Por información más detallada sobre Appium y para qué sirve, ver manual :
Appium-en-windows-android.docx

Appium Tests

Repositorio: https://github.com/abstracta/appiumTests

Descripción :

Este proyecto fue creado como un template de un proyecto para automatizar mobile usando Java como lenguaje de programación.

La idea de este proyecto es que en él se encuentren los tests de la aplicación que queramos automatizar y mediante la librería de Appium se logre conectar a un dispositivo ya sea virtual o real, y se lancen los casos de prueba en el mismo, facilitando la prueba en distintos dispositivos sin tener que probar manualmente la aplicación.

Requerimientos :

Es necesario seguir el manual que se encuentra más arriba sobre Appium, ya que en él se encuentran las instrucciones de todo lo que necesitaremos para abrir este proyecto y poder utilizarlo correctamente. También es necesario un IDE para poder abrir el proyecto (usado en este manual : Intellij Idea). Por último, necesitaremos el proyecto, que puede ser descargado desde Github de Abstracta llamado “appium Tests”.

Frameworks y Librerías :

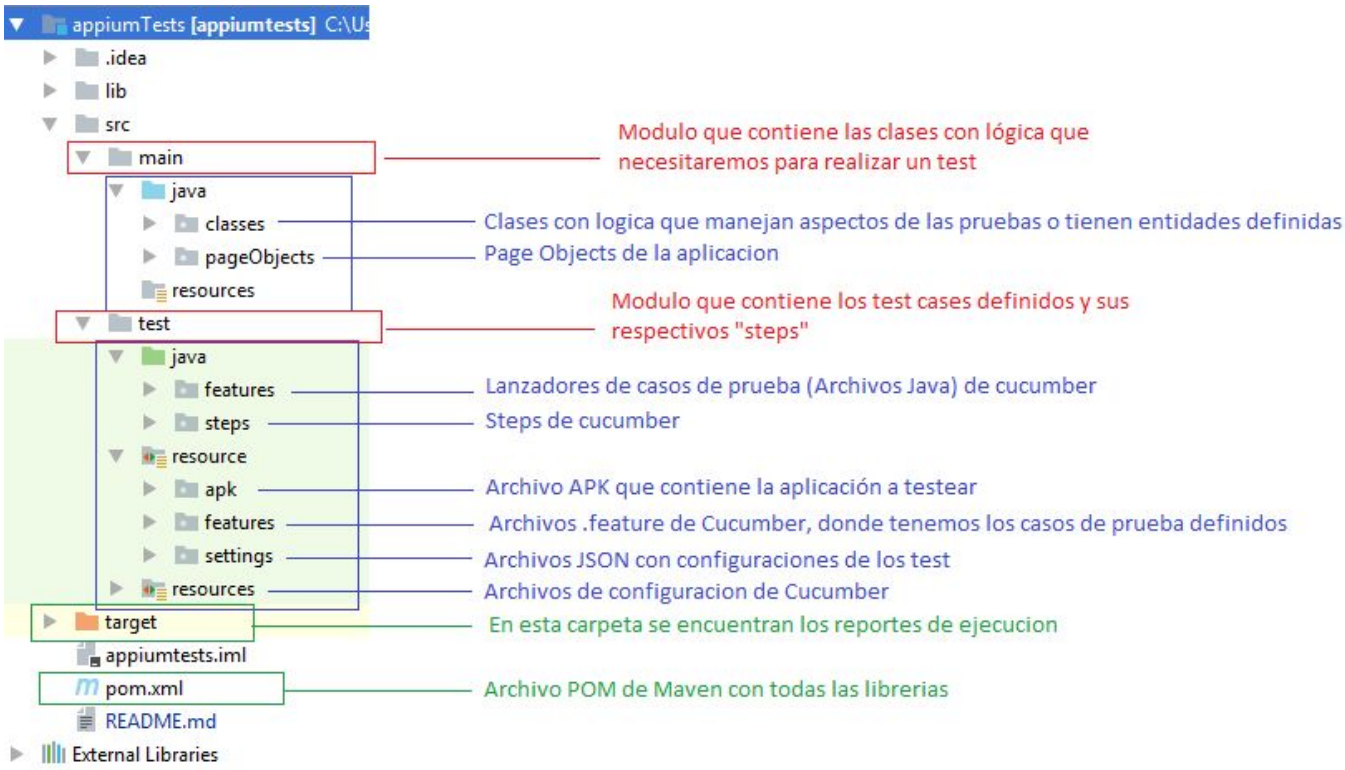
El proyecto importa ciertas librerías para un mejor uso del proyecto y una mejor organización, por lo cual se usan las siguientes librerías y frameworks :

- **Maven :** Para el control de versionado de las librerías del proyecto.
- **Cucumber :** Framework de pruebas para implementar metodologías como el Behaviour Driven Development (BDD), que permite ejecutar descripciones funcionales en texto plano como pruebas de software automatizadas.
  Estas descripciones funcionales, se escriben en un lenguaje específico de dominio, legible por el área de negocio, denominado “Gherkin”, el cual sirve simultáneamente como documentación de apoyo al desarrollo y de las pruebas automatizadas.
- **Appium (librería) :** Es la librería que se conecta con el servidor de Appium y realiza finalmente las acciones en el dispositivo.
- **Cucumber-Reporting (librería) :** Es una librería que genera un reporte más extenso y más profesional usando cucumber, generando gráficas y otros

datos de interés para su ejecución mediante Maven o ejecutando desde Jenkins (No funciona ejecutando desde el IDE).

- **TestiNg :** Framework que se eligió para interactuar con Cucumber y correr las “features” de Cucumber, permite anotaciones, es flexible, pasaje de parámetros, etc.

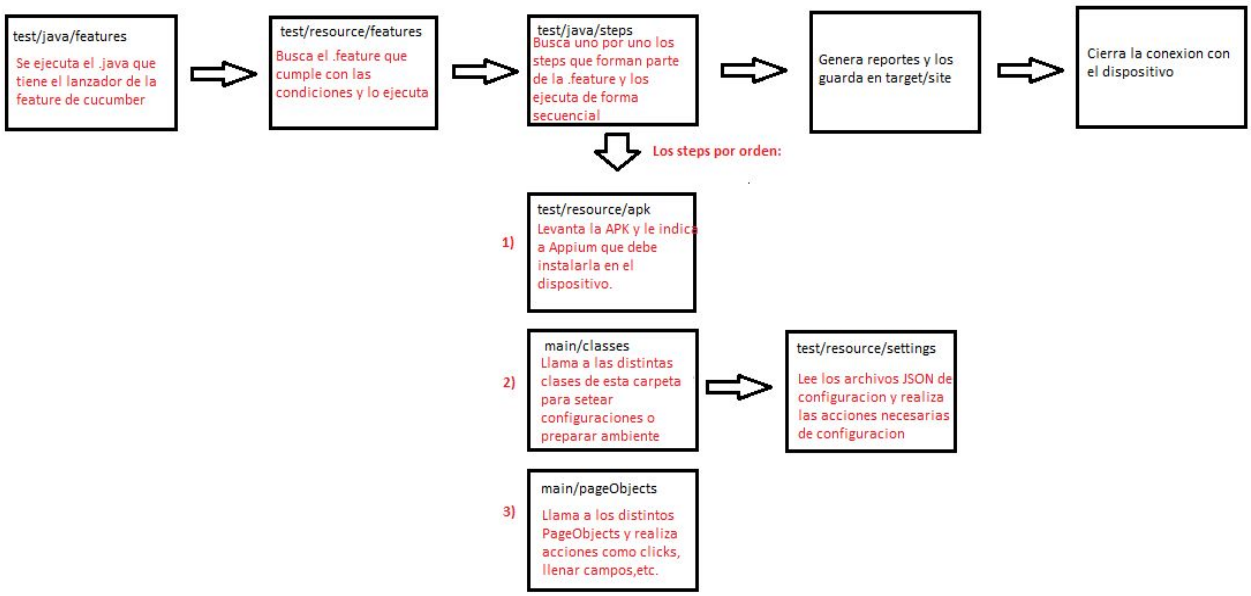
Estructura:



Puntos importantes a tener en cuenta :

- Para que la clase java (la que lanza los tests) encuentre los archivos .feature de cucumber, los mismos deberán estar en la misma carpeta bajo el módulo “resource”. En este caso tenemos la carpeta “feature” en test/java y en resource, por lo que funciona correctamente. Se puede cambiar la carpeta pero es necesario especificarlo en el cucumber Options del lanzador de la prueba.
- Para obtener reportes más completos, podemos lanzar esta prueba desde la consola usando Maven. Para esto es necesario ejecutar **> mvn install** desde la consola.
- A la hora de importar el proyecto seleccionar las opciones de descargar documentación y orígenes.

Flujo de ejecución :



Usando Cucumber :

Cucumber nos permite generar casos de prueba de forma rápida usando texto plano, lo que lo hace legible tanto para desarrolladores como para personas con un perfil funcional / de negocios.

Hay dos partes importantes de Cucumber: por un lado las “Features” y por el otro los “Steps”. Cada feature es un archivo que puede contener uno o más casos de prueba que definimos usando lenguaje de texto, que a su vez contiene uno o más steps (las instrucciones que realizan). Estos steps se traducen a métodos concretos que ejecutan código escrito por el automatizador, y que son definidos en lenguaje natural.
La manera de que un texto plano sea mapeado a un método es debido a que se utilizan annotations de Java en el método, por lo cual Cucumber busca el metodo que tenga esa annotation en lenguaje natural. Un ejemplo es :

```
@Given("^The device is setted and the application called \"([^\"]*)\" is opened$")
public void openDevice(String applicationName)  throws Exception {
}
```

Por lo cual, si en el caso de prueba llamamos al step “The device is setted and the application called ‘123’ is opened” , Cucumber buscará un método que tenga esa annotation y lo ejecutará.

Ejemplo de una Feature:

```
Feature: Search courses
  In order to ensure better utilization of courses
  Potential students should be able to search for courses

  Scenario: Search by topic
    Given there are 240 courses which do not have the topic "biology"
    And there are 2 courses A001, B205 that each have "biology" as one of the topics
    When I search for "biology"
    Then I should see the following courses:
      | Course code |
      | A001        |
      | B205        |
```

En una feature se escribe en texto plano, el archivo debe guardarse con formato .feature, y contiene (al menos) lo siguiente (puede contener otras cosas como precondiciones) :

Feature : “Una descripción de todos los casos que tiene este archivo feature”
Scenario : “Una descripción del caso de prueba”
Luego tenemos los steps que son siempre llamados con algunas palabras reservadas para dar sentido al texto, como “**Given, When, And, Then, ***”, únicamente sirven para dar sentido, no ejecuta ningun codigo en si.
Después de haber escrito la palabra reservada se llama al método en texto plano (ej: “I search for ‘biology’”). Cabe destacar que **los steps pueden contener parámetros.**

Un ejemplo de cómo se encuentra definido un step :

```
public class LoginSteps {
        private static final Logger LOGGER = Logger.getLogger(LoginSteps.class.getName());

        @Given("^I navigate to the mock application$")
        public void given_I_navigate_to_the_mock_application(){
                LOGGER.info("Entering: I navigate to the mock application");
        }

        @When("^I try to login with '(.+)' credentials$")
        public void when_I_try_to_login(String credentialsType){
                LOGGER.info("Entering: I try to login with " +
                        credentialsType + " credentials");
        }

        @Then("^I should see that I logged in '(.+)'$")
        public void then_I_login(String outcome){
                LOGGER.info("Entering: I should see that I logged in " + outcome);
        }
}
```

El step se define como un método normal en java, y por arriba de la definicion del metodo se escribe @ una palabra reservada y entre paréntesis y comillas el texto plano por el cual será llamado en la feature.

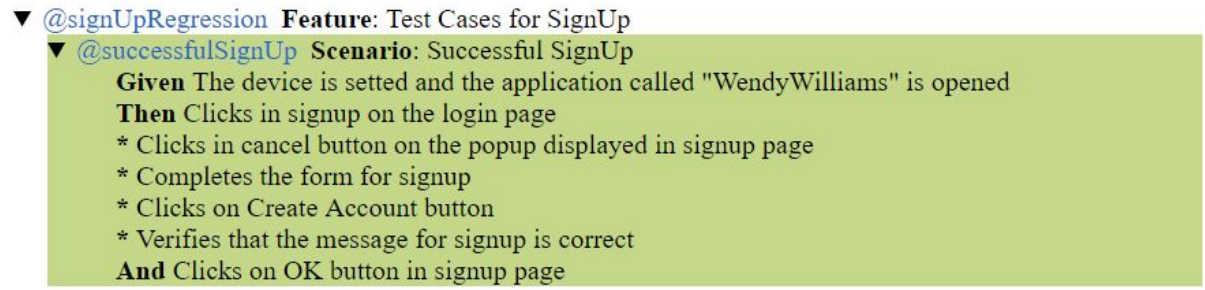
- En la feature se puede tener parametrización usando una tabla como bien muestra el ejemplo, donde podemos ejecutar siempre el mismo caso de prueba pero con distintos datos.

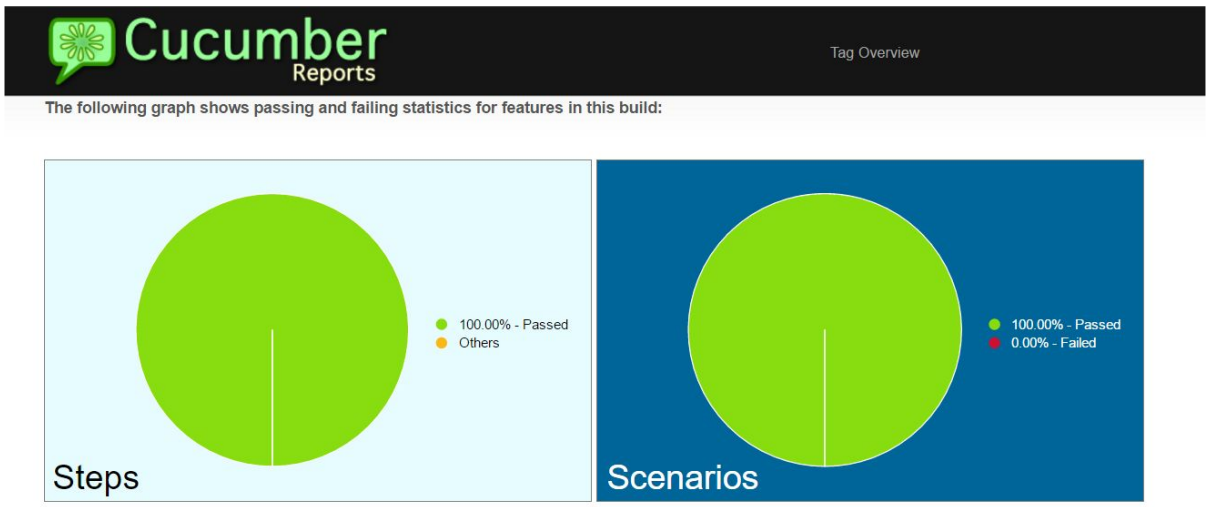
- Cada caso de prueba puede tener uno o más tags que lo identifiquen, y que sirve para luego poder ejecutar únicamente ese caso de prueba, o todos los casos de prueba que contengan ese tag.
- Por mas informacion sobre cucumber y todo su potencial : http://toolsqa.com/cucumber/cucumber-tutorial/

Reportes :

Cada ejecución de cucumber deja un reporte creado por el framework, explicando cómo resultó la misma. Este reporte debe buscarse en la carpeta “target/site/cucumber-pretty” en el caso de haberlo ejecutado desde el IDE, o en “target/site/cucumber-reports” si se ejecutó desde Maven o Jenkins.

Ejemplos :





Feature Statistics

	Scenarios			Steps						
Feature	Total	Passed	Failed	Total	Passed	Failed	Skipped	Pending	Duration	Status
Test Cases for SignUp	1	1	0	7	7	0	0	0	1 min and 2 secs and 498 ms	passed
1	**1**	**1**	**0**	**7**	**7**	**0**	**0**	**0**	**1 min and 2 secs and 498 ms**	**Totals**