

# QAOnlineAssessment-noCSiteCore

In the context of this assessment, we needed to write three scenarios: Two scenarios were about application web, and the other one was a REST-API test scenario. So two different approaches were needed in order to achieve this.

For Scenario 1 and 3, since they are about web applications (google.com & expedia.com). I needed the technologies, the tools and the frameworks that will help reach the goal that is making the automatic test:

IDE: The one I used was Eclipse for Enterprise Java Developers (eclipse IDE 2020-09)

Java: the default one was jdk.11, but I downgraded it to jdk.8 since I wanted to use the tool.jar file to be able to run Junit test

Maven: I create a maven project for its flexibility; since I could use the maven dependencies to import all the jar files needed in the context of this project. I used the version 1.7

BDD Framework: as Behaviour Data Driven framework I did use Cucumber which can easily be integrated with Selenium. I installed this plugin in my IDE.

In the project, I used the **chrome driver**, so my scripts launch the Chrome browser.

I did use **selenium** version 3.141.59

The structure of my project is as follow:

Project name

| **Features folder**

| **Drivers folder**

| **PageObjects**

| **StepDefinitions**

| **testRunner**

| **utilities**

In the **Features** folders, you can find the feature file (the file that defines the step with gherkin language ie cucumber file)

In **Drivers**, you will find there the driver used to open the browser in our case ChromeDriver

In **PageObjects**, that's where you'll find the Page Object, the object that will help us to locate html elements in the pages. This object is an attribute of object stepDefinitions

In **StepDefinitions**: you'll find the class that implements all the steps found in the feature file written in Gherkin Language.

In **testRunner**: there you can find the test class that helps us to run our scenario as a test class and generate some reports in more readable format.

**Utilities**: there you can find the utilities classes like generate random value I used in first scenario to create separate file to store the different print screens required.

Finally, I'll add that I did create two projects separately for the only reason that the two scenario, even if they used the same technologies, frameworks... have different context

For the second scenario (weather REST-API). I did use the tool Postman. I've created one collection in which I added a folder 'GET' where I added a request.

Since the requirement was about testing the weather for **tomorrow** in "**New York**", I had to call <http://api.openweathermap.org/data/2.5/forecast> which can return me a **5-day weather forecast** for the given city. In the **Json response**, I could then extract the weather for the date of tomorrow. More details can be found in the test scripts I did make in Postman. To get the response in Celsius, I set the units parameters to '**metric**'