EJERCICIO DE APIS

Autor: Leonardo Páez

URL repositorio: https://github.com/leopass1997/NTTDataKarateTest/tree/feature/develop (el desarrollo se encuentra sobre la rama feature/develop)

Objetivo

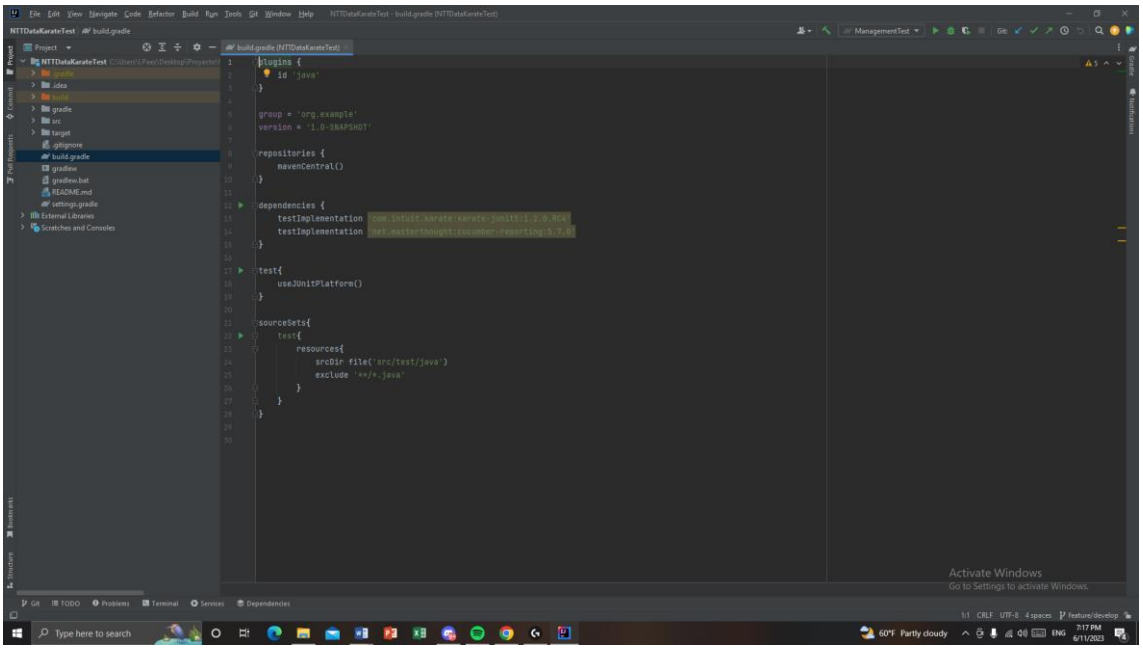
La página https://petstore.swagger.io/ proporciona la documentación sobre apis de una "PetStore". Utilizando un software para pruebas de servicios REST realizar las siguientes pruebas, identificando las entradas, capturando las salidas, test, variables, etc, en cada uno de los siguientes casos:

- Añadir una mascota a la tienda
- Consultar la mascota ingresada previamente **(Búsqueda por ID)**
- Actualizar el nombre de la mascota y el estatus de la mascota a **"sold"**
- Consultar la mascota modificada por estatus **(Búsqueda por estatus)**

Desarrollo

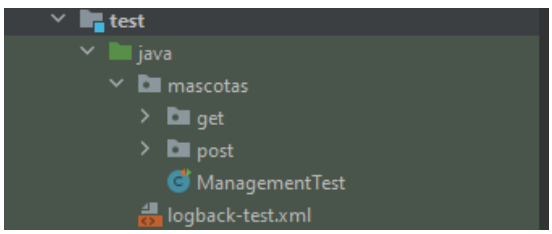
Abrir el proyecto intelijIDEA y generar un proyecto gradle. Este tipo de proyecto nos servirá para instalar las dependencias en el archivo build.gradle que trae consigo.

Las dependencias utilizadas son para el framework de karate y la de cucumber para la generación de reportes. Adicionalmente se descargan plugins de java para el compilador y se realiza una configuración para que tome los feature tests adentro de la carpeta java.

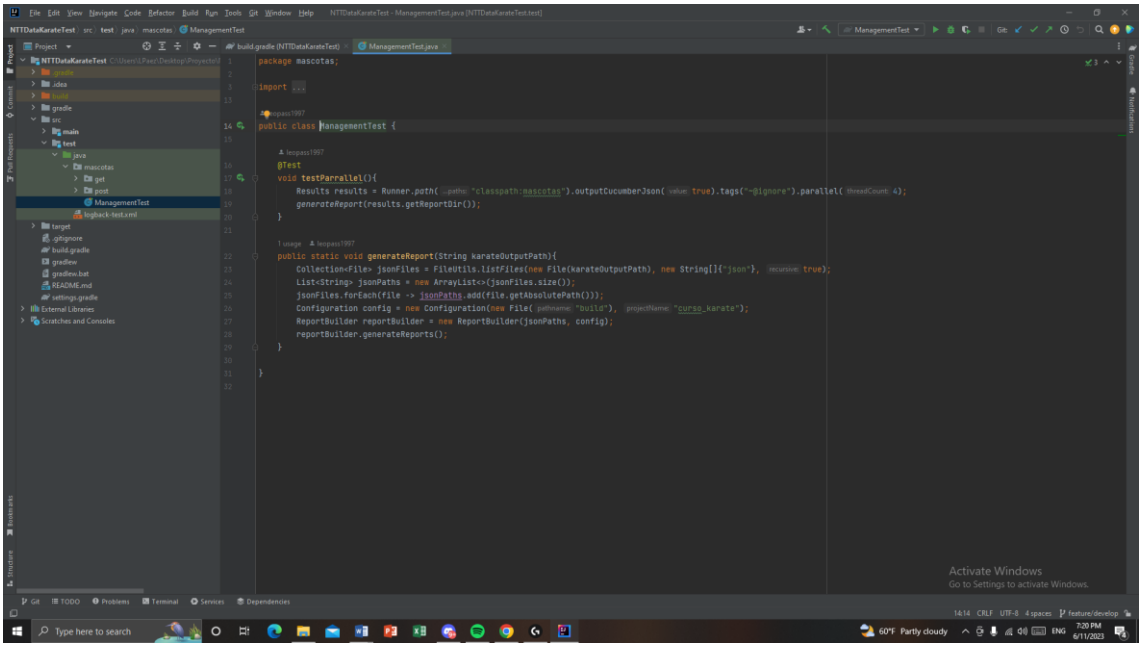


Se genera la siguiente estructura en el árbol del proyecto

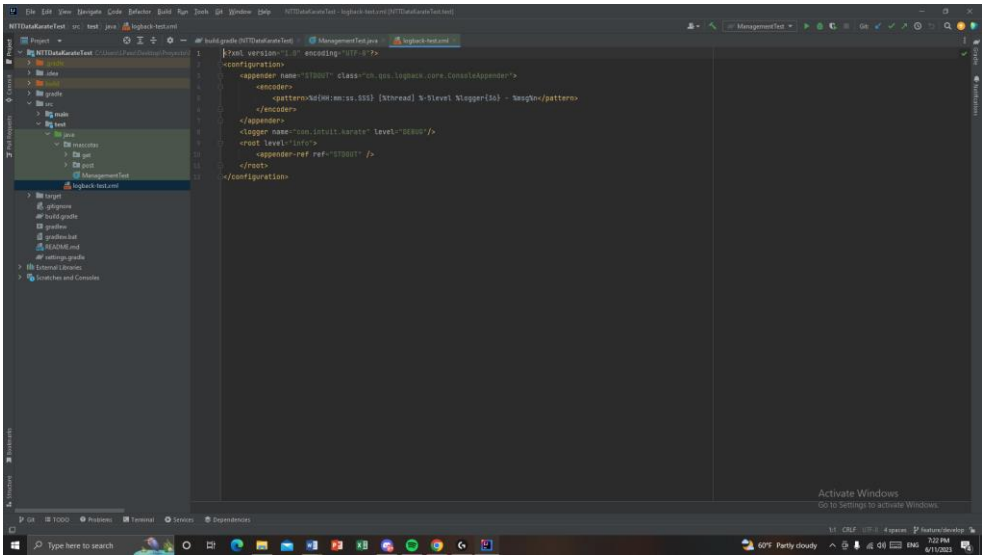
Tenemos el paquete mascotas que a su vez contiene los métodos get y post que vamos a utilizar para probar la API.



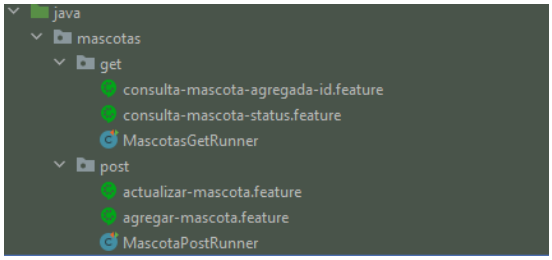
La clase managmentTest tiene la programación de 4 hilos para ejecutar los features en paralelo y adicionalmente tiene el desarrollo para mejorar los reportes de karate con la dependencia de cucumber que colocamos en el build.gradle



En el logback-test.xml se realiza la configuración de lo que sse va a mostrar en los logs de la terminal cuando se ejecute los test en paralelo



Dentro de nuestros paquetes get y post tenemos los features correspondientes a cada prueba de funcionalidad solicitada y adicionalmente se implementa una clase runner la cual sirve para correr los features dentro de ese paquete



Los features y clase runner desarrollados dentro del paquete get son los siguientes

```
Feature: Consulta de Mascota por Identificador

  Scenario: Consulta de Mascota por ID
    * call read("../post/agregar-mascota.feature@CrearNuevaMascota")
    Given url "https://petstore.swagger.io/v2/pet/" + animalId
    When method get
    Then status 200
    And match $ contains { id: "#(animalId)" }
```

```
Feature: Consulta de Mascota por Estado

  Scenario: Consulta de Mascota por Estado
    * call read("../post/actualizar-mascota.feature@ActualizarEstadoMascota")
    Given url "https://petstore.swagger.io/v2/pet/findByStatus?status=" + estadoId
    When method get
    Then status 200
    And match $ contains deep { id: "#(idAnimal)" }
    And match $ contains deep { status: "#(estadoId)" }
```

```
package mascotas.get;

import com.intuit.karate.junit5.Karate;

leopass1997
public class MascotasGetRunner {

    leopass1997
    @Karate.Test
    Karate mascotaGet() { return Karate.run().relativeTo(getClass()); }

}
```

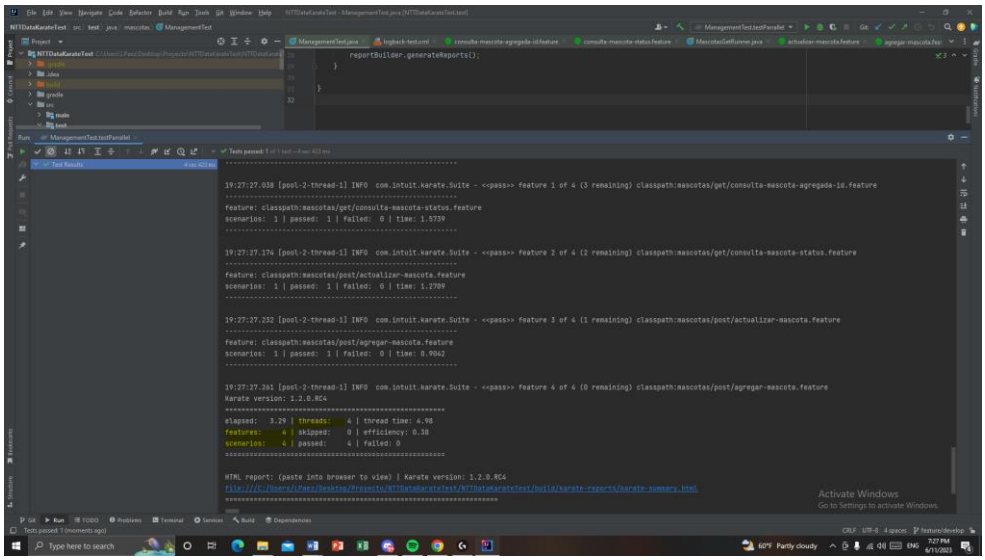
Los features y clase runner desarrollados dentro del paquete post son los siguientes

```
Feature: Actualizar Mascota

  @ActualizarEstadoMascota
  Scenario Outline: Actualizar el nombre de la mascota y el estatus de la mascota
    * call read("../post/agregar-mascota.feature@CrearNuevaMascota")
    Given url "https://petstore.swagger.io/v2/pet/" + animalId
    And form field name = <name>
    And form field status = <estado>
    When method post
    Then status 200
    And  def estadoId = <estado>
    And  def idAnimal = animalId

    Examples:
      | name            | estado |
      | "Lagarto Feliz" | "sold" |
```

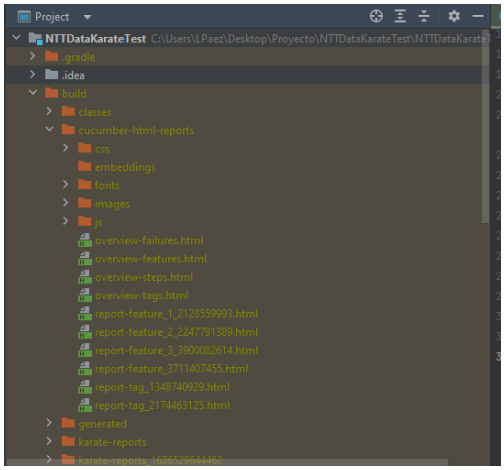
```
Feature: Agregar Mascota

  @CrearNuevaMascota
  Scenario: Agregar una nueva mascota a la tienda
    Given url "https://petstore.swagger.io/v2/pet"
    And request { "id": 777, "category": { "id": 777, "name": "reptile" }, "name": "Lacoste", "photoUrls": [ "https://definicion.de/lagarto/" ], "tags": [ { "i
    When method post
    Then status 200
    And def animalId = $.id
```

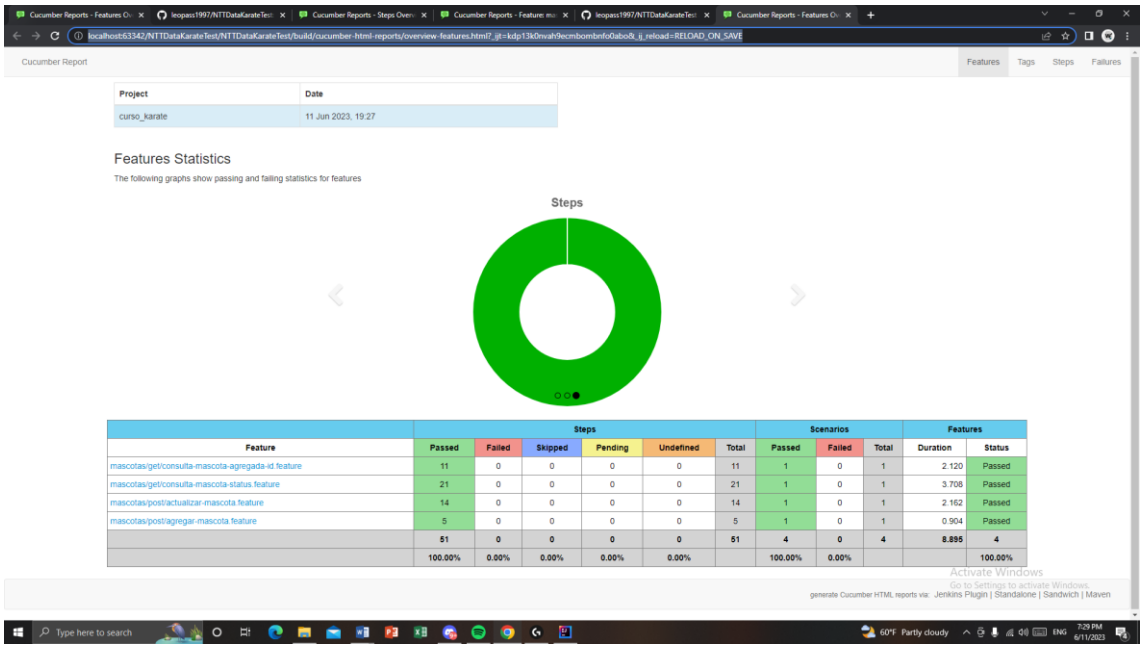
```java
package mascotas.post;

import com.intuit.karate.junit5.Karate;

leopass1997
public class MascotaPostRunner {

    leopass1997
    @Karate.Test
    Karate mascotaPost() { return Karate.run().relativeTo(getClass()); }
}
```

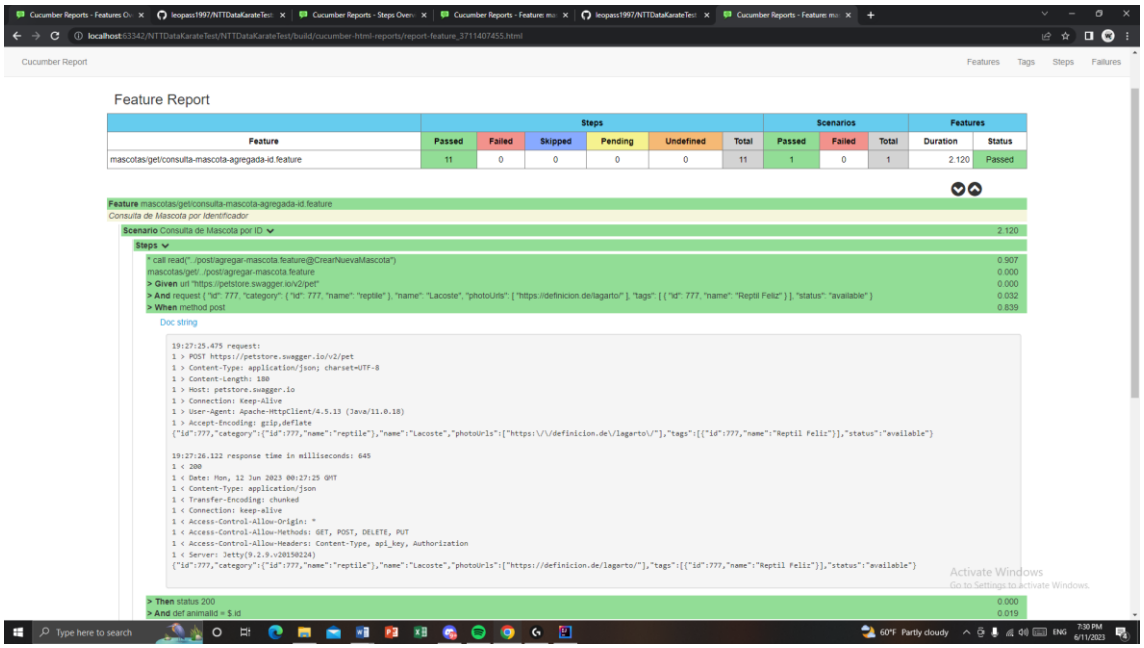
Ejecucion de features en paralelo con la ayuda de la clase managmentTest. Validamos la ejecución de todos los features y la cantidad de hilos ejecutados según lo desarrollado en la clase ya mencionada.



Si la ejecución fue exitosa en el árbol podremos ir a consultar nuestro reporte de karate mejorado con cucumber







Conclusiones

- Los 4 casos de prueba solicitados para la prueba del api resultaron exitosos. La evidencia de cada uno de estos se puede revisar dentro del log de la terminal o desde nuestro reporte de karate generado.
- Aunque no se realizaron pruebas de interfaz o frontend, el framework Karate resulta ser igual de potente que serenity bdd a la hora de realizar validaciones de resultados esperados por una acción. Y aunque no hubo la necesidad de incorporar código java para brindar alguna funcionalidad a los features, de igual manera Karate permite integrar código java a los features para mejorar el desarrollo de las pruebas.
- Finalmente, tanto sertenity bdd y karate usan el lenguaje alto nivel llamado gherkin. Siendo fácil de entender el paso a paso de cada una de estas pruebas y su objetivo a cumplir cuando finalice la ejecución del escenario.