

## Para la automatización de servicios rest utilizando ScreenPlay.

Se utilizo:

Un servicio get de la poke api, con url= <https://pokeapi.co/api/v2/berry/{id o nombre}>

Utilizando postman se realizaron pruebas manuales para conocer el funcionamiento del servicio, se probó cual era el id mínimo que recibía, el id máximo, se analizó la estructura de la respuesta y la respuesta al enviar una id o nombre que no existiera.

The screenshot shows a Postman interface for a GET request to `https://pokeapi.co/api/v2/berry/1`. The response is displayed in JSON format. The JSON structure is as follows:

```
1 {
2   "firmness": {
3     "name": "Firm",
4     "url": "https://pokeapi.co/api/v2/berry-firmness/2/"
5   },
6   "flavors": [
7     {
8       "flavor": {
9         "name": "spicy",
10        "url": "https://pokeapi.co/api/v2/berry-flavor/1/"
11      },
12      "potency": 10
13    },
14    {
15      "flavor": {
16        "name": "dry",
17        "url": "https://pokeapi.co/api/v2/berry-flavor/2/"
18      },
19      "potency": 0
20    },
21    {
22      "flavor": {
23        "name": "sweet",
24        "url": "https://pokeapi.co/api/v2/berry-flavor/3/"
25      },
26      "potency": 0
27    }
28  ]
29 }
```

Annotations in the image:

- A red box highlights the `flavors` array in the JSON response, with a red arrow pointing to it and the text: "Una respuesta correcta del servicio siempre retorna un json con un atributo llamado flavors, el cual contiene una lista de tamaño 5, y esto se tuvo en cuenta para realizar los Asserts".
- A red box highlights the status bar in the top right corner, showing "Status: 200 OK", with a red arrow pointing to it and the text: "Una consulta bien hecha retorna un código de respuesta 200".

GET ▼ https://pokeapi.co/api/v2/berry/1/ Como se puede observar retorna el atributo id con el id que fue enviado al realizar la consulta

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

| Key | Value | Description |
|-----|-------|-------------|
| Key | Value | Description |

Body Cookies Headers (26) Test Results Status: 200 OK 1

Pretty Raw Preview Visualize JSON ▼ ≡

```
34      },
35      {
36        "flavor": {
37          "name": "sour",
38          "url": "https://pokeapi.co/api/v2/berry-flavor/5/"
39        },
40        "potency": 0
41      }
42    ],
43    "growth_time": 3,
44    "id": 1,
45    "item": {
46      "name": "cheri-berry",
47      "url": "https://pokeapi.co/api/v2/item/126/"
48    },
49    "max_harvest": 5,
50    "name": "cheri",
51    "natural_gift_power": 60,
52    "natural_gift_type": {
53      "name": "fire",
54      "url": "https://pokeapi.co/api/v2/type/10/"
55    },
56    "size": 20,
57    "smoothness": 25,
58    "soil_dryness": 15
59  }
```

También retorna el nombre de la baya

En los atributos marcados fue lo que se tuvo en cuenta para realizar las aserciones.

GET ▼ https://pokeapi.co/api/v2/berry/0/ Al enviar un id o nombre inexistente en la base de datos se obtiene una respuesta "Not found" y un código 404

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

| Key | Value | Description |
|-----|-------|-------------|
| Key | Value | Description |

Body Cookies Headers (11) Test Results Status: 404 Not Found 1

Pretty Raw Preview Visualize Text ▼ ≡

1 Not Found

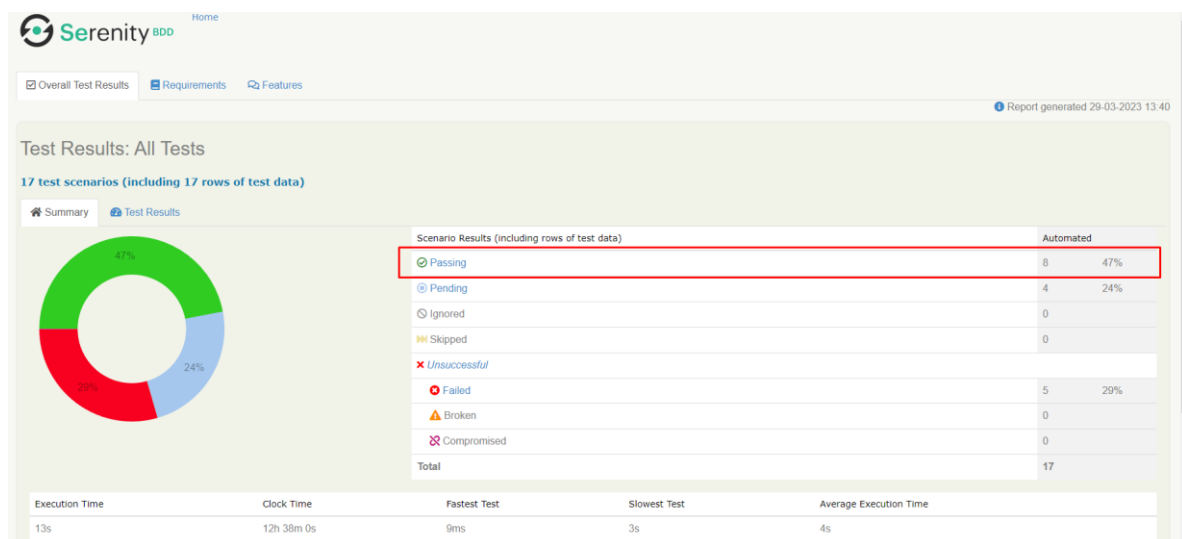
Después de analizar el servicio que se va automatiza se crea el siguiente escenario outline.

```
Feature: Ver informacion de una baya pokemon
yo como administrador de los servicios rest de la pokeApi
quiero realizar peticiones al servicio de las bayas pokemon
para ver toda la informacion acerca de una baya

Scenario Outline: Ver informacion de una baya pokemon
    Given que estoy apuntando con un endpoint a la api de bayas de la pokeapi
    When envio la peticion get con el <idnombre> de la baya
    Then recibo un <codigo> de respuesta
    And la <informacion> acerca de la baya

    Examples:
    | idnombre | codigo | informacion |
    | "1"      | 404    | "jaboca"    |
    | "1"      | 404    | "cheri"     |
    | "1"      | 200    | "cheri"     |
    | "cheri"   | 200    | "cheri"     |
    | "63"     | 200    | "jaboca"    |
    | "jaboca"  | 200    | "jaboca"    |
    | "65"     | 404    | "Not Found" |
    | "0"      | 404    | "Not Found" |
```

Y se obtienen los resultados que se ven a continuación.



Examples:

| # | Idnombre | Codigo | Informacion |
|---|----------|--------|-------------|
| 1 | "1"      | 404    | "jaboca"    |
| 2 | "1"      | 404    | "cheri"     |
| 3 | "1"      | 200    | "cheri"     |
| 4 | "cheri"  | 200    | "cheri"     |
| 5 | "63"     | 200    | "jaboca"    |
| 6 | "jaboca" | 200    | "jaboca"    |
| 7 | "65"     | 404    | "Not Found" |
| 8 | "0"      | 404    | "Not Found" |

| Steps  | Outcome | ⌚     |
|--|---------|-------|
| Example: {idnombre="1", codigo=404, informacion="jaboca"}      | FAILURE | 2.56s |
| Example: {idnombre="1", codigo=404, informacion="cheri"}       | FAILURE | 0.19s |
| Example: {idnombre="1", codigo=200, informacion="cheri"}       | SUCCESS | 0.2s  |
| Example: {idnombre="cheri", codigo=200, informacion="cheri"}   | SUCCESS | 0.17s |
| Example: {idnombre="63", codigo=200, informacion="jaboca"}     | SUCCESS | 0.16s |
| Example: {idnombre="jaboca", codigo=200, informacion="jaboca"} | SUCCESS | 0.18s |
| Example: {idnombre="65", codigo=404, informacion="Not Found"}  | SUCCESS | 1.69s |
| Example: {idnombre="0", codigo=404, informacion="Not Found"}   | SUCCESS | 0.15s |
|  | FAILURE | 5.49s |

Se crean los dos primeros escenarios para que fallen a propósito.

Un servicio put de regres.in, con url= <https://regres.in/api/users/{id}>

Utilizando postman se realizaron pruebas manuales para conocer el funcionamiento del servicio.

PUT <https://regres.in/api/users/2>

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

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

```

1  {
2    "nombre": "Pedzo",
3    "edad": 33,
4    "trabajo": "QA"
5  }

```

Body Cookies Headers (15) Test Results Status: 200 OK

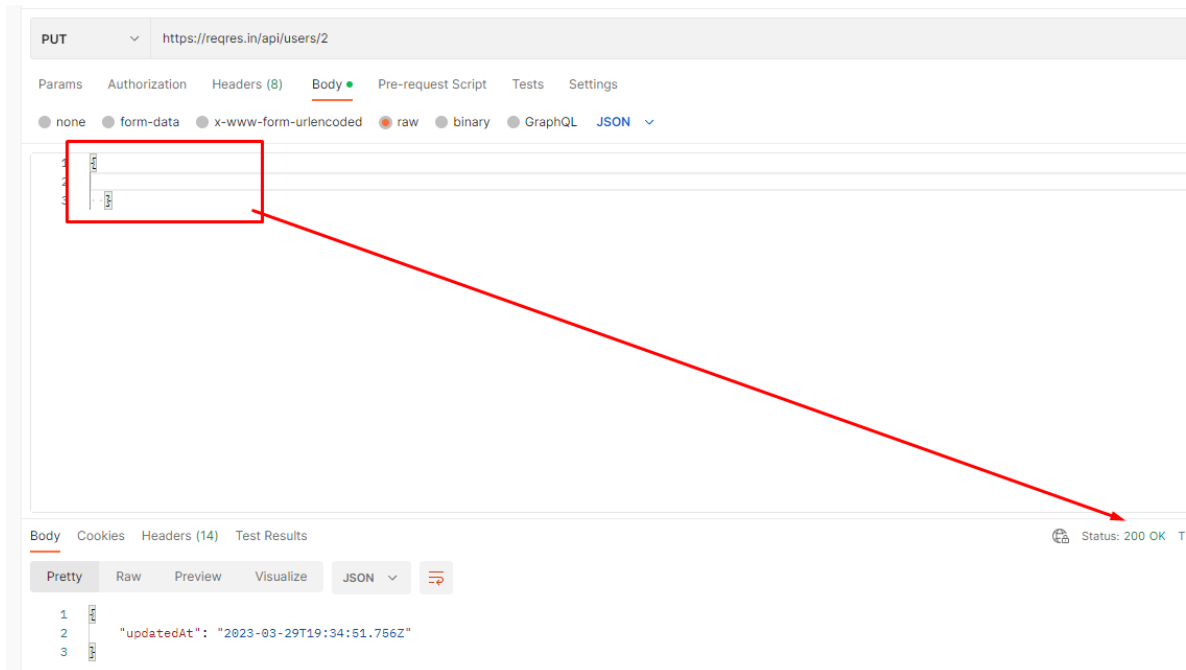
Pretty Raw Preview Visualize JSON

```

1  {
2    "nombre": "Pedzo",
3    "edad": 33,
4    "trabajo": "QA",
5    "updatedAt": "2023-03-29T19:27:30.883Z"
6  }

```

Al actualizar un usuario se crean todos los atributos y se genera uno nuevo con la fecha de la actualizacion



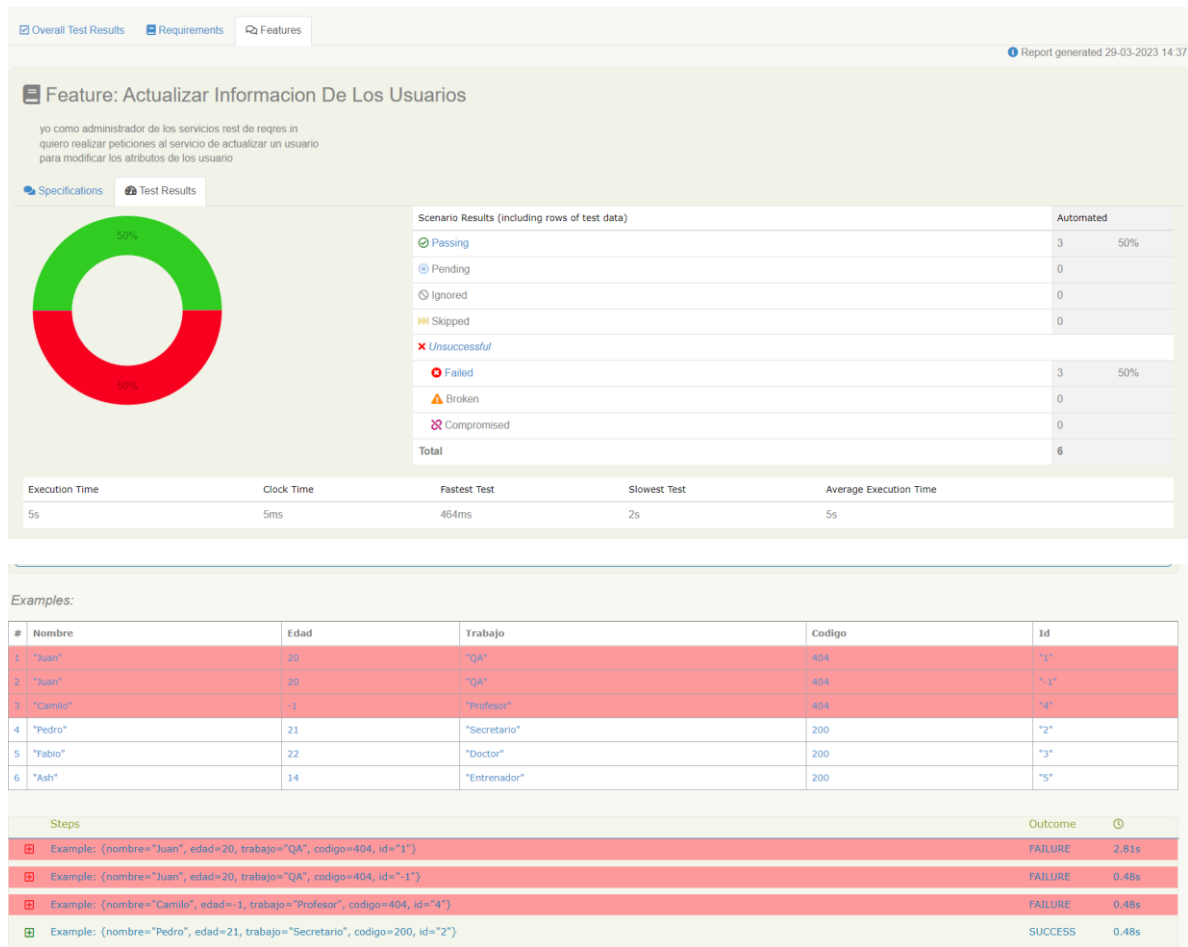
Al ser un servicio para pruebas es difícil hacerlo fallar, pero bajo mi consideración debo retornar un código de status diferente y no debió realizar la actualización.

```
Feature: Actualizar información de los usuarios
  yo como administrador de los servicios rest de regres.in
  quiero realizar peticiones al servicio de actualizar un usuario
  para modificar los atributos de los usuario

Scenario Outline: Actualizar información de un usuario
  Given que estoy apuntando con un endpoint a la api de regres.in
  When envío la petición put con el <id> de el usuario el <nombre>, <edad> y <trabajo>
  Then se recibe un <codigo> de respuesta
  And la información actualizada del usuario con la fecha de actualización

Examples:
  | nombre | edad | trabajo | codigo | id |
  | "Juan" | 20   | "QA"    | 404    | "1" |
  | "Juan" | 20   | "QA"    | 404    | "-1" |
  | "Camilo" | -1  | "Profesor" | 404    | "4" |
  | "Pedro" | 21   | "Secretario" | 200    | "2" |
  | "Fabio" | 22   | "Doctor"  | 200    | "3" |
  | "Ash"   | 14   | "Entrenador" | 200    | "5" |
```

Con la información obtenida se crea el escenario outline y se obtienen los siguientes resultados.



Para la automatización de servicios rest utilizando el framework karate.

Se utilizo:

Un servicio post de reqres.in, con url= [https://reqres.in /api/users](https://reqres.in/api/users)

Con la utilización de postman se realizaron pruebas manuales para conocer el funcionamientos de los servicios.

POST `https://reqres.in/api/users`

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

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

```
1 {
2   "name": "juan",
3   "job": "pintor"
4 }
```

Body Cookies Headers (13) Test Results Status: 201 Created

Pretty Raw Preview Visualize JSON


```
1 {
2   "name": "juan",
3   "job": "pintor",
4   "id": "666",
5   "createdAt": "2023-03-29T20:00:17.707Z"
6 }
```

Al realizar una petición que funcione correctamente  
retorna los atributos enviados junto con dos atributos nuevos

Al ser servicios usados para pruebas es difícil hacerlos fallar correctamente, por ende, solo se tomó en cuenta los atributos enviados y los nuevos que se crean para realizar las aserciones.

Teniendo en cuenta lo anterior se creó el escenario outline.

```
1 >> Feature: Crear un usuario en reqres.in
2 @Create
3 >> Scenario Outline: crear un usuario
4 * url "https://reqres.in"
5 * path "/api/users"
6 * request {"name": "#(name)", "job": "#(job)"}
7 When method post
8 Then status 201
9 And match $.name == '#(name)'
10 And match $.job == '#(job)'
11 And match $.createdAt != null
12 And match $.id != null
13 And def idUser = $.id
14
15 Examples:
16 | name | job |
17 | pedro | QA |
18 | juan | profe |
19 | Juan | profe |
20 | JuAn | profe |
21
```



4

0

Karate Labs

Scenarios

2023-03-29 12:07:45 p. m.

[1.1:15] crear un usuario

[1.2:16] crear un usuario

[1.3:17] crear un usuario

[1.4:18] crear un usuario

Summary | Tags | Feature: `src/test/java/features/crearusuario.feature` | Crear un usuario en regres.in

|                                     |   |          |
|-------------------------------------|---|----------|
| Scenario: [1.1:15] crear un usuario |   | ms: 1161 |
| 4                                   | " url "https://regres.in"                     | 62       |
| 5                                   | " path "/api/users"                           | 1        |
| 6                                   | " request {"name": "#(name)","job": "#(job)"} | 46       |
| 7                                   | When method post                              | 1022     |
| 8                                   | Then status 201                               | 0        |
| 9                                   | And match \$.name == "#(name)"                | 27       |
| 10                                  | And match \$.job == "#(job)"                  | 1        |
| 11                                  | And match \$.createdAt != null                | 3        |

|                                     |   |         |
|-------------------------------------|---|---------|
| Scenario: [1.2:16] crear un usuario |   | ms: 458 |
| 4                                   | " url "https://regres.in"                     | 1       |
| 5                                   | " path "/api/users"                           | 0       |
| 6                                   | " request {"name": "#(name)","job": "#(job)"} | 0       |
| 7                                   | When method post                              | 455     |
| 8                                   | Then status 201                               | 0       |
| 9                                   | And match \$.name == "#(name)"                | 1       |
| 10                                  | And match \$.job == "#(job)"                  | 1       |
| 11                                  | And match \$.createdAt != null                | 0       |

|                                     |   |         |
|-------------------------------------|---|---------|
| Scenario: [1.3:17] crear un usuario |   | ms: 418 |
| 4                                   | " url "https://regres.in"                     | 0       |
| 5                                   | " path "/api/users"                           | 0       |
| 6                                   | " request {"name": "#(name)","job": "#(job)"} | 0       |
| 7                                   | When method post                              | 416     |
| 8                                   | Then status 201                               | 0       |
| 9                                   | And match \$.name == "#(name)"                | 1       |
| 10                                  | And match \$.job == "#(job)"                  | 0       |
| 11                                  | And match \$.createdAt != null                | 0       |

|                                     |   |         |
|-------------------------------------|---|---------|
| Scenario: [1.4:18] crear un usuario |   | ms: 431 |
| 4                                   | " url "https://regres.in"                     | 0       |
| 5                                   | " path "/api/users"                           | 0       |
| 6                                   | " request {"name": "#(name)","job": "#(job)"} | 0       |
| 7                                   | When method post                              | 429     |
| 8                                   | Then status 201                               | 0       |

Un servicio delete de regres.in, con url= <https://regres.in/api/users/{id}>

DELETE

https://regres.in/api/users/2

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

| Key | Value | Description |
|-----|-------|-------------|
| Key | Value | Description |

Body

Cookies

Headers (12)

Test Results

Pretty

Raw

Preview

Visualize

Text

1

Status: 204 No Content

Con el delete solo obtenemos el  
codigo de status 204, por ende,  
no se puede hacer mucho con este  
servicio.

Por ende el escenario outline creado fue muy sencillo



```

> Feature: eliminar un usuario en regres.in

> Scenario Outline: eliminar un usuario
  Given url "https://regres.in"+"api/users/"+"#(id)"
  When method delete
  Then status 204
  And match "'"+204+"'" == '#(code)'
```

Examples:

| id | code  |
|----|-------|
| 1  | '204' |
| 2  | '204' |
| -2 | '400' |

Y se obtuvieron los siguientes resultados.

| Summary   Tags   Feature: src/test/java/features/delete.feature   eliminar un usuario en regres.in |         |
|--|---------|
| Scenario: [1.1:10] eliminar un usuario   | ms: 888 |
| 4 Given url "https://regres.in"+"api/users/"+"#(id)"   | 54      |
| 5 When method delete   | 802     |
| 6 Then status 204  | 0       |
| 7 And match "'"+204+"'" == '#(code)'   | 32      |
| Scenario: [1.2:11] eliminar un usuario   | ms: 467 |
| 4 Given url "https://regres.in"+"api/users/"+"#(id)"   | 1       |
| 5 When method delete   | 465     |
| 6 Then status 204  | 0       |
| 7 And match "'"+204+"'" == '#(code)'   | 1       |
| Scenario: [1.3:12] eliminar un usuario   | ms: 435 |
| 4 Given url "https://regres.in"+"api/users/"+"#(id)"   | 0       |
| 5 When method delete   | 434     |
| 6 Then status 204  | 0       |
| 7 And match "'"+204+"'" == '#(code)'   | 1       |

## Conclusiones

Karate es una herramienta bastante sencilla de usar a comparación de Serenity, pero siento que limita mucho, aunque puede ser por el poco tiempo que llevo usándola, pero es una manera de realizar pruebas rápidas sin necesidad de programar, conociendo poquitas herramientas que ofrece este framework se pueden hacer muchas cosas.

En resumen, Serenity es más adecuado para pruebas complejas y grandes proyectos de automatización de pruebas de software, mientras que Karate es más adecuado para pruebas de API y proyectos más pequeños con una curva de aprendizaje más rápida.