Project Documentation: Area Info Lookup Application

**Project Overview**

The **Area Info Lookup Application** is an Ionic-Angular-based application that allows users to search for weather information for a specific city and save their favorite cities for quick access. The application integrates with the OpenWeatherMap API to fetch real-time weather data. This documentation outlines the planning, thought process, and rationale behind the changes made during the development of the project.

**Features**

1. **Search Weather Information**:
   * Users can input a city name to fetch weather details such as temperature, condition, and wind speed.
2. **Save to Favorites**:
   * Users can save weather data for specific cities to a favorites list.
3. **Responsive Design**:
   * The application is designed to work seamlessly on both desktop and mobile devices.
4. **Lazy Loading**:
   * Modules are lazy-loaded to improve performance and reduce initial load time.

**Planning and Thought Process**

**1. Project Structure**

* The project is structured to follow Angular's modular design principles:
* **App Module (app.module.ts)**:
  + Serves as the root module, bootstrapping the application and configuring global providers.
* **Home Module (home.module.ts)**:
  + Encapsulates the logic and UI for the home page, making it reusable and modular.
* **Routing (app-routing.module.ts)**:
  + Configures lazy loading for the HomePageModule to optimize performance.

**2. Key Changes and Rationale**

**2.1. app.module.ts**

* **Purpose**:
  + Acts as the root module for the application.
* **Changes**:
  + Added provideHttpClient to configure the HTTP Client service for API communication.
  + Imported Forms Module to enable two-way data binding for user input.
* **Rationale**:
  + provideHttpClient is the modern way to configure HTTP services in Angular, replacing the deprecated HttpClientModule.
  + Forms Module is essential for handling user input in the search bar.

**2.2. home.page.ts**

* **Purpose**:
  + Implements the logic for fetching weather data and managing the favorites list.
* **Changes**:
  + Added methods for:
    - Fetching weather data (fetchWeather).
    - Saving weather data to favorites (saveToFavourites).
    - Loading favorites from localStorage (loadFavourites).
  + Used provideHttpClient to configure HTTP services locally.
* **Rationale**:
  + Encapsulating logic in the HomePage component ensures separation of concerns.
  + Using localStorage for favorites provides a simple and persistent storage solution.

**2.3. home.module.ts**

* **Purpose**:
  + Encapsulates the HomePage component and its dependencies.
* **Changes**:
  + Imported necessary modules (IonicModule, FormsModule, RouterModule).
  + Configured routing for the HomePage.
* **Rationale**:
  + Modularizing the HomePage improves maintainability and scalability.
  + Lazy-loading the HomePageModule reduces the initial load time of the application.

**2.4. app-routing.module.ts**

* **Purpose**:
  + Configures the application's routing.
* **Changes**:
  + Added lazy loading for the HomePageModule.
* **Rationale**:
  + Lazy loading improves performance by loading modules only when needed.

**2.5. app.component.ts**

* **Purpose**:
  + Serves as the root component of the application.
* **Changes**:
  + Added CUSTOM\_ELEMENTS\_SCHEMA to allow the use of custom Ionic components.
* **Rationale**:
  + Ensures compatibility with Ionic's custom elements.

**2.6. app.component.html**

* **Purpose**:
  + Defines the global layout of the application.
* **Changes**:
  + Added a global header and footer.
  + Included a router-outlet for dynamic content rendering.
* **Rationale**:
  + The header and footer provide a consistent layout across the application.
  + The router-outlet enables dynamic loading of page content based on the route

**2.7. app.component.scss**

* **Purpose**:
  + Styles the global layout of the application.
* **Changes**:
  + Styled the header, footer, and content area for a clean and modern look.
  + Added responsive design for mobile compatibility.
* **Rationale**:
  + Ensures a visually appealing and user-friendly interface.

**2.8. home.page.html**

* **Purpose**:
  + Defines the UI for the home page.
* **Changes**:
  + Added input fields for city name and buttons for fetching weather data and saving to favorites.
  + Must display weather data and the favorites list using Ionic components (ion-card, ion-list).
* **Rationale**:
  + Provides an intuitive interface for interacting with the application.

**2.9. home.page.scss**

* **Purpose**:
  + Styles the UI for the home page.
* **Changes**:
  + Styled input fields, buttons, weather cards, and the favorites list.
  + Added responsive design for mobile compatibility.
* **Rationale**:
  + Ensures a visually appealing and consistent design for the home page

**2.10. environment.ts and environment.prod.ts**

* **Purpose**:
  + Exports the environment for used during development builds
* **Changes**:
  + Defined the weatherAPIKey and stored the value there which limits factors like security risks
  + Imported the environment to access the APIKey in the home page typescript

**Challenges and Solutions**

1. **Issue**: HttpClientModule not responding in the constructor.or returning deprecated
   * **Solution**: Used provideHttpClient globally in main.ts and app.module.ts to ensure the service is available.
2. **Issue**: Styling inconsistencies across components.
   * **Solution**: Implemented styling for individual components
3. **Issue**: Application logging an error in the console specifying that duplicate ionic tags located in some components

* Solution: Ensured that the html components do not have matching ionic tags

**Conclusion**

This documentation serves as a guide for understanding the thought process behind the implementation and provides a foundation for future enhancements, the reality is that it is not 100% functional and there were some occurrences where I did not manage to achieve the desired outcome.