Reto de Automatización QA - BackEnd

Autor: RICO ELESCANO LESLY BRIGGITTE

Estrategia de Automatización con Karate DSL, Cucumber y Gherkin:

Subido en el repositorio git: https://github.com/Lesly-Rico123/Automatizacion-ServeRest.git

- Enfoque BDD (Behavior Driven Development): Utilicé la sintaxis Gherkin para definir escenarios en lenguaje natural, facilitando la colaboración entre QA, desarrollo y negocio.
- Framework Karate DSL: Implementé pruebas automatizadas para APIs REST utilizando Karate, aprovechando su capacidad para validar respuestas JSON
- Integración con Cucumber: Aunque Karate ya usa Gherkin, la integración con Cucumber permite estructurar los escenarios en archivos .feature y ejecutar pruebas con runners JUnit o TestNG.
- Modularidad y escalabilidad: Organicé el proyecto en carpetas por funcionalidades, reutilizando features como precondiciones y manteniendo una arquitectura limpia.
- Data Driven Testing: Utilicé Scenario Outline y tablas Examples para validar múltiples casos con distintos datos de entrada.
- Informes automáticos: Karate genera reportes visuales en target/karate-reports, útiles para análisis y documentación funcional viva.

Organización modular: Separé la lógica de pruebas en carpetas específicas:

- 📁 header/: Archivos JSON reutilizables para configurar encabezados
- 📁 response/: Archivos JSON de referencia para comparar respuestas esperadas de la API.

Beneficios obtenidos:

- Estandarización en la validación de respuestas.
- Reutilización de headers comunes entre servicios.
- Facilidad para mantener pruebas ante cambios en el backend.

🔹 Descripción del Servicio de Usuario:

El servicio de usuario fue diseñado para gestionar operaciones CRUD de manera eficiente y segura. Incluye los siguientes métodos:

- crearUsuario(): El método POST /usuarios, registra un nuevo usuario en el sistema con validaciones de datos.
- ActualizarUsuario(id): El método PUT /usuarios/{_id}, actualiza la información de un usuario existente según su identificador.
- eliminarUsuario(id): El método DELETE /usuarios/{_id}, elimina un usuario del sistema.
- listarUsuarioPorId(id): El método GET /usuarios/{_id} , recupera los datos de un usuario específico.
- listarUsuarios(): El método GET /usuarios, devuelve una lista completa de usuarios registrados.

Este servicio sigue principios de diseño limpio, con separación de responsabilidades y validaciones centralizadas. Puede integrarse fácilmente con controladores REST y pruebas automatizadas en frameworks como Karate DSL o Postman.

📘 Descripción del archivo README.md de Automatización con Karate DSL:

Este archivo documenta la estrategia de ejecución de pruebas continuas para el servicio de usuarios en la API ServeRest, utilizando **Karate DSL** con sintaxis **Gherkin**. Está escrito en español y orientado a testers y desarrolladores que deseen ejecutar pruebas por tags específicos

@PruebaFinal: Ejecuta todos los métodos del recurso /usuarios.

- Si se desea ejecutar una funcionalidad específica, se cambia el tag por uno de los siguientes:

Tag	Método HTTP	Funcionalidad asociada
@CrearUsuario	POST /usuarios	Crear usuarios
@ListarUsuarios	GET /usuarios	Listar todos los usuarios
@BuscarUsuarioPorId	GET /usuarios/{id}	Consultar usuario específico
@ActualizarUsuario	PUT /usuarios/{id}	Editar usuario
@EliminarUsuarioId	DELETE /usuarios/{id}	Eliminar usuario por ID

Esto permite ejecutar pruebas moduladas, integrarlas fácilmente en pipelines CI/CD, y generar reportes por funcionalidad.

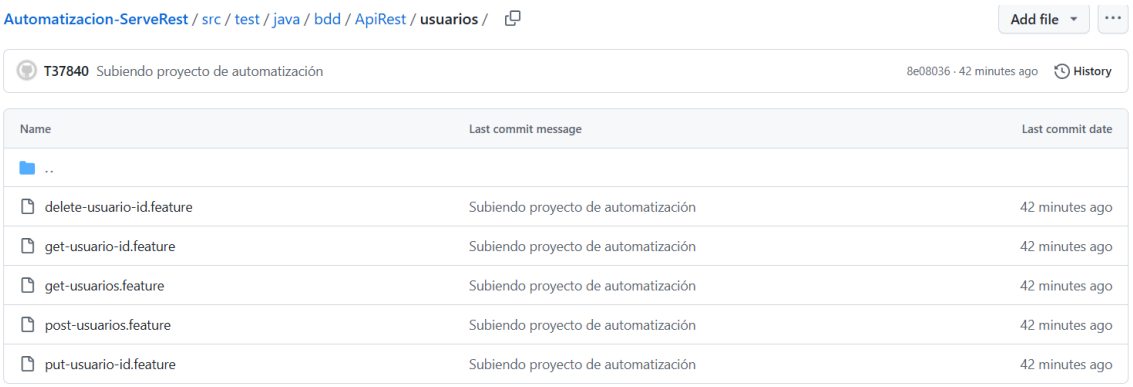
📁 **Estructura de pruebas automatizadas para el servicio de usuarios:**

Dentro de mi proyecto Automatizacion-ServeRest, organizo los escenarios de prueba por funcionalidades específicas en la ruta: src/test/java/bdd/ApiRest/usuarios/

Cada archivo .feature representa una operación REST sobre el recurso **usuarios**, siguiendo principios BDD con Gherkin. Los archivos incluyen:

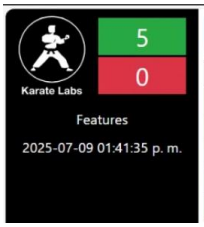
Archivo	**Método**	**Propósito**
post-usuarios.feature	POST	Crear usuarios nuevos
get-usuarios.feature	GET	Listar todos los usuarios
get-usuario-id.feature	GET	Consultar usuario específico por ID
put-usuario-id.feature	PUT	Editar información de un usuario
delete-usuario-id.feature	DELETE	Eliminar usuario (validando restricciones)

Esta estructura permite una fácil lectura, mantenimiento y ejecución independiente de cada escenario.



Automatizamos las pruebas de todos los métodos del servicio de usuarios utilizando Karate DSL, estructurando cada funcionalidad (crear, listar, listarId, editar, eliminar) en archivos .feature con sintaxis Gherkin. Las pruebas fueron ejecutadas exitosamente,

con generación automática de reportes visuales que validan cobertura y estabilidad del servicio.”



Tags | Timeline

Feature	Title	Passed	Failed	Scenarios	Time (ms)
bdd/ApiRest/usuarios/delete-usuario-id.feature	Eliminar usuario por ID	3	0	3	1147
bdd/ApiRest/usuarios/get-usuario-id.feature	Buscar usuario por ID	2	0	2	274
bdd/ApiRest/usuarios/get-usuarios.feature	Listar usuarios	8	0	8	1069
bdd/ApiRest/usuarios/post-usuarios.feature	Crear usuario	9	0	9	1064
bdd/ApiRest/usuarios/put-usuario-id.feature	Actualizar usuario	4	0	4	467

📊 Descripción del reporte de pruebas:

Este informe fue generado automáticamente por Karate el **09 de julio de 2025** a la **1:41 p. m.**, mostrando los resultados de los escenarios ejecutados en archivos .feature para el recurso /usuarios.

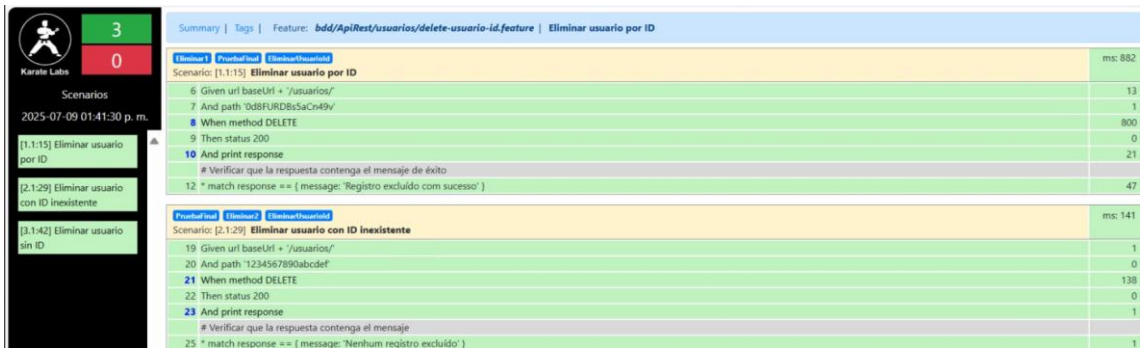
Archivo .feature	Escenario cubierto	Pasaron	Fallaron	Tiempo (ms)
delete-usuario-id.feature	Eliminar usuario por ID	3	0	1147
get-usuario-id.feature	Consultar usuario por ID	2	0	274
get-usuarios.feature	Listar todos los usuarios	8	0	1069
post-usuario.feature	Crear nuevos usuarios	9	0	1064
put-usuario.feature	Actualizar datos de usuario	4	0	467

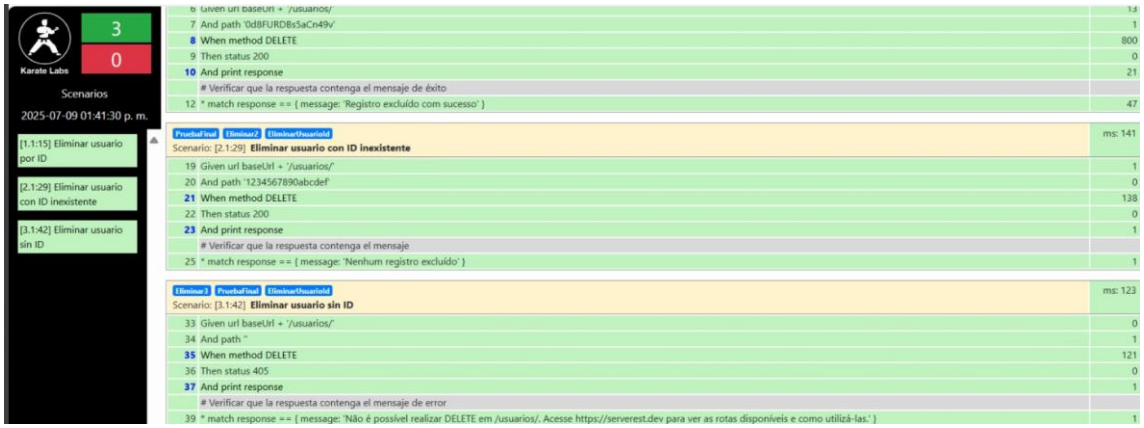
✅ Resumen del resultado:

- **Total de pruebas ejecutadas:** 5 archivos .feature
- **Total de escenarios ejecutados:** 26
- **Total de escenarios pasados:** 26
- **Total de fallos:** 0
- **Resultado general: Éxito completo**

□ Método Eliminar:

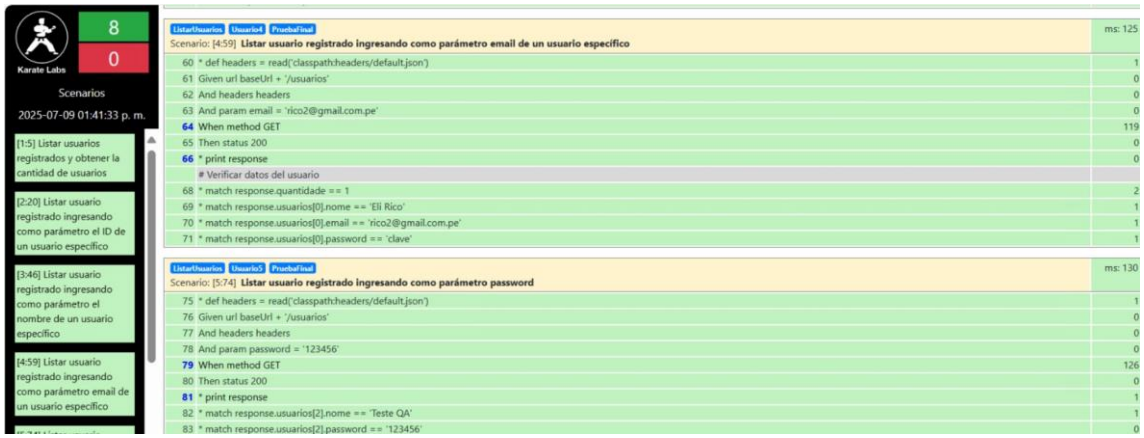
Se ejecutaron tres escenarios diseñados para validar el comportamiento del endpoint DELETE /usuarios bajo diferentes condiciones:

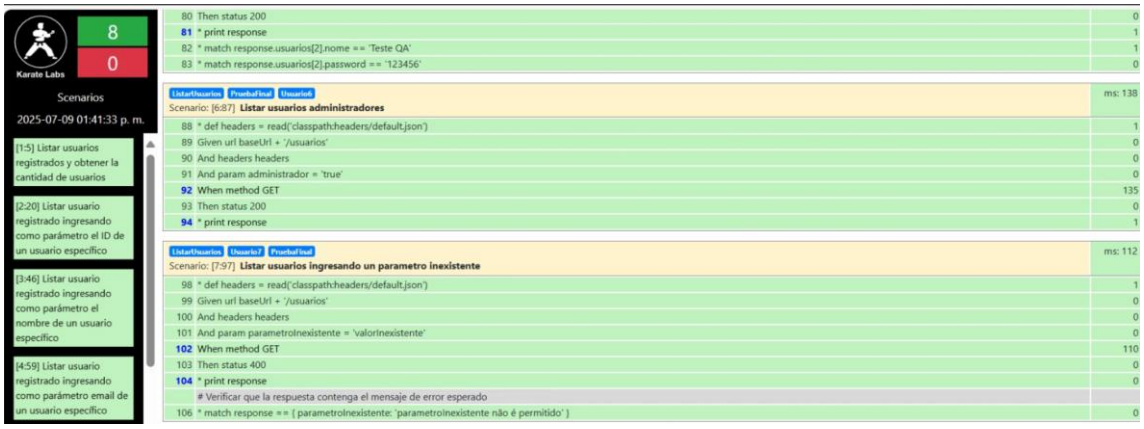


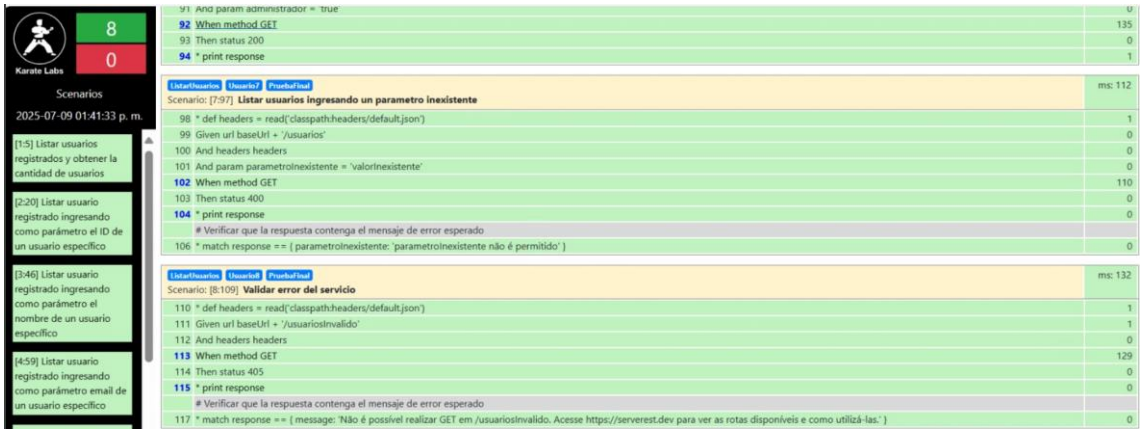


Método Litar:

Se ejecutaron tres escenarios diseñados para validar el comportamiento del endpoint GET /usuarios bajo diferentes condiciones:

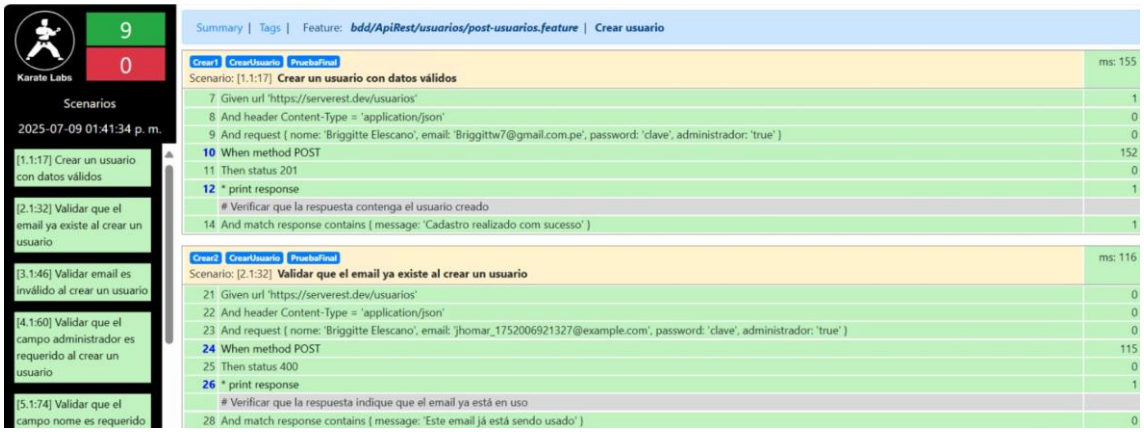


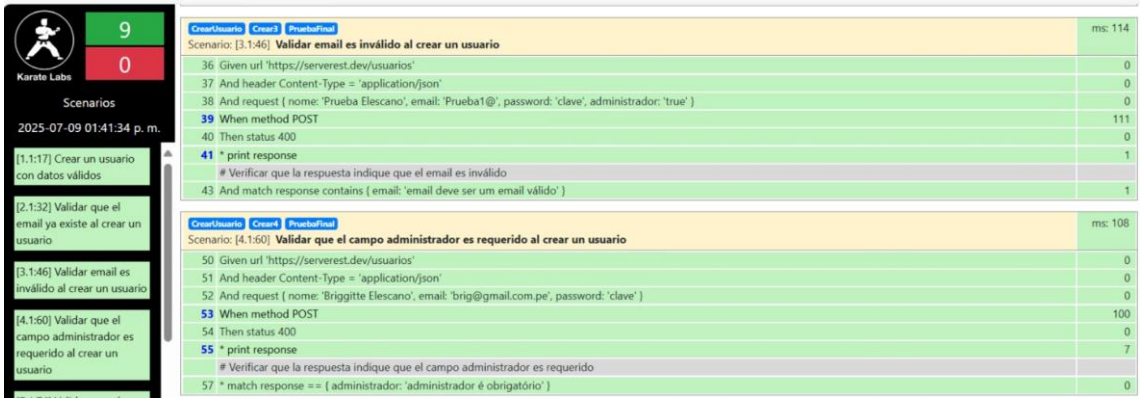


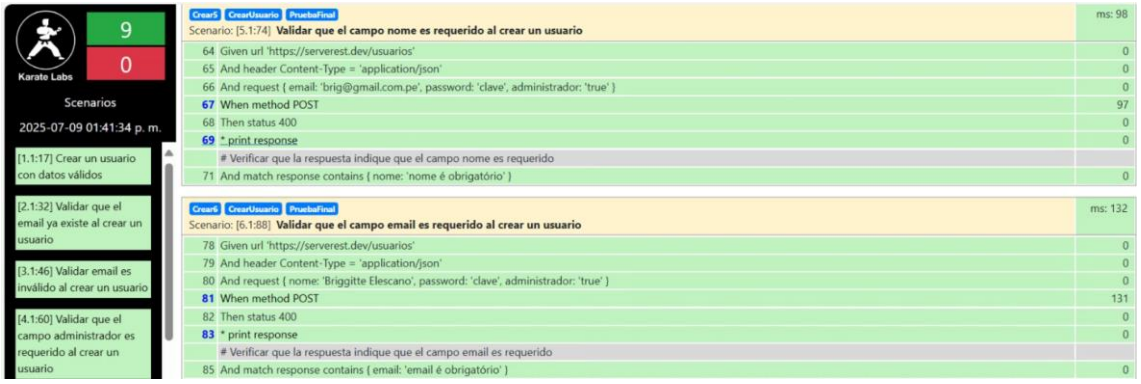


Método Crear:

Se ejecutaron tres escenarios diseñados para validar el comportamiento del endpoint Post /usuarios bajo diferentes condiciones:

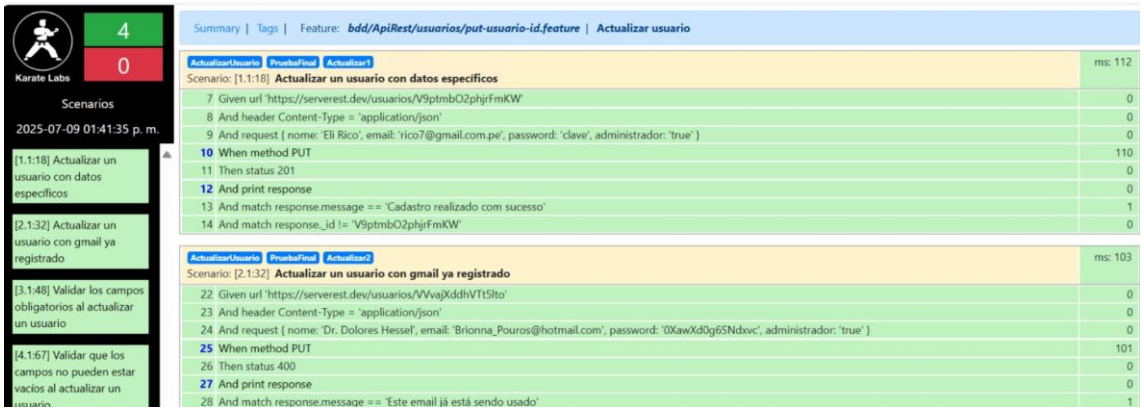






Método Actualizar:

Se ejecutaron tres escenarios diseñados para validar el comportamiento del endpoint PUT /usuarios bajo diferentes condiciones:


4
0
Karate Labs
Scenarios
2025-07-09 01:41:35 p. m.
[1.1:18] Actualizar un usuario con datos específicos
[2.1:32] Actualizar un usuario con gmail ya registrado
[3.1:48] Validar los campos obligatorios al actualizar un usuario
[4.1:67] Validar que los campos no pueden estar vacíos al actualizar un usuario
Summary | Tags | Feature: bdd/ApiRest/usuarios/put-usuario-id.feature | Actualizar usuario
ActualizarUsuario PruebaFinal Actualizar1
Scenario: [1.1:18] Actualizar un usuario con datos específicos
ms: 112
7 Given url 'https://serverest.dev/usuarios/V9ptmbO2phjrFmKW' 0
8 And header Content-Type = 'application/json' 0
9 And request { nome: 'Eli Rico', email: 'rico7@gmail.com.pe', password: 'clave', administrador: 'true' } 0
10 When method PUT 110
11 Then status 201 0
12 And print response 0
13 And match response.message == 'Cadastro realizado com sucesso' 1
14 And match response._id != 'V9ptmbO2phjrFmKW' 0
ActualizarUsuario PruebaFinal Actualizar2
Scenario: [2.1:32] Actualizar un usuario con gmail ya registrado
ms: 103
22 Given url 'https://serverest.dev/usuarios/VVvajXddhVTt5Ito' 0
23 And header Content-Type = 'application/json' 0
24 And request { nome: 'Dr. Dolores Hessel', email: 'Brionna_Pouros@hotmail.com', password: '0XawXd0g6SNdxvc', administrador: 'true' } 0
25 When method PUT 101
26 Then status 400 0
27 And print response 0
28 And match response.message == 'Este email já está sendo usado' 1


4
0
Karate Labs
Scenarios
2025-07-09 01:41:35 p. m.
[1.1:18] Actualizar un usuario con datos específicos
[2.1:32] Actualizar un usuario con gmail ya registrado
[3.1:48] Validar los campos obligatorios al actualizar un usuario
[4.1:67] Validar que los campos no pueden estar vacíos al actualizar un usuario
28 And match response.message == 'Este email já está sendo usado' 1
ActualizarUsuario PruebaFinal Actualizar3
Scenario: [3.1:48] Validar los campos obligatorios al actualizar un usuario
ms: 135
36 Given url 'https://serverest.dev/usuarios/V9ptmbO2phjrFmKW' 1
37 And header Content-Type = 'application/json' 0
38 And request { } 0
39 When method PUT 131
40 Then status 400 0
41 And print response 0
42 And match response.nome == 'nome é obrigatório' 1
43 And match response.password == 'password é obrigatório' 0
44 And match response.email == 'email é obrigatório' 0
45 And match response.administrador == 'administrador é obrigatório' 0
ActualizarUsuario Actualizar4 PruebaFinal
Scenario: [4.1:67] Validar que los campos no pueden estar vacíos al actualizar un usuario
ms: 118
52 Given url 'https://serverest.dev/usuarios/V9ptmbO2phjrFmKW' 0
53 And header Content-Type = 'application/json' 0
54 And request { nome: '', email: '', password: '', administrador: '' } 0
55 When method PUT 115
56 Then status 400 0
57 And print response 0
58 And match response.nome == 'nome não pode ficar em branco' 1
59 And match response.email == 'email não pode ficar em branco' 0
60 And match response.password == 'password não pode ficar em branco' 0
61 And match response contains 0