FLIGHT FINDER APPLICATION – PLANNING LOGIC DOCUMENT

🧠 1. PROJECT TITLE

Flight Finder – Smart Search for Flights

🎯 2. OBJECTIVE

The primary objective is to develop a user-friendly web application that empowers individuals to efficiently search for flights. The application will enable users to specify their source, destination, desired travel dates, and passenger count. It aims to present real-time or near real-time flight data, offering robust filtering capabilities by price, duration, and airline to enhance the user's decision-making process for booking travel.

⚙️ 3. TECHNOLOGY STACK

Layer	Technology	Rationale
Frontend	React.js	Chosen for its component-based architecture, efficient UI rendering, and vast community support, enabling a dynamic and responsive user interface.
Backend	Node.js + Express.js	Selected for its non-blocking, event-driven architecture, making it ideal for handling concurrent requests from the frontend and external API calls. Express.js provides a robust framework for building RESTful APIs.
Database	MongoDB	A NoSQL database chosen for its flexibility in handling unstructured flight data (e.g., cached flight results, user preferences, booking details if implemented), scalability, and ease of integration with Node.js.
API Source	Flight APIs (Mock/ Skyscanner/ RapidAPI)	Initial development will use mock data for rapid prototyping. Integration with commercial APIs (e.g., Skyscanner, RapidAPI) will be pursued for real-time data upon project maturity, ensuring accurate flight information.

Layer	Technology	Rationale
Deployment	Vercel / Netlify (Frontend), Render / Railway (Backend)	These platforms offer simplified continuous deployment, cost-effectiveness, and automatic scaling, ensuring the application is always available and performs well.

🛠️ 4. DEVELOPMENT LOGIC

➤ FRONTEND (REACT.JS)

- **Search Form:** A primary component featuring input fields for origin city/airport, destination city/airport, departure date, return date (optional), and number of passengers. Autocomplete suggestions for locations will enhance usability.
- **Validation:** Client-side validation to ensure all mandatory fields are filled and dates are valid (e.g., departure before return, future dates only). User-friendly error messages will guide corrections.
- **API Integration:** Utilize `Axios` for making asynchronous HTTP requests to the Node.js backend to fetch flight data. Proper handling of loading states and error responses will be implemented.
- **Result Rendering:** Dynamically display flight results as interactive cards. Each card will show essential details such as airline logo, flight number, departure/arrival times, duration, number of stops, and price. Pagination or infinite scrolling will manage large datasets.
- **Filtering & Sorting:** Implement user controls for sorting results by price (low-to-high, high-to-low) and duration (shortest-to-longest). Additional filters by airline, number of stops, and departure/arrival time ranges will be considered.
- **User Experience (UX):** Incorporate loading spinners during API calls, clear empty state messages, and informative error notifications.
- **Routing:** `React Router` will manage client-side navigation, allowing for distinct URLs for the search page, results display, and a simulated booking/confirmation page.

➤ BACKEND (NODE.JS + EXPRESS)

- **API Endpoints:**
  - `GET /api/flights?source=XYZ&destination=ABC&date=YYYY-MM-DD&passengers=N`: Primary endpoint to query and retrieve flight

data. This will proxy requests to the external flight API or serve cached/mock data.
    - `POST /api/book`: (Optional) Endpoint to simulate a booking process. It will accept flight details and user information, returning a simulated confirmation.
- **Data Handling:** Responsible for making secure calls to the 3rd-party flight API, parsing the raw JSON response, and transforming it into a consistent format suitable for the frontend. Includes error handling for external API failures.
- **Middleware:**
    - `CORS (Cross-Origin Resource Sharing)`: To allow requests from the React frontend running on a different origin.
    - `Error Handling`: Centralized error handling middleware to catch and process exceptions, providing meaningful error responses to the client.
    - `Request Validation`: Basic validation of incoming query parameters for the `/flights` endpoint to prevent malformed requests.
- **External API Interaction:** Manage API keys securely, handle rate limits (if applicable), and implement retry mechanisms for transient failures.

🗃️ 5. DATA FLOW LOGIC

The application's data flow follows a clear, unidirectional path:

`User Inputs → React Form → Axios Request → Node.js Backend → Flight API → Response (JSON) → Node.js Backend (Processing) → React Frontend (Display Cards)`

🧪 6. TESTING STRATEGY

- **Frontend Testing:**
    - **Unit Testing:** Using Jest and React Testing Library to test individual React components in isolation (e.g., form input changes, button clicks, state updates).
    - **Integration Testing:** Verifying the interaction between components (e.g., search form submitting correctly to display results).
- **Backend Testing:**
    - **Unit Testing:** Jest for individual functions and API utility methods.

    - **Integration Testing:** Postman or Supertest to test REST API endpoints, ensuring they return correct data and handle various inputs/errors as expected.
- **Bug Tracking:** A simple tracking system using GitHub Issues will be maintained to log, prioritize, and manage bugs and feature requests throughout the development lifecycle.

📦 7. DEPLOYMENT PLAN

Step	Tool Used	Description
Frontend Hosting	Netlify or Vercel	The React application will be built and deployed as static assets. These platforms offer seamless CI/CD integration with GitHub, automatic SSL, and CDN for fast global access.
Backend Hosting	Railway or Render	The Node.js Express server will be deployed to a cloud platform. These services provide persistent servers, environment variable management for API keys, and scalability features.
Domain Mapping	Provided by Hosting Services	Both frontend and backend hosting services provide free subdomains (e.g., `myapp.netlify.app`, `myapi.railway.app`). A custom domain can be mapped if acquired, but for the initial phase, default domains will suffice.

📌 8. EXPECTED OUTPUT

Upon successful completion, users should be able to:

- **Effortlessly Search Flights:** Input their travel preferences (source, destination, date, passengers) and initiate a search.
- **View Comprehensive Results:** See a clear, organized list of available flights with key details such as airline, times, duration, and price.
- **Filter and Sort Results:** Refine their search by applying filters (e.g., number of stops, airlines) and sorting options (e.g., price, duration).
- **(Optional) Simulate a Booking Process:** If implemented, users can click on a flight to view more details and proceed through a mock booking flow, confirming their selection without actual transaction.