* **About Application:**

This is a simple user application where user enters an address in application to get the weather condition of last seven days at the particular location.

UI consists of a text-box and submit button. Once user enters location in text box and clicks on submit button, User gets graphical representation of weather conditions for last 7 days.

* **Test Scenarios:**
  + **In scope**

I considered following scenarios for testing:

1. Verify Different inputs and results for the input

|  |  |
| --- | --- |
| **#** | **Test-Scenarios** |
| 1 | VERIFY ALL ELEMENTS PRESENT IN WEATHER APP OVERVIEW PAGE |
| 2 | ENTER VALID CITY AND VERIFY SEARCH RESULTS |
| 3 | ENTER INVALID CITY AND VERIFY SEARCH RESULTS |
| 4 | ENTER EXACT ADDRESS AND VERIFY SEARCH RESULTS |
| 5 | ENTER INVALID STRING AND VERIFY SEARCH RESULTS |
| 6 | ENTER CITY FROM DIFFERENT COUNTRY AND VERIFY SEARCH RESULTS |
| 7 | ENTER COUNTRY NAME AND VERIFY SEARCH RESULTS |
| 8 | ENTER CONTINENT NAME AND VERIFY SEARCH RESULTS |
| 9 | ENTER ZIP CODE AND VERIFY SEARCH RESULTS |
| 10 | ENTER INVALID ZIP CODE AND VERIFY SEARCH RESULTS |
| 11 | ENTER LARGE TEXT AND VERIFY SEARCH RESULTS |
| 12 | ENTER CITY NAME WITH SPECIAL CHARACTERS ('$tockton’) AND VERIFY SEARCH RESULTS |

1. Verify values coming from Proxy API are rendered correctly on UI

|  |  |
| --- | --- |
| **#** | **Test-Scenarios** |
| 1 | CREATE A JSON FILE FROM PROXY API |
| 2 | VERIFY UI VALUES ARE MATCHING WITH BACKEND API |

* + **Out of scope**
* Performance of APIs or UI is not considered.
* JavaScript frontend validations are not currently present so some of the failures in the scripts are attributed to that.
* Comparing UI values to backend API is done for positive scenario.
* Cross browser Testing
* **Why Protractor Automation tool?**
* Protractor is a JavaScript based tool is equipped with Jasmine Framework. Though its primarily supports Angular application, with some support non angular application is also covered.
* Since Protractor is Node.js module, Containerization of the tool along with CI tool becomes easy as we don’t have install JVM for java or a separate framework like Maven,
* This tool is scalable with Selenium grid.
* Handling of RESTful APIs is much simpler in Protractor.
* Some of the JavaScript libraries are easily accessible. ( like Jasmine-reporter or writing test-results in json file)

* **Automation Framework:**

Following is the configuration of the Automation Framework.

1. I am currently using Page Object Model Framework.
2. All the selectors for the elements and Methods to interact with these elements are listed in Object Repository(ObjRep 🡪 WeatherOverview.js)
3. All the utility functions related to Getting data from API and writing the data in output stream are located in ObjRep 🡪 ApiUtil.js
4. Input stream provides the Test-data for different test-scenarios from a “TestData.json” file.
5. Outpur Stream records all the test-results in Testresults.js and Data from API in different files.

Input Stream

TestResults.JS

Output Stream

VerifyProxyAPI\_spec.js

WeatherUIOverview\_spec.js

**Conf. Js**

TestData.JS

WeatherAPI.Json

LocationAPI.Json

ApiUtil. JS

WeatherOverview.js

**Advantages**

1. Above Page Object Model Framework is scalable and we can expand it as the application grows.
2. Easy to Integrate with CI tools.
3. Easy to write test-cases in TDD method.
4. Verification of API Integration is seamless.

* **Shortcomings**

1. Primarily a UI automation framework.
2. Some of the code can be better Organized ( like adding promise chaining for Request module to interact with APIs)
3. Non-Angular applications have some challenges with Protractor(I had to add browser.waitForAngularEnabled(false) and explicit waits browser.wait for browser synchronization.)