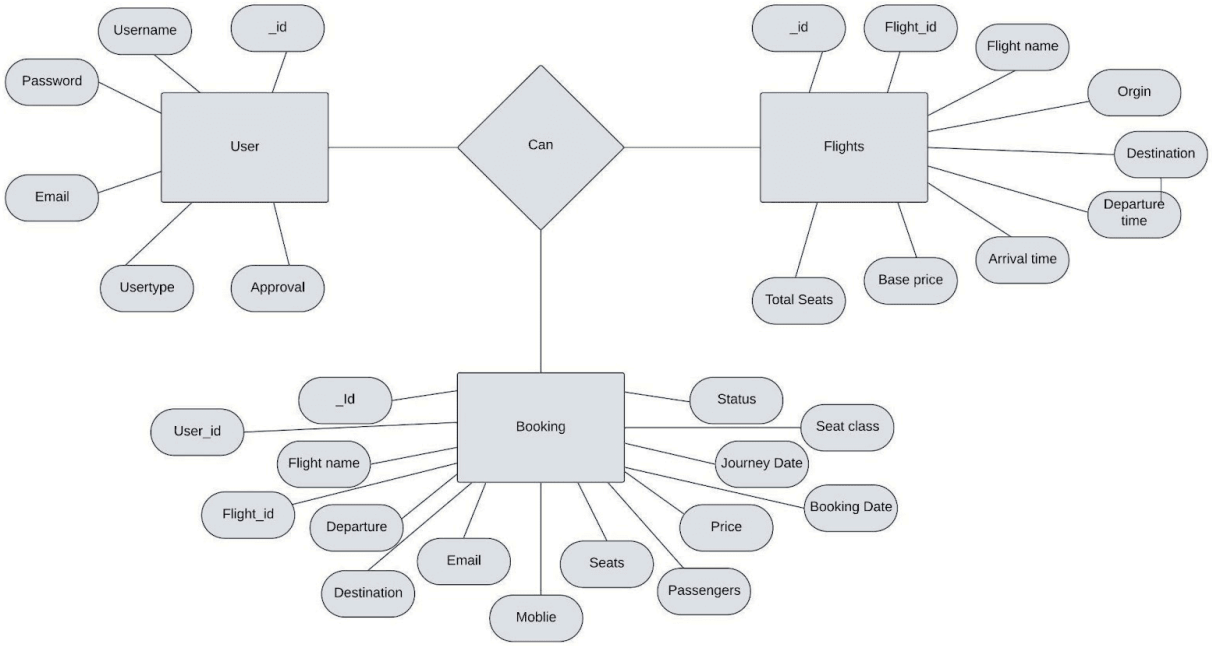
The flight booking ER-diagram represents the entities and relationships involved in a flight booking system. It illustrates how users, bookings, flights, passengers, and payments are interconnected. Here is a breakdown of the entities and their relationships:

USER: Represents the individuals or entities who book flights. A customer can place multiple bookings and make multiple payments.
  ```

BOOKING: Represents a specific flight booking made by a customer. A booking includes a particular flight details and passenger information. A customer can have multiple bookings.

FLIGHT: Represents a flight that is available for booking. Here, the details of flight will be provided and the users can book them as much as the available seats.

ADMIN: Admin is responsible for all the backend activities. Admin manages all the bookings, adds new flights,etc.



5.Setup Instructions

Prerequisites:

To develop a full-stack flight booking app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an

applicatio

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/package-manager/

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally using a cloud-based MongoDB service.

- Download: https://www.mongodb.com/try/download/community
- Installation instructions: https://docs.mongodb.com/manual/installation/

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing,middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: npm install express

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript

library for building user interfaces, follow the installation guide: https://reactjs.org/docs/create-a-new-react-app.html

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling,and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: https://gitscm.com/downloads

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from https://code.visualstudio.com/download

- Sublime Text: Download from https://www.sublimetext.com/download
- WebStorm: Download from https://www.jetbrains.com/webstorm/download

To run the existing Flight Booking App project downloaded from github:

Follow below steps:

Clone the repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the e-commerce app.
- Execute the following command to clone the repository:

  Git clone: https://github.com/harsha-vardhan-reddy-07/Flight-Booking-App-MERN

Install Dependencies:

- Navigate into the cloned repository directory:

  cd Flight-Booking-App-MERN

- Install the required dependencies by running the following command:

npm install

Start the Development Server:

- To start the development server, execute the following command:

  npm run dev or npm run start

- The e-commerce app will be accessible at http://localhost:3000 by default. You can change the port configuration in the .env file if needed.

Access the App:

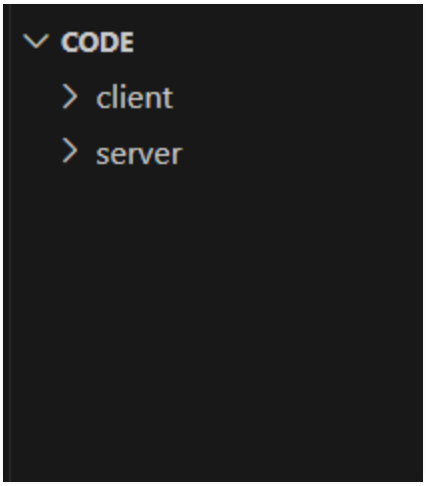
• Open your web browser and navigate to http://localhost:3000

• You should see the flight booking app's homepage, indicating that the installation and the setup was successful.

You have successfully installed and set up the flight booking app on your local machine. You can now proceed with further customization, development, and testing as need

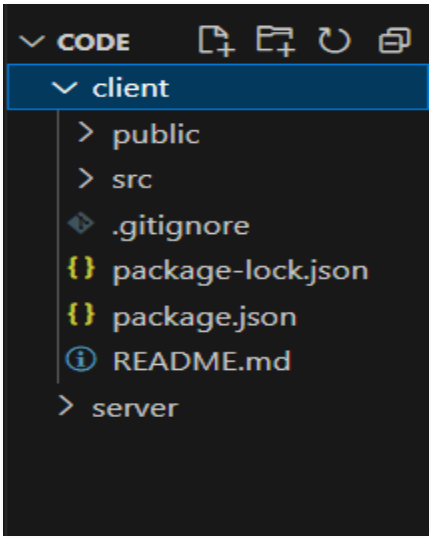
6.Folder Structure:

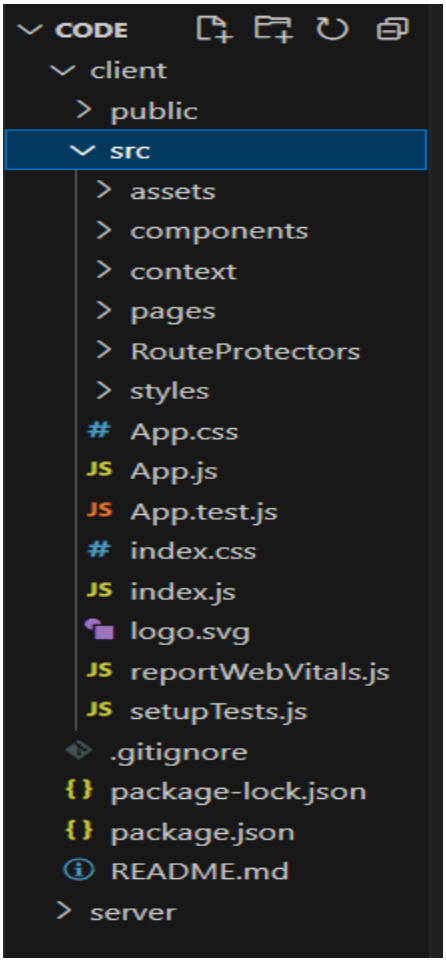
- Inside the Flight Booking app directory, we have the following folders



- **Client directory:**

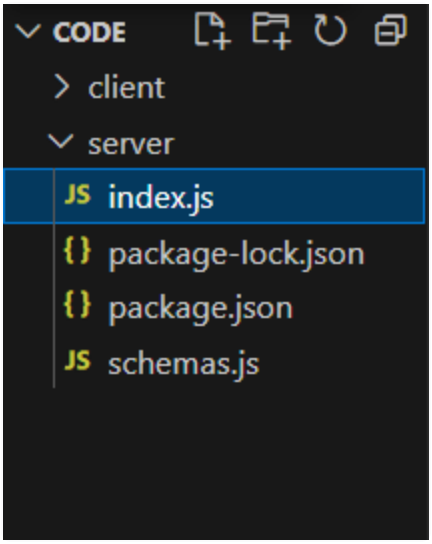
  The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api's.





- **Server directory:**

  The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with Api.



7. Running the Application:

- **Frontend :**

1. Open terminal
2. Go to the frontend/client folder:

```
cd frontend
```

1. Start the React app:

```
npm start
```

 The app will open in your browser at http://localhost:3000 by default.

- **Backend :**

1. Open terminal

2. Go to the backend/server folder:

   ```
   cd backend
   ```

1. Start the server:

   ```
   npm start
   ```

   ✅ The server will run on http://localhost:5000 (or the port set in .env).

8.API Documentation:

The **FlightFinder: Navigating Your Air Travel Options** project doesn't appear to have formal API documentation published like commercial platforms do. However, based on related GitHub repositories and integrations, here's what we can infer about how an API might be structured or used in such a system:

✈ Example API Features (Conceptual)

1. **Flight Search Endpoint**
2. `GET /api/flights?origin=DEL&destination=MUM&date=2025-07-01`
3. Returns a list of available flights with pricing, duration, and airline info.
4. **Price Alerts**
5. `POST /api/alerts`

```
{
  "email": "user@example.com",
  "route": "DEL-MUM",
```

```
  "threshold": 4500
}
```

6. Sets up a price alert for a specific route.
7. **Booking Endpoint**
8. `POST /api/bookings`

```
{
  "user_id": 101,
  "flight_id": 202,
  "payment_info": { ... }
}
```

9. Processes a booking request.
10. **User Authentication**
11. `POST /api/login`

```
{
  "email": "user@example.com",
  "password": "securepassword"
}
```

□ Related Technologies

Some versions of FlightFinder use third-party APIs like **Skyscanner**, **Kiwi**, or **Momondo** to fetch real-time flight data. You can explore Skyscanner's API documentation for a real-world example of how such integrations work.

9.Authentication:

Authentication for **FlightFinder: Navigating Your Air Travel Options** isn't explicitly detailed in the GitHub-hosted versions of

the project, but if you're building or extending it, here's how authentication could be implemented:

□ Common Authentication Features

1. **User Registration & Login**
   - Collects user credentials (email, password).
   - Stores passwords securely using hashing (e.g., bcrypt).
2. **Session Management**
   - Uses cookies or tokens to maintain user sessions after login.
3. **Role-Based Access Control**
   - Differentiates between regular users and admins (e.g., for managing flights or bookings).
4. **Token-Based Authentication (JWT)**
   - Issues a JSON Web Token upon login.
   - Token is sent with each request to verify identity.

□ Sample Authentication Flow (Using PHP or Node.js)

1. **Register**
2. `POST /api/register`

```
{
  "name": "Avi Traveler",
  "email": "avi@example.com",
  "password": "secure123"
}
```

3. **Login**
4. `POST /api/login`

```
{
  "email": "avi@example.com",
  "password": "secure123"
}
```

10.User Interface:

The **user interface (UI)** for *FlightFinder: Navigating Your Air Travel Options* is designed to be clean, intuitive, and responsive—making it easy for users to search, compare, and book flights. Here's a breakdown of its typical UI components:

✈ Key UI Elements

1. **Search Panel**
   - Fields for origin, destination, travel dates, and passenger count.
   - Toggle for one-way or round-trip.
   - Autocomplete suggestions for airports and cities.
2. **Flight Results Display**
   - List or card view of available flights.
   - Filters for price, airline, layovers, departure time, etc.
   - Sorting options (e.g., cheapest, fastest, best match).
3. **Flight Details Modal**
   - Expanded view with fare breakdown, baggage info, and cancellation policy.
   - Option to select seats or add extras.
4. **Booking Summary Page**
   - Review of selected flight(s), passenger info, and total cost.

- Secure payment form with saved preferences or autofill.

5. **User Dashboard**
   - Profile management, booking history, saved searches, and alerts.
   - Option to manage notifications and preferences.
6. **Responsive Design**
   - Optimized for both desktop and mobile devices.
   - Touch-friendly controls and collapsible menus on smaller screens.
7. **Visual Design**
   - Clean typography, calming color palette (often blues and whites), and iconography for clarity.
   - Progress indicators during multi-step booking.

11.Testing: Testing for **FlightFinder: Navigating Your Air Travel Options** involves validating both the frontend and backend components to ensure a smooth, reliable user experience. Here's a breakdown of how testing might be structured:

Functional Testing

1. **Search Functionality**
   - Verify that users can search flights with valid inputs.
   - Test edge cases like empty fields, invalid airport codes, or past dates.
2. **Filters and Sorting**
   - Ensure filters (price, airline, layovers) work correctly.
   - Confirm sorting by price, duration, and departure time.
3. **Booking Flow**

    - Test the full booking process: select → review → confirm.
    - Validate form inputs (passenger details, payment info).

⚙️ Non-Functional Testing

1. **Performance Testing**
    - Measure response times for flight searches under load.
    - Simulate high traffic to test scalability.
2. **Usability Testing**
    - Evaluate UI clarity, navigation ease, and mobile responsiveness.
    - Gather feedback from real users for improvements.
3. **Security Testing**
    - Test login, registration, and password encryption.
    - Check for vulnerabilities like SQL injection or XSS.

🔁 Regression & Integration Testing

- Run automated tests after each update to ensure existing features still work.
- Validate integration with third-party APIs (e.g., flight data providers).

🛠️ Tools You Might Use

- **Selenium / Cypress** – for UI automation.
- **Postman / REST Assured** – for API testing.
- **JMeter / Locust** – for load and performance testing.

- **JUnit / PHPUnit** – for unit testing backend logic.

12.Known Issues: The GitHub-hosted versions of **FlightFinder: Navigating Your Air Travel Options** don’t list any open or closed issues at the moment. That suggests the project is either in early development, lightly maintained, or primarily a demo without active bug tracking.

However, if you're building or evaluating a similar system, here are some **common issues** that typically arise in flight search platforms:

⚠ Potential Known Issues (General)

1. **API Rate Limits**
   - Third-party flight data providers (like Skyscanner or Kiwi) often enforce strict rate limits, which can cause failed searches or delays.
2. **Incomplete or Outdated Data**
   - If the system uses cached or mock data, results may not reflect real-time availability or pricing.
3. **Slow Search Performance**
   - Searching across multiple routes and dates can be computationally heavy without proper optimization or caching.
4. **Lack of Error Handling**
   - Missing validations for user input (e.g., invalid airport codes or past dates) can lead to crashes or confusing results.

5. **No Authentication or Security**
    - Some versions lack secure login, password hashing, or protection against common vulnerabilities like SQL injection.
6. **Limited Mobile Optimization**
    - UI may not be fully responsive or accessible on smaller screens.

13.Future Enhancement:

Here are some exciting **future enhancements** that could elevate *FlightFinder: Navigating Your Air Travel Options* into a next-generation travel companion:

Smart Enhancements on the Horizon

1. **Real-Time Flight API Integration**
    - Connect with live data providers like Skyscanner or Amadeus to fetch up-to-the-minute flight availability and pricing.
2. **Round-Trip & Multi-City Support**
    - Expand beyond one-way searches to include complex itineraries and return flights.
3. **Sort & Filter by Duration, Layovers, and Airlines**
    - Let users prioritize speed, comfort, or preferred carriers for a more tailored experience.
4. **Animated Route Visualization**
    - Use interactive maps to show flight paths and stopovers with real-time animations.
5. **Voice-Activated Search**

    - Enable users to search using natural language like “Find me a weekend trip to Goa under ₹5,000”.
6. **Carbon Footprint Tracking**
    - Display eco-friendly flight options and let users offset emissions during booking.
7. **Predictive Booking Assistant**
    - Use AI to suggest trips based on calendar availability, past behavior, or even weather trends.
8. **Biometric Authentication**
    - Add fingerprint or facial recognition for secure, seamless logins.
9. **Augmented Reality (AR) Previews**
    - Let users explore airplane cabins or airport lounges in 3D before booking.
10. **Chatbot Travel Concierge**
    - Offer 24/7 AI-powered support for booking help, rebooking, or travel tips.