



---

**SERENITY BDD - KARATE**

---

Presentado por: Jessica Andrea López

Presentado a: Juan Esteban Pineda



29 DE MARZO DE 2023

SOFKAU  
Automatización de Pruebas

## Tabla de Contenido

<b>Serenity BDD</b> .....	2
<b>CREATE</b> - <a href="https://restful-booker.herokuapp.com/auth">https://restful-booker.herokuapp.com/auth</a> .....	2
<b>GET</b> - <a href="https://reqres.in/api/unknown">https://reqres.in/api/unknown</a> .....	5
<b>Karate</b> .....	8
<b>DELETE</b> - <a href="https://reqres.in/api/users/2">https://reqres.in/api/users/2</a> .....	9
<b>PUT</b> - <a href="https://reqres.in/api/users/2">https://reqres.in/api/users/2</a> .....	10
<b>PUT</b> - <a href="https://restful-booker.herokuapp.com/booking/:id">https://restful-booker.herokuapp.com/booking/:id</a> .....	13
<b>Conclusiones</b> .....	16

## Serenity BDD

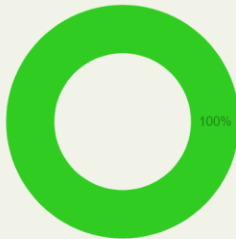
### Passing Tests

12 tests across 2 scenarios | [Download](#)

**Note that results include data-driven scenarios containing tests , which may also contain results other than tests .**

Summary

Test Results



Scenarios	Automated	
✓ Passing	12	100%
⏸ Pending	0	
⌛ Ignored	0	
⏭ Skipped	0	
✗ Unsuccessful		
✗ Failed	0	
⚠ Broken	0	
⚡ Compromised	0	
Total	12	

Tags

[Create Auth Token](#) [Get Resource](#)

### Passing Tests

12 tests across 2 scenarios | [Download](#)

**Note that results include data-driven scenarios containing tests , which may also contain results other than tests .**

Summary

Test Results

#### Automated Tests

Show 25 entries

Filter

Feature	Scenario	Steps	Start Time	Duration	Result
Create Authentication Token	Create a new authentication token for update and delete requests in Booking API (6 examples)	4	14:44:03	10s 026ms	✓
Obtain Resources from ReqRes API	Obtain resources from the ReqRes API (6 examples)	4	14:44:35	10s 519ms	✓

Showing 1 to 2 of 2 entries

Previous 1 Next

**CREATE** - <https://restful-booker.herokuapp.com/auth>

Explore previamente la aplicación que pretende automatizar en Postman (Respuestas esperadas).

# Auth - CreateToken

1.0.0

Creates a new auth token to use for access to the PUT and DELETE /booking

POST

https://restful-booker.herokuapp.com/auth

Example 1:

```
curl -X POST \
  https://restful-booker.herokuapp.com/auth \
  -H 'Content-Type: application/json' \
  -d '{
    "username" : "admin",
    "password" : "password123"
  }'
```

https://restful-booker.herokuapp.com/auth

Save



POST

https://restful-booker.herokuapp.com/auth

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Cookies

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

Beautify

```
1 {
2   "username": "admin",
3   "password": "password123"
4 }
```

Body Cookies Headers (8) Test Results

200 OK 91 ms 271 B Save Response

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "token": "1ccaf78db3a88cc"
3 }
```

POST

https://restful-booker.herokuapp.com/auth

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Cookies

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

Beautify

```
1 {
2   "username": "",
3   "password": "password123"
4 }
```

Body Cookies Headers (8) Test Results

200 OK 491 ms 272 B Save Response

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "reason": "Bad credentials"
3 }
```

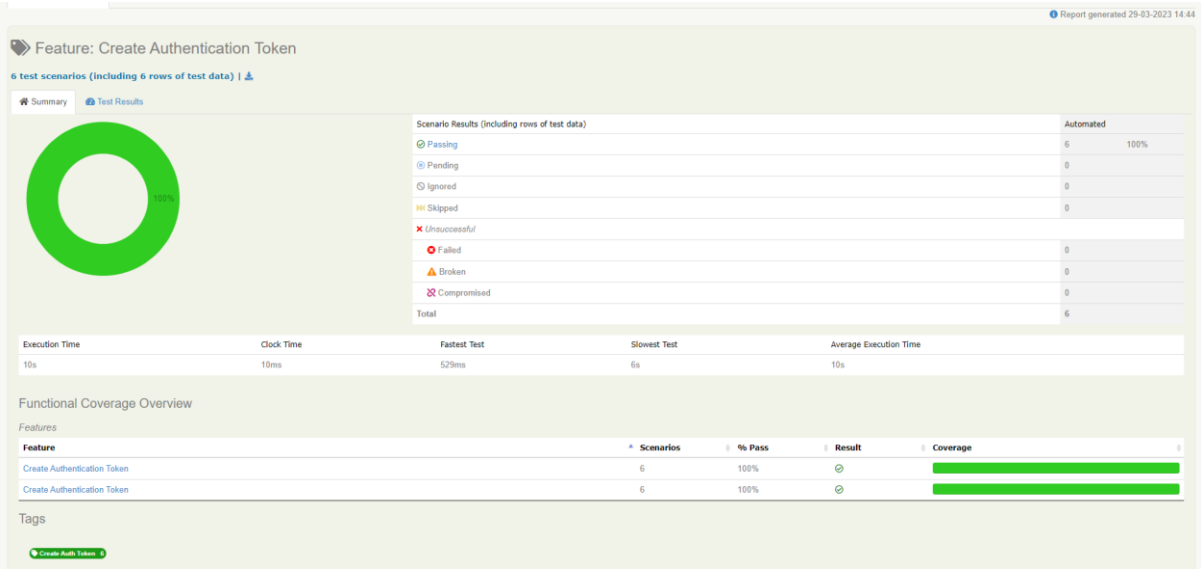
Defina claramente la característica que desea automatizar.

La característica que se desea automatizar es la creación de un nuevo token de autenticación para poder realizar solicitudes de actualización y eliminación en la API de Booking. Se especifica que se deben probar diferentes casos de prueba utilizando diferentes credenciales para comprobar que se pueda generar un token válido.

```
Feature: Create Authentication Token
  As a user of Booking API
  I want to create a new authentication token with my credentials
  So that I can access for update and delete requests in Booking API

@CreateAuthToken
Scenario Outline: Create a new authentication token for update and delete requests in Booking API
  Given I am in the Booking API
  When I make a POST request to create a new authentication token with valid username <username> and password <password> credentials
  Then the status code should be <code>
  And the response body should contain a valid authentication token
  Examples:
    | username | password | code |
    | "admin"  | "password123" | 200 |
    | ""       | "password123" | 200 |
    | ""       | ""           | 200 |
    | "admin"  | "\n"         | 200 |
    | "admin " | "password123" | 200 |
    | "admin"  | " password123" | 200 |
```

Genere un reporte con las evidencias de la ejecución.



Create a new authentication token for update and delete requests in Booking API

Cast

Scenario Outline

Given I am in the Booking API  
When I make a POST request to create a new authentication token with valid username <username> and password <password> credentials  
Then the status code should be <code>  
And the response body should contain a valid authentication token

Examples:

#	Username	Password	Code
1	"admin"	"password123"	200
2	--	"password123"	200
3	--	--	200
4	"admin"	"	200
5	"admin"	"password123"	200
6	"admin"	" password123"	200

Steps

	Outcome	
Example: {username="admin", password="password123", code=200}	SUCCESS	6.83s
Example: {username="", password="password123", code=200}	SUCCESS	0.59s
Example: {username="", password="", code=200}	SUCCESS	0.54s
Example: {username="admin", password="", code=200}	SUCCESS	0.57s
Example: {username="admin ", password="password123", code=200}	SUCCESS	0.53s
Example: {username="admin", password=" password123", code=200}	SUCCESS	0.54s
	SUCCESS	10.03s

GET - https://reqres.in/api/unknown

Request

/api/unknown

Response

200

https://reqres.in/api/unknown

Save

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	KEY	VALUE	DESCRIPTION		Bulk Edit
--	-----	-------	-------------	--	-----------

Body

Cookies

Headers (17)

Test Results

200 OK

250 ms

1.42 KB

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1
2  "page": 1,
3  "per_page": 6,
4  "total": 12,
5  "total_pages": 2,
6  "data": [
7    {
8      "id": 1,
9      "name": "cerulean",
10     "year": 2000,
11     "color": "#98B2D1",
12     "pantone_value": "15-4020"
13   },
14   {
15     "id": 2,
16     "name": "fuchsia rose",
17     "year": 2001,
18     "color": "#C74375",
19     "pantone_value": "17-2031"
20   },
21   ]
```

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method: GET, URL: `https://reqres.in/api/users`, Send button.
- Params Tab:** Query Params table with columns: KEY, VALUE, DESCRIPTION, Bulk Edit.
- Response Bar:** Status: 200 OK, Time: 543 ms, Size: 1.7 KB, Save Response button.
- Body Tab:** JSON view showing the response data.

The JSON response is as follows:

```
1 {
2   "page": 1,
3   "per_page": 6,
4   "total": 12,
5   "total_pages": 2,
6   "data": [
7     {
8       "id": 1,
9       "email": "george.bluth@reqres.in",
10      "first_name": "George",
11      "last_name": "Bluth",
12      "avatar": "https://reqres.in/img/faces/1-image.jpg"
13    },
14    {
15      "id": 2,
16      "email": "janet.weaver@reqres.in",
17      "first_name": "Janet",
18      "last_name": "Weaver",
19      "avatar": "https://reqres.in/img/faces/2-image.jpg"
20    }
21  ]
22 }
```

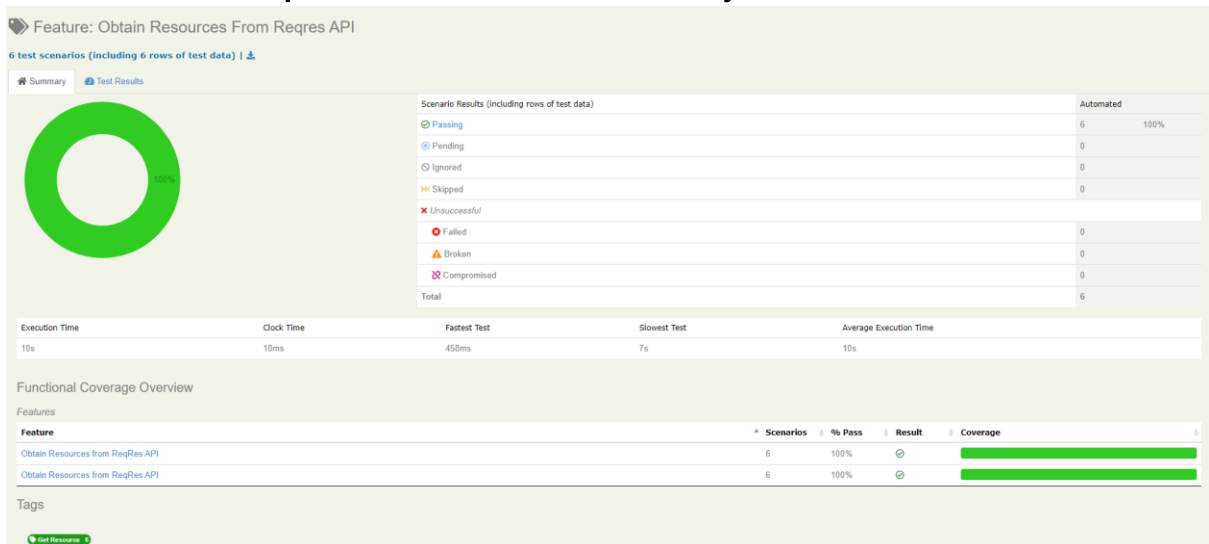
### Defina claramente la característica que desea automatizar.

La característica que se desea automatizar es la obtención de una lista de recursos a través de una solicitud GET a la API de ReqRes. El objetivo de esta característica es permitir a los usuarios obtener información detallada de todos los recursos de un tipo específico, como usuarios, productos o publicaciones, y verificar que la API está disponible mediante la validación de la respuesta de código de estado.

```
Feature: Obtain Resources from ReqRes API
  As a user of ReqRes API
  I want to get a list of resources by resource type
  So that I can view the details of all resources of a specific type

  @GetResource
  Scenario Outline: Obtain resources from the ReqRes API
    Given the ReqRes API is available
    When I make a GET request to retrieve the <resource> list
    Then the response status code should be <code>
    And the response body should contain a list of <resource>
    Examples:
      | resource | code |
      | "users"  | 200  |
      | "unknown" | 200  |
      | "products" | 200  |
      | "posts"  | 200  |
      | " "      | 404  |
      | "/"      | 404  |
```

Genere un reporte con las evidencias de la ejecución.





Obtain resources from the ReqRes API

Call

### Scenario Outline

Given the ReqRes API is available  
 When I make a GET request to retrieve the <resource> list  
 Then the response status code should be <code>  
 And the response body should contain a list of <resource>

Examples:

#	Resource	Code
1	"users"	200
2	"unknown"	200
3	"products"	200
4	"posts"	200
5	--	404
6	"/"	404

Steps	Outcome	⌚
Example: (resource="users", code=200)	SUCCESS	7.12s
Example: (resource="unknown", code=200)	SUCCESS	0.46s
Example: (resource="products", code=200)	SUCCESS	0.69s
Example: (resource="posts", code=200)	SUCCESS	0.57s
Example: (resource="", code=404)	SUCCESS	0.79s
Example: (resource="/", code=404)	SUCCESS	0.57s
	SUCCESS	10.52s

# Karate

3

0

Features  
 2023-03-29 02:40:03 p. m.

[Tags](#) | [Timeline](#)

Feature	Title	Passed	Failed	Scenarios	Time (ms)
src/test/java/features/deleteUser.feature	Delete User	4	0	4	3027
src/test/java/features/putBooking.feature	Update Booking	2	0	2	1169
src/test/java/features/putUser.feature	Update User	3	0	3	1396

Explore previamente la aplicación que pretende automatizar en Postman (Respuestas esperadas).

## DELETE - <https://reqres.in/api/users/2>

The screenshot displays a REST client interface with a 'Request' tab on the left and a 'Response' tab on the right. The Request tab shows the URL `https://reqres.in/api/users/2` and the HTTP method `DELETE`. The Response tab shows the status code `204`. Below the tabs, there are sections for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Query Params' section is expanded, showing a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The table contains one row with 'Key' and 'Value'. The 'Body' tab is selected, showing a 'Pretty' view of the response body, which is empty. The 'Save Response' button is visible in the top right corner.

KEY	VALUE	DESCRIPTION
Key	Value	Description

### Defina claramente la característica que desea automatizar.

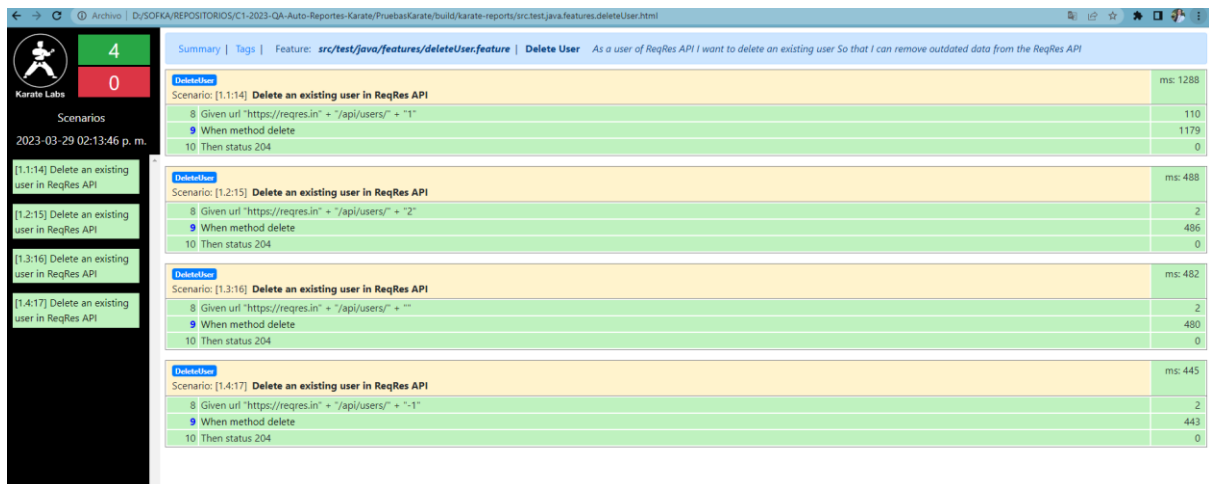
La característica que se desea automatizar es la funcionalidad de eliminar un usuario existente en la API de ReqRes. Esta característica permitirá a los usuarios eliminar datos de la API de ReqRes mediante el envío de una solicitud HTTP DELETE a la URL correspondiente que contiene el ID del usuario a eliminar. La eliminación exitosa del usuario se confirmará mediante la recepción de un código de estado HTTP 204.

```
Feature: Delete User
  As a user of ReqRes API
  I want to delete an existing user
  So that I can remove outdated data from the ReqRes API

  @DeleteUser
  Scenario Outline: Delete an existing user in ReqRes API
    Given url "https://reqres.in" + "/api/users/" + <id>
    When method delete
    Then status 204

    Examples:
      | id |
      | "1" |
      | "2" |
      | "" |
      | "-1" |
```

Genere un reporte con las evidencias de la ejecución.



The screenshot shows a web browser displaying a Karate Labs report. The left sidebar has a 'Karate Labs' logo and a 'Scenarios' section with a list of test scenarios. The main content area shows a 'Summary' tab with a table of test results. The table has columns for 'Scenario', 'Step', 'Status', and 'Time'. The scenarios are 'Delete an existing user in ReqRes API' and 'Delete an existing user in ReqRes API'. The steps are 'Given url', 'When method delete', and 'Then status 204'. The status is 'Pass' for all steps. The time is 'ms: 1288' for the first scenario and 'ms: 488' for the second scenario.

Scenario	Step	Status	Time
[1.1:14] Delete an existing user in ReqRes API	8 Given url "https://reqres.in" + "/api/users/" + "1"	Pass	110
	9 When method delete	Pass	1179
	10 Then status 204	Pass	0
[1.2:15] Delete an existing user in ReqRes API	8 Given url "https://reqres.in" + "/api/users/" + "2"	Pass	2
	9 When method delete	Pass	486
	10 Then status 204	Pass	0
[1.3:16] Delete an existing user in ReqRes API	8 Given url "https://reqres.in" + "/api/users/" + ""	Pass	2
	9 When method delete	Pass	480
	10 Then status 204	Pass	0
[1.4:17] Delete an existing user in ReqRes API	8 Given url "https://reqres.in" + "/api/users/" + "-1"	Pass	2
	9 When method delete	Pass	443
	10 Then status 204	Pass	0

Explore previamente la aplicación que pretende automatizar en Postman (Respuestas esperadas).

**PUT** - <https://reqres.in/api/users/2>



The screenshot shows a Postman interface with a 'Request' tab and a 'Response' tab. The 'Request' tab shows a PUT request to 'https://reqres.in/api/users/2' with a JSON body. The 'Response' tab shows a 200 status code and a JSON body.

Request	Response
<code>/api/users/2</code>	<code>200</code>
<pre>{   "name": "morpheus",   "job": "zion resident" }</pre>	<pre>{   "name": "morpheus",   "job": "zion resident",   "updatedAt": "2023-03-29T19:23:49.291Z" }</pre>

The image displays two screenshots of the ReqRes API interface, showing the process of updating a user's information via a PUT request.

**Top Screenshot:**

- URL:** `https://reqres.in/api/users/2`
- Method:** PUT
- Body (JSON):**

```
{  "name": "morpheus",  "job": "zion resident"}
```
- Response (JSON):**

```
{  "name": "morpheus",  "job": "zion resident",  "updatedAt": "2023-03-29T19:25:53.264Z"}
```
- Status:** 200 OK, 500 ms, 790 B

**Bottom Screenshot:**

- URL:** `https://reqres.in/api/users/2`
- Method:** PUT
- Body (JSON):**

```
{  "name": "Jessica",  "job": "Engineer"}
```
- Response (JSON):**

```
{  "name": "Jessica",  "job": "Engineer",  "updatedAt": "2023-03-29T20:47:58.924Z"}
```
- Status:** 200 OK, 312 ms, 780 B

## Defina claramente la característica que desea automatizar.

La característica que se desea automatizar es la funcionalidad de actualizar la información de un usuario existente en la API de ReqRes. Esta característica permitirá a los usuarios actualizar la información del usuario mediante el envío de una solicitud HTTP PUT a la URL correspondiente que contiene el ID del usuario a actualizar, junto con los nuevos datos de nombre y trabajo. La actualización exitosa del usuario se confirmará mediante la recepción de un código de estado HTTP 200 y la verificación de que el campo `updatedAt` del usuario actualizado no sea nulo. Esto asegura que la información en la API de ReqRes sea precisa y actualizada.


```

Feature: Update User
  As a user of ReqRes API
  I want to update an existing user's information
  So that I can ensure the accuracy of the data in the ReqRes API

  @UpdateUser
  Scenario Outline: Update an existing user's information in ReqRes API
    Given url "https://reqres.in" + "/api/users/" + "2"
    And request {"name": <name>,"job": <job>}
    When method put
    Then status 200
    And match $.updatedAt != null

    Examples:
      | name      | job      |
      | "Jessica" | "Engineer" |
      | ""        | ""        |
      | " "      | "*/"     |
  
```

Genere un reporte con las evidencias de la ejecución.

 <div>3</div> <div>0</div> <p>Scenarios</p> <p>2023-03-29 02:13:52 p. m.</p> <p>[1.1:16] Update an existing user's information in ReqRes API</p> <p>[1.2:17] Update an existing user's information in ReqRes API</p> <p>[1.3:18] Update an existing user's information in ReqRes API</p>	<div>Summary   Tags   Feature: <code>src/test/java/features/putUser.feature</code>   <b>Update User</b> As a user of ReqRes API I want to update an existing user's information So that I can ensure the accuracy of the data in the ReqRes API</div> <table border="1"> <thead> <tr> <th>UpdateUser</th> <th>ms: 472</th> </tr> </thead> <tbody> <tr> <td colspan="2">Scenario: [1.1:16] Update an existing user's information in ReqRes API</td> </tr> <tr> <td>8 Given url "https://reqres.in" + "/api/users/" + "2"</td> <td>1</td> </tr> <tr> <td>9 And request {"name": "Jessica","job": "Engineer"}</td> <td>0</td> </tr> <tr> <td>10 When method put</td> <td>441</td> </tr> <tr> <td>11 Then status 200</td> <td>0</td> </tr> <tr> <td>12 And match \$.updatedAt != null</td> <td>30</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>UpdateUser</th> <th>ms: 469</th> </tr> </thead> <tbody> <tr> <td colspan="2">Scenario: [1.2:17] Update an existing user's information in ReqRes API</td> </tr> <tr> <td>8 Given url "https://reqres.in" + "/api/users/" + "2"</td> <td>1</td> </tr> <tr> <td>9 And request {"name": "", "job": ""}</td> <td>0</td> </tr> <tr> <td>10 When method put</td> <td>466</td> </tr> <tr> <td>11 Then status 200</td> <td>0</td> </tr> <tr> <td>12 And match \$.updatedAt != null</td> <td>2</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>UpdateUser</th> <th>ms: 459</th> </tr> </thead> <tbody> <tr> <td colspan="2">Scenario: [1.3:18] Update an existing user's information in ReqRes API</td> </tr> <tr> <td>8 Given url "https://reqres.in" + "/api/users/" + "2"</td> <td>2</td> </tr> <tr> <td>9 And request {"name": " ", "job": "*/"}</td> <td>1</td> </tr> <tr> <td>10 When method put</td> <td>456</td> </tr> <tr> <td>11 Then status 200</td> <td>0</td> </tr> <tr> <td>12 And match \$.updatedAt != null</td> <td>1</td> </tr> </tbody> </table>	UpdateUser	ms: 472	Scenario: [1.1:16] Update an existing user's information in ReqRes API		8 Given url "https://reqres.in" + "/api/users/" + "2"	1	9 And request {"name": "Jessica","job": "Engineer"}	0	10 When method put	441	11 Then status 200	0	12 And match \$.updatedAt != null	30	UpdateUser	ms: 469	Scenario: [1.2:17] Update an existing user's information in ReqRes API		8 Given url "https://reqres.in" + "/api/users/" + "2"	1	9 And request {"name": "", "job": ""}	0	10 When method put	466	11 Then status 200	0	12 And match \$.updatedAt != null	2	UpdateUser	ms: 459	Scenario: [1.3:18] Update an existing user's information in ReqRes API		8 Given url "https://reqres.in" + "/api/users/" + "2"	2	9 And request {"name": " ", "job": "*/"}	1	10 When method put	456	11 Then status 200	0	12 And match \$.updatedAt != null	1
UpdateUser	ms: 472																																										
Scenario: [1.1:16] Update an existing user's information in ReqRes API																																											
8 Given url "https://reqres.in" + "/api/users/" + "2"	1																																										
9 And request {"name": "Jessica","job": "Engineer"}	0																																										
10 When method put	441																																										
11 Then status 200	0																																										
12 And match \$.updatedAt != null	30																																										
UpdateUser	ms: 469																																										
Scenario: [1.2:17] Update an existing user's information in ReqRes API																																											
8 Given url "https://reqres.in" + "/api/users/" + "2"	1																																										
9 And request {"name": "", "job": ""}	0																																										
10 When method put	466																																										
11 Then status 200	0																																										
12 And match \$.updatedAt != null	2																																										
UpdateUser	ms: 459																																										
Scenario: [1.3:18] Update an existing user's information in ReqRes API																																											
8 Given url "https://reqres.in" + "/api/users/" + "2"	2																																										
9 And request {"name": " ", "job": "*/"}	1																																										
10 When method put	456																																										
11 Then status 200	0																																										
12 And match \$.updatedAt != null	1																																										

Explore previamente la aplicación que pretende automatizar en Postman (Respuestas esperadas).

## PUT - <https://restful-booker.herokuapp.com/booking/:id>

### Booking - UpdateBooking

1.0.0

Updates a current booking

PUT

<https://restful-booker.herokuapp.com/booking/:id>

JSON example usage:

XML example usage:

URLEncoded example usage:

```
curl -X PUT \
  https://restful-booker.herokuapp.com/booking/1 \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json' \
  -H 'Cookie: token=abc123' \
  -d '{
    "firstname" : "James",
    "lastname" : "Brown",
    "totalprice" : 111,
    "depositpaid" : true,
    "bookingdates" : {
      "checkin" : "2018-01-01",
      "checkout" : "2019-01-01"
    },
    "additionalneeds" : "Breakfast"
  }'
```

<https://restful-booker.herokuapp.com/auth>

Save

Send

POST

<https://restful-booker.herokuapp.com/auth>

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {
2   ... "username" : "admin",
3   ... "password" : "password123"
4 }
```

Body

Cookies

Headers (8)

Test Results

200 OK 91 ms 271 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

Send

Copy

```
1 {
2   "token" : "1ccaf78db3a88cc"
3 }
```



Cookie

token=1ccaf78db3a88cc

https://restful-booker.herokuapp.com/booking/2

PUT https://restful-booker.herokuapp.com/booking/2 Send

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies

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

```
1 {
2   "firstname": "Susan 2",
3   "lastname": "Wilson",
4   "totalprice": 525,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-09-03",
8     "checkout": "2021-05-07"
9   }
10 }
11
```

Body Cookies Headers (8) Test Results 200 OK 112 ms 418 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "firstname": "Susan 2",
3   "lastname": "Wilson",
4   "totalprice": 525,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-09-03",
8     "checkout": "2021-05-07"
9   },
10   "additionalneeds": "Breakfast"
11 }
```

PUT https://restful-booker.herokuapp.com/booking/1 Send

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies

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

```
1 {
2   "firstname": "Susan 2",
3   "lastname": "Wilson",
4   "totalprice": 525,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-09-03",
8     "checkout": "2021-05-07"
9   }
10 }
11
```

Body Cookies Headers (8) Test Results 403 Forbidden 362 ms 252 B Save Response

Pretty Raw Preview Visualize Text

```
1 Forbidden
```

## Defina claramente la característica que desea automatizar.

La característica que se desea automatizar es la funcionalidad de actualizar la información de una reserva existente en la API de Restful Booker. Para poder actualizar una reserva, se requiere un token de autenticación, el cual se obtiene mediante el escenario "Retrieve Token".

Una vez que se tiene el token, se puede proceder a actualizar la información de la reserva mediante el envío de una solicitud HTTP PUT a la URL correspondiente que

contiene el ID de la reserva a actualizar, junto con los nuevos datos de nombre, apellido, precio total, depósito pagado y fechas de reserva.


En este escenario en particular, se está verificando que si el usuario no tiene el token de autenticación requerido, entonces la actualización de la reserva no se llevará a cabo y se recibirá un código de estado HTTP 403. Esto asegura que la información de la API de Restful Booker solo pueda ser actualizada por usuarios autorizados y autenticados.

```
Feature: Update Booking
  As a user of Restfull Booker
  I want to update an existing booking's information
  So that I can update the information

  @Token
  Scenario: Retrieve Token
    Given url "https://restful-booker.herokuapp.com/auth"
    And request {"username": "admin", "password": "password123"}
    When method post
    Then status 200
    And def token = "token=" + response.token
    And karate.write('token.txt', token)
    And print token

  @Token
  @UpdateBooking
  Scenario: not Update an existing booking's information in Restfull Booker if the user doesn't have the token
    Given url "https://restful-booker.herokuapp.com" + "/booking/" + "2"
    And headers {"Content-Type": "application/json", "Accept": "application/json", "Cookie": token}
    And request {"firstname": "Susana", "lastname": "Wil", "totalprice": 525, "depositpaid": true, "bookingdates": {"checkin": "2023-09-03", "checkout": "2023-05-07"}}
    When method put
    Then status 403
```

Genere un reporte con las evidencias de la ejecución.

<div><div></div><div><div>2</div><div>0</div></div><div>Karate Labs</div></div> <div>Scenarios</div> <div>2023-03-29 02:39:59 p. m.</div> <div>[1:7] Retrieve Token</div> <div>[2:18] not Update an existing booking's information in Restfull Booker if the user doesn't have the token</div>	Summary   Tags   Feature: <code>src/test/java/features/putBooking.feature</code>   Update Booking   As a user of Restfull Booker I want to update an existing booking's information So that I can update the information		
Token		ms: 838	
Scenario: [1:7] Retrieve Token			
8	Given url "https://restful-booker.herokuapp.com/auth"		1
9	And request {"username": "admin", "password": "password123"}		48
10	When method post		646
11	Then status 200		0
12	And def token = "token=" + response.token		2
13	And karate.write('token.txt', token)		118
14	And print token		23
Token UpdateBooking		ms: 331	
Scenario: [2:18] not Update an existing booking's information in Restfull Booker if the user doesn't have the token			
19	Given url "https://restful-booker.herokuapp.com" + "/booking/" + "2"		1
20	And headers {"Content-Type": "application/json", "Accept": "application/json", "Cookie": token}		2
21	And request {"firstname": "Susana", "lastname": "Wil", "totalprice": 525, "depositpaid": true, "bookingdates": {"checkin": "2023-09-03", "checkout": "2023-05-07"}}		1
22	When method put		328
23	Then status 403		0



## Conclusiones

Serenity BDD y Karate son dos herramientas diferentes para la automatización de pruebas de software, cada una con sus propias fortalezas y enfoques.

Serenity BDD es una herramienta más completa que permite automatizar pruebas no solo de API, sino también de UI y de integración. Esta herramienta se enfoca en la escritura de pruebas en un estilo de lenguaje natural, como Gherkin, que facilita la comprensión de las pruebas. Además, Serenity BDD proporciona informes detallados y personalizables que ayudan a analizar los resultados de las pruebas de forma eficaz.

Por otro lado, Karate se enfoca en la automatización de pruebas de API y utiliza un lenguaje específico de dominio (DSL) simple y fácil de entender. Karate se destaca por su simplicidad y facilidad de uso, lo que la hace adecuada para principiantes.

En general, la elección entre Serenity BDD y Karate dependerá de las necesidades específicas del proyecto de automatización de pruebas. Si se necesita una herramienta más completa que permita la automatización de pruebas de UI e integración, Serenity BDD puede ser más adecuada. Si se enfoca en la automatización de pruebas de API y se busca una herramienta simple y fácil de usar, Karate puede ser una mejor opción.