Aspirante: Mairon Delgado García

Cargo: Tester junior

Empresa: CINTE

Prueba técnica: Assessment Kata APIs – Categoría 7

Introducción:

En este documento se realizará una documentación sobre las incidencias encontradas en la API “Fake Store API”. Para ello se realizará un informe simple en donde se mostrarán los casos de prueba para cada endpoint, los features y escenarios utilizados, las respuestas del servidor y lo que se esperaba que se enviara.

Para la realización de esta prueba se hizo uso de la herramienta Karate, JUNIT para correr los test, y Visual Studio Code como entorno de desarrollo. Se organizaron las carpetas de forma modular para que la prueba estuviera más ordenada y fácil de testear.

Karate muestra una interfaz muy útil para observar de forma gráfica los resultados de los test, organizados los features creados.

1. Productos.

1.1. Camino feliz.

Caso de prueba	#1. Obtener todos los ids y crear un nuevo producto, copiando la información de un producto ya creado previamente.
Método	GET y POST
Respuesta esperada	200 y 201
Respuesta obtenida	200 y 2021
Resultado	**Passed**

*Nota: Fake Store API presenta limitaciones con la visualización de los nuevos productos creados. Solo permite la visualización cuando se crean directamente con post, cuando se hacen combinaciones como la mostrada anteriormente no permite su visualización, y solo muestra el id.

Caso de prueba	#2. Crear un nuevo producto con un JSON externo. Validar que la respuesta del servidor sea los mismos datos enviados en la solicitud
Método	POST
Respuesta esperada	201
Respuesta obtenida	201
Resultado	**Passed**

Caso de prueba	#3. Actualizar producto usando el método PUT y PATCH
Método	PUT y PATCH
Respuesta esperada	200 Y 200
Respuesta obtenida	200 Y 200
Resultado	**Passed**

Caso de prueba	#4. Eliminar producto
Método	DELETE
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#5. Validar tipos de datos*
Método	GET
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

*Nota: El endpoint tiene un campo adicional llamado “rating”, el cual no se muestra en la documentación de la API. La prueba se realizó teniendo en cuenta lo mencionado.

1.2. Errores encontrados

Caso de prueba	#5. Obtener un producto sin un id existente
Método	GET
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#6. Eliminar un producto con un id que no existe
Método	DELETE
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#7. Validar tipos de datos. Intentar enviar un dato que no corresponde con los tipos de datos configurados desde la API
Método	GET
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#8. Validar campos faltantes
Método	GET
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#9. Crear nuevo producto con campos faltantes*
Método	POST

Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

*Nota: Si se configura para que envíe un estado 200, el endpoint almacenará la información de todos. Es importante tener en cuenta esto, porque quiere decir que no se está controlando que tipo de datos se están enviando al servidor, lo que podría generar incongruencias y error pues se estaría guardando información que no requiere el sistema.

Caso de prueba	#10. Validar precios de la tienda demasiado altos.*
Método	PUT
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

*Nota: Aunque el caso de prueba no falló, es importante tener en cuenta este tipo de validaciones, pues es posible que en muchos contextos un precio de 1^51, no sea algo muy coherente, por lo que se recomendaría limitarlo a cierta cantidad en específico.

2. Carritos

Para tener en cuenta:

La estructura en la documentación de la API no es la correcta, la correcta es la que aparece al traer la información con “GET”. Es importante tener en cuenta esto para poder tener un body en nuestro Feature adecuado y pertinente.

-Información que aparece en la documentación de la API:

-Información que realmente envía el endpoint:


```
{
  "id": 1,
  "userId": 1,
  "date": "2020-03-02T00:00:00.000Z",
  "products": [
    {
      "productId": 1,
      "quantity": 4
    },
    {
      "productId": 2,
      "quantity": 1
    },
    {
      "productId": 3,
      "quantity": 6
    }
  ],
  "__v": 0
},
```


2.1. Camino feliz

Caso de prueba	#1. Obtener todos los carritos
Método	GET
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#2. Obtener el id de un carrito en específico.
Método	GET
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#3. Obtener el id del usuario
Método	GET
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#4. Obtener los productos de un carrito
Método	GET
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#5. Crear un nuevo carrito
Método	POST
Respuesta esperada	201
Respuesta obtenida	201
Resultado	**Passed**

Caso de prueba	#6. Crear un carrito dentro de otro carrito
Método	POST
Respuesta esperada	404
Respuesta obtenida	404
Resultado	**Passed**

Caso de prueba	#7. Añadir nuevos productos a un carrito
Método	PUT
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#8. Añadir el mismo producto varias veces*
Método	POST
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Nota: La API no actualiza los carritos, es una limitación de la API, quizá un comportamiento esperado al ser creada para hacer pruebas, por lo tanto, los datos que se actualizan no persisten.

Caso de prueba	#9. Eliminar carritos
Método	POST
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Nota general: Fake Store API no soporta la eliminación de productos dentro del carrito, lo cual consiste en una posible limitación de la API.

2.2. Errores encontrados

Caso de prueba	#10. Obtener un carro que no existe
Método	GET
Respuesta esperada	400

Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#11. Eliminar un carro que no existe
Método	DELETE
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#12. Crear un carro sin productos
Método	POST
Respuesta esperada	400
Respuesta obtenida	201
Resultado	**Failed**

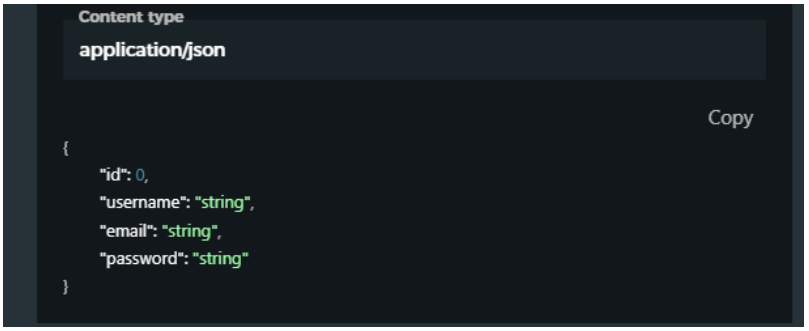
Caso de prueba	#13. Agregar un producto sin cantidad al carrito
Método	POST
Respuesta esperada	400
Respuesta obtenida	201
Resultado	**Failed**

Caso de prueba	#14. Intentar cambiar el dueño de un carrito
Método	GET
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

3. Users.

Para este endpoint se verifica la documentación del endpoint y lo que regresa al llamar los datos con GET. Se evidencia que hay inconsistencia, pues la documentación solo muestra un body con 4 campos, mientras que lo que devuelve el endpoint tiene 12 campos. Esto podría considerarse como una falla en la documentación que hace que se puedan generar fallas y confusiones por parte del equipo de desarrollo.

-Documentación API:



-Campos que devuelve el endpoint

3.1. Camino feliz

Caso de prueba	#1. Obtener todos los usuarios
Método	GET
Respuesta esperada	200
Respuesta obtenida	200

Resultado	**Passed**

Caso de prueba	#2. Obtener usuario por id
Método	GET
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#3. Validar tipos de datos
Método	GET
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#4. Crear nuevo usuario y validar creación*
Método	GET y POST
Respuesta esperada	200 y 201
Respuesta obtenida	200 y 201
Resultado	**Passed**

Nota: El endpoint no permite validar la información de los nuevos usuarios creados, solo devuelve id con “Null”, lo cual podría corresponder a una falla en un entorno real. Lo esperado es que al consultar el nuevo usuario creado, muestre la información relacionada a la creación.

Caso de prueba	#5. Editar un usuario existente
Método	PUT

Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

Caso de prueba	#6. Eliminar usuarios
Método	DELETE
Respuesta esperada	200
Respuesta obtenida	200
Resultado	**Passed**

3.2. Errores encontrados

Caso de prueba	#7. Obtener un usuario con id no existente
Método	GET
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#8. Crear nuevo usuario con información faltante*
Método	POST
Respuesta esperada	400
Respuesta obtenida	201
Resultado	**Failed**

*Nota: En este caso el endpoint debería devolver 400, porque según la documentación, los campos username, password y email son obligatorios.

Caso de prueba	#9. Crear nuevo usuario con un username ya creado en otro usuario*

Método	POST
Respuesta esperada	400
Respuesta obtenida	201
Resultado	**Failed**

*Nota: Esto depende de la regla de negocio, sin embargo, en esta prueba se asume que dos o más usuarios no deberían tener un mismo username, pues generaría inconsistencia de datos y riesgos de autenticación.

Caso de prueba	#10. Crear nuevo usuario con una contraseña con pocas validaciones*
Método	POST
Respuesta esperada	400
Respuesta obtenida	201
Resultado	**Failed**

*Nota: Aunque depende de la regla de negocio, usualmente se necesita que la contraseña cumpla con ciertos parámetros de seguridad. Por esa razón se espera que la respuesta sea 400 y no 201

Caso de prueba	#11. Crear nuevo usuario sin contraseña*
Método	POST
Respuesta esperada	400
Respuesta obtenida	201
Resultado	**Failed**

*Nota: Se espera que el sistema pida la contraseña como campo obligatorio, según lo mencionado en la documentación del endpoint.

Caso de prueba	#12. Tratar de eliminar un usuario con un id que no existe.
Método	DELETE
Respuesta esperada	400

Respuesta obtenida	200
Resultado	**Failed**

Caso de prueba	#13. Intentar cambiar el id de un usuario ya existente
Método	PUT
Respuesta esperada	400
Respuesta obtenida	200
Resultado	**Failed**

4. Auth

La documentación menciona que una autenticación exitosa debería tener un estado 200, pero realmente devuelve 201(lo más acorde), por lo que existe incongruencia o desactualización en la documentación.

1. Happy path

Caso de prueba	#1. Inicio de sesión con credenciales válidas y que el token que retorne sea un “String”
Método	POST
Respuesta esperada	201
Respuesta obtenida	201
Resultado	**Passed**

Caso de prueba	#2. Validar tiempo de respuesta (menor a 1 segundo)
Método	POST
Respuesta esperada	201
Respuesta obtenida	201
Resultado	**Passed**

Caso de prueba	#3. Validar que token no esté vacío
Método	POST
Respuesta esperada	201
Respuesta obtenida	201
Resultado	**Passed**

4.2. Casos de prueba con respuesta negativas

Caso de prueba	#4. Login con un usuario que no existe
Método	POST
Respuesta esperada	401
Respuesta obtenida	401
Resultado	**Passed**

Caso de prueba	#5. Login con una contraseña invalida
Método	POST
Respuesta esperada	401
Respuesta obtenida	401
Resultado	**Passed**

Caso de prueba	#6. Login sin enviar ninguna información
Método	POST
Respuesta esperada	400
Respuesta obtenida	400
Resultado	**Passed**

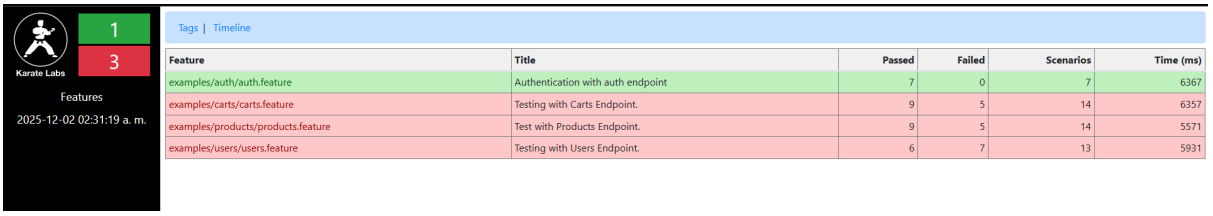
Caso de prueba	#7. Validar JSON body mal escrito
Método	POST
Respuesta esperada	400
Respuesta obtenida	400
Resultado	**Passed**

5. Estabilidad del sistema

Se realizaron 3 peticiones en paralelo para evaluar la estabilidad del servicio con el método:

Runner.path("classpath:examples").parallel(3);

Con el objetivo de simular a 3 usuarios diferentes realizando peticiones en el sistema. Al ejecutar la prueba, se evidencia que no se enviaron estados 500 y los tiempos de respuesta de los endpoints estuvieron dentro del rango habitual



6. Resultados generales.

De los 4 módulos evaluados (products, carts, users, auth/build), se evidencia que hay 3 endpoints que presentan errores o inconsistencias (products, carts, users), mientras que 1 endpoint tuvo resultados satisfactorios (auth/build). Es importante tener en cuenta los casos de prueba para cada uno de los endpoints con el objetivo de mejorar la consistencia de los datos, así como garantizar que la API responda adecuadamente ante escenarios positivos y negativos.

Conclusiones:

- Algunos endpoints envían errores como satisfactorios en escenarios no esperados.

- No hay persistencia de los datos, lo que hace complicado la consulta de nueva información registrada.
- La API en general no se encuentra bien documentada, hay campos que no muestra que son obligatorios, o campos que no se saben que se encuentran en el endpoint hasta que se consultan y se valida que hay diligenciarlos.
- A diferencia de los otros endpoints, el endpoint de autenticación fue el único en el cual se obtenían respuesta del servidor acorde a lo solicitado.
- El endpoint de autenticación fue el único que tuvo resultados satisfactorios en los test.