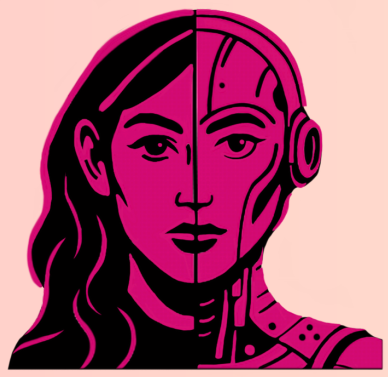
PRUEBAS DE API

Interfaz de Programación de Aplicaciones

Guía Rápida de REST API

Mujeres Testing - Latam



Creado por Fransa J. Aravena

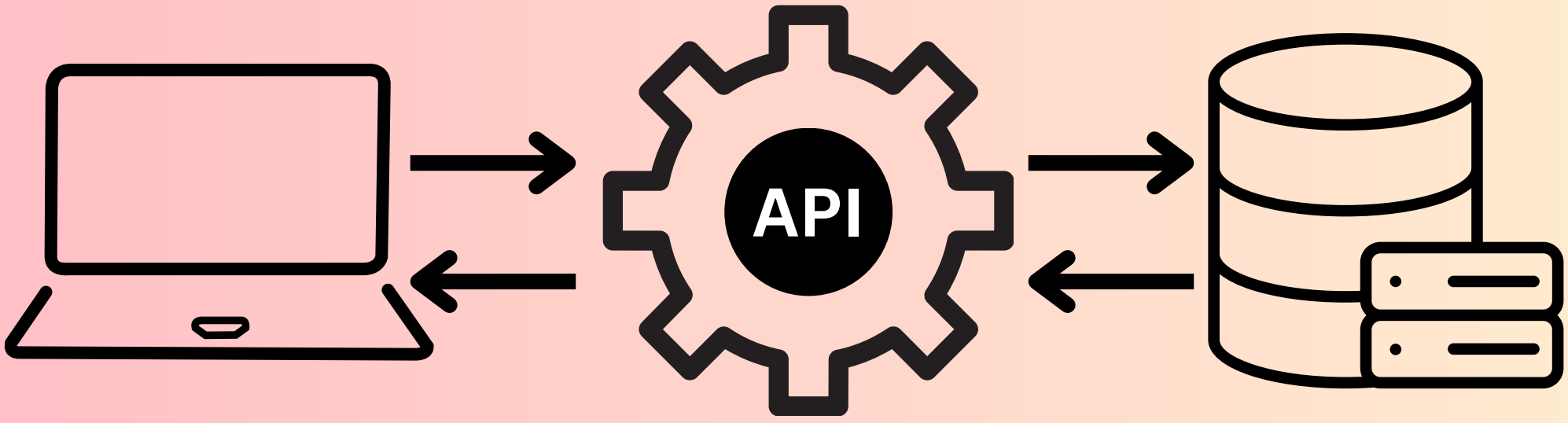
¿QUÉ ES UNA API?

API (Interfaz de Programación de Aplicaciones)

Es un **conjunto** de **reglas** y **protocolos** que permite que dos aplicaciones diferentes se comuniquen entre sí. Es como un puente conector de sistemas para intercambiar información o ejecutar acciones.

OBJETVOS

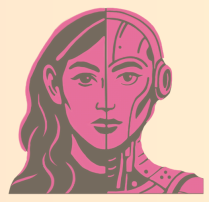
- **Conectar** sistemas.
- **Facilitar** la integración entre aplicaciones, servicios o dispositivos.
- **Reutilizar** funcionalidades sin tener que reprogramarlas desde cero.
- **Estandarizar** la comunicación entre software.



Creado por Fransa J. Aravena

TIPOS DE API

Característica	REST	SOAP	GraphQL	gRPC
Modelo baso en...	Recursos y métodos HTTP	Mensajes XML estructurados y contratos rígidos (WSDL).	Consultas (queries y mutations) definidas por el cliente.	Llamadas a procedimientos remotos (RPC).
Formato de datos	Generalmente JSON, aunque soporta XML y otros.	Sólo XML.	JSON (estándar).	Protocol Buffers (binario, eficiente).
Flexibilidad	Medio: devuelve el recurso completo, a veces más datos de los necesarios.	Bajo: estructura fija y estricta.	Muy alta: el cliente pide solo los campos que necesita.	Medio: contratos predefinidos en archivos .proto.
Facilidad de pruebas	Alta: fácil de probar con Postman, Playwright, curl.	Media: requiere validar XML y WSDL.	Media-Alta: necesita construir queries, pero es muy usado en frontend.	Media-Baja: requiere tooling especial para compilar protos.
Casos de uso	APIs web, microservicios, CRUD de aplicaciones.	Sistemas empresariales críticos (banca, seguros, legacy).	Apps web/móviles que optimizan consumo de datos (ej. GitHub, Facebook).	Sistemas distribuidos de alta performance, streaming, IoT, microservicios internos.



Creado por Fransa J. Aravena

PRUEBAS MÁS RELEVANTES

Validar los códigos de estado HTTP

Revisa el cuerpo de la respuesta
Tipos de datos correctos, estructura consistente, campos obligatorios presentes.

Confirma validación de entradas
Con datos inválidos o incompletos, campos vacíos, demasiado largos.

Prueba la seguridad
Endpoints que requieren autenticación, restringidos. Tokens caducados o inválidos.

Evalúa consistencia de mensajes de error
Mensajes claros y útiles. Que no revelen información sensible.

Revisa headers y metadata
Authorization, Content-Type y Cache-Control

Prueba el rendimiento y límites
Confirma comportamiento en caso de muchas requests y tiempos.

Valida compatibilidad y versionado
Que endpoint exponga su versión. Cambios no rompen clientes antiguos.

Creado por Fransa J. Aravena

CÓDIGOS DE ESTADO HTTP

Los más relevantes son:

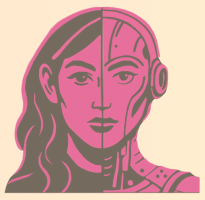
Código	Significado	Descripción
200	OK	Petición exitosa.
201	Created	Recurso creado con éxito.
400	Bad Request	Petición mal formada o con datos inválidos.
401	Unauthorized	Falta autenticación (token/API key inválido).
403	Forbidden	Autenticado, pero sin permisos.
404	Not Found	El recurso no existe.
500	Internal Server Error	Error genérico en el servidor.
502	Bad Gateway	El servidor actuó como proxy y recibió una respuesta inválida.
503	Service Unavailable	Servicio no disponible (mantenimiento o sobrecarga).
504	Gateway Timeout	El servidor no recibió respuesta a tiempo.

Creado por Fransa J. Aravena

REST API

CONCEPTOS CLAVE

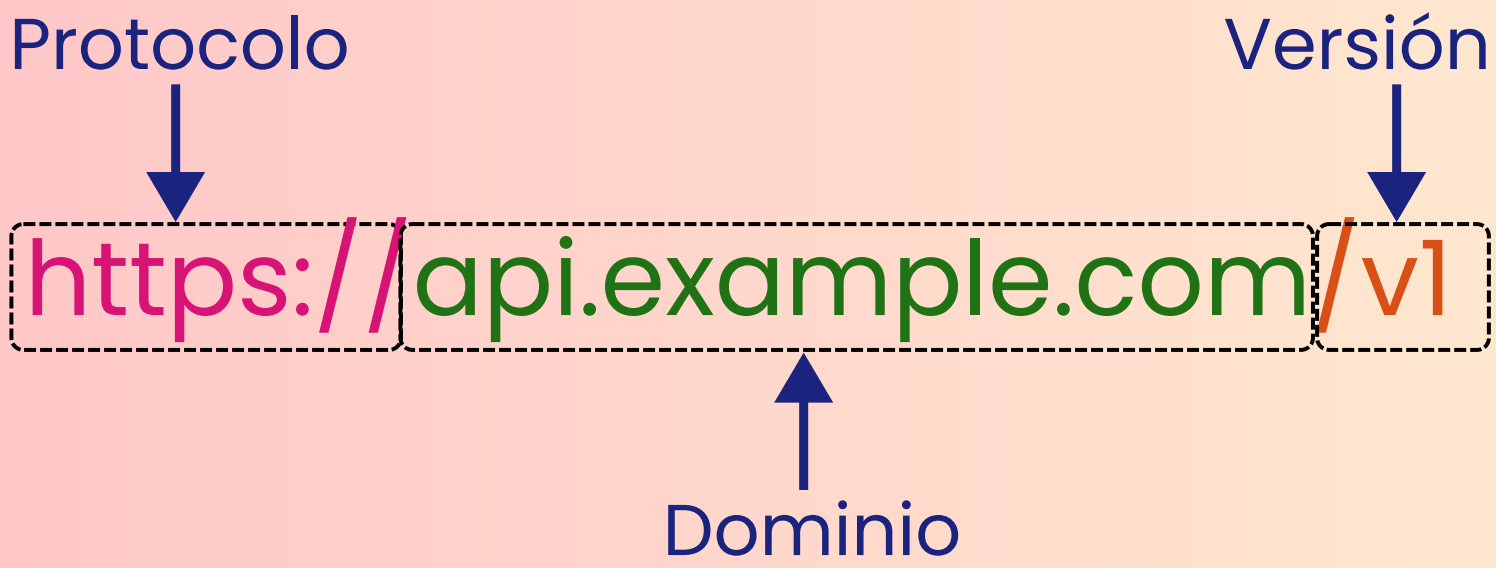
Creado por Fransa J. Aravena



URL BASE

URL BASE:
Es el punto de entrada de la API, a partir de esta se construyen los endpoints agregando rutas específicas de recursos.

EJEMPLO:



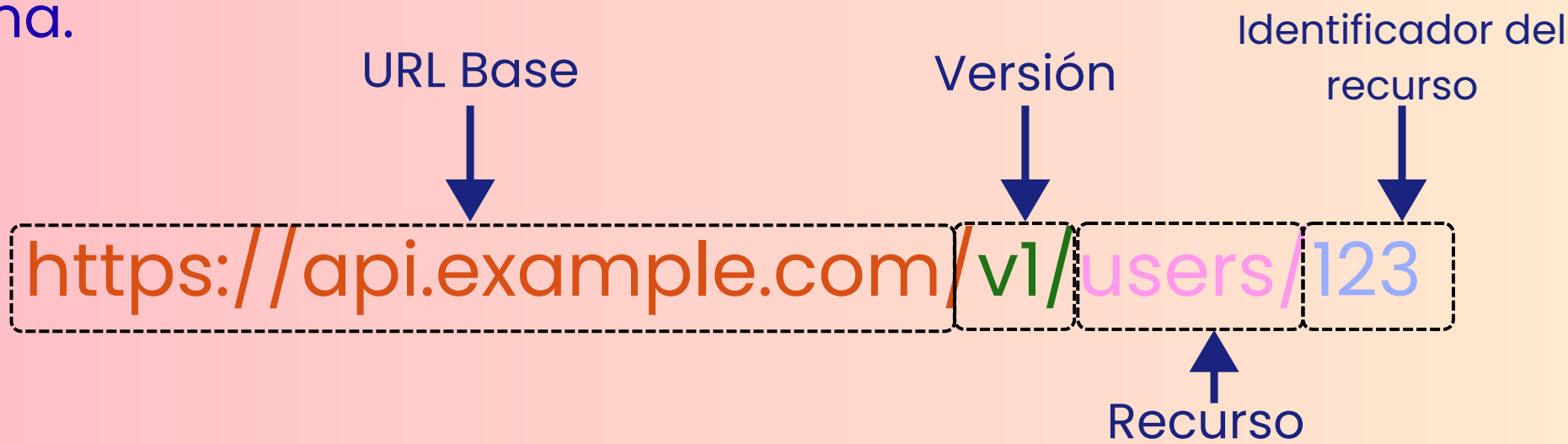
Importancia:

- **Consistencia**: Todos los endpoints comparten misma raíz.
- **Facilidad** de configuración: En Postman o Playwright, definiendo una vez la URL base y luego agregando ruta específica, *evitando repetición*.
- **Versionado**: Para que clientes usen distintas versiones sin romper compatibilidad, o diferenciar la versión estable de la nueva, entre otros.

Creado por Fransa J. Aravena

ENDPOINT

Un endpoint es la dirección de un recurso en la API. Gracias a los endpoints, podemos crear, leer, actualizar o eliminar datos en un sistema.



Método	Endpoint	Descripción
GET	https://api.example.com/v1/**users**	Lista de usuarios
GET	https://api.example.com/v1/**users/123**	Detalles de usuario {123}
POST	https://api.example.com/v1/**users**	Crea un nuevo usuario, indicando metada en body
PUT	https://api.example.com/v1/**users/123**	Actualizar usuario completo
PATCH	https://api.example.com/v1/**users/123**	Actualización parcial de un usuario
DELETE	https://api.example.com/v1/**users/123**	Eliminar usuario

Creado por Fransa J. Aravena

MÉTODOS

GET

Obtener datos

Propósito
Recuperar información del servidor sin modificarla.

Características

- *Idempotente*: Múltiples llamadas producen el mismo resultado.
- *Seguro*: No modifica el estado del servidor.
- *Sin cuerpo*: No debe incluir body.

Ejemplo

GET /api/users

POST

Crear recursos

Propósito
Crear nuevos recursos o procesar datos.

Características

- *No idempotente*: Múltiples llamadas pueden crear recursos duplicados.
- *No seguro*: Modifica el estado del servidor.
- *Con cuerpo*: Incluye datos en el body.

Ejemplo

POST /api/users

Body
{
"name": "María López",
“email": "maria@e.com",
"role": "user"
}

PUT

Actualizar completamente

Propósito
Reemplazar completamente un recurso existente.

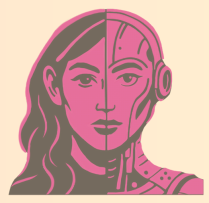
Características

- *Idempotente: Múltiples llamadas producen el mismo resultado.*
- *Reemplazo total: Todos los campos del recurso se actualizan.*
- *Puede crear: Si recurso no existe, puede crearlo.*

Ejemplo

PUT /api/users/123

Body
{
"name": "María update",
"email": "m.new@e.com",
"role": "admin", "status":
"active"
}



Creado por Fransa J. Aravena

MÉTODOS

PATCH

Actualizar parcialmente

Propósito

Modificar solo campos específicos de un recurso

Características

- *Puede ser Idempotente*: Dependiendo de implementación.
- *Actualización* parcial: Sólo campos enviados.
- *Más eficiente:* Para cambios menores

Ejemplo

PATCH /api/users/123

Body
{
"email":
"m.new2@e.com"
}

DELETE

Eliminar

Propósito

Eliminar un recurso del servidor

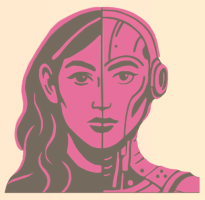
Características

- *Idempotente*: Múltiples llamadas tienen el mismo efecto.
- *Sin cuerpo*: Generalmente no incluye body
- *Destructivo*: Elimina permanentemente el registro.

Ejemplo

DELETE /api/users/123

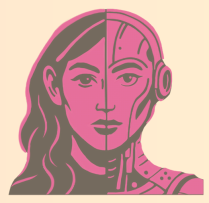
Creado por Fransa J. Aravena



METADATA

Información adicional que acompaña a la petición, pero que no es parte directa de los datos del recurso. Describen como debe procesarse la petición.

Header	Función	Ejemplo
Authorization	Envía credenciales para autenticación (token, Basic Auth, etc.).	Authorization: Bearer eyJhbGciOi...
Cache-Control	Indica cómo manejar la caché.	Cache-Control: no-cache
Postman-Token	Generado por Postman para diferenciar requests y evitar caché.	<calculated when request is sent>
Host	Especifica el dominio del servidor al que va la petición.	Host: api.example.com
User-Agent	Identifica el cliente que hace la request.	User-Agent: PostmanRuntime/7.45.0
Accept	Indica los tipos de contenido aceptados por el cliente.	Accept: application/json
Accept-Encoding	Lista de compresiones soportadas para la respuesta.	Accept-Encoding: gzip, deflate, br
Connection	Define si la conexión se mantiene abierta o se cierra tras la respuesta.	Connection: keep-alive
api_key	Header personalizado de autenticación usado por algunas APIs.	api_key: special-key



Creado por Fransa J. Aravena

HERRAMIENTAS DE PRUEBA

Pruebas Manuales

Insomnia

swagger

POSTMAN

httpie

Thunder Client

Hoppscotch

Pruebas Automatizadas

Playwright

REST ASSURED

POSTMAN

Karate Labs

cypress

Pruebas de Performance

k6

BlazeMeter

Artillery

LOAD RUNNER

LOCUST

APACHE JMeter™

Creado por Fransa J. Aravena

EJEMPLOS DE TESTING AUTOMATIZADO

Validando 200 OK y user id

Herramienta: Postman con Javascript

GET https://api.example.com/v1/users/123

Params · Authorization · Headers (9) · Body · Scripts · Settings

Pre-request

Post-response

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response has user data", function () {
    const json = pm.response.json();
    pm.expect(json).to.have.property("id", 123);
});
```

Consejo: Usar variables para configurar URL base en un sólo lugar y luego llamarlo en endpoint con "{{URL}}".

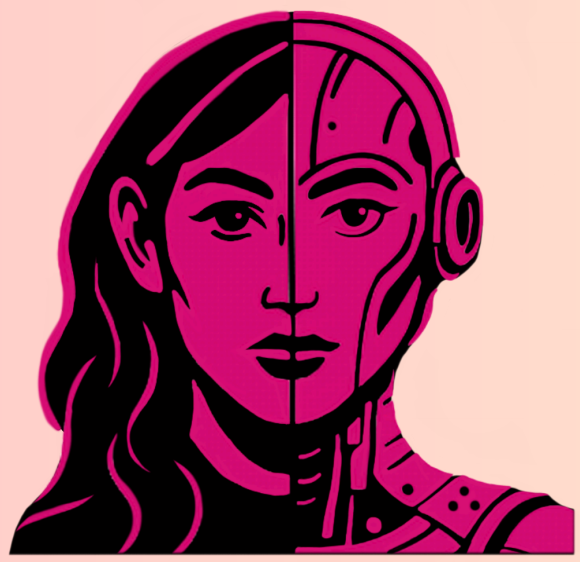
Herramienta: Playwright con Typescript

```
import { test, expect } from '@playwright/test';

test('GET /users/123 returns 200', async ({ request }) => {
  const res = await request.get('https://api.example.com/v1/users/123');
  expect(res.status()).toBe(200);

  const body = await res.json();
  expect(body).toHaveProperty('id', 123);
});
```

Consejo: configurar URL base en playwright.config.ts

Conectando y apoyando a Mujeres en LATAM para crecer juntas en el camino del Testing

Mujeres Testing - Latam



Creado por Fransa J. Aravena

Septiembre, 2025