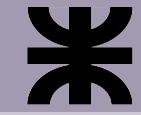


Rest API

Introducción a las APIs

Qué es una API: Una API (Application Programming Interface, es una interfaz de programación de aplicaciones) es un conjunto de reglas y definiciones que permiten que dos aplicaciones se comuniquen entre sí. Usadas para interactuar con bases de datos, servicios web, sistemas operativos, etc.

Una API (es una interfaz de programación de aplicaciones)

Ejemplo : cuando usas una app de clima en tu celular, esta no mide el clima, sino que consulta una API (como la de OpenWeather) que devuelve los datos.

Tipos de APIs:

APIs de sistema (ej: llamadas al sistema operativo).

APIs de librerías (ej: funciones de una librería de matemáticas).

APIs web (las que vamos a ver en REST).

¿Qué es REST?

REST (Representational State Transfer): es un estilo arquitectónico, no un estándar formal para diseñar APIs.

Define cómo deben diseñarse las APIs para ser fáciles de usar, escalables y predecibles.

No es un estándar, sino un conjunto de principios.

Se basa en el protocolo HTTP, que ya usamos en la web.

Principios de REST

Cliente-Servidor: el cliente (navegador, app móvil, etc.) y el servidor (API) están separados.

Stateless: cada request debe tener todo lo necesario (ej: token de autenticación, parámetros). El servidor no recuerda “el estado” entre llamadas.

Cacheable: las respuestas pueden guardarse en caché para optimizar rendimiento.

Interfaz uniforme: siempre se usan recursos y métodos HTTP estándar (GET, POST, PUT, DELETE).

JSON (JavaScript Object Notation) como formato de intercambio de datos

JSON es el más usado porque es simple, ligero y legible.

Datos estructurados de manera simple y entendible.

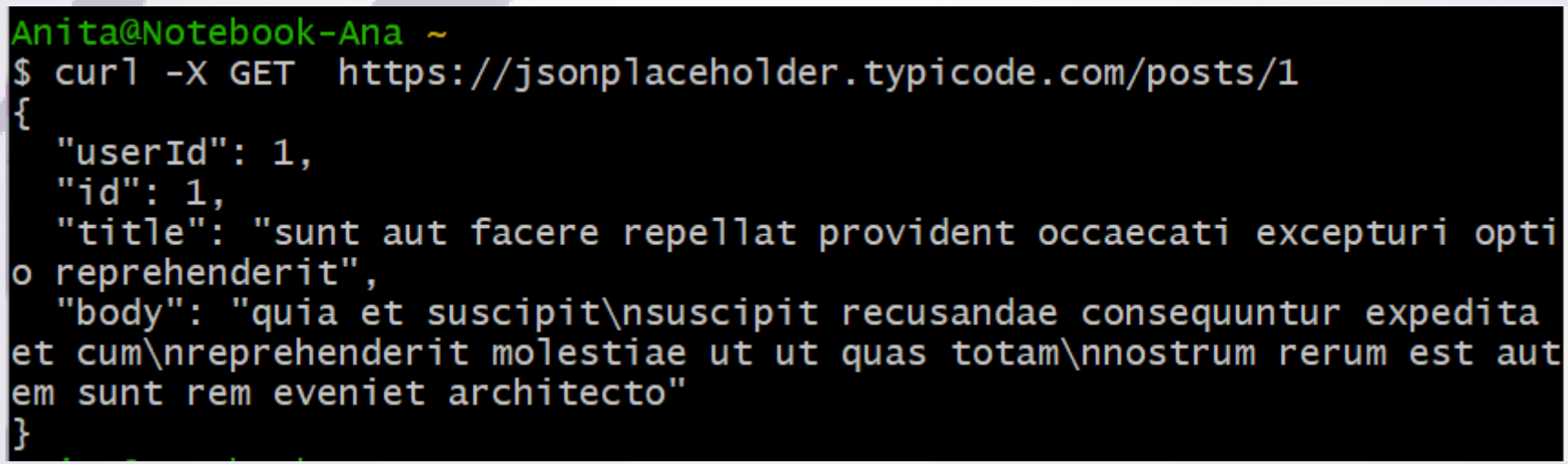
Ejemplo:

```
{
  "id": 101,
  "nombre": "Técnicas Digitales III",
  "alumnos": 15
}
```

Otros formatos: XML, YAML, pero JSON domina por compatibilidad con JavaScript y frameworks modernos.

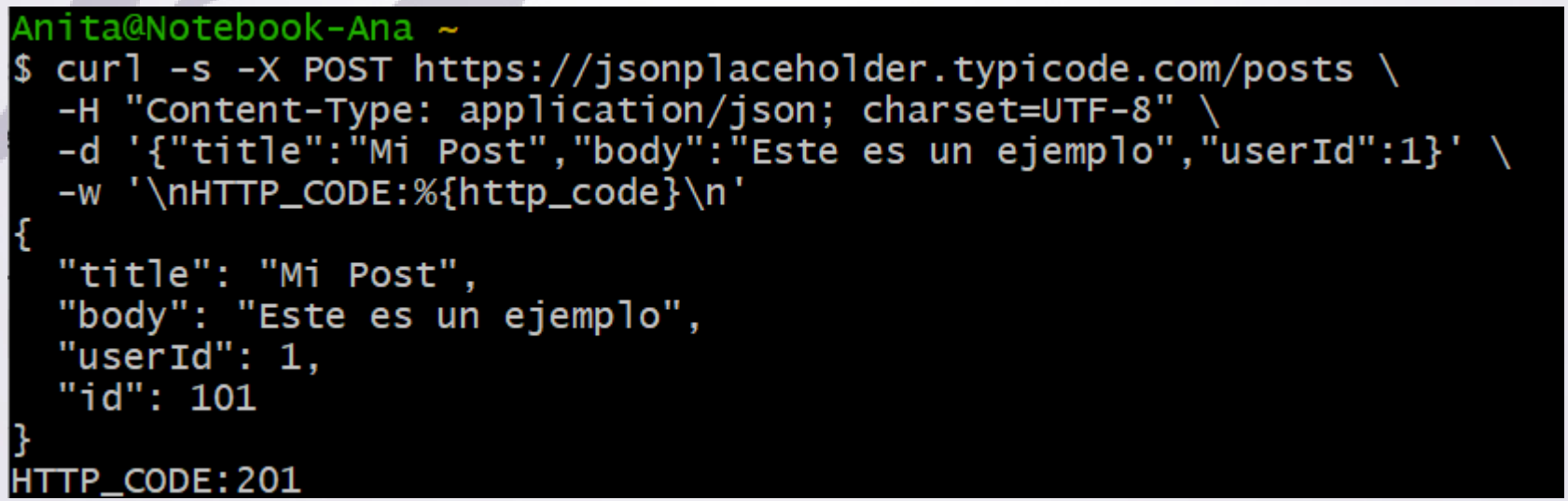
Ejemplo: con API pública de prueba JSONPlaceholder.
curl es una herramienta de consola para hacer peticiones HTTP

```
$ curl -X GET  https://jsonplaceholder.typicode.com/posts/1
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio
reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita
et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est
autem sunt rem eveniet architecto"
}
```



Ejemplo:

```
$ curl -s -X POST https://jsonplaceholder.typicode.com/posts \
  -H "Content-Type: application/json; charset=UTF-8" \
  -d '{"title":"Mi Post","body":"Este es un ejemplo","userId":1}' \
  -w '\nHTTP_CODE:%{http_code}\n'
{
  "title": "Mi Post",
  "body": "Este es un ejemplo",
  "userId": 1,
  "id": 101
}
HTTP_CODE:201
```



Buenas prácticas en REST API

Usar nombres consistentes en recursos: /users, /products.

Métodos HTTP correctos:

- GET /users → obtener usuarios
- POST /users → crear usuario
- PUT /users/1 → actualizar usuario
- DELETE /users/1 → borrar usuario

Códigos de estado:

200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 404 Not Found.

Seguridad: autenticación y autorización con JWT u OAuth2.

Versionado: incluir /v1/ o similar en la URL.

Ventajas y desventajas de REST

Ventajas:

- Fácil de entender.
- Usa HTTP estándar.
- Compatible con múltiples lenguajes.

Desventajas:

- Ineficiente para consultas complejas (trae más datos de los necesarios).
- Puede requerir varias llamadas.
- Alternativas: GraphQL (consultas personalizadas), gRPC (alto rendimiento en microservicios).

Request en Flask

• Flask es un framework para desarrollo web escrito en Python

• El contenido que un cliente web envía al servidor va almacenado en la Request.servidor va almacenado en la Request.

• En Flash la Request se representa mediante un objeto y para poder utilizar lo debemos importar:

from flask import Flask, request

El objeto request representa lo que envía el cliente.

Tipo de request

- El objeto request se puede utilizar para saber el tipo de petición que nos hace el cliente

- GET, POST, DELETE …

- Esto lo sabemos con el atributo . Method

Métodos principales:

- GET: se utiliza para traer información de un servidor

- POST: se utiliza para enviar datos al servidor

- PUT: actualizar datos.

- DELETE: borrar.

Ejemplo:

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/enviar', methods=['POST'])
def recibir():
    datos = request.json
    return f"Recibido: {datos['nombre']}"
```