API BASICS

Erick Santillan
feat.
Curies Community

Introducción

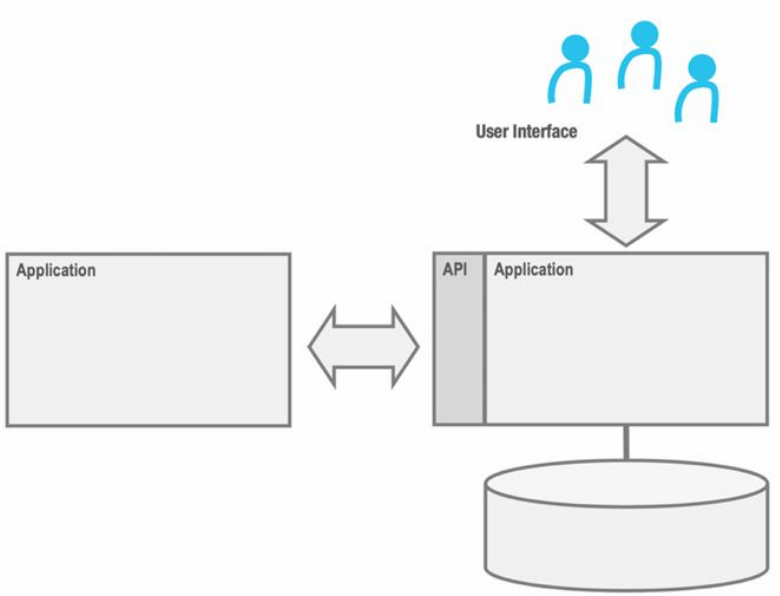
(Aquí es donde me presento)

¿Qué es una API?

Cuando los usuarios interactúan con una aplicación, lo hacen a través de su interfaz de usuario (UI). Sin embargo, cuando las aplicaciones necesitan comunicarse entre sí, dependen de una Interfaz de Programación de Aplicaciones (API). Las APIs están diseñadas específicamente para habilitar interacciones máquina-a-máquina, permitiendo que los sistemas intercambien información y ejecuten operaciones de manera eficiente tanto a nivel técnico como a nivel de procesos de negocio. (**Bayer, T., & Polley, T. (2025).** ***The API Gateway Handbook***)

Una API (Application Programming Interface o Interfaz de Programación de Aplicaciones) es un conjunto de reglas y protocolos que permite que dos aplicaciones se comuniquen entre sí. (**Gemini 3**)

Una API es simplemente una *interfaz* que un software ofrece para que otro software interactúe con él. (**Microsoft Copilot**)



¿Para qué sirve una API?

Como su definición nos lo dice, para que una aplicación se pueda **comunicar** con otra(s).

Imaginemos que estamos haciendo un programa que se quisiera comunicar con ChatGPT o con Google Drive, ¿Como podriamos hacerle? No podemos ponerle unas “manos” para que pueda acceder al chat o al drive.

En esos casos es cuando las APIs son clave.

Ejemplos

🌐 API WEB

Interfaz accesible por internet mediante protocolos como HTTP o HTTPS. Permite que aplicaciones se comuniquen con servicios remotos. **Ejemplo:** APIs REST, SOAP, GraphQL.

🖥️ API DE SISTEMA OPERATIVO

Conjunto de funciones que el sistema expone para que las aplicaciones interactúen con recursos internos como archivos, procesos, memoria o hardware. **Ejemplo:** API de archivos (fs), API de procesos, API de notificaciones.

📚 API COMO BIBLIOTECA DE CÓDIGO

Es la interfaz que ofrece una librería de código para que el desarrollador use sus funciones, clases o módulos. **Ejemplo:** Lodash, React, NumPy, OpenCV.

🔧 API PARA HARDWARE

Permite que el software se comunique con dispositivos físicos: sensores, cámaras, Bluetooth, robots, etc. **Ejemplo:** Web Bluetooth, API del acelerómetro, API de cámaras.

⚙️ API DE COMUNICACIÓN ENTRE PROCESOS (IPC)

Mecanismos que permiten que dos procesos intercambien datos sin usar la web. Incluye sockets, colas de mensajes, memoria compartida, pipes, RPC. **Ejemplo:** ZeroMQ, Redis Pub/Sub, sockets locales.

Vamos bien? O un diagrama? Otro ejemplo? Algo?

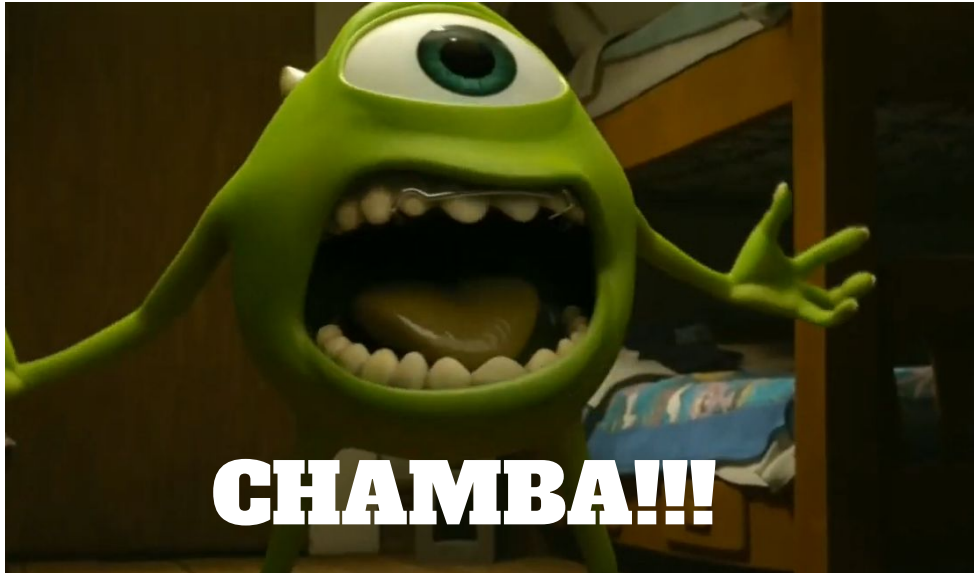
(Aquí va el diagrama en caso de que la banda quiera, de lo contrario, mmmm, un meme)

En este curso nos enfocaremos en las APIs...

WEB

¿Por qué en este tipo de API?

Actualmente son el estándar en la industria y generalmente son muy solicitadas para puestos de trabajo.



API Web

Las APIs web permiten un acceso programático sencillo a cualquier aplicación de servidor mediante unas pocas líneas de código, habilitando la ejecución remota de cualquier acción que dicha aplicación pueda realizar: por ejemplo, leer o actualizar la dirección de un cliente, obtener información de salud del servidor, enviar un SMS o detectar gatos en un video. Las APIs no requieren conocer el código ni la lógica interna; mientras exista la autorización correspondiente, cualquier persona puede utilizarlas, no solo sus desarrolladores.

Una API web es:

- Una interfaz remota para aplicaciones
- Una interfaz basada en el protocolo HTTP
- Una interfaz hacia una implementación
- Una interfaz diseñada para que otros la utilicen

Lauret, A. (2025). *The Design of Web APIs* (2nd ed.). Manning Publications.

Tipos de API Web

🌐 REST

🧱 SOAP

🧬 GraphQL

⚡ gRPC

🔔 Webhooks

🔄 WebSockets

P.D. No me funen, gracias :)

SOAP

Proporciona servicios web basados en XML con contratos estrictos definidos mediante WSDL (Web Services Description Language).

- Usa mensajes XML estructurados en un “sobre” SOAP.
- Requiere un contrato formal (WSDL).
- Soporta seguridad avanzada y transacciones

```python
from zeep import Client

client = Client("https://example.com/service?wsdl")
result = client.service.GetCustomer(id=123)
print(result)
```

GraphQL

Permite que el cliente solicite exactamente los datos que necesita mediante un lenguaje de consultas.

- Un único endpoint procesa todas las consultas.
- El cliente define los campos a devolver.
- Evita overfetching y underfetching.

```python
import requests

query = """
{
  user(id: 123) {
    name
    email
  }
}
"""

response = requests.post("https://api.example.com/graphql", json={"query": query})
print(response.json())
```

gRPC

Permite comunicación de alto rendimiento entre servicios usando HTTP/2 y Protocol Buffers.

- Se define un archivo .proto.
- Se genera código cliente/servidor automáticamente.
- Usa comunicación binaria eficiente

```python
import grpc
import user_pb2
import user_pb2_grpc

channel = grpc.insecure_channel("localhost:50051")
stub = user_pb2_grpc.UserServiceStub(channel)

request = user_pb2.UserRequest(id=123)
response = stub.GetUser(request)

print(response)
```

Webhooks

Recibe notificaciones automáticas desde otro sistema cuando ocurre un evento.

- El cliente registra una URL.
- El servidor envía solicitudes HTTP POST cuando ocurre un evento.
- Es un modelo push, no requiere polling.

```python
from flask import Flask, request

app = Flask(__name__)

@app.route("/webhook", methods=["POST"])
def webhook():
    event = request.json
    print("Evento recibido:", event)
    return "OK", 200

app.run(port=5000)
```

Websockets

Permite comunicación bidireccional en tiempo real entre cliente y servidor.

- Establece una conexión persistente.
- Permite enviar y recibir mensajes sin reconectar.
- Ideal para aplicaciones en tiempo real.

```python
from websocket import create_connection

ws = create_connection("wss://example.com/realtime")

ws.send("Hola servidor")
print(ws.recv())

ws.close()
```

Vamos a enfocarnos en REST, ya que actualmente es el estilo de arquitectura más utilizado y donde existe mayor CHAMBAAA!

REST

Permite exponer recursos a través de operaciones estándar HTTP, como lectura, creación, actualización o eliminación de datos.

- Usa métodos HTTP (GET, POST, PUT, DELETE).
- Cada recurso tiene una URL única.
- Normalmente intercambia datos en JSON.
- Es stateless: cada petición contiene toda la información necesaria.

```python
import requests

response = requests.get("https://api.example.com/users/123")
if response.status_code == 200:
    print(response.json())
```

#NoMeCuentesTuVidaCrack

#MuchoTexto

Orígenes...

REST son las siglas de Representational State Transfer, es un término acuñado por Roy Fielding en su tesis doctoral *Architectural Styles and the Design of Network-based Software Architectures* (Universidad de California, Irvine, 2000), para describir un estilo arquitectónico orientado a aplicaciones escalables y débilmente acopladas que se comunican a través de redes.

Es importante destacar que las definiciones originales de Fielding se centraban más en la arquitectura de aplicaciones web que en el diseño de APIs, y buscaban transmitir cómo debería comportarse una aplicación bien diseñada. Las prácticas modernas en el diseño de APIs RESTful se han construido sobre los principios establecidos por Fielding para aplicaciones RESTful.

RESTful se refiere a las APIs que siguen de manera estricta los principios y restricciones del estilo REST. (Los que propuso el tio Roy)

La Transferencia de Estado Representacional (REST) se refiere al acto de transferir —o comunicar— la representación del estado de un recurso. El concepto de recurso es fundamental en el contexto de las aplicaciones RESTful. De hecho, los recursos son el elemento central alrededor del cual se diseñan y estructuran las APIs RESTful.

Una especificación de API es un conjunto de endpoints que representan distintos recursos que pueden ser manipulados a través de la API, junto con las acciones que se pueden ejecutar sobre ellos. En las APIs REST construidas sobre HTTP, dichas acciones se representan mediante métodos HTTP, como GET, POST o PUT.

La tesis original…

Haro Peralta, J. (2022). *Microservice APIs: With examples in Python* (MEAP ed., Version 10). Manning Publications.

“ChatGPT dame el resumen de los principios de REST”

Arquitectura Cliente-Servidor: Separación estricta entre cliente (interfaz de usuario) y servidor (lógica y datos), lo que permite evolucionar cada parte de manera independiente.

Interacción Stateless: Cada petición HTTP debe contener toda la información necesaria; el servidor no almacena estado de sesión entre solicitudes.

Cacheabilidad: Las respuestas deben indicar si pueden ser cacheadas para mejorar rendimiento y reducir carga en el servidor.

Sistema en Capas: La arquitectura puede componerse de múltiples capas (proxies, gateways, balanceadores) sin que el cliente necesite conocerlas.

Interfaz Uniforme: Uso consistente de recursos, URIs, métodos HTTP y representaciones, lo que simplifica la interacción y reduce acoplamiento.

Selvaraj, S. (2024). *Mastering REST APIs: Boosting your web development journey with advanced API techniques*. Apress.

AHORA SI VAMOS A LA ACCIÓN

Vamos al ejercicio práctico...

Hacerle peticiones a una API.

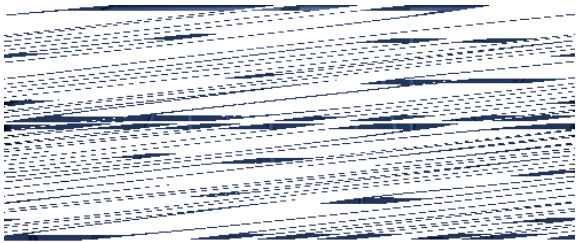
Necesitamos algo que nos ayude a traernos la información de una API. Existen extensiones en VSCode, herramientas como Postman o inclusive desde la misma terminal de linux/windows

Pero vamos a lo sencillo, una extensión de VSCode

Pokemon API

Disney API

Rick and Morty API



Star Wars API