API REST

ÍNDICE

1. Qué es una API

Es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet, siendo un conjunto de reglas y protocolos que lo permiten. En otras palabras, es una interfaz que permite que dos o más sistemas se comuniquen y compartan datos y servicios.

Las APIs son utilizadas en la mayoría de los sistemas de software modernos, desde aplicaciones móviles hasta servicios web, y conocer su terminología es fundamental para desarrollar y utilizar aplicaciones de manera eficiente y efectiva. Por ejemplo, la aplicación de planilla de horarios expone una API que solicita el nombre completo de un empleado y un rango de fechas. Cuando recibe esta información, procesa internamente la planilla de horarios del empleado y devuelve la cantidad de horas trabajadas en ese rango de fechas.

Las API son fundamentales para la comunicación y la integración de aplicaciones y sistemas, y son utilizadas por muchas empresas y organizaciones para conectar sus sistemas y aplicaciones a otros sistemas externos, por lo que se puede pensar en una API web como una puerta de enlace entre los clientes y los recursos de la Web.

Clientes: quieren acceder a la información desde la web.
Recursos: son la información que diferentes aplicaciones proporcionan a sus clientes.

Existen distintos tipos de API:

- **La API de SOAP** (Simple Object Access Protocol) es un protocolo de comunicación que utiliza XML (Lenguaje de Marcado Extensible) para intercambiar información entre diferentes sistemas.
- **Las API de RPC** (Remote Procedure Call) permiten que una aplicación invoque una función o método en un sistema remoto a través de una red.
- **La API de WebSocket** es un protocolo de comunicación bidireccional que permite que los datos se intercambien entre una aplicación web y un servidor en tiempo real.
- **API de REST** es una forma de permitir que diferentes aplicaciones se comuniquen entre sí a través de internet utilizando el protocolo HTTP.

2. Qué es REST

La transferencia de estado representacional (REST) es una arquitectura de software que impone condiciones sobre cómo debe funcionar una API. En un principio, se creó como una guía para administrar la comunicación en una red compleja como Internet.

Los desarrolladores de API pueden diseñar API por medio de varias arquitecturas diferentes. Las que siguen el estilo arquitectónico de REST se llaman API REST. Los servicios web que implementan una arquitectura de REST son llamados servicios web RESTful. El término API RESTful suele referirse a las API web RESTful, sin embargo, los términos API REST y API RESTful se pueden utilizar de forma intercambiable.

3. API REST

Las API REST se comunican a través de solicitudes HTTP para realizar funciones estándar de bases de datos, como crear, leer, actualizar y eliminar registros (también conocido como CRUD) dentro de un recurso.

Por ejemplo, una API REST usaría una solicitud GET para recuperar un registro. Una solicitud POST crea un nuevo registro. Una solicitud PUT actualiza un registro y una solicitud DELETE lo elimina. Todos los métodos HTTP se pueden utilizar en las llamadas a la API. Una API REST bien diseñada es similar a un sitio web que se ejecuta en un navegador web con funcionalidad HTTP incorporada.

Algunos términos básicos que se utilizan en las API Rest son las siguientes:

1. **Endpoint:** Es la dirección URL específica de un recurso en la API Rest.
2. **Método HTTP:** Es el verbo utilizado para definir la acción que se va a realizar sobre el recurso. Los métodos más comunes son: GET, POST, PUT y DELETE.
3. **Parámetros:** Son valores adicionales que se envían con la solicitud HTTP para ayudar a definir la acción que se desea realizar. Los parámetros pueden ser incluidos en la URI o en el cuerpo de la solicitud.
4. **URI:** Es una cadena de caracteres que identifica el recurso y se utiliza en conjunto con el método HTTP para acceder a él. Se compone de *endpoint* y los *parámetros*.
5. **Payload:** Es la información adicional que se envía en la solicitud HTTP. Puede estar en formato JSON o XML y puede contener datos adicionales para la creación o actualización de un recurso.
6. **Autenticación:** Es el proceso de verificación de la identidad de un usuario o aplicación para permitir el acceso a los recursos protegidos. Se realiza mediante la inclusión de credenciales de usuario o de aplicación en la solicitud HTTP.
7. **Respuesta HTTP:** Es la respuesta que devuelve la API Rest después de procesar la solicitud HTTP. Puede estar en formato JSON o XML y puede incluir datos adicionales o mensajes de error.
8. **Códigos de estado HTTP:** Son códigos numéricos que indican el resultado de una solicitud HTTP. Los más comunes son 200 *OK* (éxito), 400 *Bad Request* (solicitud incorrecta), 401 *Unauthorized* (no autorizado), 404 *Not Found* (recurso no encontrado) y 500 *Internal Server Error* (error interno del servidor).
9. **Caché:** Es un mecanismo utilizado para almacenar temporalmente los datos de una respuesta HTTP para reducir el tiempo de carga de una página o aplicación. La API Rest puede incluir información adicional en la respuesta HTTP para indicar si los datos se pueden almacenar en caché y por cuánto tiempo.

4. Funcionamiento de las API REST

Como ya sabemos, el cliente es quien envía solicitudes al servidor, que a su vez responde con información o datos. Entonces, ¿cómo funciona?

1. Solicitud del cliente: El cliente envía una solicitud HTTP al servidor, especificando el recurso que desea acceder y la acción que quiere realizar (por ejemplo, crear un nuevo usuario).

2. Métodos HTTP: Los métodos HTTP utilizados son:

➡ **GET:** Recupera la representación de un recurso.
➡ **POST:** Crea un nuevo recurso.
➡ **PUT:** Actualiza completamente un recurso.
➡ **DELETE:** Elimina un recurso.
➡ **PATCH:** Actualiza parcialmente un recurso.

3. Respuesta del servidor: El servidor recibe la solicitud, la procesa y devuelve una respuesta HTTP al cliente. La respuesta puede contener:

➡ **Datos en formato JSON o XML:** Los datos del recurso solicitado.
➡ **Código de estado HTTP:** Indica si la solicitud fue exitosa o no (por ejemplo, 200 *OK*, 404 *Not Found*).

4. Identificación de recursos: Cada recurso tiene un URI (*Uniform Resource Identifier*) que lo identifica.

5. Sin estado: Las API REST son sin estado, lo que significa que cada solicitud del cliente debe contener toda la información necesaria para comprender la solicitud, y el servidor no mantiene información sobre interacciones anteriores.

Además de *sin estado* también puede ser:

- **Cliente-Servidor:** La API opera bajo una arquitectura cliente-servidor.
- **Cacheable:** Las respuestas pueden ser almacenadas en caché para mejorar el rendimiento.
- **Interfaz uniforme:** La API debe tener una interfaz uniforme para acceder a los recursos, lo que facilita la comprensión y el uso de la API.
- **Estructura jerárquica:** Los recursos son identificados por URIs (Uniform Resource Identifiers).

5. Beneficios que ofrecen las API REST

Los sistemas que implementan API REST pueden escalar de forma eficiente porque REST optimiza las interacciones entre el cliente y el servidor. La tecnología sin estado elimina la carga del servidor porque este no debe retener la información de solicitudes pasadas del cliente. El almacenamiento en caché bien administrado elimina de forma parcial o total algunas interacciones entre el cliente y el servidor. **Todas estas características admiten la escalabilidad**, sin provocar cuellos de botella en la comunicación que reduzcan el rendimiento.

Los servicios web RESTful admiten una separación total entre el cliente y el servidor. Simplifican y desacoplan varios componentes del servidor, de manera que cada parte pueda evolucionar de manera independiente. Los cambios de la plataforma o la tecnología en la aplicación del servidor no afectan la aplicación del cliente. La capacidad de ordenar en capas las funciones de la aplicación **aumenta la flexibilidad aún más**. Por ejemplo, los desarrolladores pueden efectuar cambios en la capa de la base de datos sin tener que volver a escribir la lógica de la aplicación.

También **son independientes** de la tecnología que se utiliza. Puede escribir aplicaciones del lado del cliente y del servidor en diversos lenguajes de programación, sin afectar el diseño de la API. También puede cambiar la tecnología subyacente en cualquiera de los lados sin que se vea afectada la comunicación.

Presenta un **formato de datos flexible**, pues aunque comúnmente se usa con JSON, REST puede trabajar con otros formatos como XML, YAML o incluso texto plano.

6. Qué es el formato JSON

Es un formato de intercambio de datos para los humanos y las máquinas. Su principal utilidad se da en el intercambio de datos entre sistemas informáticos. Se dice que es ligero porque los datos no suelen ocupar mucho, como sí ocurre con otros lenguajes de intercambio de datos. Es el formato de intercambio de datos utilizado en la API REST.

Fue creado a partir de un subconjunto del lenguaje de programación JavaScript, y utiliza convenciones familiares a los programadores de lenguajes de la familia C, como C++, Java, Python, entre otros.

Tiene algunas características que lo han hecho especialmente adecuado y utilizado en el ámbito del desarrollo, especialmente en la web:

- Su facilidad de escritura para los humanos, así como la lectura
- El hecho de ser especialmente ligero, es decir, con poco peso en bytes
- Es fácil de analizar y de generar
- Es compatible con la mayoría de los lenguajes de programación usados en la actualidad

7. Funcionamiento de JSON

JSON es comúnmente utilizado para transmitir datos en aplicaciones web entre el cliente y el servidor. Un uso común de JSON es obtener datos de un servidor web a través de una solicitud HTTP. Este intercambio de datos puede ser entre dos servidores completos o entre un servidor y un cliente.

También se utilizan para almacenar configuraciones de las aplicaciones, ya sean introducidas en el código o como soporte al almacenamiento de las configuraciones introducidas por el usuario.

Debido a su naturaleza independiente, JSON se puede usar para intercambiar datos entre diferentes lenguajes de programación. Por ejemplo, un servidor puede estar escrito en PHP y el cliente web que consume los datos del backend PHP puede estar escrito en Javascript, Sin embargo, ambos pueden enviar y recibir datos en formato JSON.

8. Características

Se caracteriza por: su sintaxis, siendo bastante simple ya que consiste en pares de "nombre : valor" que se separan por comas y se encierran entre llaves {} para formar objetos, o se encierran entre corchetes [] para formar arreglos (más conocido como *Array*, pues son simples colecciones de elementos,). Los valores pueden ser cualquier cosa que se pueda representar en un lenguaje de programación.

Ejemplo:
Tenemos un arreglo (array) de personas (objetos) con sus atributos y valores.

```
[
   {
      'nombre': 'Facundo',
      'edad': 30,
      'ciudad': 'Madrid'
   },
   {
      'nombre': 'Saturnina',
      'edad': 98,
      'ciudad': 'Barcelona'
   },
   {
      'nombre': 'Adoración',
      'edad': 55,
      'ciudad': 'Sevilla'
   }
]
```

También es un **formato ligero**, lo que significa que no consume muchos recursos ni en su construcción ni a la hora de transferirlo por Internet.

Además, es **independiente del lenguaje de programación**, lo que significa que se puede usar en cualquier lenguaje que pueda analizar, generar y manipular datos en este formato. Esto hace que sea muy flexible y adaptable a diferentes situaciones.

9. Ventajas de JSON

Por muchos motivos JSON se ha convertido en un formato ampliamente usado en el mundo del desarrollo, especialmente en la web. Unas pocas de las ventajas que ofrece serían:

Facilidad de uso y lectura por humanos: pues es bastante fácil su lectura y escritura, con estructura simple y clara que facilita la comprensión de los datos a simple vista.

Eficiencia en la transmisión de datos: lo hace ideal para la comunicación entre el cliente y el servidor en aplicaciones web, y, en general, en cualquier sistema distribuido.

Compatibilidad con numerosos lenguajes de programación: convirtiéndolo en una opción versátil para el intercambio de datos. Todos los lenguajes actuales soportan JSON, sería difícil encontrar uno que no admita este formato.