

Santiago Ramirez Arenas

A continuación se adjuntan las capturas de pantalla de las peticiones realizadas desde Postman, en donde se evidencia la URL a la que se hace la petición, el estatus de la petición y el body enviado/recibido según corresponda.

## REPORTE EJECUCIÓN PRUEBAS

Imagen 1: Petición Get de la pagina de Jinkan que retorna información de un anime por id

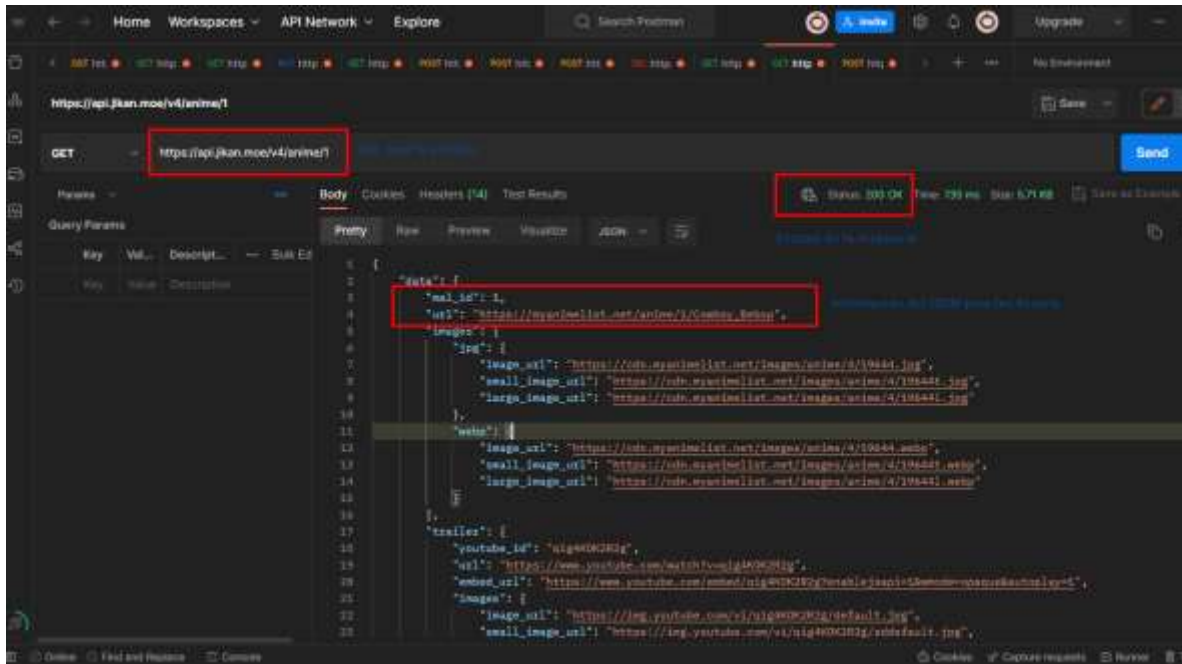


Imagen 2: Petición Post de la pagina de Placeholder para ingresar un registro de una foto

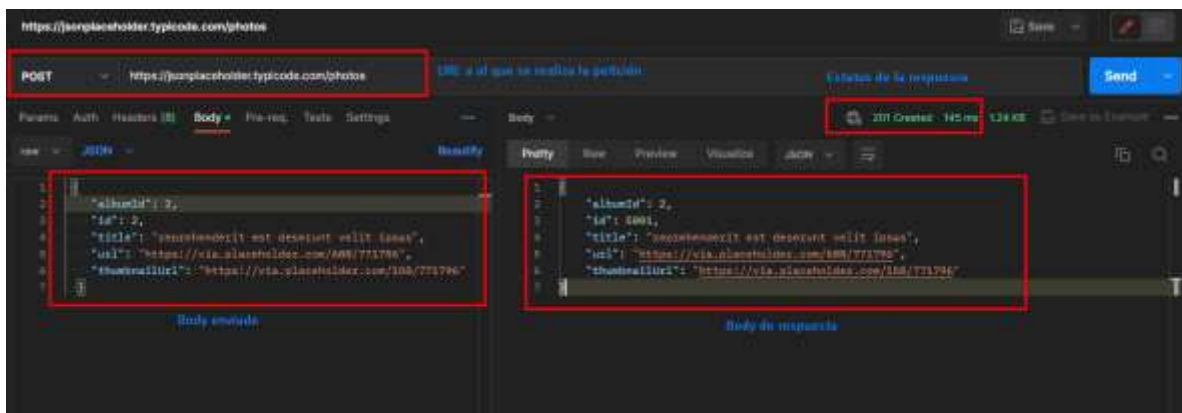


Imagen 3: Petición Put de la pagina de Regress para actualizar la información de un usuario

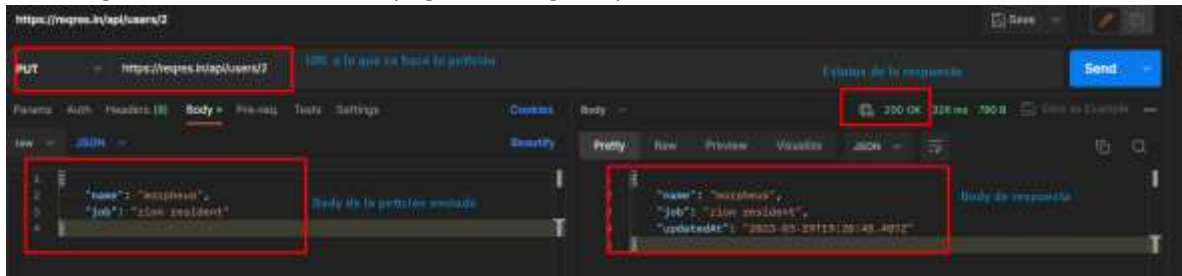
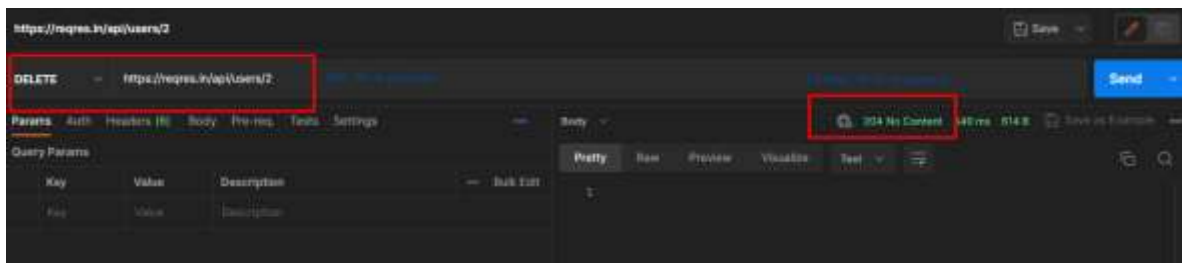


Imagen 4: Petición Delete de la pagina de Regress para eliminar la información de un usuario



A continuación, se mostrará un resumen del reporte generado por Serenity tras la ejecución de las pruebas ejecutadas de forma automatizada

Imagen 5: Resumen del reporte proporcionado por SerenityBDD

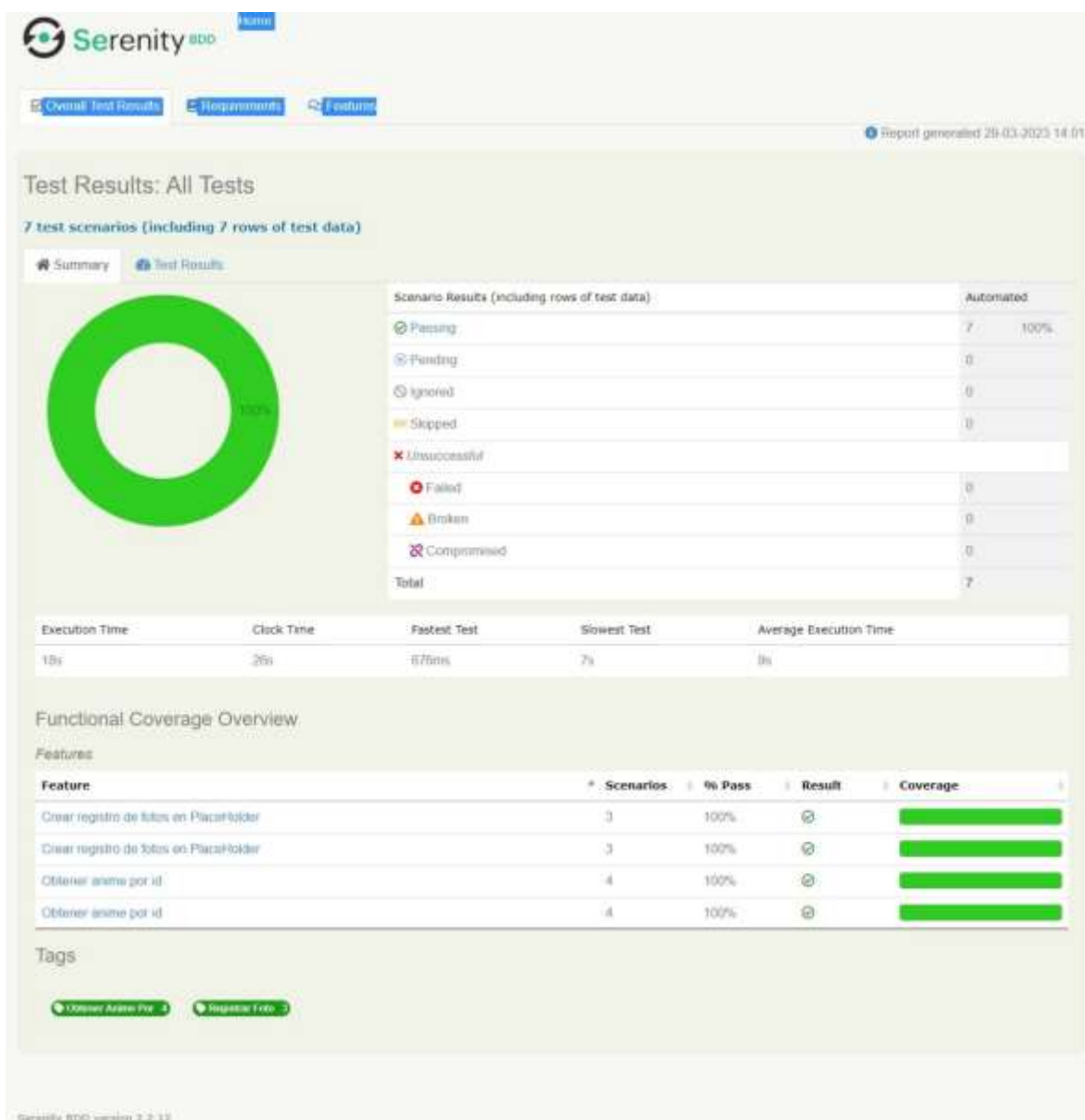


Imagen 6: evidencia ejecución de los steps de cada escenario para el servicio GET Anime Jikan

**Serenity BDD** Home > Obtener Anime Por Id > seleccion anime por id

Overall Test Results Requirements Features Report generated 28-03-2025 14:30

Obtener Anime Por Id **Obtener Anime Por (Tag)**

**seleccion anime por id**

**Call**

**Scenario Outline**

Given que el usuario esta usando la API de Anime Jikan  
 When se envia una peticion para obtener un anime por su <id>  
 Then se deberia observar el <estatushttp> junto con la <url> del anime y el <mal\_id> del anime

**Examples:**

#	Id	Estatus Http	Url	Mal
1	"1"	200	"https://myanimelist.net/anime/1/Cowboy_Bebop"	1
2	"6"	200	"https://myanimelist.net/anime/6/Trigun"	6
3	"10000"	404	"Not found"	1
4	"A"	404	"Not found"	1

Steps	Outcome	⌚
Example: {id="1", estatusHttp=200, url="https://myanimelist.net/anime/1/Cowboy_Bebop", mal_id=1}	SUCCESS	7.19s
Given que el usuario esta usando la API de Anime Jikan	SUCCESS	0.1s
When se envia una peticion para obtener un anime por su "1"	SUCCESS	5.85s
Then se deberia observar el 200 junto con la "https://myanimelist.net/anime/1/Cowboy_Bebop" del anime y el 1 del anime	SUCCESS	1.09s
Example: {id="6", estatusHttp=200, url="https://myanimelist.net/anime/6/Trigun", mal_id=6}	SUCCESS	0.72s
Given que el usuario esta usando la API de Anime Jikan	SUCCESS	0.01s
When se envia una peticion para obtener un anime por su "6"	SUCCESS	0.65s
Then se deberia observar el 200 junto con la "https://myanimelist.net/anime/6/Trigun" del anime y el 6 del anime	SUCCESS	0.05s
Example: {id="10000", estatusHttp=404, url="Not found", mal_id=1}	SUCCESS	0.68s
Given que el usuario esta usando la API de Anime Jikan	SUCCESS	0.01s
When se envia una peticion para obtener un anime por su "10000"	SUCCESS	0.69s
Then se deberia observar el 404 junto con la "Not found" del anime y el 1 del anime	SUCCESS	0s
Example: {id="A", estatusHttp=404, url="Not found", mal_id=1}	SUCCESS	0.68s
Given que el usuario esta usando la API de Anime Jikan	SUCCESS	0.01s
When se envia una peticion para obtener un anime por su "A"	SUCCESS	0.65s
Then se deberia observar el 404 junto con la "Not found" del anime y el 1 del anime	SUCCESS	0s
	SUCCESS	9.56s

Serenity BDD version 2.2.8

Imagen 7: evidencia ejecución de los steps de cada escenario para el servicio POST crear registro de foto

**Serenity BDD** Home > Crear Registro De Fotos En Placeholder > Crear registro de fotos en Placeholder

Overall Test Results Requirements Features Report generated 29-03-2023 14:31

Crear Registro De Fotos En Placeholder Registrar Foto (Tag)

✓ Crear registro de fotos en Placeholder

Call

### Scenario Outline

Given que el usuario ingresa a la API de Placeholder en la sección de Photos  
 When el usuario ingresa la información de <albumId>, <id>, <title>, <url> y <thumbnailUrl>  
 Then el usuario debe ver el registro de la foto con los datos ingresados y un estatus <code>

Examples:


#	Album	Id	Title	Url	Thumbnail Url	Code
1	1	1	"Titulo"	" Prueba"	"Prueba1URL"	201
2	2	2	"Titulo2"	" Prueba2"	"Prueba2URL"	201
3	3	3	"Titulo3"	" Prueba3"	"Prueba3URL"	201

Steps	Outcome	Time
Example: {albumId=1, id=1, title="Titulo", url=" Prueba", thumbnailUrl="Prueba1URL", code=201}	SUCCESS	3.94s
Given que el usuario ingresa a la API de Placeholder en la sección de Photos	SUCCESS	0.1s
When el usuario ingresa la información de 1, 1, "Titulo", " Prueba" y "Prueba1URL"	SUCCESS	5.39s
Then el usuario debe ver el registro de la foto con los datos ingresados y un estatus 201	SUCCESS	0.32s
Example: {albumId=2, id=2, title="Titulo2", url=" Prueba2", thumbnailUrl="Prueba2URL", code=201}	SUCCESS	0.87s
Given que el usuario ingresa a la API de Placeholder en la sección de Photos	SUCCESS	0.02s
When el usuario ingresa la información de 2, 2, "Titulo2", " Prueba2" y "Prueba2URL"	SUCCESS	0.82s
Then el usuario debe ver el registro de la foto con los datos ingresados y un estatus 201	SUCCESS	0.01s
Example: {albumId=3, id=3, title="Titulo3", url=" Prueba3", thumbnailUrl="Prueba3URL", code=201}	SUCCESS	1.41s
Given que el usuario ingresa a la API de Placeholder en la sección de Photos	SUCCESS	0.01s
When el usuario ingresa la información de 3, 3, "Titulo3", " Prueba3" y "Prueba3URL"	SUCCESS	1.38s
Then el usuario debe ver el registro de la foto con los datos ingresados y un estatus 201	SUCCESS	0.01s
	SUCCESS	8.54s

Serenity BDD version 2.2.8

A continuación, se mostrará un resumen del reporte generado por Karate tras la ejecución de las pruebas para el servicio PUT actualizar usuarios ejecutadas de forma automatizada

Imagen 8: Resumen pruebas servicio PUT regress



1


0

Features

2023-03-29 02:50:51 p. m.

Tags   Timeline					
Feature	Title	Passed	Failed	Scenarios	Time (ms)
src/test/java/features/PutUserRegress.feature	Update User	3	0	3	4385

Imagen 9: evidencia ejecución de los steps de cada escenario para el servicio PUT actualizar usuario



3

0

Scenarios

2023-03-29 02:50:49 p. m.

[1.1-12] Actualizar usuario con diferentes nombres y trabajos


[1.2-13] Actualizar usuario con diferentes nombres y trabajos

[1.3-14] Actualizar usuario con diferentes nombres y trabajos

Summary   Tags   Feature: src/test/java/features/PutUserRegress.feature   Update User					
Scenario: [1.1-12] Actualizar usuario con diferentes nombres y trabajos					ms: 2403
4	Given url "https://regres.in/api/users/2"				210
5	And request { "name": "Alice", "job": "Developer" }				129
6	When method put				1960
7	Then status 200				0
8	And match response.name == name				103
Scenario: [1.2-13] Actualizar usuario con diferentes nombres y trabajos					ms: 302
4	Given url "https://regres.in/api/users/2"				1
5	And request { "name": "Bob", "job": "Designer" }				1
6	When method put				498
7	Then status 200				0
8	And match response.name == name				1
Scenario: [1.3-14] Actualizar usuario con diferentes nombres y trabajos					ms: 1400
4	Given url "https://regres.in/api/users/3"				1
5	And request { "name": "Charlie", "job": "Project Manager" }				1
6	When method put				1478
7	Then status 200				0
8	And match response.name == name				1

A continuación, se mostrará un resumen del reporte generado por Karate tras la ejecución de las pruebas para el servicio DELETE eliminar usuario ejecutadas de forma automatizada

Imagen 10: Resumen pruebas servicio DELETE regress



1


0

Features

2023-03-29 03:22:11 p. m.

Tags   Timeline					
Feature	Title	Passed	Failed	Scenarios	Time (ms)
src/test/java/features/DeleteUserRegress.feature	Delete User	3	0	3	2709

Imagen 8: evidencia ejecución de los steps de cada escenario para el servicio DELETE eliminar usuario

<div>  <div> <div>3</div> <div>0</div> </div> </div> <div> Karate Labs </div> <div> Scenarios </div> <div> 2023-03-29 03:22:09 p. m. </div> <div> <div>[1.1-10] Delete usuario con diferentes IDs y parámetros</div> <div>[1.2-11] Delete usuario con diferentes IDs y parámetros</div> <div>[1.3-12] Delete usuario con diferentes IDs y parámetros</div> </div>	<div>Summary   Tags   Feature: src/test/java/features/DeleteUserRegress.feature   Delete User</div>
	<div> <div>Scenario: [1.1-10] Delete usuario con diferentes IDs y parámetros</div> <div>ms: 1758</div> </div> <div> <div>4 Given url "https://regres.in/api/users/1"</div> <div>151</div> </div> <div> <div>5 When method delete</div> <div>1607</div> </div> <div> <div>6 Then status 204</div> <div>0</div> </div>
	<div> <div>Scenario: [1.2-11] Delete usuario con diferentes IDs y parámetros</div> <div>ms: 478</div> </div> <div> <div>4 Given url "https://regres.in/api/users/2"</div> <div>1</div> </div> <div> <div>5 When method delete</div> <div>476</div> </div> <div> <div>6 Then status 204</div> <div>0</div> </div>
	<div> <div>Scenario: [1.3-12] Delete usuario con diferentes IDs y parámetros</div> <div>ms: 474</div> </div> <div> <div>4 Given url "https://regres.in/api/users/3"</div> <div>1</div> </div> <div> <div>5 When method delete</div> <div>473</div> </div> <div> <div>6 Then status 204</div> <div>0</div> </div>

## Conclusiones de la actividad

En primer lugar resalto la facilidad con la que se realizan pruebas de API usando Karate, es una herramienta muy útil y sencilla de usar, podría describirse como una herramienta “LowCode” ya que no requiere un alto nivel de comprensión en lo que respecta a programación, además de que los informes que ofrece Karate son detalladas al igual que lo son los de SerenityBDD, en cambio, SerenityBDD ofrece una capa de abstracción adicional para las pruebas lo que hace que el código sea más fácil de mantener y, de igual forma sea más escalable, aunque requiera una mejor comprensión en cuanto a programación. Karate me parece una herramienta muy poderosa para cuando debemos automatizar pruebas más sencillas y contamos con poco tiempo dado que cuenta con una “facilidad” de implementación, pero cuando necesitamos algo más escalable considero que debemos recurrir a SerenityBDD.