

Introducción a Karate

Definición

Bienvenidos, en esta clase en donde se abordará el funcionamiento del framework Karate.

Karate es una herramienta de código abierto creada en Java, que fue diseñada para la automatización de pruebas API (SOAP y REST), simulaciones, pruebas de rendimiento e incluso la automatización de la interfaz de usuario en un marco unificado.

Esta herramienta contiene su propia sintaxis, que facilita el proceso de pruebas, sin necesidad de formatear o compilar, también puede ejecutar los casos de prueba en paralelo y realizar comprobaciones de archivos en formato JSON y XML.

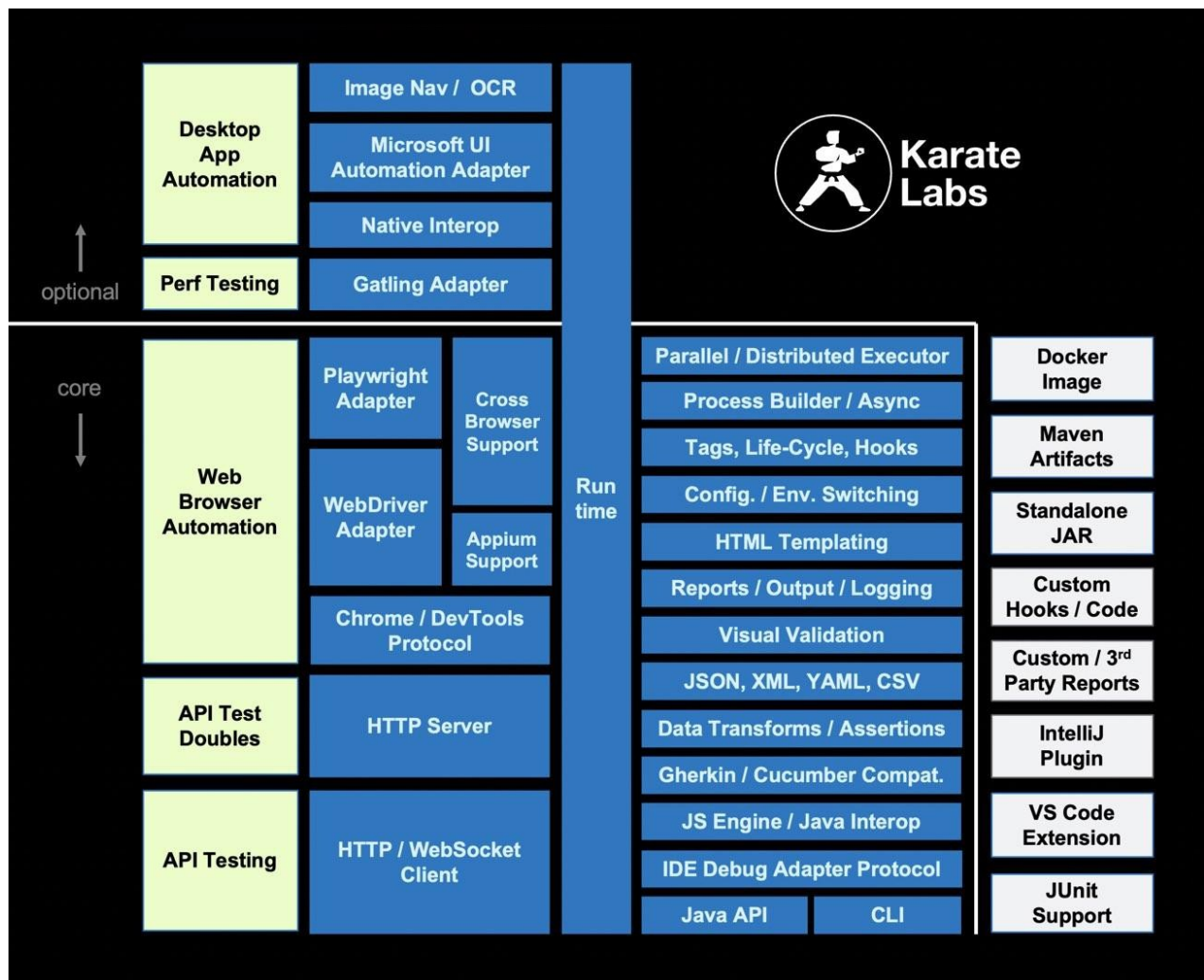


Imagen n°1: Funcionalidades y características de Karate

Debido a la simplicidad del framework, se puede integrar en varios IDE's en forma de plugins o extensiones.

Karate está diseñado bajo el enfoque BDD, también conocido como Behavior Driven Development o Desarrollo guiado por comportamiento, que se destaca por centrarse en la funcionalidad del sistema y del usuario utilizando un lenguaje común.

Conocimientos Previos

Si bien, para utilizar Karate no se necesitan conocimientos avanzados de programación en java, hay que tener en cuenta los siguientes aspectos antes de trabajar con el framework:

- Tener claro los fundamentos del Testing de software y sus conceptos asociados
- Fundamentos de las peticiones de HTTP
- Fundamentos de las estructuras JSON, XML, entre otros.
- Conocimientos de Maven o Gradle
- Conocimiento sobre reglas de negocio

Estructura

La estructura de la sintaxis de los archivos feature de Karate tienen una similitud con Gherkin, un lenguaje que también está basado en BDD, eso si, solo se pueden escribir las instrucciones en inglés, se componen de varios elementos:

- Feature: Es una descripción de alto nivel que puede abarcar múltiples escenarios.
- Scenario: Se refiere al contexto en el que se define la regla de negocio y define la estructura del "Given, When y Then".
- Given: Define el contexto inicial del sistema, es decir, asigna al sistema un estado concreto antes de la interacción en el sistema (When), estos estados pueden ser urls o rutas de acceso.
- When: Describe la acción o el evento al interactuar con el sistema, un ejemplo sería el de las peticiones HTTP.
- Then: Muestra el resultado esperado de la interacción indicada en el When.

Scenario: create and retrieve a cat

Given url 'http://myhost.com/v1/cats'

And request { name: 'Billie' }

When method **post**

Then status 201

And match response == { id: '#notnull', name: 'Billie' }

Given path **response.id**

When method **get**

Then status 200

JSON is 'native'
to the syntax

Intuitive DSL
for HTTP

Payload
assertion in
one line

Second HTTP
call using
response data

Imagen n°2: Ejemplo de sintaxis BDD en Karate

Instalación

Antes de empezar, lo primero que hay que hacer es instalar Java JDK, o también el OpenJDK, esto se debe a que Karate lo necesita para ejecutar, después, se necesita un IDE para integrarlo, hay tres opciones en particular:

- Visual Studio Code
- IntelliJ Idea
- Eclipse

También se necesita instalar ya sea Maven o Gradle, para el manejo de librerías. Karate puede ser instalado como plugin para las IDE 's anteriormente mencionadas. En esta sesión, se utilizará IntelliJ, debido a que este tiene instalado Maven y Gradle, utilizaremos este último.

Paso a Paso

Creación de proyecto base

Primero que todo, abrimos IntelliJ y creamos un proyecto utilizando el Maven Archetype para facilitar la configuración de sus propiedades. Una vez terminado, se abrirá la interfaz con el proyecto creado con el archivo build.gradle que contiene la configuración del proyecto de Karate.

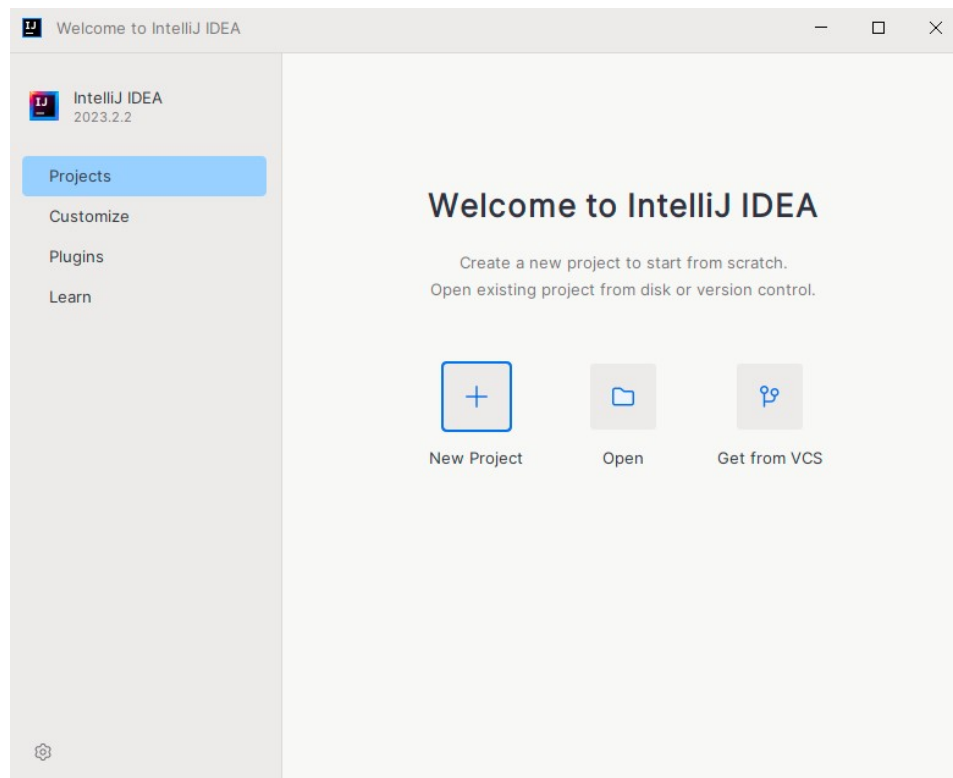


Imagen n°3: Menú inicial de IntelliJ

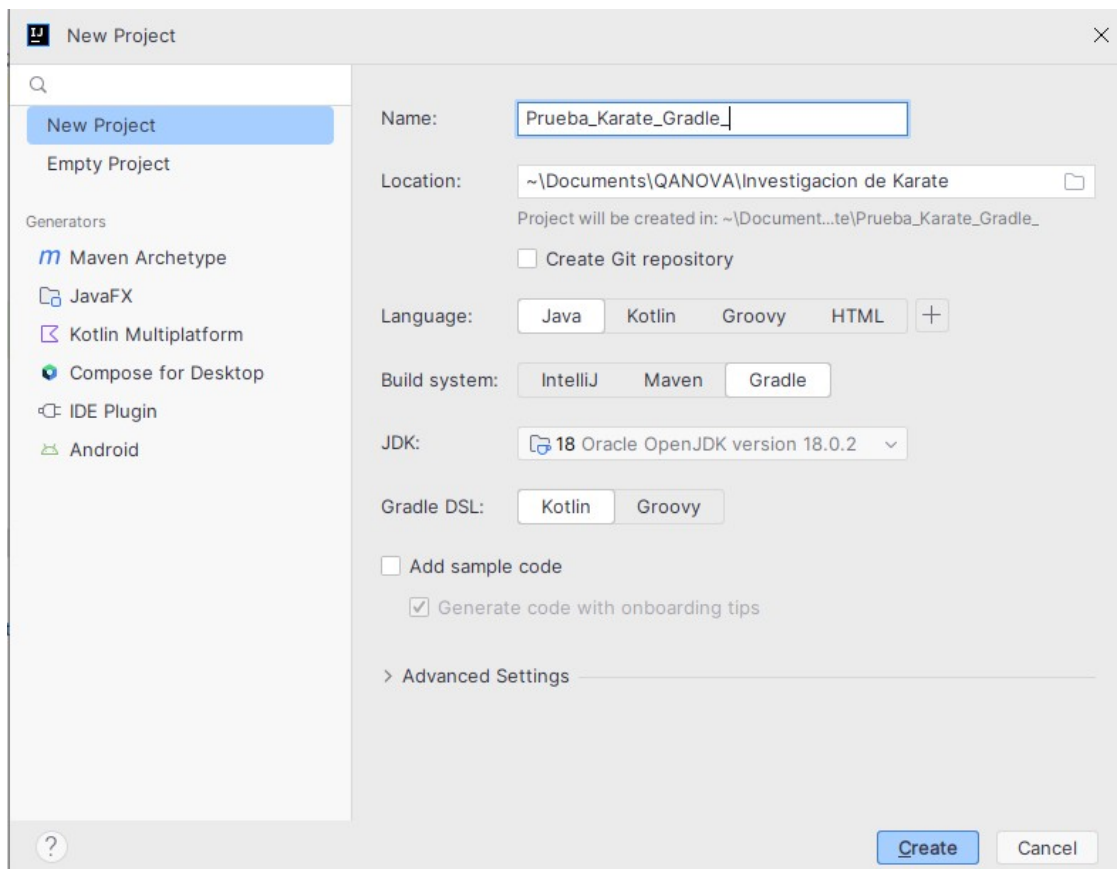


Imagen n°4: Menú de configuración de nuevo proyecto con Gradle

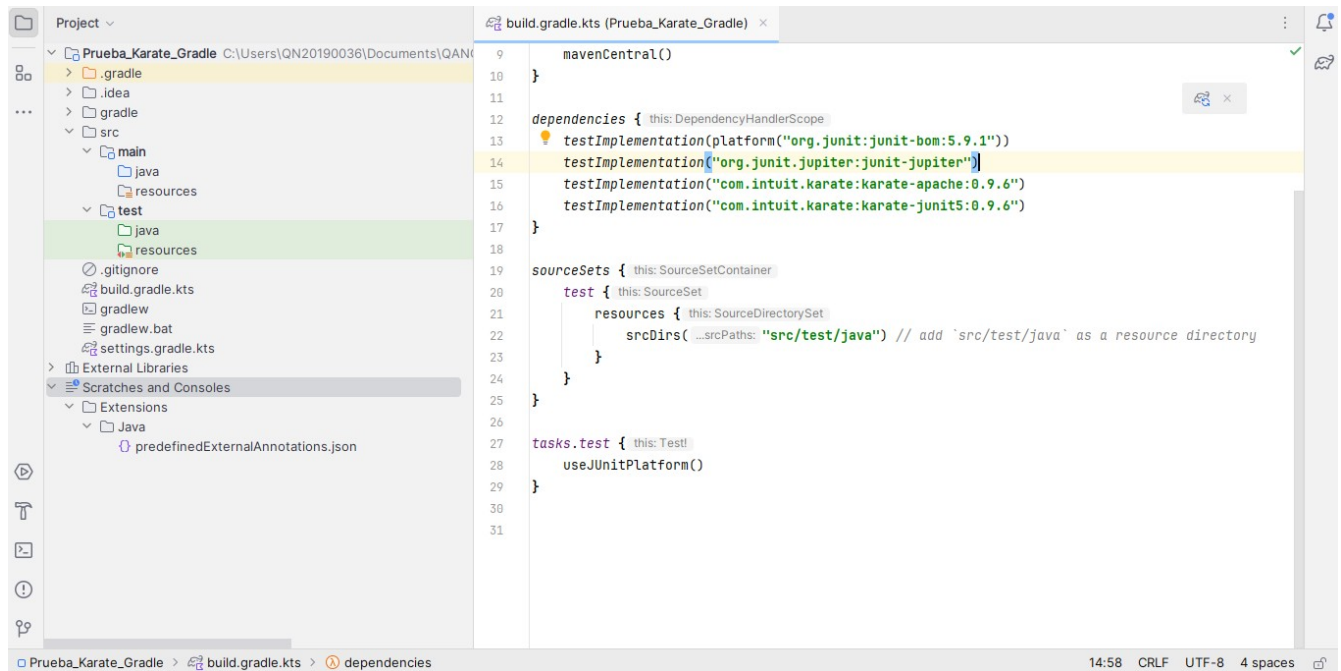


Imagen n°5: Menú del proyecto con archivo build.gradle.

Si bien, la estructura predeterminada esta estable, hay que añadir las dependencias y decirle en donde se encuentran nuestros archivos de ejecución.

En dependencies, hay que indicarle las librerías de Karate, mientras que en SourceSets, le indicamos la ruta con los archivos de ejecución Karate.

```
dependencies {  
    testImplementation(platform("org.junit:junit-bom:5.9.1"))  
    testImplementation("org.junit.jupiter:junit-jupiter")  
    testImplementation("com.intuit.karate:karate-apache:0.9.6")  
    testImplementation("com.intuit.karate:karate-junit5:1.4.0")  
}
```

Imagen n°6: Dependencias de Gradle con librerías de Karate

```
sourceSets {  
    test {  
        resources {  
            srcDirs("src/test/java")  
        }  
    }  
}
```

Imagen n°7: Ruta de archivos Karate

Elaboración nueva prueba

En esta instancia, haremos una prueba que consiste en realizar una petición GET para recolectar datos de una página llamada reqres, que se utiliza para pruebas de API, llamadas de peticiones, etc.

Primero, creamos una carpeta llamada pruebas bajo src/test/java y después, en esa carpeta, generamos un archivo Karate llamado PrimeraPrueba.feature.

Para ello, estableceremos el Feature, que tendrá como valor "First API Test" y el Scenario tendrá como valor "Run a Get API Test", después, estableceremos la url, el tipo de petición y el resultado esperado.



Imagen n°6: PrimeraPrueba.feature

Luego, lo ejecutamos y nos entregará los resultados junto con una url que nos permite verlos en detalle.

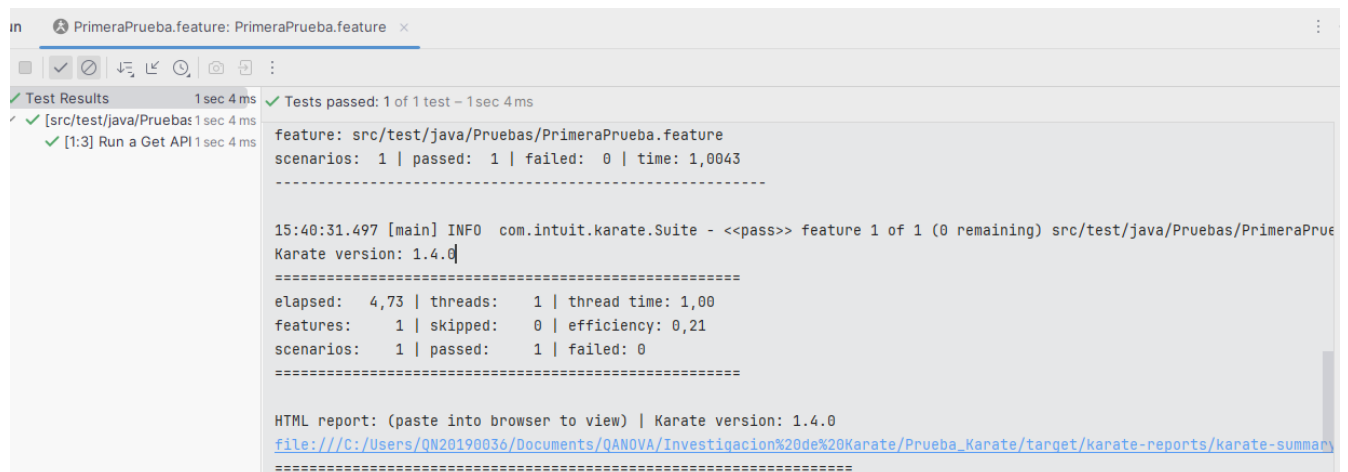


Imagen n°7: Resultado de ejecución

1

0

Scenarios
2023-09-21 03:40:30 p. m.

[1:3] Run a Get API Test

Summary | Tags | Feature: `src/test/java/Pruebas/PrimeraPrueba.feature` | First API Test # Esta prueba consiste en realizar un metodo get

Scenario: [1:3] Run a Get API Test	ms: 1004
4 Given url 'https://reqres.in/api/users?page=2'	2
15:48:29.339 karate.env system property was: null	
5 When method GET	998
15:48:29.884 request:	
1 > GET https://reqres.in/api/users?page=2	
1 > Host: reqres.in	
1 > Connection: Keep-Alive	
1 > User-Agent: Apache-HttpClient/4.5.14 (Java/21)	
1 > Accept-Encoding: gzip,deflate	
15:48:30.354 response time in milliseconds: 546	
1 < 200	
1 < Date: Thu, 21 Sep 2023 18:40:30 GMT	
1 < Content-Type: application/json; charset=utf-8	
1 < Transfer-Encoding: chunked	
1 < Connection: keep-alive	
1 < X-Powered-By: Express	
1 < Access-Control-Allow-Origin: *	
1 < Etag: W/"486-ut6vzocuidvyMf8arZpMpJ6ZRDu"	
1 < Via: 1.1 vegur	
1 < Cache-Control: max-age=14400	
1 < CF-Cache-Status: HIT	
1 < Age: 423	
1 < Report-To: {"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v3? s=GIL36chQVaa9epMx2f1latHa3EpJfhiaWQ1nShGuX2B0sh2iy1XatX7R0YvZfEnohberF7zP7z5Op8YVj7EoIuFkAoNV1eGuVrnZcqo67xzB0CdrCVHSP68udh10Q%3D%3D"}], "group": "cf-nel", "max_age": 604800}	
1 < NEL: {"success_fraction": 0, "report_to": "cf-nel", "max_age": 604800}	
1 < Vary: Accept-Encoding	
1 < Server: cloudflare	
1 < CF-RAY: 80a46fdc9e129fb-5CL	
{ "page": 2, "per_page": 6, "total": 12, "total_pages": 2, "data": [{ "id": 7, "email": "michael.lawson@reqres.in", "first_name": "Michael", "last_name": "Lawson", "avatar": "https://reqres.in/img/faces/7-image.jpg" }, { "id": 8, "email": "lindsay.ferguson@reqres.in", "first_name": "Lindsay", "last_name": "Ferguson", "avatar": "https://reqres.in/img/faces/8-image.jpg" }, { "id": 9, "email": "tobias.funk@reqres.in", "first_name": "Tobias", "last_name": "Funk", "avatar": "https://reqres.in/img/faces/9-image.jpg" }, { "id": 10, "email": "byron.fields@reqres.in", "first_name": "Byron", "last_name": "Fields", "avatar": "https://reqres.in/img/faces/10-image.jpg" }, { "id": 11, "email": "george.edwards@reqres.in", "first_name": "George", "last_name": "Edwards", "avatar": "https://reqres.in/img/faces/11-image.jpg" }, { "id": 12, "email": "rachel.howell@reqres.in", "first_name": "Rachel", "last_name": "Howell", "avatar": "https://reqres.in/img/faces/12-image.jpg" }], "support": { "url": "https://reqres.in/#support-heading", "text": "To keep ReqRes free, contributions towards server costs are appreciated!" } }	
6 Then status 200	4

Imagen n°8: Resultados Web de Karate

Una vez ejecutado, lo siguiente es elaborar una clase llamada PruebaRunner, que consiste en un archivo .java que se utilizan para realizar ejecuciones de una o varias pruebas, los resultados se mostrarán de forma similar al ejecutar los archivos .feature por separado.

pom.xml (Prueba_Karate)
PrimeraPrueba.feature
PruebaRunner.java

```

1 package Pruebas;
2 import com.intuit.karate.junit5.Karate;
3 public class PruebaRunner {
4     @Karate.Test
5     Karate testGET() {
6         return Karate.run( ...paths: "PrimeraPrueba" ).relativeTo(getClass());
7     }
8 }
9

```

Imagen n°9: PruebaRunner del Get

Ejemplos

A continuación, veremos distintos casos en los que se harán pruebas para los distintos tipos de peticiones HTTP con respuestas satisfactorias (200-299)


Petición POST

En esta sección, veremos un caso de uso en donde se necesite subir datos mediante una petición POST.

```
1 >> Feature: API Post Test
2
3 >> Scenario: Post user data
4   Given url 'https://reqres.in' + '/api/users'
5   And request {'name': 'manuel', 'job': 'chef'}
6   When method POST
7   Then status 201
```

Imagen n°10: Prueba POST

Aquí, indicamos al Given el directorio con la dirección, después, se indican los parámetros a procesar, y por ultimo, indicamos la petición y el resultado.



1

0

Features

2023-09-21 05:54:02 p. m.

Tags | Timeline

Feature	Title	Passed	Failed	Scenarios	Time (ms)
src/test/java/Pruebas/PruebaPost.feature	API Post Test	1	0	1	2025

Imagen n°11: Resultados Web de Prueba Post

Petición PUT

En esta sección, revisaremos un caso de uso con la petición PUT, que se encarga de actualizar datos, si bien se asemeja a POST debido a que ambos tienen la capacidad de enviar datos al servidor, el método POST se le puede llamar una o varias veces de forma sucesiva, por lo que siempre tiene el mismo efecto (en otros términos, es idempotente), en cambio, al enviar peticiones POST idénticas pueden tener efectos adicionales, como enviar una orden varias veces.

```
Feature: API Put Test
```

```
Scenario: Put user data
```

```
Given url 'https://reqres.in/api/users/2'
```

```
And request {"name": "steve", "job": "Store Manager"}
```

```
When method PUT
```

```
Then status 200
```


```
And print response
```

Imagen n°12: Prueba PUT

Esta petición actualiza los datos de la segunda fila de usuarios para reemplazar los datos existentes con los nuestros.

```
10:56:36.464 [main] INFO com.intuit.karate - [print] {
  "name": "steve",
  "job": "Store Manager",
  "updatedAt": "2023-09-25T13:56:33.764Z"
}
```

Imagen n°13: Resultados prueba PUT



1

0

Scenarios
2023-09-25 05:19:25 p. m.
[1:3] Put user data

Summary | Tags | Feature: `src/test/java/Pruebas/PruebaPut.feature` | API Put Test

Scenario: [1:3] Put user data		ms: 2108
4	Given url 'https://reqres.in/api/users/2'	1
5	And request {"name": "steve", "job": "Store Manager"}	65
6	When method PUT	2013
7	Then status 200	10
8	And print response	18

```

17:19:25.588 [print] {
  "name": "steve",
  "job": "Store Manager",
  "updatedAt": "2023-09-25T20:19:24.529Z"
}

```

Imagen n°14: Resultados Web Prueba PUT

Los resultados muestran que el registro ha sido actualizado e indica la fecha y hora en la que se concretó.

Petición DELETE

Esta petición se encarga de eliminar datos específicos del servidor, en este caso, borra los datos de la segunda fila.


```

Feature: API Delete Test

Scenario: Delete user data
  Given url 'https://reqres.in/api/users/2'
  When method DELETE
  Then status 204
  And print responseStatus

```

Imagen n°15: Prueba Delete



1

0

Scenarios

2023-09-25 05:06:17 p. m.

[1:3] Delete user data

Summary | Tags | Feature: `src/test/java/Pruebas/PruebaDelete.feature` | API Delete Test

Scenario: [1:3] Delete user data		ms: 2026
4	Given url 'https://reqres.in/api/users/2'	1
5	When method DELETE	2004
6	Then status 204	9
7	And print responseStatus	12
17:06:17.829 [print] 204		

Imagen n°16: Resultados Web de Prueba Delete