Weather ForeCast App

1. Instructions for Running the Weather App

Follow these steps to run the Weather App on your local machine:

Requirements:

- **Xcode 12 or later** (for SwiftUI support)
- **macOS 10.15 or later**
- **Simulator or iOS device for testing**

Steps to Run the Weather App:

1. **Clone the Repository:**

If you are using a Git repository for version control, clone the repository to your local machine.
`git clone <repository_url>`

2. **Open the Project:**
    - Open the `.xcodeproj` in **Xcode**.
3. **Set Up API Key (if applicable):**
    - Ensure that the **Weather API key** (e.g., for OpenWeatherMap) is properly configured in the app.
        - Go to **Constants.swift** and add your API key if it's missing.
4. **Build the App:**
    - Select your desired simulator or device in the Xcode toolbar.
    - Click the **Run** button (`Cmd + R`) to build and run the app.
5. **Test the App:**
    - **Home Screen:** Enter a city name and click the "Fetch Weather" button to get weather data.
    - **Weather Details:** View the weather forecast for the next 5 days, including temperature, humidity, and weather conditions.
    - **Error Handling:** Test by entering an invalid city or disconnecting from the internet to see error handling in action.

---

2. Unit Tests for Weather App

Below are the **unit test cases** for the Weather App. These tests ensure the app functions correctly and follows the MVVM architecture with the appropriate error handling, API interaction, and UI states.

Test Case 1: `WeatherViewModel` - Fetch Weather Success

This test case verifies that when a valid city is entered, the weather data is fetched correctly.

```
func testFetchWeatherSuccess() {
    // Arrange
    let expectation = XCTestExpectation(description: "Fetch weather
succeeds")
    let mockResponse = WeatherResponse(
        list: [
            DailyForecast(dt: 1678928400, temp: Temperature(min: 10,
max: 20), weather: [Weather(main: "Cloudy", description: "Cloudy",
icon: "02d")]),
            DailyForecast(dt: 1679014800, temp: Temperature(min: 12,
max: 22), weather: [Weather(main: "Sunny", description: "Clear sky",
icon: "01d")])
        ],
        city: City(name: "Test City")
    )
    mockWeatherService.mockResponse = mockResponse  // Set mock
response

    // Act
    viewModel.fetchWeather(for: "Test City")

    // Assert
    DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
        XCTAssertEqual(self.viewModel.cityName, "Test City")
        XCTAssertEqual(self.viewModel.forecasts.count, 2)
        XCTAssertEqual(self.viewModel.forecasts[0].weather[0].main,
"Cloudy")
        XCTAssertEqual(self.viewModel.forecasts[1].weather[0].main,
"Sunny")
        expectation.fulfill()
    }
```

```
    wait(for: [expectation], timeout: 2)
}
```

Test Case 2: `WeatherViewModel` - Fetch Weather Failure (No Internet)

This test simulates a failure in fetching weather data due to no internet connection.

swift
Copy code

```
func testFetchWeatherFailureNoInternet() {
    // Arrange
    let mockNetworkMonitor = MockNetworkMonitor(isConnected: false)
// Simulate no internet
    let viewModelWithNoInternet = WeatherViewModel(
        weatherService: mockWeatherService,
        persistenceService: mockPersistenceService,
        networkMonitor: mockNetworkMonitor
    )

    // Act
    viewModelWithNoInternet.fetchWeather(for: "Test City")

    // Assert
    DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
        XCTAssertEqual(viewModelWithNoInternet.errorMessage, "No
internet connection. Please check your network and try again.")
        XCTAssertTrue(viewModelWithNoInternet.forecasts.isEmpty)
    }
}
```

Test Case 3: `WeatherViewModel` - Fetch Weather Failure (API Error)

This test case verifies how the app handles API errors, such as a bad server response.

swift
Copy code

```
func testFetchWeatherFailureApiError() {
    // Arrange
```

```
    let expectation = XCTestExpectation(description: "Fetch weather
fails")
    mockWeatherService.error = URLError(.badServerResponse)  //
Simulate server error

    // Act
    viewModel.fetchWeather(for: "Test City")

    // Assert
    DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
        XCTAssertNotNil(self.viewModel.errorMessage)
        XCTAssertEqual(self.viewModel.errorMessage, "Network Error:
Unable to connect to the server.")
        XCTAssertTrue(self.viewModel.forecasts.isEmpty)
        expectation.fulfill()
    }

    wait(for: [expectation], timeout: 2)
}
```

Test Case 4: Persistence - Save Last Searched City

This test case checks if the last searched city is saved correctly in persistent storage.

swift
Copy code

```
func testSaveLastSearchedCity() {
    // Arrange
    let testCity = "Saved City"

    // Act
    viewModel.fetchWeather(for: testCity)

    // Assert
    XCTAssertEqual(mockPersistenceService.getLastSearchedCity(),
testCity)
}
```

Test Case 5: Persistence - Retrieve Last Searched City on App Launch

This test case verifies that the last searched city is retrieved correctly when the app is relaunched.

swift
Copy code

```
func testRetrieveLastSearchedCityOnInit() {
    // Arrange
    let savedCity = "Previously Searched City"
    mockPersistenceService.saveLastSearchedCity(savedCity)

    // Act
    let newViewModel = WeatherViewModel(
        weatherService: mockWeatherService,
        persistenceService: mockPersistenceService
    )

    // Assert

XCTAssertEqual(newViewModel.persistenceService.getLastSearchedCity(),
savedCity)
}
```

Test Case 6: Fetch Weather from Saved Response

This test case verifies that weather data is loaded correctly from previously saved data.

swift
Copy code

```
func testFetchWeatherFromLastSavedResponse() {
    // Arrange
    let savedResponse = WeatherResponse(
        list: [
            DailyForecast(dt: 1678928400, temp: Temperature(min: 15,
max: 25), weather: [Weather(main: "Sunny", description: "Clear sky",
icon: "01d")])
        ],
        city: City(name: "Test City")
    )
```

```
    mockPersistenceService.saveWeatherResponse(savedResponse)

    // Act
    if let response = mockPersistenceService.getWeatherResponse() {
        viewModel.cityName = response.city.name
        viewModel.forecasts = response.list
    }

    // Assert
    XCTAssertEqual(viewModel.cityName, "Test City")
    XCTAssertEqual(viewModel.forecasts.count, 1)
    XCTAssertEqual(viewModel.forecasts[0].weather[0].main, "Sunny")
}
```
