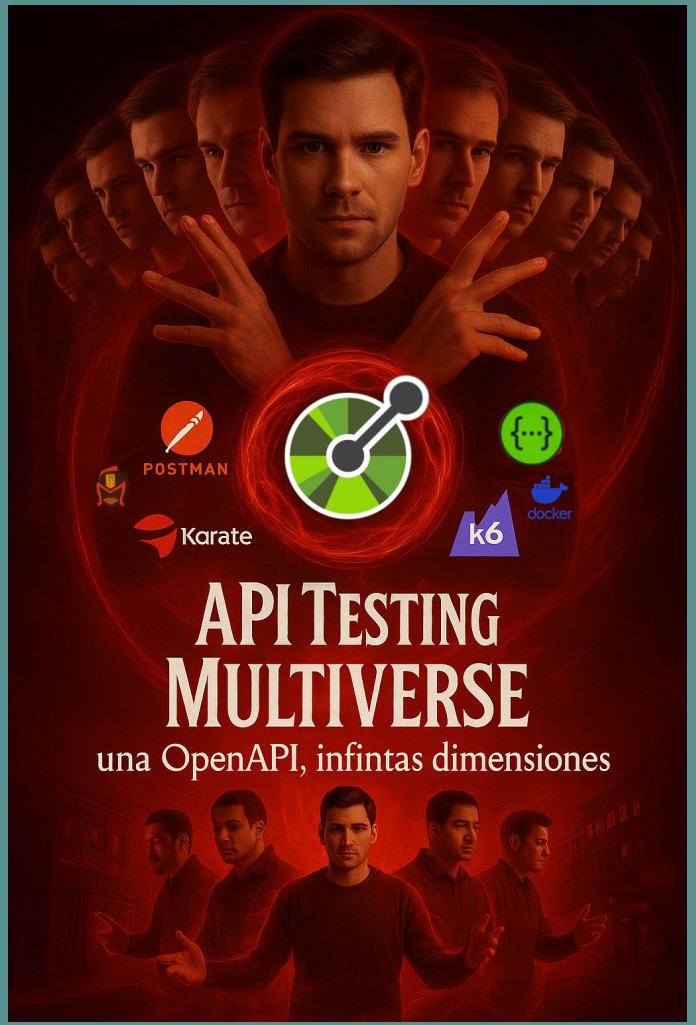

POSTMAN
Karate
k6
docker
API TESTING
MULTIVERSE
una OpenAPI, infintas dimensiones

SPEAKER

Gonzalo de Cos

Licenciado en Cs. de la Computación (UNS)
Más de 20 años de experiencia en Desarrollo de Software
Trabajé en distintas empresas y distintos roles: Developer, Tech. Leader, Data Engineer y Quality Engineer.

Actualmente, Staff QA Engineer en AVATURE

linkedin.com/in/gonzalo-de-cos/

gonzalo.de.cos@gmail.com

CONTEXTO

En el mundo actual, donde los sistemas tecnológicos están cada vez más conectados, las APIs (Application Programming Interfaces) constituyen la llave para que diferentes sistemas se comuniquen entre sí de manera eficiente.

No importa si hablamos de una app móvil, una plataforma web, un sistema bancario o una herramienta de inteligencia artificial: todo se apoya en APIs para intercambiar datos y funcionalidades.

Con el auge de la inteligencia artificial, las APIs cobran aún más protagonismo. Los modelos de IA necesitan integrarse con múltiples sistemas para acceder a datos, enviar resultados, o extender sus capacidades. Sin APIs bien diseñadas y robustas, esta integración sería lenta, frágil e insegura.

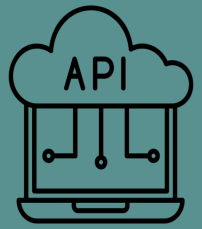
OBJETIVOS

Cuando construimos APIs, es imprescindible que sean confiables, cumplan exactamente con lo planificado y ofrezcan eficiencia y simplicidad al usarlas.

Un diseño adecuado es el pilar de estos resultados. Adoptar desde el comienzo una definición completa, robusta y bien documentada como eje del ciclo de vida de las APIs genera importantes ventajas en todas las etapas del desarrollo.

En esta charla vamos a ver cómo a partir de una estrategia de DESIGN FIRST, construyendo una especificación de OpenAPI lo más completa posible se puede lograr esto:

Construir APIs claras, colaborativas, validadas y listas para el futuro.



API FIRST

A la hora de construir una API existen diversas estrategias.
Entre ellas, subyacen las estrategias de ***DESING FIRST*** vs ***CODE FIRST***.

La estrategia de CODE FIRST puede resultar adecuada para proyectos pequeños o cuando se cuenta con poco tiempo y se necesitan obtener resultados muy rápidamente con requisitos cambiantes o aun no definidos completamente.

La estrategia de DESIGN FIRST, se basa en poner el foco en el diseño cuidadoso y completo de nuestra API. Las distintas áreas colaboran para lograr un diseño adecuado, consensuado desde el comienzo y luego a partir de allí proceder con la implementación de la misma.

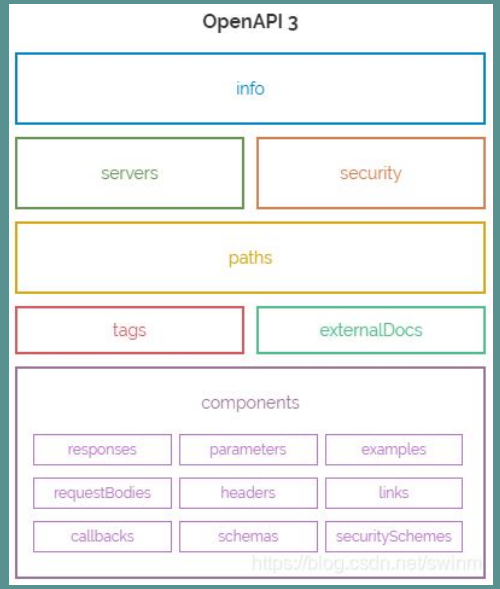
API FIRST y DESIGN FIRST se suelen usar para referenciar el mismo concepto, sin embargo vale mencionar que la estrategia de API FIRST va aún más allá y se centra en el diseño e implementación de una API como piedra angular de la construcción del sistema.
Diseñar y construir las APIs y luego el backend y el frontend del mismo.


Documentación automática
Validación de diseño
Alineación temprana
QA
Testing confiable
Contratos bien definidos
ESPECIFICACIÓN
Versionado
DEVELOPMENT
PRODUCTO
Testing temprano
Mejor gobernanza
Escalabilidad
Mantenibilidad
Mocking

OpenAPI Specification

- Una especificación OpenApi (**OpenAPI Specification - OAS**) es un estándar abierto para describir APIs RESTful de forma clara y estructurada.
- Es un formato (usualmente en JSON o YAML) que define cómo es una API: sus endpoints (rutas), métodos (GET, POST, etc.), parámetros, respuestas, modelos de datos, seguridad y más.
- Permite que tanto humanos como máquinas entiendan y usen la API sin ambigüedades.
- Facilita la generación automática de documentación, pruebas, mocks y herramientas de integración.
- Promueve la estandarización y mejora la comunicación entre desarrolladores, testers y usuarios de la API.

OpenAPI Specification

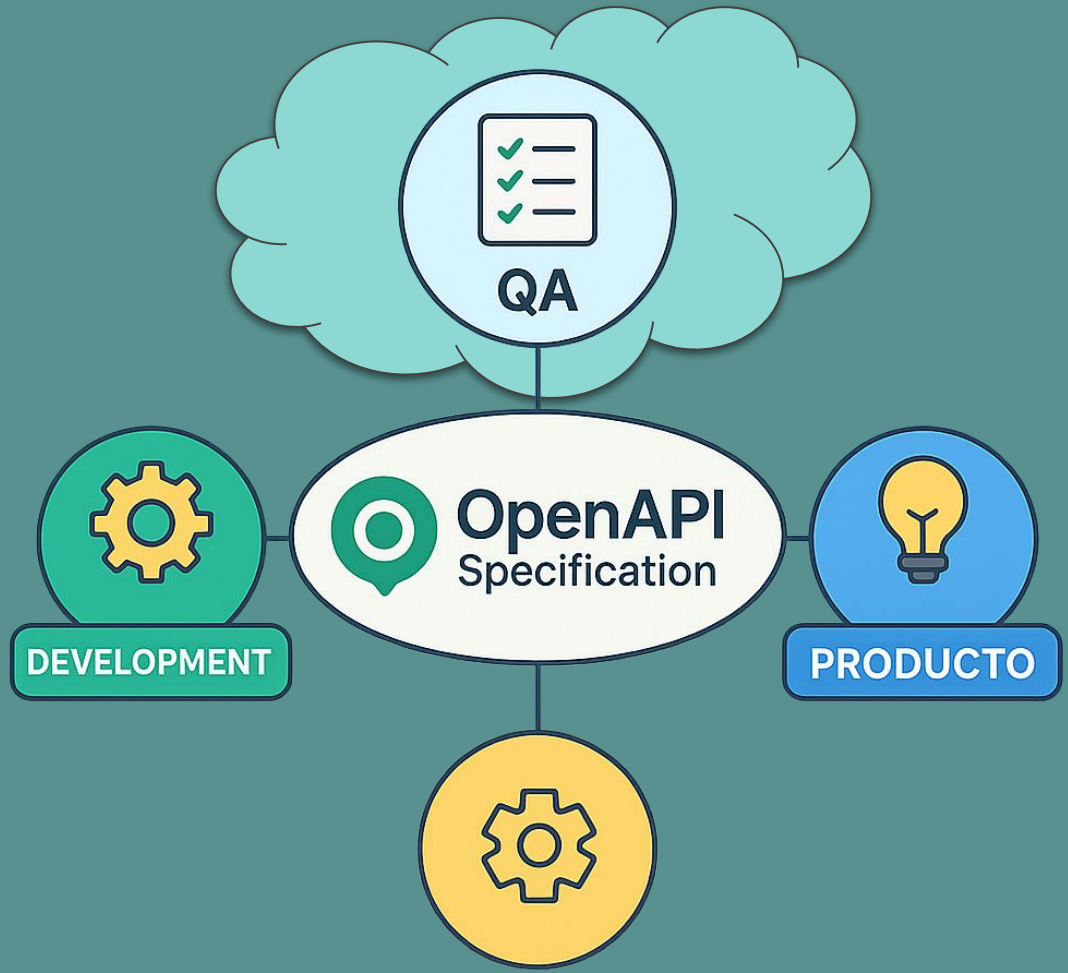



```yaml
openapi: 3.0.0
info:
  title: Star Wars API (SWAPI)
  description: API pública para acceder a datos del universo de Star Wars.
  version: 1.0.0
servers:
  - url: https://swapi.info/api

paths:
  /people:
    get:
      summary: Listar personajes
      responses:
        '200':
          description: Lista de personajes
          content:
            application/json:
              schema:
                type: object
                properties:
                  results:
                    type: array
                    items:
                      $ref: '#/components/schemas/Person'

  /people/{id}:
    get:
      summary: Obtener personaje por ID
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: Detalle del personaje
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Person'
        '404':
          description: No encontrado

  /films:
    get:
      summary: Listar peliculas
      responses:
        '200':
          description: Lista de peliculas
          content:
            application/json:
              schema:
                type: object
                properties:
                  results:
                    type: array
                    items:
                      $ref: '#/components/schemas/Film'
```

```yaml
paths:
  /key:                          # HTTP routes
    delete:                      # HTTP method
      parameters:
        - in: query              # Parameter location
          name: email            # Parameter name
          required: true
          schema:                # Parameter blueprint, contains parameter's type
            type: string
      responses:
        '200':                   # HTTP response status code
          content:
            application/json:    # HTTP response content type
              schema:            # Response blueprint, contains types
                properties:
                  status:
                    type: string
                type: object
```

Características principales

- **Modelos de datos:** Detalla las estructuras de datos para solicitudes y respuestas.
- **Definición de endpoints:** Describe rutas, métodos HTTP, parámetros y respuestas esperadas.
- **Autenticación y seguridad:** Incluye mecanismos para proteger las interacciones con la API.
- **Formatos de respuesta:** Define códigos de estado y ejemplos para manejar las respuestas correctamente.
- **Manejo de errores:** Explica los códigos y respuestas de error para facilitar la depuración.
- **Documentación interactiva:** Permite generar interfaces para explorar y probar la API
- **Versionado y compatibilidad:** Facilita el mantenimiento y evolución respetando las funcionalidades
- **Estandarización:** Promueve un lenguaje común para desarrolladores y stakeholders.
- **Formato legible por máquinas:** Se escribe en JSON o YAML, facilitando su comprensión por humanos y sistemas.


QA
OpenAPI
Specification
DEVELOPMENT
PRODUCTO

TESTING DE APIs

El testing de APIs es el proceso de verificar que una API se comporte como se espera. Esto incluye validar que devuelva los datos correctos, que responda adecuadamente ante casos límite, que maneje los errores de forma elegante y que cumpla con expectativas de rendimiento y seguridad.

Probar APIs directamente permite a los equipos de desarrollo y QA detectar errores antes, mejorar la eficiencia y mantener la estabilidad a medida que se agregan nuevas funcionalidades.

Beneficios del Shift Left Testing en APIs

✅ Más confianza en el producto
🚀 Ciclos de entrega más rápidos
🤝 Mejor colaboración entre equipos
🛠️ APIs estables, seguras y listas para escalar

Testear desde el principio = menos bugs, más valor, mayor calidad

TIPOS DE TESTING DE APIs

- **Testing de contrato**, verificar respuestas, codigo de errores, formato de datos
- **Testing Funcional**, verifica el correcto funcionamiento de la API en general, respuestas a diferentes entradas, errores inesperados, etc.
- **Testing Unitario**, de código boilerplate y lógica de negocio (desarrollo)
- **Testing de Integración y e2e,** del lado del consumidor de la API a través de workflows con múltiples API calls
- **Testing de Carga,** progresivo para comprobar límites y comportamiento general
- **Testing de Concurrencia,** verificando rate-limiting
- **Testing de Seguridad,** verificar autenticación y vulnerabilidades comunes

Ahora si….

Acompáñenme a explorar un multiverso de herramientas para testing automático de REST APIs

UNIVERSO SWAGGER

Donde comenzó todo...

Swagger, dentro del ecosistema OpenAPI, es mucho más que una simple herramienta de documentación:
es el contrato vivo de la API.

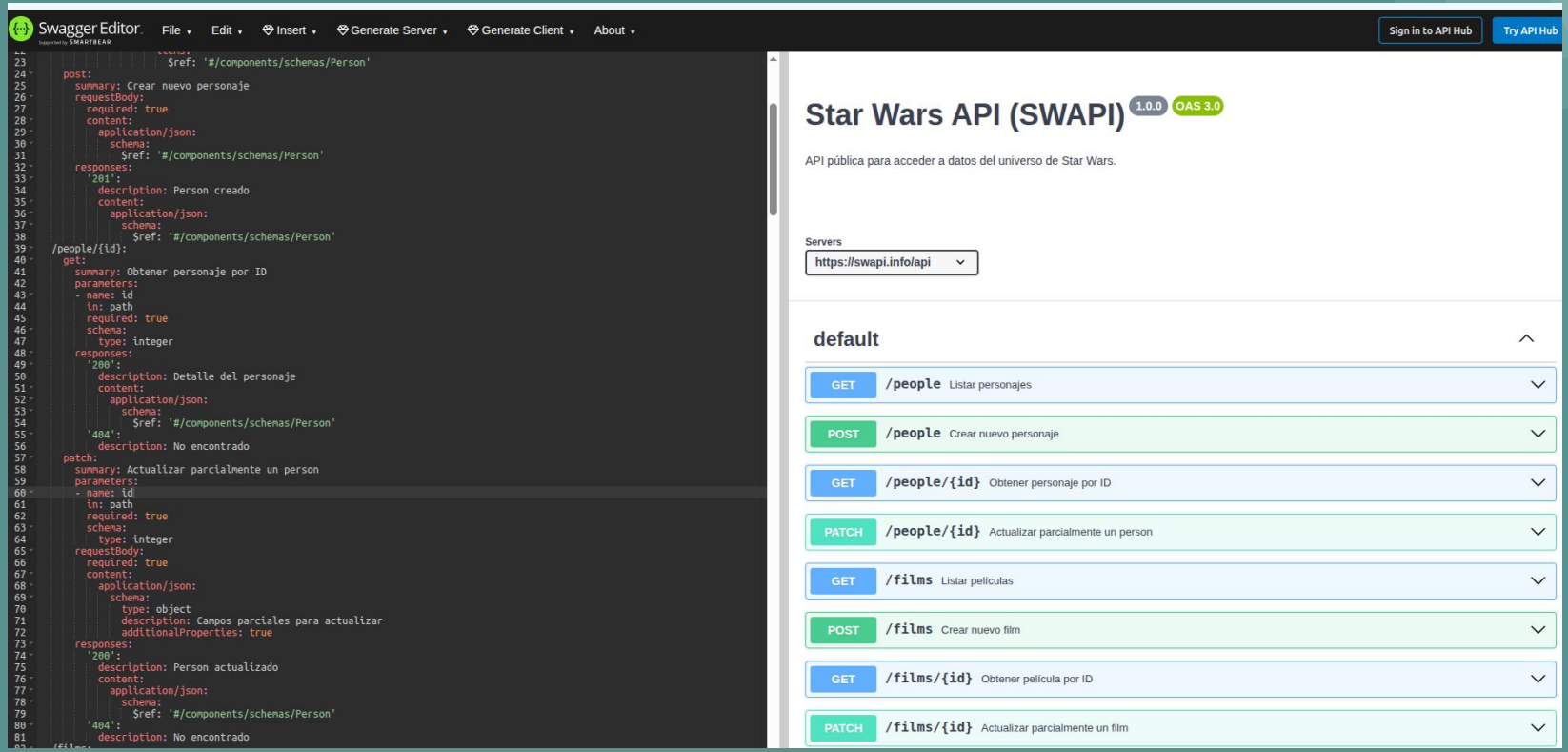
ECOSISTEMA:

SWAGGER EDITOR - SWAGGER UI - SWAGGER CODEGEN - SWAGGER PARSER y SWAGGER CORE

- Swagger Editor: edita y valida especificaciones OpenAPI
- Swagger UI: documentación interactiva para probar endpoints directamente en el navegador.
  Ideal para demos rápidas o validaciones con clientes
- Swagger Codegen: genera código cliente/servidor desde la spec

CON LICENCIA (SMART BEAR) MUCHAS MÁS HERRAMIENTAS PARA EL CICLO DE VIDA COMPLETO DE LA API

UNIVERSO SWAGGER



UNIVERSO POSTMAN

EL UNIVERSO MÁS USADO Y CONOCIDO

- POSTMAN es mucho más que testing exploratorio
- La herramienta más completa para trabajar con APIs
- Permite manejar todos los aspectos del ciclo de vida de APIs
- Provee edición de OpenApi inline, Workspaces, Tests, Reportes, POSTBOT IA, POSTMAN-CLI, integración CI/CD. POSTMAN CLOUD

- **VENTAJAS:**
  - DISPONIBLE ONLINE (SINCRONIZACIÓN) / FULL CLOUD
- **DESVENTAJAS:**
  - CREACIÓN DE CUENTA DE USUARIO OBLIGATORIA
  - NECESITA LICENCIA PARA TRABAJO EN PROYECTOS MEDIANOS Y GRANDES / CANTIDAD DE COLABORADORES

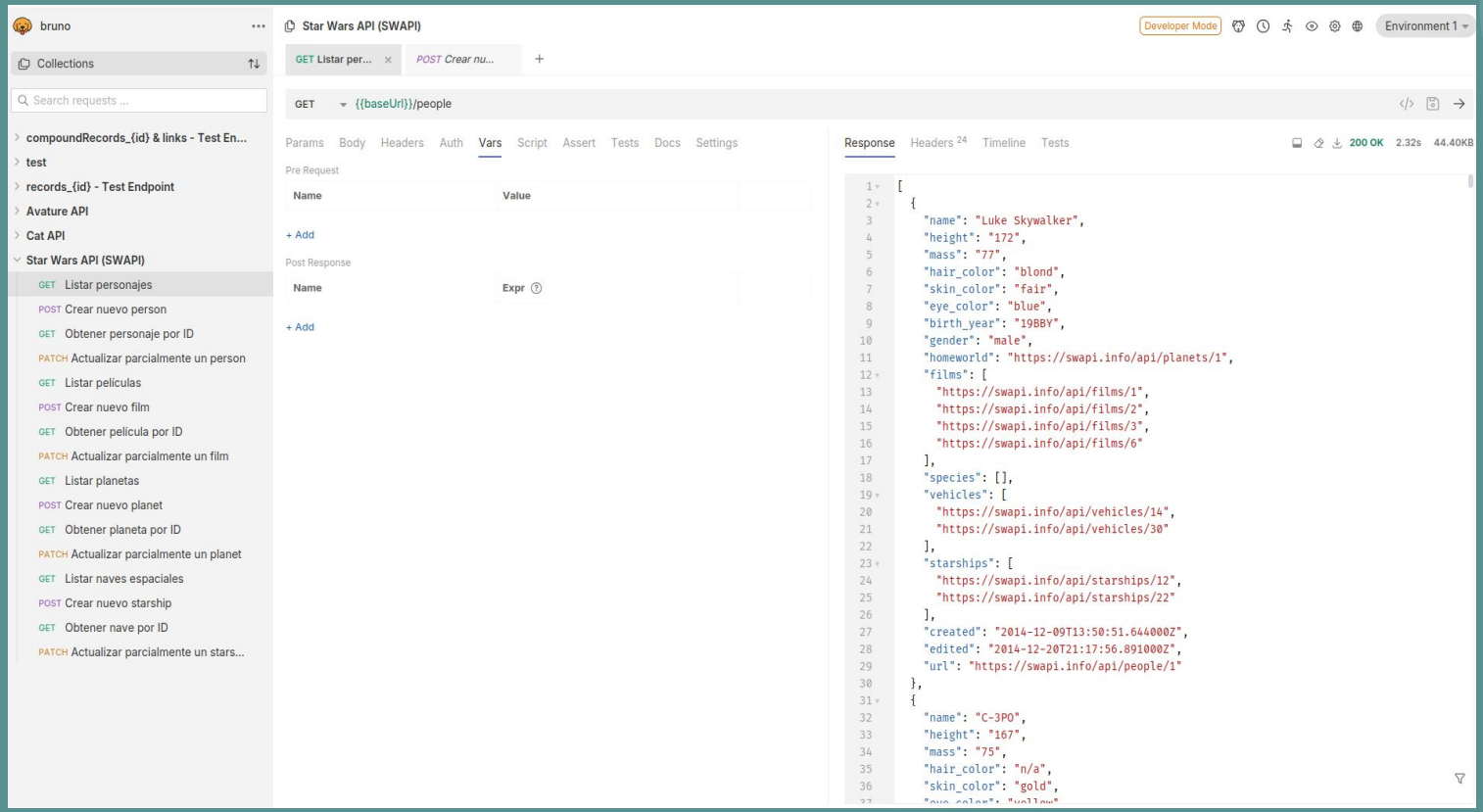
UNIVERSO BRUNO

MUY BUENA ALTERNATIVA A POSTMAN

- Herramienta OpenSource
- Tests como archivos YAML, git-friendly
- Rápido, liviano, offline, ideal para control de cambios
- Sin mocking ni performance, pero útil para escenarios básicos

- **VENTAJAS:**
  - VERSIONADO GIT EN TEXTO PLANO (Requests como archivos YAML)
  - RUTA CORTA DE APRENDIZAJE
  - IDEAL PARA USO LOCAL
- **DESVENTAJAS**:
  - PUEDE PARECER LIMITADO EN COMPARACIÓN A POSTMAN
  - UI CON BASTANTES LIMITACIONES (HISTORY - RUNNER)

UNIVERSO BRUNO



UNIVERSO PRISM-DREDD

Un stack liviano: mock + contrato

PRISM (STOPLIGHT)

- Mock server desde OpenAPI
- Función PROXY

DREDD

- Valida la implementación vs Contrato
- No genera código de SERVER ni de CLIENTE. TEST SUITE prefabricada a partir de OAS

UNIVERSO KARATE

Automatización de flujos complejos desde la especificación

- Compatible con specs OpenAPI y generación automática de tests en Gherkin
- Define flujos completos con validaciones complejas
- Soporta scripting, datos dinámicos, integración con CI/CD
- Mocking en lenguaje naural
- Permite performance testing
- Amplio soporte en plugins de integración con VsCode para generar mocks y tests y controlar la ejecución de los mismos

UNIVERSO KARATE

```
Scenario: create and retrieve a cat

Given url 'http://myhost.com/v1/cats'
And request { name: 'Billie' }
When method post
Then status 201
And match response == { id: '#notnull', name: 'Billie' }

Given path response.id
When method get
Then status 200
```

JSON is 'native' to the syntax

Intuitive DSL for HTTP

Payload assertion in one line

Second HTTP call using response data

```
Feature: payment service mock

Background:
* def id = 0
* def payments = {}

Scenario: pathMatches('/payments') && methodIs('post')
    * def payment = request
    * def id = ~~(id + 1)
    * payment.id = id
    * payments[id + ''] = payment
    * def response = payment

Scenario: pathMatches('/payments')
    * def response = $payments.*

Scenario: pathMatches('/payments/{id}') && methodIs('put')
    * payments[pathParams.id] = request
    * def response = request

Scenario: pathMatches('/payments/{id}') && methodIs('delete')
    * karate.remove('payments', '$.' + pathParams.id)
    * def response = ''

Scenario: pathMatches('/payments/{id}')
    * def response = payments[pathParams.id]
```

UNIVERSO StepCI

Generación automática de tests

- Define escenarios de pruebas legibles (yaml) directamente a partir de la spec
- Agnóstico al lenguaje de programación
- Soporta REST - SOAP - GraphQL, etc..
- Interfaz clara y ejecutable en CI/CD
- Ideal para QA funcionales en etapas tempranas del desarrollo

PERMITE GENERAR TESTS NO FUNCIONALES RÁPIDAMENTE:

- **FUZZ TESTING (SEGURIDAD)**
- **TESTS DE CARGA**
- **TESTS DE PERFORMANCE**

UNIVERSO StepCI

```yaml
version: "1.1"
name: My Workflow
env:
  host: jsonplaceholder.typicode.com
  resource: posts
components:
  credentials:
    example:
      basic:
        username: hello
        password: world
tests:
  example:
    steps:
      - name: Post a post
        http:
          url: https://${{env.host}}/${{env.resource}}
          method: POST
          auth:
            $ref: "#/components/credentials/example"
          headers:
            Content-Type: application/json
          json:
            title: Hello Step CI!
            body: This is the body
            userId: 1
          captures:
            id:
              jsonpath: $.id
          check:
            status: 201
            jsonpath:
              $.id:
                - eq: 101
                - isNumber: true
```

```yaml
version: "1.1"
name: Load Test
env:
  host: example.com
config:
  loadTest:
    phases:
      - duration: 2
        arrivalRate: 1
      - duration: 10
        arrivalRate: 2
    check:
      p99:
        - lte: 500
tests:
  example:
    steps:
      - name: GET request
        http:
          url: https://${{env.host}}
          method: GET
```

```
response_time:
  min: ..... 437
  max: ..... 500
  avg: ..... 453.3636
  med: ..... 446.5
  p95: ..... 498
  p99: ..... 500
steps:
  failed: ..... 0
  passed: ..... 22
  skipped: ..... 0
  errored: ..... 0
  total: ..... 22
tests:
  failed: ..... 0
  passed: ..... 22
  total: ..... 22
checks
  p99: ..... pass
rps: ..... 1.570531
iterations: ..... 22
duration: ..... 14008

Workflow passed after 14.008s
```

```yaml
version: "1.1"
name: Schema Testing
tests:
  example:
    steps:
      - http:
          url: https://jsonplaceholder.typicode.com/posts/1
          method: GET
          check:
            schema:
              $ref: "openapi.yml#/components/schemas/Post"
```

UNIVERSO k6

Testing de Performance desde la especificación

- Scripts JS generados desde OpenAPI
- Tests de carga, estrés, concurrencia (SOAK, PEAK, SPIKE)
- Métricas, umbrales y reportes integrables a CI/CD
- Levantar y ejecutar tests de performance rápidamente desde docker-compose
- Provee tool Open Source para generar automáticamente cliente y ejemplos de tests en TypeScript

UNIVERSO k6

JavaScript

```javascript
import http from 'k6/http';

export default function () {
  const payload = JSON.stringify({
    name: 'lorem',
    surname: 'ipsum',
  });
  const headers = { 'Content-Type': 'application/json' };
  http.post('https://quickpizza.grafana.com/api/post', payload, { headers });
}
```

LOAD TESTING

STRESS TESTING

```javascript
import http from 'k6/http';
import { sleep } from 'k6';

export const options = {
  // Key configurations for Stress in this section
  stages: [
    { duration: '10m', target: 200 }, // traffic ramp-up from 1 to a higher 200 users over 10 minutes.
    { duration: '30m', target: 200 }, // stay at higher 200 users for 30 minutes
    { duration: '5m', target: 0 }, // ramp-down to 0 users
  ],
};

export default () => {
  const urlRes = http.get('https://quickpizza.grafana.com');
  sleep(1);
  // MORE STEPS
```

UNIVERSO ZAP SECURITY

Seguridad automatizada

- ZAP (Zed Attack Proxy) permite escanear APIs REST en busca de vulnerabilidades
- El script de escaneo permite ejecutar pruebas automáticas desde una OAS
- Configurable por línea de comandos, ideal para CI/CD
- Detecta errores comunes de seguridad como inyecciones, exposición de datos sensibles, etc.

https://github.com/zaproxy/zaproxy

https://www.zaproxy.org/blog/2017-06-19-scanning-apis-with-zap/

```
docker pull owasp/zap2docker-weekly
docker run -t owasp/zap2docker-weekly zap-api-scan.py -t \
    https://www.example.com/openapi.json -f openapi
```

COMPARATIVA DE UNIVERSOS

Ventajas comparativas por enfoque

- **Swagger**: simplicidad, documentación viva, interfaz de prueba directa.
- **Postman**: ecosistema robusto, OpenApi+Mocks+Workspaces+Tests+IA+Integración en nube
- **Bruno**: enfoque local, control de versiones. Testing exploratorio
- **Prism + Dredd**: Stack liviano para contrato y mocking
- **Karate**: Tests declarativos. Documentación Viva. Mocking en lenguaje natural. Ideal para flujos complejos y CI
- **StepCI**: Simplicidad y generación declarativa de tests Funcional+Performance+CI
- **k6**: foco exclusivo en performance y métricas
- **ZAP:** Foco en seguridad. Búsqueda de vulnerabilidades mas comunes

MULTIVERSOS

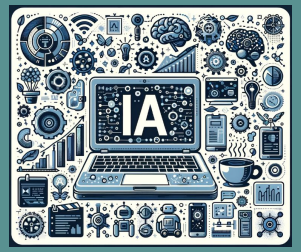
Explorar y mezclar herramientas

- Analizar combinaciones posibles para obtener nuevas dimensiones según las necesidades del proyecto.
- Elegir el entorno adecuado según presupuesto, objetivos y equipo
- Por ej: Postman + Dredd / Bruno + k6 + ZAP / Prism + Karate
- Cada combinación, una estrategia diferente

HERRAMIENTAS FUNDAMENTALES DOCKER - CI/CD

Automatización de pruebas con Docker e integración en pipelines

- Usar DOCKER para levantar rápidamente entornos de prueba con las herramientas vistas
- Infraestructura reproducible, portable y versionada
- Despliegue continuo y dinámico al cambiar la OpenApi Specification
- Integración de pruebas al pipeline con ejecución automática en GitHub Actions o Jenkins
- La idea es contar con pruebas dinámicas rápidamente como paso de validación del ciclo de desarrollo



INTELIGENCIA ARTIFICIAL

IA como copiloto

El uso de la IA es clave como ayuda para la implementación y automatización en cada etapa del ciclo de vida de las APIs:

- Generar OpenApi Specification (o partes de la misma)
- Generar Overlays
- Generar escenarios de pruebas para Karate
- Generar docker-compose files para levantar entornos
- Generar código para incluir los tests en pipelines
- Generar scripts específicos de k6
- Sugerir pruebas de flujo (workflows)
- Generar tests en Postman
- Generación de mocks
- Validación de specs
- Generar documentación
- Generar de casos de pruebas
- Generar de entornos de prueba
- Generación y debugging de código de tests
- Describir herramientas y usos específicos
- Ejemplos prácticos, ventajas y desventajas

Ahorro de tiempo, mayor cobertura, más productividad

CONCLUSIONES

Lecciones clave del viaje

- **OpenAPI** como base central de automatización.
  - Mas allá del estado del proyecto, usar OpenApi como especificación da innumerables ventajas
- **Hay vida más allá de Postman**. Existe un multiverso de herramientas distintas.
  - https://tools.openapis.org/ (Mas de 1500 herramientas para automatizar procesos de APIs)
- **Shift Happens**. Mocking y contract testing permiten un verdadero shift-left testing.
  - QA involucrado desde el arranque del proyecto, validando y elevando la calidad del producto
- **La IA** debe integrarse al día a día de QA. El uso de la IA hoy no es opcional, es esencial.
  - Agregar valor rápidamente en cada paso del proceso de desarrollo

¡Gracias por acompañarme a explorar el multiverso!

¡Happy Testing!

linkedin.com/in/gonzalo-de-cos/

gonzalo.de.cos@gmail.com

https://github.com/gonzadc/api-testing-multiverse