Mobile Developer Technical Assessment

Weather Dashboard Application

Overview

You will build a complete weather dashboard application for mobile devices that fetches weather data from the OpenWeatherMap API, implements **local data persistence** for user data, and displays it through a responsive mobile interface. This assessment evaluates your skills in a **Mobile Framework (e.g., Flutter, React Native, or Native iOS/Android)**, **API integration**, **State Management**, and **Mobile UI/UX design**.

Time Allocation: 4-6 hours

Project Requirements

Core Functionality

Build a mobile weather dashboard that allows users to:

- Search for weather information by city name
- View current weather conditions (temperature, humidity, description, etc.)
- See a 5-day weather forecast
- Save favorite cities for quick access
- View weather history for previously searched cities

Technical Specifications

1. Mobile Application (Flutter/React Native/Native)

Target: Develop for both Android and iOS (or focus on one platform if using Native). The following uses **Flutter** as an example, but adjust the framework/tools as needed.

Requirements:

- Create a single-codebase mobile application using Flutter (or preferred mobile framework).
- Implement the following components/screens:
    - **Search Screen** - City search input with validation.
    - **Weather Details Screen** - Display current weather and 5-day forecast.
    - **Favorites Screen** - Manage and display favorite cities.
    - **History Screen** - Show previously searched cities.
    - **Main Navigation** (e.g., BottomNavigationBar or Drawer).

UI/UX Requirements:

- **Responsive Design** (adapting layout for different screen sizes and orientations).
- Adherence to **Platform Conventions** (e.g., using Material Design for Android and Cupertino for iOS, or platform-specific UI patterns).
- Implement **Loading states** and **Error handling** in the UI.
- Clean, intuitive interface.

2. Data Persistence & State Management

Requirements:

- Implement a robust **State Management** solution (e.g., **Provider, BLoC/Cubit, Redux, or MobX**) to handle data flow across the application.
- Implement **Local Data Persistence** for:
    - Weather search history.
    - User favorite cities.
- **Local Storage Choice:** Use a mobile-specific local database/storage solution (e.g., **SQLite/SQFlite, Hive, Realm, or AsyncStorage**). Specify your choice and reasoning.
- **Offline Capability:** The application should display locally cached weather data or history/favorites when the device is offline.

3. API Integration

OpenWeatherMap API Integration:

- Use Current Weather Data API and 5 Day Weather Forecast API.
- Implement robust HTTP client logic.
- Handle API rate limits and errors gracefully, ensuring proper error messages are displayed in the UI for invalid cities or connection issues.

Required API Calls (Implemented client-side):

None
```
// Current weather
GET
api.openweathermap.org/data/2.5/weather?q={city}&appid={API_key}&
units=metric

// 5-day forecast
GET
api.openweathermap.org/data/2.5/forecast?q={city}&appid={API_key}
&units=metric
```

*(**Note:** Unlike the full-stack version, this assessment is client-focused. Direct server implementation is not required unless specified as a bonus.)*

Testing Requirements

Write tests using your framework's native testing tools (e.g., **flutter test** with **mockito** and **widget_test**).

Required Test Coverage (TDD Approach)

- **Unit Tests:** Test the core business logic (e.g., API services, data models, state management logic).
- **Widget/Component Tests:** Verify the UI components (e.g., **SearchBar**, **WeatherCard**) render correctly for different states (loading, error, data available).
- **Integration Tests:** Test full user flows (e.g., searching for a city, adding it to favorites, verifying local storage persistence).

Evaluation Criteria

The criteria are adjusted to emphasize mobile development expertise:

Evaluation Criteria	**Weight**	**Description**
Code Quality	20%	Clean, readable, and well-organized code. Proper error handling. Code reusability and modularity.
Functionality & UI/UX	25%	All required features implemented. **Adherence to mobile platform conventions and responsive design.** Intuitive user experience.
State Management & Architecture	30%	**Robust and scalable state management implementation.** Efficient data flow. Proper use of local persistence and application architecture patterns (e.g., MVVM, BLoC).

Testing & TDD	25%	Comprehensive test coverage (Unit, Widget, Integration). Tests written before implementation (TDD).

Bonus Points

- **Backend Service:** Implementation of a simple, serverless backend (e.g., Firebase, AWS Lambda) for storing favorites instead of local-only storage.
- **Advanced Features:** Background data fetching/notifications, weather maps.
- **Performance:** Code demonstrating memory and performance optimization techniques specific to the mobile platform.
- **Continuous Integration/Delivery (CI/CD):** Setup for automated builds/tests.

Submission Guidelines

1. Create a GitHub repository with your solution.
2. Include a detailed **README** with setup instructions, required dependencies, and how to run tests.
3. Provide a brief video demo (2-3 minutes) showing the application in action.
4. Specify which mobile framework you chose (Flutter, React Native, Native iOS/Android).
5. Submit the repository link and any additional notes.