UNIVERSIDAD PRIVADA-DE-TACNA

INGENIERIA DE SISTEMAS

TITULO:

TRABAJO ENCARGADO 01: FRAMEWORKS DE PRUEBAS

CURSO:

CALIDAD Y PRUEBAS DE SOFTWARE

DOCENTE(ING):

Patrick Cuadros Quiroga

Alumnos:

Cutipa Machaca Arnold Felix	(2019064040)
Poma Chura Jhon Romario	(2019064022)
Valdez Apaza Rafael Jesus	(2019063317)
Sagua Ramos, Gustavo Alonso	(2018062254)
Romero Roque, Angelica Beatriz	(2019063327)
Yucra Mamani, Vanessa	(2019063635)

Tacna – Perú

2021

Comparativa de Frameworks de Pruebas de APIs

October 2021

1. Resumen

En el presente documento se hablará de un tema importante como son los Frameworks de pruebas, abordaremos los framewors como Karate DSL y JMeter, se hablará qué son, ventajas y como aplicarlos. También se dará una explicación básica Testing de Software, cada principio y ejemplos prácticos.

En las pruebas de software, la automatización nos permite una ejecución de pruebas más rápida y constante en comparación con las pruebas manuales. En la actualidad, para realizar pruebas automatizadas funcionales se cuenta con herramientas como Selenium, TestNG, Sikuli, Cypress, Watir, TestCafe, WebdriverIO, entre otros. En cada una de estas contamos con opciones para realizar comparaciones entre resultados obtenidos y resultados esperados, y realizar reportes.

Un framework se encarga de encapsular funciones genéricas y comunes del software que se requiera, a la vez que estandariza la generación de código. Esto agiliza el alcance de requerimientos para así disminuir el tiempo que se dedica a tareas de bajo nivel requeridas para el proyecto.

2. Abstract

In this document we will talk about an important topic such as test Frameworks, we will address framewors such as Karate DSL and JMeter, what they are, advantages and how to apply them will be discussed. A basic explanation of Software Testing, each principle and practical examples will also be given.

In software testing, automation enables faster and more consistent test execution compared to manual testing. Currently, to perform automated functional tests, there are tools such as Selenium, TestNG, Sikuli, Cypress, Watir, TestCafe, WebdriverIO, among others. In each of these we have options to make comparisons between obtained results and expected results, and to make reports.

A framework is in charge of encapsulating generic and common functions of the software that is required, at the same time that it standardizes the generation of code. This streamlines the scope of requirements to decrease the time spent on low-level tasks required for the project.

3. Introducción

El Testing de Software es la realización de pruebas sobre el mismo, con el fin de obtener información acerca de su calidad.

Este proceso tiene varios objetivos bien definidos, como son:

- Encontrar defectos o bugs con el finde ser subsanados.

- Eso nos va a permitir aumentar la confianza en el nivel de calidad del software. Aunque por muchas pruebas que realicemos no vamos a poder afirmar al 100 porciento que ese software no contiene errores, a base de hacer pruebas y subsanar errores, aumentamos nuestra confianza en el mismo.
- Esta calidad nos va a facilitar la toma de decisiones. Por ejemplo, si tenemos una entrega que debe pasar a producción, si tiene muchos defectos y la calidad no es buena, a lo mejor debemos posponerlo y subsanar primero los errores, pero si la calidad es buena y no tiene errores, lo pasamos a producción.
- Evitar la aparición de nuevos efectos, ya que si aumento la calidad de mi software y hacemos las cosas bien, es menos probable que aparezcan nuevos defectos que si hago las cosas de cualquier forma y con una calidad baja.

4. Desarrollo

KARATE DSL

Karate es una herramienta de código abierto que combina la automatización de pruebas de API, mocks, pruebas de rendimiento e incluso la automatización de la interfaz de usuario en un marco único y unificado. Utiliza la sintaxis Gherkin, popularizada por Cucumber, como lenguaje neutro y de fácil comprensión incluso para las personas no técnicas, que se enfoca en el desarrollo guiado por comportamiento (BDD).

Funcionalidades principales de Karate:

- Setup inicial sencillo, rápido y directo.
- BDD unificado desde un mismo fichero. No es necesario definir los steps en otras ubicaciones.
- Pruebas simples, concisas, legibles y fáciles de mantener.
- Posibilidad de utilizar clases Java para utilidades complejas o funciones de ayuda que facilita su depuración y mantenibilidad.
- No se requieren conocimientos de programación avanzados. Acciones habituales sobre APIs implementadas mediante lenguaje natural.
- Posibilidad de ejecutar pruebas en paralelo.
- Reutilización de features pudiendo ser usadas como precondiciones o funciones de otros pruebas.
- Implementación sencilla de asertos complejos.
- Soporte nativo para aserciones sobre JSON y XML.
- Permite lectura de ficheros CSV (Data Driven Testing)
- Motor de JavaScript integrado.
- Importación de archivos csv o json con información. En cucumber, las variables están estáticas en una tabla.
- Documentación y ejemplos completos.

– Soporte websockets.Informes HTML completos y muy visuales.

¿Dónde lo puedes aplicar?

Casos de uso habituales:
Karate es una buena opción cuando se desee disponer de una suite automatizada de pruebas sobre servicios REST y no se precisen acciones excesivamente complejas para realizar dichas comprobaciones. Es decir, por su sencillez y facilidad de uso, resulta ideal para disponer rápidamente de una batería completa de pruebas sin apenas conocimientos de programación y utilizando un lenguaje común que puede ser comprendido tanto por la parte técnica como la de negocio. Esto hace que las pruebas sirvan a su vez como documentación funcional del API sobre el que se lanzan las pruebas (concepto de Living documentation).

Dónde no es recomendable:
En casos donde para la correcta ejecución de las pruebas se requiera un setup complejo de datos o del entorno puede no resultar una buena opción. No tanto porque desde Karate no se puedan realizar dichas acciones, pero a costa de perder su potencial de claridad y sencillez.

¿Cómo funciona?

Entorno utilizado, prerrequisitos Prerrequisitos:

– Java 8+

– Maven o Gradle

– Eclipse o IntelliJ IDE

Setup inicial

Para un proyecto maven se necesita añadir las siguientes dependencias:

```
<dependency>
<groupId>com.intuit.karate</groupId>
<artifactId>karate-apache</artifactId>
<version>0.9.4</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>com.intuit.karate</groupId>
<artifactId>karate-junit4</artifactId>
<version>0.9.4</version>
<scope>test</scope>
</dependency>
```

Figura 1: Setup Inicial

Para ir viendo las principales funcionalidades de Karate se irán recorriendo los casos de uso más habituales:

GET

La prueba consiste en obtener un listado de los usuarios del sistema y comprobar que la respuesta es correcta.

```
Feature: Get Tests on reqres.in
Scenario: Get users list
Given url 'https://reqres.in' + '/api/users' + '?page=2'
When method GET
Then status 200
```

Figura 2: Get

Como se puede observar, existen definiciones constantes como por ejemplo las url que se van a volver a usar en más de una ocasión y que se pueden sacar y definirlas para su uso en toda la feature en el apartado “Background”. Para ello se usa la palabra clave en karate “def”.

```
Feature: Get Tests on reqres.in
Background:
* def urlBase = 'https://reqres.in'
* def usersPath = '/api/users'
Scenario: Get users list
Given url urlBase + usersPath + '?page=2'
When method GET
Then status 200
```

Figura 3:

Por ejemplo conocer si uno de los usuarios de la lista se llama “Emma” y que los identificadores de todos los usuarios no sean “nulos”.Se puede ampliar esta prueba añadiendo además comprobaciones adicionales sobre el contenido de la respuesta obtenida.

```
Feature: Get Tests on reqres.in
Background:
* def urlBase = 'https://reqres.in'
* def usersPath = '/api/users'
Scenario: Get users list and check value in field
Given url urlBase + usersPath
When method GET
Then status 200
And match $..first_name contains 'Emma'
And match $..id contains '#notnull'
```

Figura 4:

POST

Este test verifica la creación de un usuario, por lo que debemos añadir un “BODY” a la petición. En este caso, veremos cómo conseguirlo haciendo uso además del concepto de “Scenario Outline” para parametrizar tanto la petición como los resultados esperados en cada caso.

Esta forma de realizar pruebas resulta muy útil cuando deseamos ejercitar una misma funcionalidad bajo diferentes casuísticas.

```
Feature: Login and register Tests on reqres.in
Background:
* def urlBase = 'https://reqres.in'
* def loginPath = '/api/login'
Scenario Outline: As a <description>, I want to get the corresponding response_code
Given url urlBase + loginPath
And request { 'email': <username> , 'password': <password> }
When method POST
* print response
Then response.status == <status_code>
```

Figura 5:

PUT

El siguiente test actualiza el valor de un atributo de un usuario y se comprueba que efectivamente se ha modificado.

```
Feature: Post/Put/Patch/Delete Tests on reqres.in
Background:
* def urlBase = 'https://reqres.in'
* def usersPath = '/api/users'
Scenario: Put user
Given url urlBase + usersPath + '/2'
And request { name: 'morpheus updated',job: 'leader updated' }
When method PUT
Then status 200
And match $.name == 'morpheus updated'
```

Figura 6:

EJECUCIÓN DE LAS PRUEBAS

Para la ejecución de los tests se define una clase Java en la que se configura su rutina así como la ubicación de los features.

```
package examples;import com.intuit.karate.KarateOptions;
import com.intuit.karate.junit4.Karate;
import org.junit.runner.RunWith;
@RunWith(Karate.class)
@KarateOptions(features = "classpath:examples", tags = "~@ignore")
class ExamplesRunner {
}
```

Figura 7:

En dicha ejecución, Karate espera que exista un archivo “karate-config.js” con las variables de configuración. En el que se pueden definir variables como el environment o funciones JavaScript reutilizables.

Para lanzar la ejecución se puede hacer de dos formas: desde nuestro IDE o por línea de comandos. Para el segundo caso se podrá ejecutar el siguiente comando Maven:

```
mvn clean test -Dtest=ExamplesRunner
```

Figura 8:

REPORTES

Cuando se ejecutan los tests se genera un informe por cada feature. El informe propuesto por Karate está en formato html y su estructura es la siguiente:

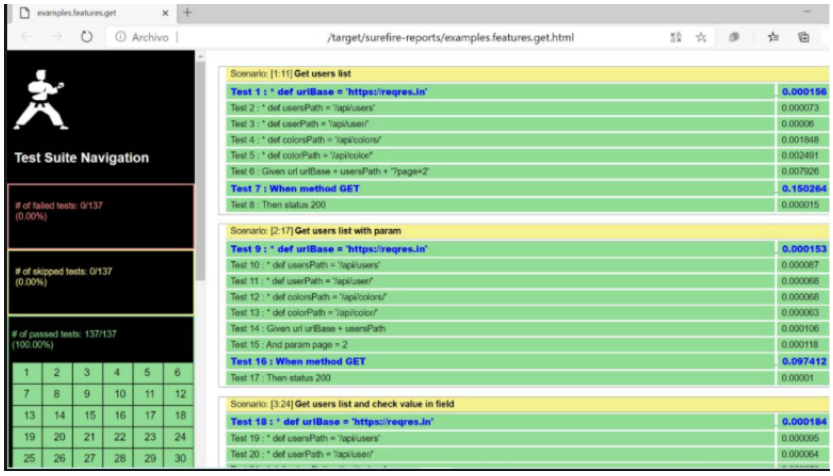

Figura 9:

JMETER

JMeter es una herramienta de testing cuyas funcionalidades se pueden resumir en tres:

- Diseñar un testplan, esto es, generar un fichero .jmx
- Ejecutar un testplan
- Ver de distintas formas los resultados de la ejecución de un testplan (vía listeners)

Para diseñar un testplan, JMeter dispone de una interfaz GUI a modo de diseñador, en la que el tester puede ir agregando componentes de manera visual, y ejecutar los componentes agregados, viendo el resultado. Una vez finalizado el diseño del testplan, la herramienta permite grabar este como un fichero .jmx.

Los datos generados por la herramienta para cada petición se procesan o bien con un tipo de componente que proporciona la interfaz GUI llamados listeners, o bien con herramientas externas. Los listeners permiten ver los resultados de una o mas ejecuciones de múltiples maneras (cada listener de una manera).

Ventajas e inconvenientes

JMeter es una herramienta ideal para realizar pruebas de rendimiento de aplicaciones web. Sus principales debilidades son:

- JMeter NO se comporta como un navegador. Esto tiene varias implicaciones importantes: por defecto no guarda ni envía cookies, no interpreta código JavaScript, ... Cualquier funcionalidad de estas debe ser implementada específicamente en el testplan

- Con JMeter el tester trabaja a nivel de protocolos: el desarrollador de un testplan ha de descender a este nivel, por lo que normalmente el tester tiene que apoyarse en herramientas adicionales durante el desarrollo de un testplan, como Firebug, HttpFox, SoapUI, Badboy, ...
- Los tipos de aplicaciones que se pueden testear con JMeter dependen de los protolos que implementen las interfaces de acceso a la aplicación. Con JMeter se pueden testear los siguientes tipos de interfaces: HTTP, HTTPS, SOAP (sobre HTTP), XML-RPC (sobre HTTP), FTP, LDAP, POP3, IMAP, SMTP, JMS, JDBC y TCP.

5. Conclusiones

Este proceso, aunque como hemos explicado es uno de los elementos más importante en el proceso de desarrollo, a día de hoy sigue siendo uno de los más olvidados por las empresas a la hora de desarrollar un proyecto de software.

Invertir en este tipo de procesos supone una garantía en la calidad del producto y que éste no se lance al mercado con bugs. Esto no quiere decir que no se produzcan defectos que subsanar en fases posteriores, pero sí que serán menos y que tendrán un impacto económico inferior.

Que el gran desafío sea presentar un proyecto de calidad y que la experiencia del usuario final sea satisfactoria.

6. Recomendaciones

- Cada caso de prueba debe definir el resultado de salida esperado que se comparará con el realmente obtenido.
- El programador debe evitar probar sus propios programas, ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas.
- Se debe inspeccionar a conciencia el resultado de cada prueba, para así, poder descubrir posibles síntomas de defectos.
- Al generar casos de prueba, se deben incluir tanto datos de entrada válidos como no válidos.
- Las pruebas deben centrarse en dos objetivos: probar si el software no hace lo que debe hacer o viceversa, es decir, si provoca efectos secundarios adversos
- No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto, dedicando pocos recursos a las pruebas siempre hay defectos.
- La experiencia parece indicar que donde hay un defecto hay otros, es decir, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto.
- Las pruebas son una tarea tanto o más creativa que el desarrollo de software. Siempre se han considerado las pruebas como una tarea destructiva y rutinaria.

7. Webgrafia

- https://apiumhub.com/es/tech-blog-barcelona/karate-framework-testeo-apis/
- https://stackshare.io/karate-dsl/alternatives
- https://www.iberpixel.com/blog/la-importancia-del-testing-en-el-desarrollo-de-aplicaciones/
- https://www.sngular.com/es/automatizacion-de-pruebas-con-karate-i/