

FASTBOOK 04

APIs y sockets

Core Desarrollo



Qualentum Lab

04. APIs y sockets

En este fastbook, abordaremos otro tema vinculado a la arquitectura: los canales de comunicación entre aplicaciones. Desgranaremos el concepto API y nos detendremos en la interfaz REST y en el protocolo SOAP.

Autor: Jaime Morales

≡ APIs

≡ REST

≡ SOAP

≡ Sockets

≡ Conclusiones

APIs



Qualentum Lab

Una API (del inglés, *application programming interface*; en español, interfaz de programación de aplicaciones) es una pieza de código que permite a las aplicaciones comunicarse entre sí y compartir información y funcionalidades.

Existen muchas definiciones que abordan lo que es el concepto de API, pero, sobre todo, debemos quedarnos con esta característica: que **es un módulo de software que interactúa o se comunica con otro**.

Las APIs pueden tener una o varias funcionalidades, también pueden ser privadas o públicas.

LAS APIs PRIVADAS

LAS APIs PÚBLICAS

Las privadas son de empresa, por ejemplo, la de un banco, a cuya API solo podrán acceder las personas autorizadas.

LAS APIs PRIVADAS**LAS APIs PÚBLICAS**

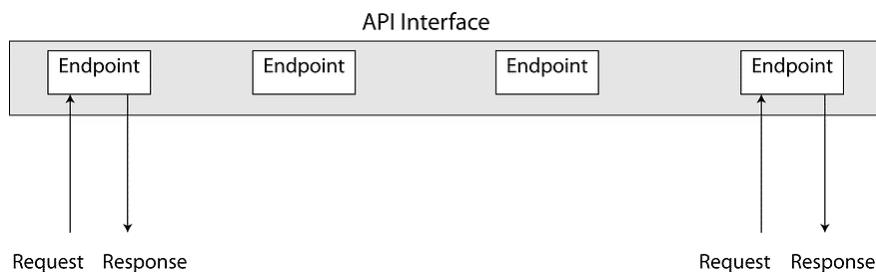
Un ejemplo de API pública sería la que ofrece la [Comunidad de Madrid](#) para que los ciudadanos puedan obtener información de eventos, del tiempo, etc., sin necesidad de que estén validados.

Adicionalmente, las APIs también pueden **establecerse en entornos locales**, es decir, que solo desde un entorno privado, una red interna pueda ser consumida. Normalmente, se desarrollan para dar mayor funcionalidad a APIs que ya están expuestas o evitar que alguien externo consuma las mismas ya que acceden a información secreta o confidencial habitualmente.

En este contexto, es importante diferenciar entre un *endpoint* y una API, ya que en muchas ocasiones podemos caer en el error de confundirlos.

Básicamente, una API es una interfaz que ofrece una serie de funcionalidades (una o varias). En cambio, un endpoint es la URL a la que se va a acceder.

A continuación, en la imagen se muestra la diferencia entre ambos:



Tal y como vemos en la ilustración, una API es el concepto más genérico y el *endpoint* es la URL que realiza la funcionalidad en concreto.

REST



De acuerdo con [Red Hat](#), una API Rest es una **interfaz de programación de aplicaciones que se ajusta a los límites de la arquitectura REST** y permite interacción con los servicios web de Restful. Se considera un contrato a la definición de la información que se va a intercambiar entre el cliente y el servidor, también conocido como **consumidor** (quien realiza la llamada) y el **productor** (quien expone la llamada).

REST no es un protocolo ni un estándar, sino un conjunto de límites que definen la arquitectura de la aplicación.

Cuando el cliente envía una solicitud del recurso requerido lo envía en uno de estos formatos (el formato JSON es el más extendido, popular y el más habitual):

- JSON.
- HTML
- XLT.
- Python.
- PHP.
- Sin formato.

Además de este mensaje, hay que tener en cuenta las cabeceras y los parámetros que se envían en una solicitud HTTP de la API RESTful, ya que contienen información de la solicitud, de los metadatos, el identificador de los recursos, etc.

Para que una API se considere una API REST necesita tener las siguientes **características**:

- Arquitectura **cliente-servidor** compuesta de clientes, servidores y recursos con la gestión de solicitudes a través de HTTP.
- Comunicación entre el cliente y el servidor sin estado**, lo cual implica que no se almacena la información del cliente entre las solicitudes de GET y que cada una de ellas es independiente y está desconectada del resto.
- Datos que pueden **almacenarse en caché y optimizar las interacciones** entre el cliente y el servidor.
- Una **interfaz uniforme** entre los elementos, para que la información se transfiera de forma estandarizada. Para ello, deben cumplirse las siguientes condiciones:

Primera condición

Los recursos solicitados deben ser identificables e independientes de las representaciones enviadas al cliente.

Segunda condición

El cliente debe poder manipular los recursos a través de la representación que recibe, ya que esta contiene suficiente información para permitirlo.

Tercera condición

Los mensajes autodescriptivos que se envíen al cliente deben contener la información necesaria para describir cómo debe procesarla.

Cuarta condición

Debe contener hipertexto o hipermedios, lo cual significa que cuando el cliente acceda a algún recurso, debe poder utilizar hipervínculos para buscar las demás acciones que se encuentren disponibles en ese momento.



Un **sistema en capas** que organiza cada uno de los servidores en jerarquías invisibles para el cliente (los encargados de la seguridad, del equilibrio de carga, etc.) que participan en la recuperación de la información solicitada.



Código disponible según se solicite (opcional), es decir, la capacidad para enviar códigos ejecutables del servidor al cliente cuando se requiera, lo cual amplía las funciones del cliente.

Es tan grande la comunidad que utiliza REST que existen **herramientas automáticas para documentar las funcionalidades** que ofrece un servicio o aplicación basada en REST, las más conocidas son Swagger, RAML y OpenAPI.

A continuación, vamos a explicar cada una de ellas:

Swagger

Recoge las reglas que se utilizan para documentar las APIs, de esta manera, quien las necesite debe conocer en detalle el contrato para realizar la llamada, los parámetros que va a recibir y el tipo (numérico, caracteres...).

Existe además una interfaz cuyo nombre es [Swagger UI](#), el cual utiliza un documento JSON o YAML que lo hace interactivo.



RAML

Restful API modeling language es una herramienta para la definición del modelado de aplicaciones REST que posee un lenguaje bastante sencillo y comprensible para sus usuarios.



OpenAPI

Un estándar para la descripción de interfaces de programación, que nace del proyecto de Swagger. OpenAPI es una especificación, es decir, una descripción abstracta que no está ligada a una aplicación técnica concreta. Hasta la versión 2.0, esta especificación todavía se llamaba Swagger, después, fue bautizada con nombre propio: **OpenAPI**. Sin embargo, las herramientas proporcionadas por **SmartBear**, la empresa que la desarrolló originalmente sigue existiendo con el nombre de Swagger.

Con OpenAPI, una API **puede describirse de manera uniforme**. Esto se conoce como definición API y se genera en un formato legible por máquina. En particular, se utilizan dos lenguajes: **YAML** y **JSON**.



SOAP



SOAP (*simple object access protocol*) tuvo su origen al inicio de la década de los 90. Su objetivo era **posibilitar la comunicación entre un cliente y el navegador de internet**. Como sabemos, la comunicación en internet funciona a través de los protocolos HTTP, HTTPS, FTP o TCP. Por lo tanto, SOAP se convirtió esencial para ofrecer los servicios web, interfaces usadas por los consumidores de dichas APIs.

El framework de SOAP **determina la forma que debe adoptar dicha solicitud**. Dentro de esta definición de la solicitud también pueden incluirse datos específicos de la aplicación, lo que es un punto fuerte de SOAP. De esta forma, los servicios web pueden desplegar aplicaciones diferentes. Para que puedan utilizarse como servicios web sin necesidad de tener la misma sintaxis, SOAP establece unas **reglas básicas**.

El framework SOAP se basa en el metalenguaje XML y, como resultado, un documento XML bien formado.

A continuación, compartimos un ejemplo de solicitud en SOAP:

```
POST /example HTTP/1.1
Host: example.org
Content-Type: text/xml; charset=utf-8
...
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
  SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  <SOAP-ENV:Header>
    ...
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>

</SOAP_ENV:Envelope>
```

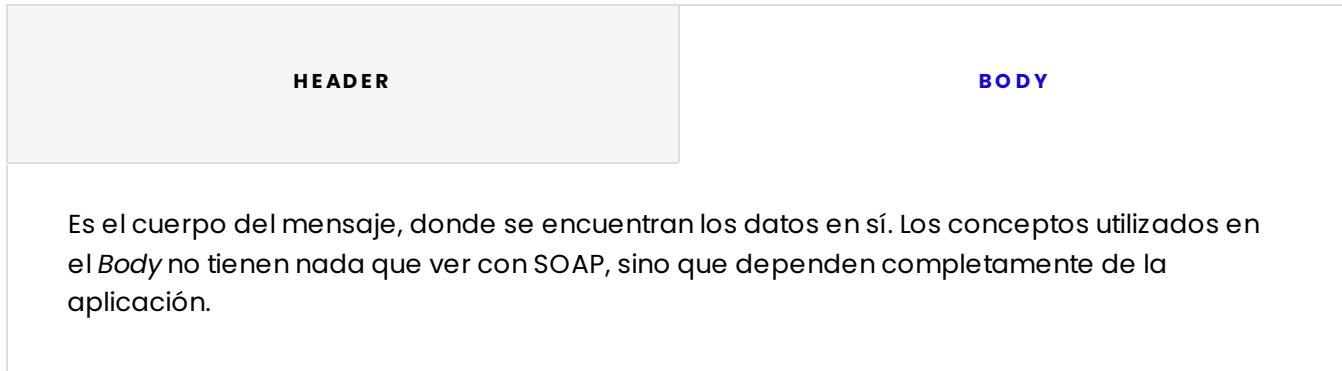
En este ejemplo, la solicitud comienza con un encabezado HTTP. A continuación, sigue el llamado *envelope* (sobre) de SOAP, porque envuelve el contenido real del mensaje como un sobre.

Los elementos principales de SOAP son el *Header* y el *Body*.

HEADER

BODY

Es el encabezado de la solicitud SOAP que contiene metadatos como el cifrado que se ha utilizado. Su uso es opcional.



Es el cuerpo del mensaje, donde se encuentran los datos en sí. Los conceptos utilizados en el *Body* no tienen nada que ver con SOAP, sino que dependen completamente de la aplicación.



Si quieres saber más sobre estos elementos principales, clica en este [enlace](#).

Comparación entre REST y SOAP

A continuación, se muestra una tabla con la **comparativa** entre SOAP y REST.

	SOAP	REST
Significado	Protocolo simple de acceso a objetos.	Transferencia de estado representacional.
¿Qué es?	SOAP es un protocolo para la comunicación entre aplicaciones.	REST es un estilo de arquitectura para diseñar interfaces de comunicación.
Diseño	La API de SOAP expone la operación.	La API de REST expone los datos.

Protocolo de transporte	SOAP es independiente y puede funcionar con cualquier protocolo de transporte.	REST solo funciona con HTTPS.
Formato de datos	SOAP solo admite el intercambio de datos XML.	REST admite XML, JSON, texto plano y HTML.
Rendimiento	Los mensajes SOAP son más grandes, lo que hace que la comunicación sea más lenta.	REST tiene un rendimiento más rápido debido a los mensajes más pequeños y al soporte de almacenamiento en caché.
Escalabilidad	SOAP es difícil de escalar. El servidor mantiene el estado al almacenar todos los mensajes anteriores intercambiados con un cliente.	REST es fácil de escalar. No tiene estado, por lo que cada mensaje se procesa de manera independiente de los mensajes anteriores.
Seguridad	SOAP admite el cifrado con sobrecargas adicionales.	REST admite cifrado sin afectar al rendimiento.
Caso de uso	SOAP es útil en aplicaciones antiguas y API privadas.	REST es útil en aplicaciones modernas y API públicas.



Amplía información en este [enlace](#).

Sockets



Qualentum Lab

Nació en los años 80 dando respuesta a la necesidad del sistema Unix de Berkeley de diseñar una tecnología capaz de establecer la comunicación entre procesos en red semejante a la comunicación por correo o por teléfono. ¿Pero qué es un socket?

Un socket es el punto final de conexión, es decir, un dispositivo o elemento electrónico que se genera gracias al sistema operativo y permite el envío de información de los procesos que hagan uso de estos.

Por lo tanto, su función principal es la **interconectividad**, ya que permite enviar y recibir información entre procesos incluso de diferentes máquinas.

Ventajas y desventajas de los sockets:

Ventajas

- Permite el intercambio del flujo de datos entre un cliente y un servidor de manera fiable y ordenada.
- Se beneficia de la funcionalidad API.

Desventajas

- Es dependiente de la respuesta del servidor.
- En el tipo de sockets no orientado a la conexión, el protocolo UDP no garantiza que los datos lleguen en el mismo orden en que se enviaron.

En [IBM](#) puedes encontrar una amplia y detallada clasificación de los distintos sockets.

Algunos de ellos son:

SOCK_DGRAM

Proporciona diagramas en forma de mensajes sin conexión de una longitud máxima ya fijada, en concreto, mensajes cortos, como un servidor de nombres o de horas, dado que el orden y la fiabilidad de la entrega de mensajes no están garantizados. Un socket de datagrama da soporte al flujo bidireccional de datos, que no está secuenciado, no es fiable ni está duplicado.

Un proceso que recibe mensajes en un socket de datagrama puede encontrar mensajes duplicados o en un orden diferente al pedido enviado. Sin embargo, se conservan los límites de registro en los datos. Los sockets de datagramas modelan de cerca las instalaciones que se encuentran en muchas redes de commutación de paquetes contemporáneas.

SECUENCIA_SOCKETS

Ofrece secuencias de bytes bidireccionales secuenciadas con un mecanismo de transmisión para los datos de secuencia. Este tipo de socket transmite datos de forma fiable, en orden y con prestaciones fuera de banda. En el dominio UNIX, el tipo de socket *SOCK_STREAM* funciona como un conducto. En el dominio de Internet, el tipo de socket *SOCK_STREAM* se implementa en el protocolo de control de transmisiones/*internet protocol* (TCP/IP).

SOCK_SEQPAQUETE

Proporciona un flujo de información secuenciado, fiable y no duplicado.

SOCK_CONN_DGRAM

Ofrece un servicio de datagramas orientado a conexiones. Este tipo de socket da soporte al flujo bidireccional de datos, que se secuencian y no se duplican, pero no es fiable. Puesto que se trata de un servicio orientado a la conexión, el socket debe estar conectado antes de la transferencia de datos.

Conclusiones



En este fastbook, hemos definido el concepto API para después conocer más en detalle dos de ellas: **SOAP** y **REST**. Tal y como hemos analizado tras compararlas, las APIs REST son más actuales, más ágiles y consumen menos recursos, de ahí que ahora mismo el futuro esté en aprender cómo trabajar con las APIs REST. No obstante, hasta que llegue ese futuro, también es importante conocer **SOAP** ya que, en la actualidad, **se usa en muchos sectores** para diferentes aplicaciones y servicios.

Por último, hemos visto qué son los **sockets**, esos **canales de intercomunicación** entre procesos no relacionados para el intercambio de datos, ya sea a nivel local o entre redes.

¡Enhорabuena! Fastbook superado



Qualentum.com