

Documentation
For
KeyForecast Java Application

Submitted By
Name :- Piyush Bhatia
Roll No :- 2010990525
Mob. No :- 9896502619

1. Project Title :-

KeyForecast

2. Abstract :

This project presents a weather forecasting application developed using JavaCore for functionality, JavaX Swing for the user interface, and leveraging TestNG for comprehensive test coverage. The application integrates RESTAssured to retrieve weather data from a Weather API, providing users with accurate and up-to-date forecasts. Additionally, Copilot AI, an advanced AI tool, is employed to enhance code generation and streamline development processes. The user interface offers an intuitive experience, allowing users to easily navigate and visualize weather forecasts for desired locations. The combination of robust functionality, efficient UI design, comprehensive testing, and advanced AI assistance ensures the reliability, accuracy, and usability of the weather forecasting application.

3. Introduction :

Keyforecast application! Powered by JavaCore, JavaX Swing, TestNG, and RESTAssured, our platform delivers real-time weather data with unparalleled accuracy. Leveraging Copilot AI for streamlined development, our intuitive interface ensures seamless navigation for users. Stay informed and ahead of the weather effortlessly with our cutting-edge solution.

4. Functional Overview :

- **Live Weather Snapshot:** Offering up-to-the-minute insights into temperature, humidity, wind speed, and a visual weather icon specific to the user's location
- **Location-Driven Forecast:** Enabling users to easily access weather updates through city names or coordinates.
- **User-Friendly Interface:** Intuitive navigation through interactive elements like dropdown menus, search bars, and buttons.
- **Adaptive Layout:** Seamlessly adjusting content and design for optimal viewing across various devices and screen orientations.

5. Key Features:

Real-Time Weather Overview: Provides instant updates on temperature, humidity, wind speed, and current weather conditions for designated locations.

2. Precision Location Search: Enables users to swiftly access weather details by entering city names or ZIP codes, ensuring accurate forecasts tailored to specific locales.

3. Dynamic Responsiveness: Guarantees a seamless user experience across a variety of devices and screen sizes through adaptive design, maintaining consistency in functionality and presentation.

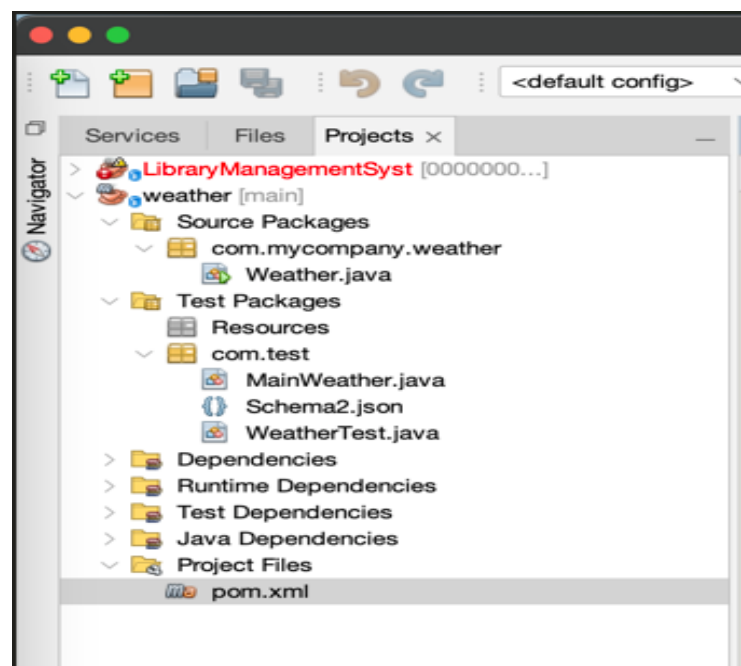
4. Application Deployment Strategy: Implements the transfer of application files to a compatible web server or hosting platform supporting essential technologies, ensuring smooth deployment and operation.

5. Thorough Application Testing: Validates post-deployment functionality by accessing the application through a web browser, confirming the accuracy and timeliness of weather data retrieval.

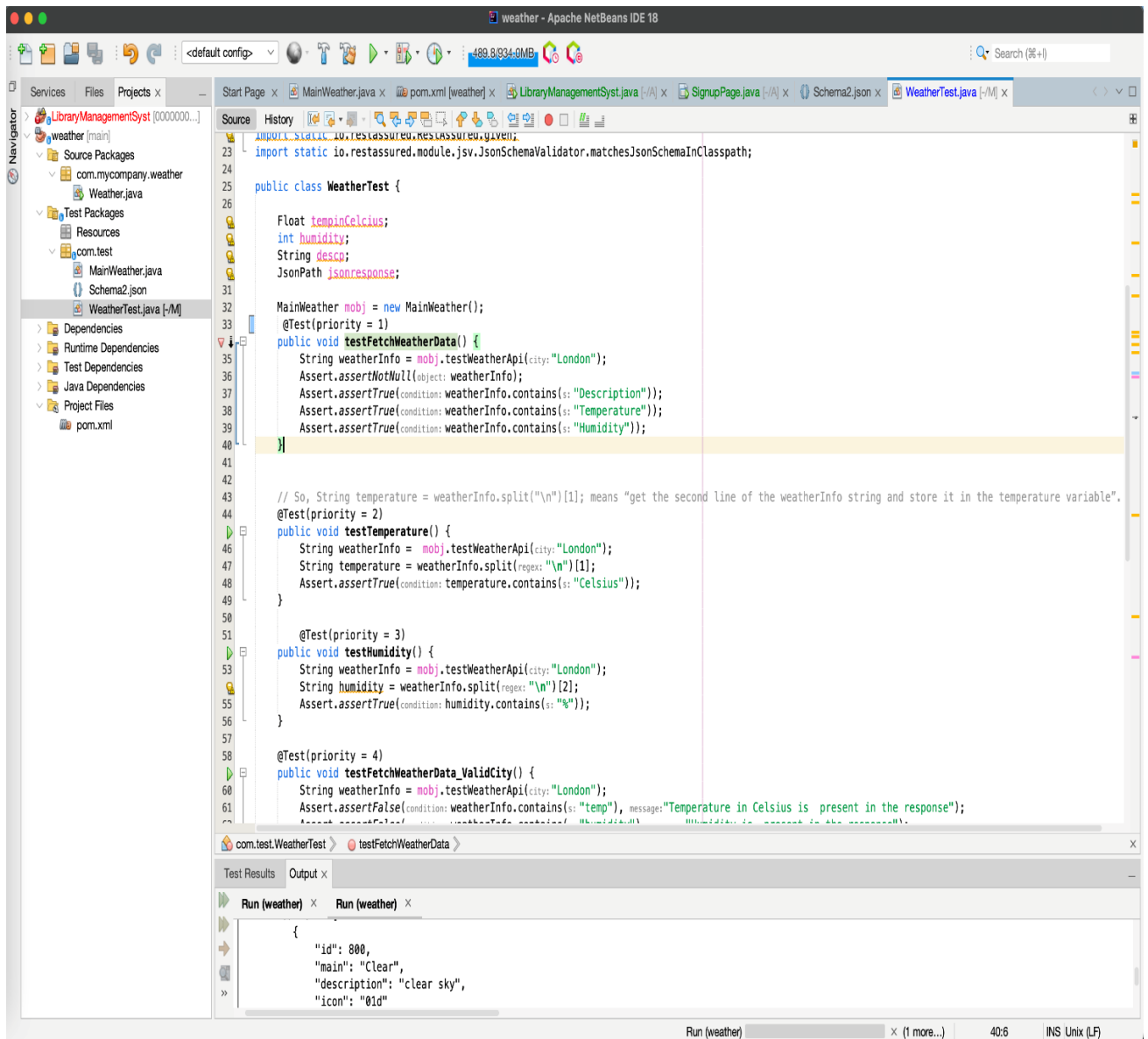
6. Implementation :

In order to efficiently cover the end-to-end flow of the application and save time for API testing process, an automation framework was implemented using RestAssured with Java and use Maven project so that we can add dependencies in pom.xml file. The chosen framework architecture follows TestNG Framework.

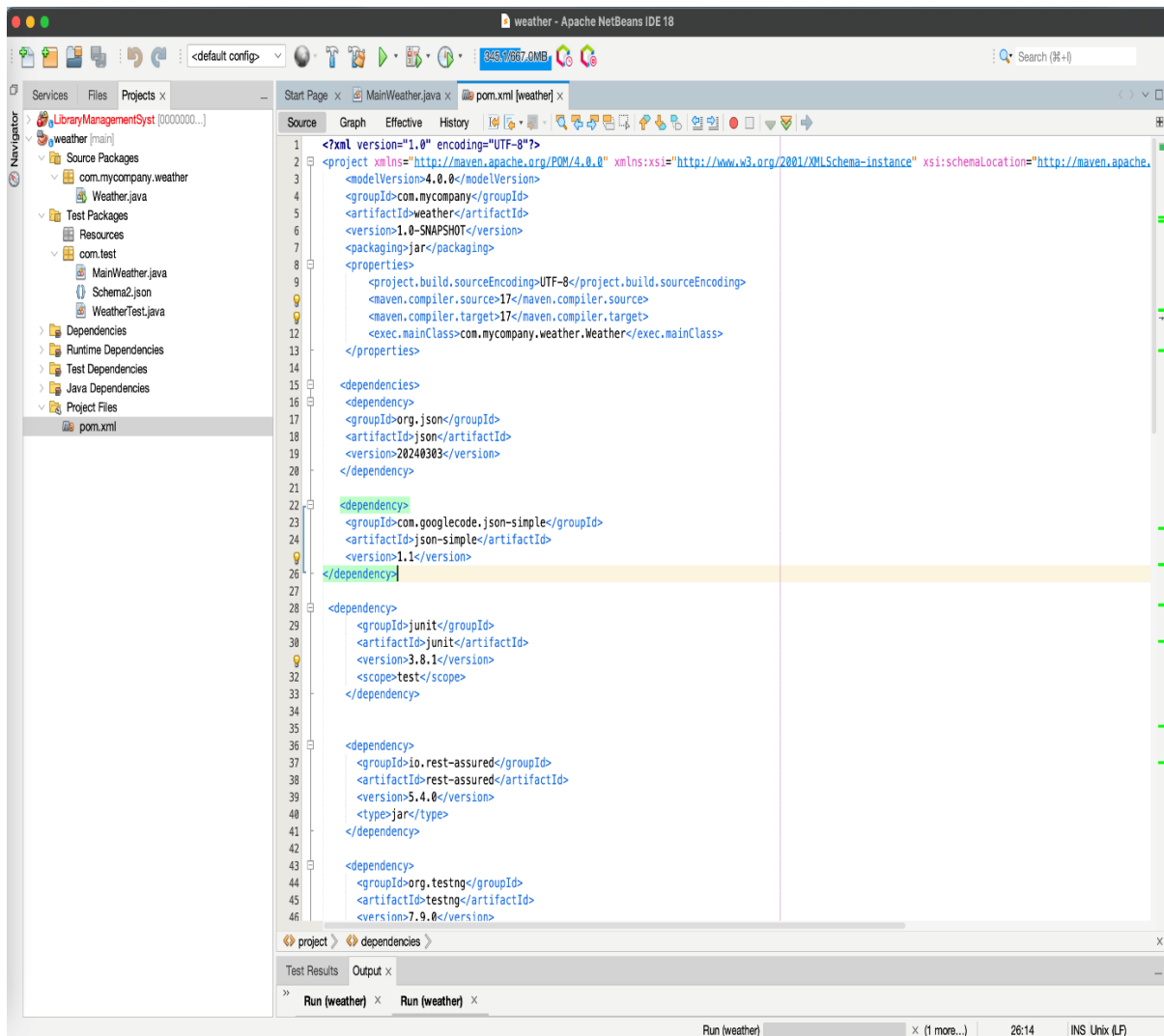
The automation framework is structured as a Maven project with the following directory layout:



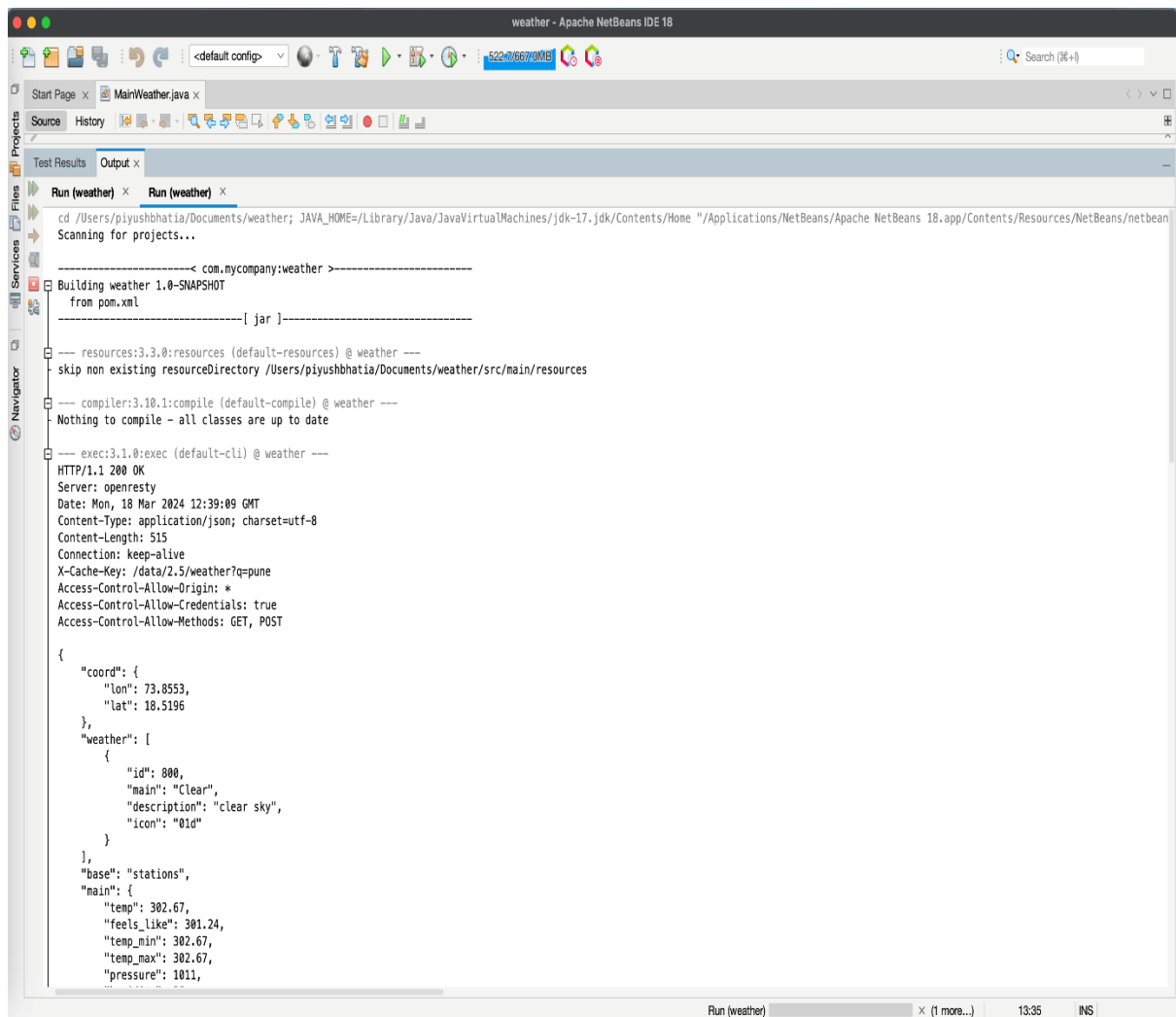
a. MainWeather class structure: In this class we passed base URI of application:



C. POM.xml : We Added all dependencies required in the pom.xml file

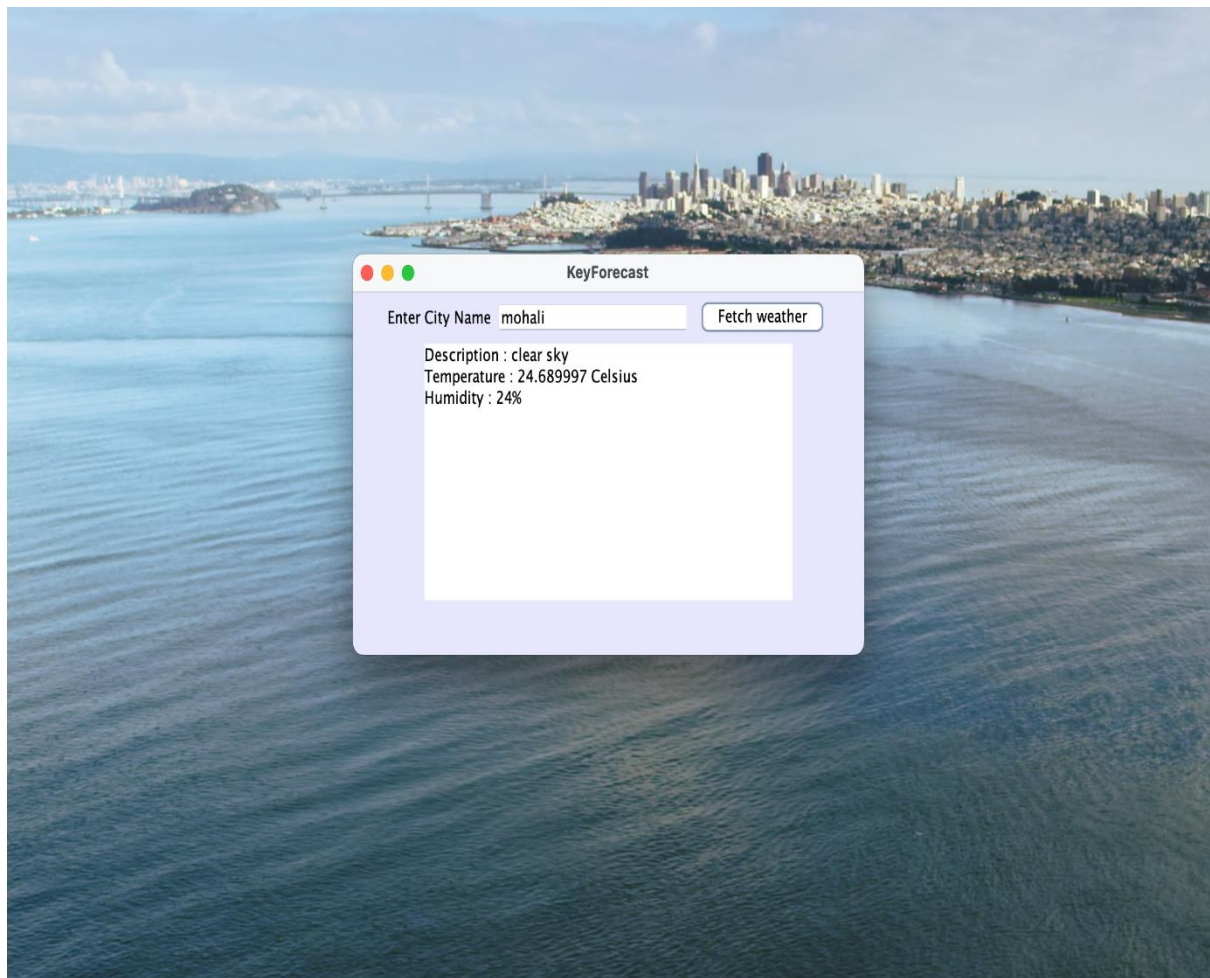


d. Response body :

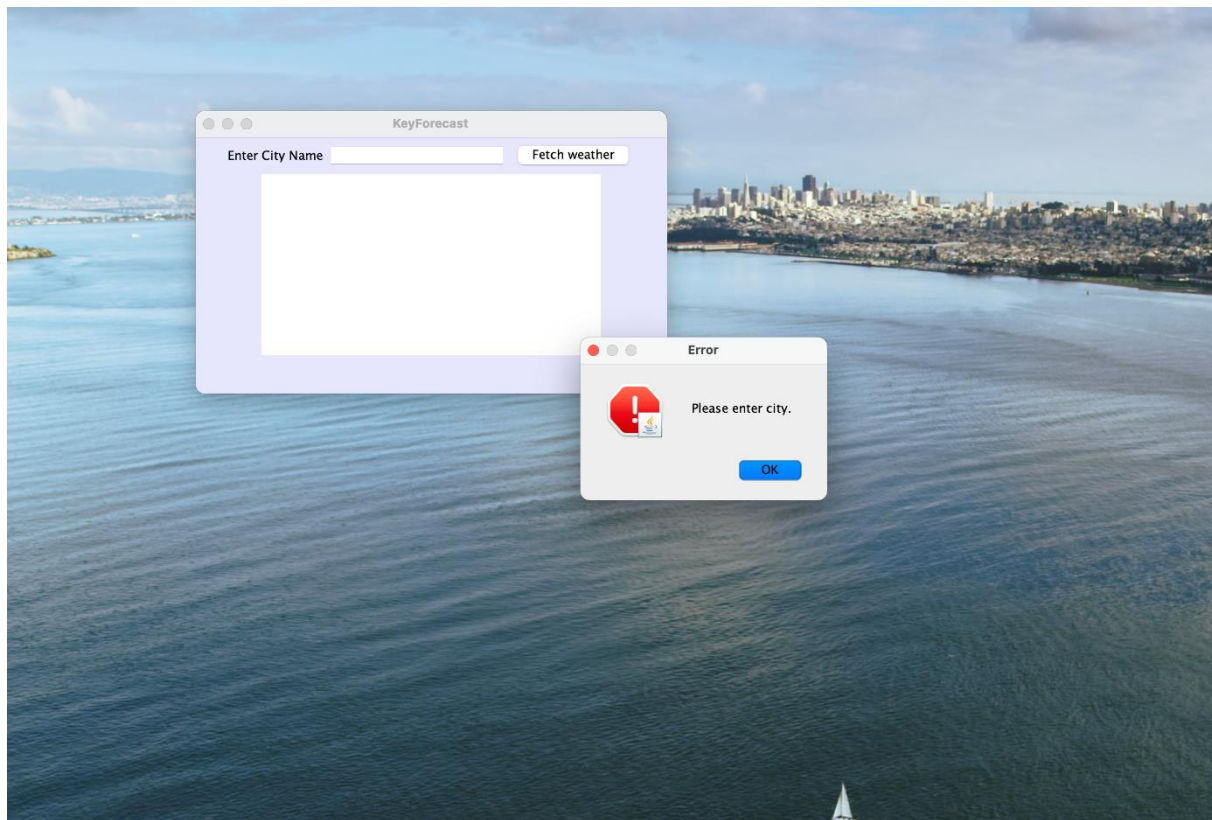


e. UI Snashots:

UI withcity Name:



UI without city:



Co-pilot:

Find - Open... free public w... https://api.o... Java Action... Find - Open... Allure Report copilot.microsoft.com Allure Rep

Copilot

Certainly! Here's a concise description for a **Weather API test**:

A Weather API test involves using an API to retrieve weather-related data. It can be used to validate various aspects of weather-related functionality in applications. For instance:

- **Current Weather Data:** Verify that the API returns accurate real-time weather conditions for a specific location.
- **Hourly Forecast:** Test the API's ability to provide hourly weather predictions for the next few days.
- **Daily Forecast:** Ensure that the API delivers accurate weather forecasts for an extended period (e.g., 16 days).
- **Climatic Forecast:** Validate long-term weather predictions (up to 30 days).
- **Global Weather Alerts:** Test the API's ability to push notifications for weather warnings from national agencies.

In summary, a Weather API test ensures that weather-dependent mobile and web applications function correctly under different weather scenarios 1 2.

Learn more 1 tomorrow.io 2 meteomatics.com 3 openweathermap.org 4 weatherapi.com

1 of 30

You

This Basic App is written in java ,RestAssured and validate the Request with TESTNG. make a short description for Weather API testing framework using the given info

Copilot

Certainly! Based on the information provided, here's a concise description for a **Weather API testing framework**:

New topic Ask me anything... 0/2000

Feedback

Test Cases Passed:

77 pom.xml (EndToEndTesting) Weather.java x JSONException.java UserTest.java schema.json schema2.json MainWeather.java

```
1 package api.test;
2
3 import io.restassured.RestAssured;
4 import io.restassured.path.json.JsonPath;
5 import io.restassured.path.json.exception.JsonPathException;
6 import io.restassured.response.Response;
7 import org.testng.Assert;
8 import org.testng.annotations.BeforeTest;
9 import org.testng.annotations.Test;
10
11 import static io.restassured.RestAssured.given;
12 import static io.restassured.module.json.schema.JsonSchemaValidator.matchesJsonSchemaInClasspath;
13
14 new
15 public class Weather {
16
17     no usages
18     Float tempInCelsius;
19     no usages
20     int humidity;
21     no usages
22     String @ssss;
23     no usages
24     JsonPath jsonresponse;
25     // @BeforeTest
26     // public void testWeatherApi(){
27     //     RestAssured.baseURI= "http://api.openweathermap.org/";
28     //     String response = given().queryParam("q","Mumbai").queryParam("appid","c71a3c0964298436fd10d0c75736dff8").get("
29     //     .then().log().all()
30     //     .assertThat()
31     //     .statusCode(200).extract().response().asString();
32     //     System.out.println(response);
33     //     //
34     //     jsonresponse = new JsonPath(response);
35     //     Float temp = jsonresponse.get("main.temp");
36 }
```

Run Weather x

Default Suite 2 sec 705 ms

- ✓ SwaggerTesting-master 2 sec 705 ms
- ✓ Weather 2 sec 705 ms
- ✓ testFetchWeatherData 1 sec 541 ms
- ✓ testTemperature 210 ms
- ✓ testHumidity 209 ms
- ✓ testFetchWeatherData_ValidCity 244 ms
- ✓ testFetchWeatherData_EmptyCity 501 ms

✓ Tests passed: 5 of 5 tests - 2 sec 705 ms

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/j
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See <http://www.slf4j.org/codes.html#StaticLoggerBinder> fo
HTTP/1.1 200 OK
Server: openresty
Date: Mon, 18 Mar 2024 12:37:22 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 472
Connection: keep-alive
X-Cache-Key: /data/2.5/weather?q=London
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST
{
 "coord": {
 "lon": 0,
 "lat": 51.5074

terminal Local x Local (2) x +

port successfully generated to /var/folders/zh/2hvdbr7d3_l9nfvsx_xlqpc80808gn/T/6539358167288126283/allure-report
starting web server...

24-03-18 18:08:30.421:INFO:main: Logging initialized @4392ms to org.eclipse.jetty.util.log.StdErrLog
24-03-18 18:08:31.727 java[38207:1461867] WARNING: Secure coding is automatically enabled for restorable state! However, not on all supported macOS versions of this application. Opt-in to secure co
ng explicitly by implementing UIApplicationDelegate.applicationSupportsSecureRestorableState:.
over started at <http://127.0.0.1:33812/>. Press <ctrl+c> to exit

ppetTesting-master > src > test > resources > schema2.json LF UTF-8 4 spaces

6. Challenges Faced

-API Integration: Integrating and working with external APIs, such as the OpenWeather API, can be challenging

-Asynchronous Programming: Working with asynchronous JavaScript, especially when making API calls or handling user interactions, can be complex.

-Performance Optimization: Optimizing the application's performance, such as reducing load times, minimizing network requests, caching data, and optimizing images, is crucial for a smooth user experience.

7. References

OpenWeather API Documentation:

-Website: OpenWeather API Documentation

-Access to real-time weather data, key for fetching and displaying weather information in the application.