MAYO 25, 2025

INFORME RETO TÉCNICO - QA BACK

Automatización en la API “Serve Rest”

AUTOR: ALFREDO RAICO CHÁVEZ

1. INTRODUCCIÓN

Este documento describe la estrategia de automatización de pruebas implementada para los servicios backend de la API “Serverest.dev”. Mi enfoque principal ha sido garantizar la **fiabilidad y calidad de la API** a través de pruebas automatizadas eficientes, desarrolladas con **Karate DSL**.

2. HERRAMIENTAS

Las herramientas que se usa en este proyecto son:

- **Karate DSL**: Este es el framework principal para la automatización de pruebas de API. Su sintaxis basada en Gherkin (Given-When-Then) y su capacidad para manejar HTTP, JSON y JavaScript lo hacen ideal para el testing de backend.
- **Java Development Kit (JDK):** Karate se ejecuta sobre la Máquina Virtual de Java (JVM), por lo que el JDK es el motor que permite que las pruebas se compilen y ejecuten.
- **Apache Maven:** Esta es la herramienta de gestión de proyectos y construcción que usa. Maven es fundamental para manejar las dependencias del proyecto (como las librerías de Karate) y para ejecutar las pruebas desde la línea de comandos.
- **Visual Studio Code (VS Code):** Como entorno de desarrollo integrado se usó VS Code, proporcionó el editor de código, resaltado de sintaxis, depuración y la interfaz para escribir y organizar los tests de Karate.

3. ESTRATEGIA DE AUTOMATIZACIÓN

La estrategia se basa en un modelo de **pruebas de API**, lo que significa que se interactúa directamente con los endpoints HTTP de Serverest.dev. Esto permite validar la lógica de negocio y la consistencia de los datos de forma ágil y precisa, sin depender de una interfaz de usuario.

Los pilares de mi estrategia son:

1. **Cobertura Funcional Esencial**: Me he centrado en automatizar los flujos críticos de la API, cubriendo las operaciones fundamentales de **Creación, Lectura, Actualización y Eliminación (CRUD)** para la gestión de **usuarios**. Esto incluye escenarios de éxito, validaciones de datos, y manejo de errores (por ejemplo, intentar acceder con credenciales inválidas o buscar un usuario inexistente).
2. **Reutilización Inteligente**: Para que el proyecto sea fácil de mantener y escalar, he puesto especial énfasis en la reutilización de código. He consolidado las funciones comunes en un **archivo helpers** dedicado. Aquí se gestionan la tarea como:
    - La **generación de datos dinámicos** (como nombres de usuario o emails únicos), lo cual es clave para que las pruebas sean independientes y no generen conflictos por datos duplicados.

4. PATRONES DE DISEÑO UTILIZADOS

Para construir un conjunto de pruebas robusto y comprensible, he aplicado varios patrones de diseño:

1. **Sintaxis Gherkin "Given-When-Then"**: Karate DSL utiliza la estructura de Gherkin, lo que hace que cada escenario de prueba sea muy legible. Esto permite que incluso personas sin un perfil técnico puedan entender el propósito de cada test.
2. **Modularización con call read()**: Al dividir las funcionalidades en diferentes "features" y utilizar “* call read()” (por ejemplo, para el manejo de funciones helper), logró un alto nivel de modularidad. Cada componente cumple una función específica y puede ser invocado donde sea necesario, reduciendo la duplicidad.
3. **Encapsulamiento JavaScript**: Para la lógica de preparación de datos o transformaciones más complejas, he embebido funciones JavaScript directamente en los features. Esto me da flexibilidad y mantiene la lógica cerca de su uso, dentro del ecosistema de Karate.
4. **Validación Robusta de Respuestas JSON**: He utilizado extensivamente el comando * match de Karate junto con sus "asserts" especiales (#string, #number, #array, etc.). Esto no solo verifica que los valores sean correctos, sino también que la estructura y los tipos de datos de las respuestas JSON cumplan con lo esperado por la API.

5. CONCLUSIÓN

En resumen, la implementación de estas pruebas automatizadas con Karate DSL ha sido una estrategia efectiva para validar la API de Serverest.dev. La combinación de su sintaxis intuitiva, la capacidad de reutilizar código y las potentes herramientas de validación de JSON me permite mantener un conjunto de pruebas ágil, comprensible y fácil de expandir, contribuyendo significativamente a la calidad del backend.