

Pruebas de Herramientas de automatización Serenity VS Karate (Services Rest)

Proyecto: Servicios REST REQRES

Jesús Miguel Molina Mendoza



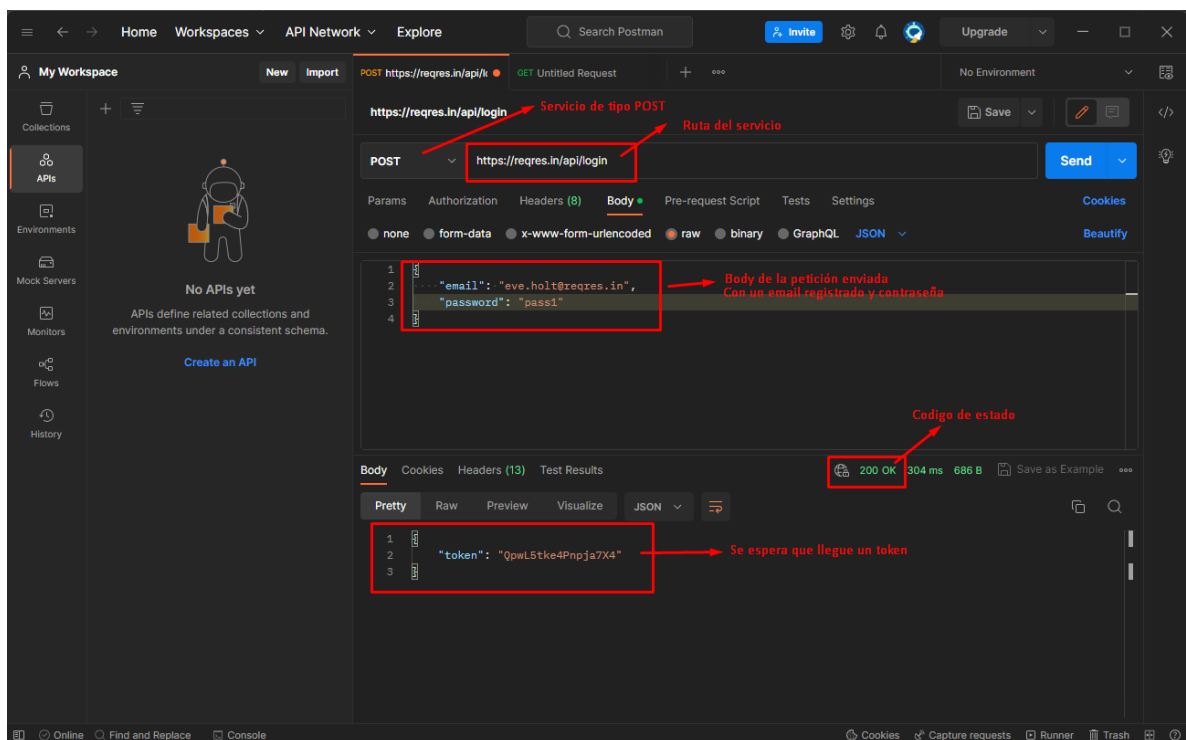
Sofka U
Training League QA
2023

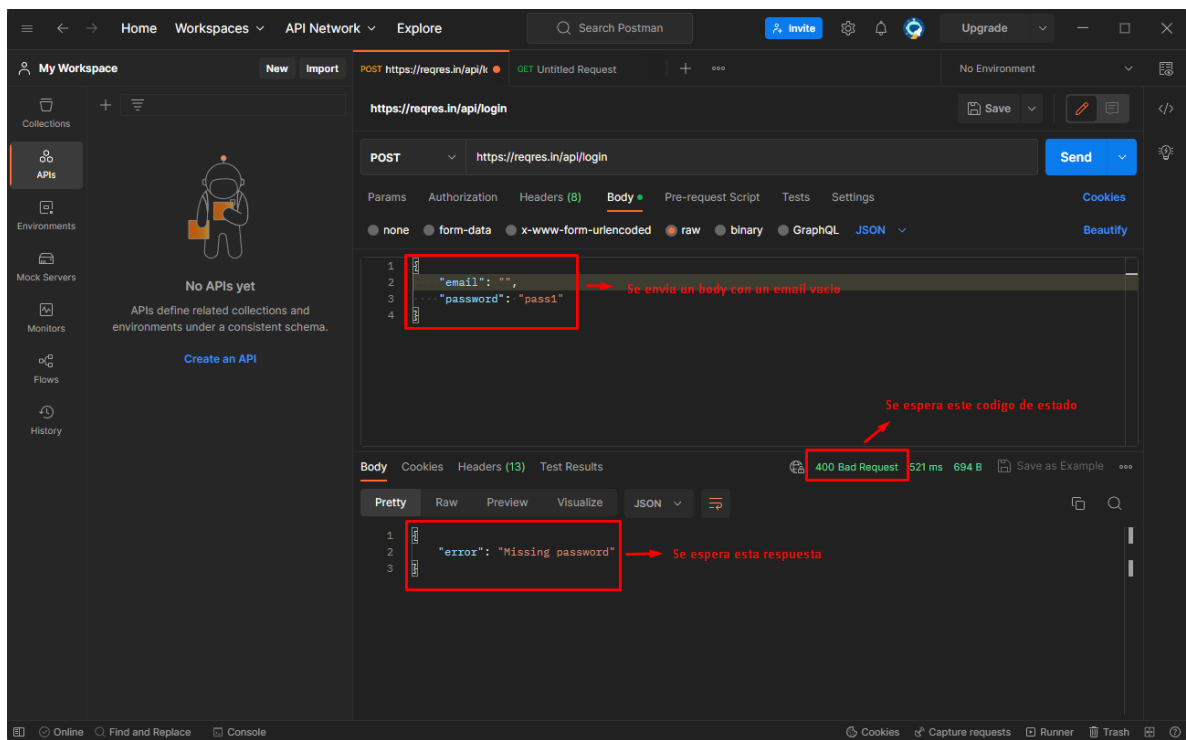
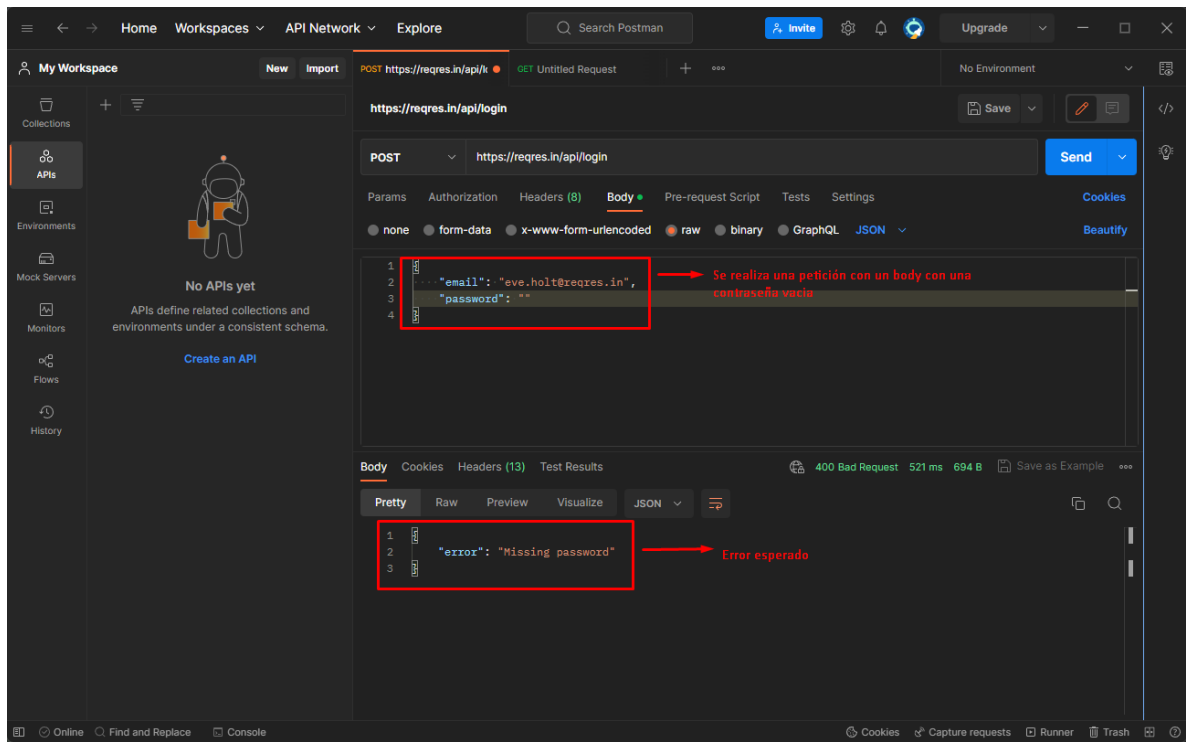
1. Método **[POST]** LOGIN: <https://regres.in/api/login>

```
Feature: login to the system
  AS regres user
  I WANT TO login to the system
  THAT I can use the system services

@Login
Scenario Outline: login to the system
  Given the user is on the login page
  When the user send a login request with the email <email> and the password <password>
  Then user then sees a response with a status code <statusCode> and a token

Examples:
  | email | password | statusCode |
  | "eve.holt@regres.in" | "pass111" | 200 |
  | "janet.weaver@regres.in" | "" | 400 |
  | "" | "pass333" | 400 |
  | "miguel.mendoza@gmail.com" | "pass444" | 400 |
```





Home Workspaces API Network Explore Search Postman Invite Upgrade

My Workspace New Import POST https://reqres.in/api/login GET Untitled Request No Environment

collections APIs Environments Mock Servers Monitors Flows History

No APIs yet
APIs define related collections and environments under a consistent schema.
Create an API

POST https://reqres.in/api/login

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "email": "jesus.miguel@gmail.com",
3   "password": "pass1"
4 }
```

Se envia un body que contiene un email no registrado

Codigo de estado esperado

400 Bad Request 308 ms 694 B Save as Example

Body Cookies Headers (13) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": "user not found"
3 }
```

Respuesta esperada

Online Find and Replace Console Cookies Capture requests Runner Trash

Reporte Serenity

Test Results: All Tests

4 test scenarios (including 4 rows of test data)

Summary

Test Results

100%

Scenario Results (including rows of test data)		Automated	
✓ Passing		4	100%
⌚ Pending		0	
⌚ Ignored		0	
⌚ Skipped		0	
✗ Unsuccessful			
✗ Failed		0	
⚠ Broken		0	
⚠ Compromised		0	
Total		4	

Execution Time	Clock Time	Fastest Test	Slowest Test	Average Execution Time
6s	6ms	533ms	4s	6s

Functional Coverage Overview

Features

Feature	Scenarios	% Pass	Result	Coverage
list a user	0	0%	⌚	
login to the system	4	100%	✓	
login to the system	4	100%	✓	

Tags

Login

2. Método [GET] SINGLE USER: <https://reqres.in/api/users/>

```
Feature: List a user
  AS reqres admin
  I WANT to generate a list of users
  SO THAT I can know who the registered users are

@ListUser
Scenario Outline: List a user
  Given the admin is on the list page
  When the user makes a request with the user id <id>
  Then the user should see a response containing the user id <id>, email <email>, name <name>, last name <lastName> and status code <statusCode>

Examples:
  | id | email                | name   | lastName | statusCode |
  | 1  | "george.bluth@reqres.in" | "George" | "Bluth"   | 200        |
  | 2  | "janet.weaver@reqres.in" | "Janet"  | "Weaver"  | 200        |
  | 3  | "emma.wong@reqres.in"   | "Emma"   | "Wong"    | 200        |
```

The screenshot shows the Postman interface with a GET request to `https://reqres.in/api/users/2`. The URL is highlighted with a red box and labeled "URL de la petición". The "Query Params" section shows a parameter `id` with the value `2`, also highlighted with a red box and labeled "ID". The "Body" tab is selected, showing a status of `200 OK` with a response time of `322 ms` and a size of `1 KB`. This status is highlighted with a red box and labeled "Codigo de estado". The response body is shown in "Pretty" format, containing a JSON object with user details. The entire response body is highlighted with a red box and labeled "Body respuesta esperado".

```
{
  "data": {
    "id": 2,
    "email": "janet.weaver@reqres.in",
    "first_name": "Janet",
    "last_name": "Weaver",
    "avatar": "https://reqres.in/img/faces/2-image.jpg"
  },
  "support": {
    "url": "https://reqres.in/#support-heading",
    "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
  }
}
```

Reporte Serenity

Feature: List A User

3 test scenarios (including 3 rows of test data) | [Download](#)

Summary

Test Results

100%

Scenario Results (including rows of test data)		Automated
✓ Passing		3100%
⌚ Pending		0
⌚ Ignored		0
⌚ Skipped		0
✗ Unsuccessful		
✗ Failed		0
⚠ Broken		0
⚠ Compromised		0
Total		3

Execution Time	Clock Time	Fastest Test	Slowest Test	Average Execution Time
4s	4ms	265ms	4s	4s

Functional Coverage Overview

Features

Feature	Scenarios	% Pass	Result	Coverage
List a user	3	100%	✓	<div></div>
List a user	3	100%	✓	<div></div>

Tags

List User 3

3. Método [PUT] UPDATE USER: <https://reqres.in/api/users/>

Feature: Update User

AS regres admi

I WANT TO be able to update the data of an existing user

SO THAT I can keep updated information of registered users

Scenario Outline: Update a User

Given url "https://reqres.in/" + "api/users/" + <id>

And request {"name": "#(name)", "job": "#(job)"}

When method put

Then status <statusCode>

And match \$ == {"name": "#(name)", "job": "#(job)", "updatedAt": "#notnull"}

Examples:

id	name	job	statusCode
1	Jesus	Tester	200
2	Miguel	Ingeniero	200
3	Mafer	Enfermera	200

The screenshot shows the Postman interface with a PUT request configured. The URL is `https://reqres.in/api/users/2`. The method is `PUT`. The body is a JSON object: `{"name": "Jesus", "job": "Tester"}`. The response is a `200 OK` status with a JSON body: `{"name": "Jesus", "job": "Tester", "updatedAt": "2023-03-29T16:46:58.411Z"}`. Red arrows point to various elements: the request method, the URL, the JSON body, the response status, and the response JSON body.

Tipo de petición

ID de la petición


URL de la petición

JSON de la petición

codigo de estado

JSON de petición

Reporte Karate



Karate Labs

1

0

Features

2023-03-29 11:56:46 a. m.

Tags | Timeline

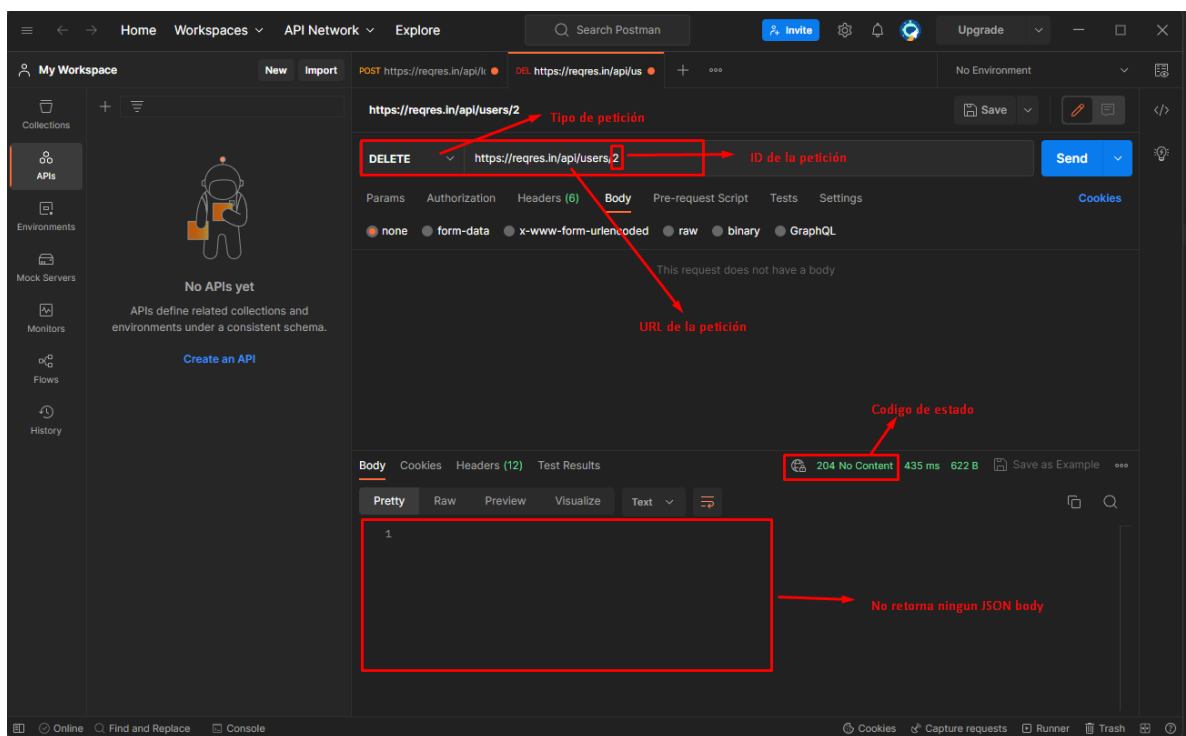
Feature	Title	Passed	Failed	Scenarios	Time (ms)
src/test/java/users/put/put-user.feature	Update User	3	0	3	2514

4. Método **[DELETE]** DELETE USER: <https://reqres.in/api/users/>

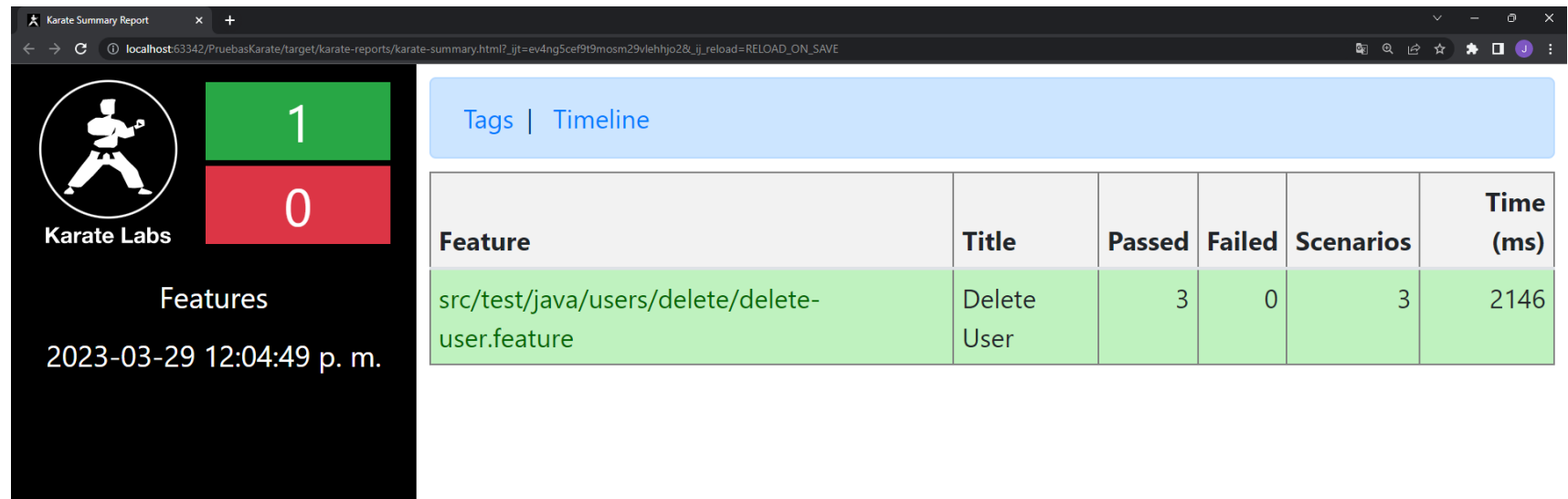
```
Feature: Delete User
  AS regres admi
  I WANT TO be able to delete an existing user
  SO THAT I can keep the list of registered users updated

Scenario Outline: Delete a User
  Given url "https://reqres.in/" + "api/users/" + <id>
  When method delete
  Then status <statusCode>

Examples:
  | id | statusCode |
  | 1  | 204        |
  | 2  | 204        |
  | 3  | 204        |
```



Reporte Karate



Conclusiones

Tanto Karate Framework como Serenity con Rest Assured son herramientas que sirven para automatizar pruebas a API

Karate Framework proporciona una solución completa para la automatización de pruebas de API y pruebas de integración de microservicios, lo que significa que no se requieren herramientas adicionales para realizar pruebas de extremo a extremo. Es fácil de aprender y usar, y es altamente legible y fácil de mantener.

Por otro lado, Rest Assured es una biblioteca que se utiliza en conjunto con un lenguaje de programación para automatizar pruebas de integración de API. Es fácil de aprender y usar para aquellos con experiencia en Java y proporciona una manera rápida y fácil de automatizar pruebas de integración de API y verificar el comportamiento de la API en diferentes situaciones. También es altamente personalizable y extensible.

Finalmente, la elección entre Karate Framework y Rest Assured dependerá de las necesidades específicas del proyecto y la experiencia del equipo de QA en lenguajes de programación.