**Weather Application Design Flow Document**

**Overview**

The Weather Application is a web-based tool that allows users to:

* Fetch and display weather data for a specific city.
* View details such as temperature, humidity, wind speed, and weather description.
* Handle errors gracefully (e.g., invalid city name or API issues).

This document outlines the flow of the application, including component interactions, service usage, and user actions.

**1. Flowchart for Weather Application**

**Step-by-Step Flow**

**Initial Page Load (Dashboard)**

**Component:** AppComponent (Main Dashboard)  
**Action:** On app load, the dashboard displays a search input field and a button to fetch weather data.  
**Flow:**

1. **The AppComponent initializes and displays the WeatherComponent.**
2. **The WeatherComponent renders a search input field and a "Get Weather" button.**
3. **The user can enter a city name and click the button to fetch weather data.**

**Fetch Weather Data**

**Component: WeatherComponent  
Action: The user enters a city name and clicks the "Get Weather" button.  
Flow:**

1. **The WeatherComponent captures the city name from the input field.**
2. **The WeatherService is called with the city name to fetch weather data from the OpenWeatherMap API.**
3. **If the API call is successful:**
   * **The weather data (e.g., temperature, humidity, wind speed) is displayed in the UI.**
   * **The weatherData object is updated with the API response.**
4. **If the API call fails:**
   * **An error message is displayed (e.g., "City not found").**
   * **The errorMessage variable is updated with the error details.**

**Service Interaction:**

* **The WeatherService uses Angular's HttpClient to make a GET request to the OpenWeatherMap API.**
* **The API URL is constructed with the city name and API key.**

**Display Weather Data**

**Component: WeatherComponent  
Action: Display the fetched weather data.  
Flow:**

1. **The WeatherComponent receives the weather data from the WeatherService.**
2. **The data is displayed in a structured format:**
   * **City name and country.**
   * **Weather description (e.g., "Clear sky").**
   * **Temperature in Celsius.**
   * **Humidity and wind speed.**
3. **If no data is available, a message like "No weather data found" is displayed.**

**Error Handling**

**Component: WeatherComponent  
Action: Handle errors during the API call.  
Flow:**

1. **If the API call fails (e.g., invalid city name or network error):**
   * **The errorMessage variable is updated with a user-friendly error message.**
   * **The weatherData object is set to null.**
2. **The error message is displayed in the UI.**

**Search for Weather Data**

**Component: WeatherComponent  
Action: The user can search for weather data by entering a city name.  
Flow:**

1. **The user types a city name into the search input field.**
2. **When the user clicks the "Get Weather" button:**
   * **The getWeather() method in the WeatherComponent is triggered.**
   * **The WeatherService fetches the weather data for the specified city.**
3. **The UI updates dynamically to display the weather data or error message.**

**2. Diagram of the Flow**

**Here’s how the flow of the Weather Application can be represented visually:**

**Copy**

**[Initial Page Load (Dashboard)]**

**--> [Display Search Input and Button]**

**--> [User Enters City Name and Clicks "Get Weather"]**

**--> [Call WeatherService to Fetch Weather Data]**

**--> [If Successful] --> [Display Weather Data]**

**--> [If Failed] --> [Display Error Message]**

**3. Services**

**WeatherService**

**The WeatherService is responsible for:**

* **Fetching weather data from the OpenWeatherMap API.**
* **Handling API errors and returning meaningful responses.**

**Service Functions:**

1. **getWeather(city: string): Observable<any>**
   * **Fetches weather data for the specified city.**
   * **Constructs the API URL with the city name and API key.**
   * **Returns the weather data as an Observable.**

**Service Interaction in Components:**

* **The WeatherComponent calls the getWeather() method to fetch weather data.**
* **The WeatherService uses Angular's HttpClient to make HTTP requests.**

**typescript**

**Copy**

**@Injectable({**

**providedIn: 'root'**

**})**

**export class WeatherService {**

**private apiUrl = 'https://api.openweathermap.org/data/2.5/weather';**

**private apiKey = environment.weatherApiKey;**

**constructor(private http: HttpClient) {}**

**getWeather(city: string): Observable<any> {**

**const url = `${this.apiUrl}?q=${city}&appid=${this.apiKey}&units=metric`;**

**return this.http.get<any>(url).pipe(**

**catchError(error => {**

**console.error('Error fetching weather data:', error);**

**return throwError(() => new Error('Failed to retrieve weather data.'));**

**})**

**);**

**}**

**}**

**4. Components**

**AppComponent**

* **Acts as the main container for the application.**
* **Displays the WeatherComponent.**

**typescript**

**Copy**

**@Component({**

**selector: 'app-root',**

**template: `**

**<h1>Welcome to Weather App</h1>**

**<app-weather></app-weather>**

**`,**

**standalone: true,**

**imports: [WeatherComponent]**

**})**

**export class AppComponent {}**

**WeatherComponent**

* **Handles user input and displays weather data.**
* **Calls the WeatherService to fetch weather data.**

**typescript**

**Copy**

**@Component({**

**selector: 'app-weather',**

**standalone: true,**

**imports: [CommonModule, FormsModule],**

**templateUrl: './weather.component.html',**

**styleUrls: ['./weather.component.css']**

**})**

**export class WeatherComponent {**

**city: string = '';**

**weatherData: any = null;**

**errorMessage: string = '';**

**constructor(private weatherService: WeatherService) {}**

**getWeather(): void {**

**if (this.city) {**

**this.weatherService.getWeather(this.city).subscribe({**

**next: (data) => {**

**this.weatherData = data;**

**this.errorMessage = '';**

**},**

**error: (error) => {**

**this.errorMessage = error.message || 'Failed to retrieve weather data.';**

**this.weatherData = null;**

**}**

**});**

**} else {**

**this.errorMessage = 'Please enter a city name.';**

**this.weatherData = null;**

**}**

**}**

**}**

**WeatherComponent Template**

* **Displays the search input, button, and weather data.**

**html**

**Copy**

**<div class="weather-container">**

**<h1>Weather Application</h1>**

**<div class="input-container">**

**<input type="text" [(ngModel)]="city" placeholder="Enter city name" />**

**<button (click)="getWeather()">Get Weather</button>**

**</div>**

**<div \*ngIf="weatherData">**

**<h2>{{ weatherData.name }}, {{ weatherData.sys.country }}</h2>**

**<p>{{ weatherData.weather[0].description | titlecase }}</p>**

**<h3>{{ weatherData.main.temp }} °C</h3>**

**<p>Humidity: {{ weatherData.main.humidity }}%</p>**

**<p>Wind Speed: {{ weatherData.wind.speed }} m/s</p>**

**</div>**

**<div \*ngIf="errorMessage" class="error-message">**

**<p>{{ errorMessage }}</p>**

**</div>**

**</div>**

**Run HTML**

**5. Conclusion**

**This design flow document outlines the structure and functionality of the Weather Application. It defines the interactions between components, services, and user actions, ensuring a smooth and intuitive user experience. By following this flow, developers can implement the application efficiently while maintaining clean and organized code.**