Temario:

1 Qué es una api.

2 Flujo de comunicación.

3 Api Rest.

4 Fectch.

¿Qué es una API?

Saber cómo funciona una API es como develar los secretos sobre la fórmula que utilizan los softwares, los sistemas y las aplicaciones para interactuar, integrarse y comunicarse.

Es descubrir cómo las Tecnologías de la Información pueden ser más eficientes, generando menos trabajo y más productividad.

La expresión Application Programming Interface o en español, Interfaz de Programación de Aplicaciones, originó el acrónimo API.

Las API son «traductores» cuya función es conectar sistemas, software y aplicaciones. Con las API es posible ofrecerle una experiencia de uso más familiar a las personas.

Las API le permiten al usuario final utilizar una aplicación, software o incluso una simple hoja de cálculo, consultando, cambiando y almacenando datos de diferentes sistemas, sin que el usuario tenga que ingresar a ellos, directamente.

El propósito de una API es intercambiar datos entre diferentes sistemas, la mayoría de las veces estos intercambios de datos tienen como objetivo automatizar procesos manuales y / o permitir la creación de nuevas funcionalidades

Por ejemplo, un software de marketing recibe datos de un cliente potencial y usando una API, se envía la información de un cliente potencial calificado al software CRM. En este momento se da un intercambio de datos para automatizar un proceso manual.

Otro ejemplo es cuando un software CRM cuenta con un botón al lado del número de teléfono que permite realizar una llamada mediante la integración con una proveedora de telefonía VOIP para dejar la llamada grabada. En este caso, se realiza el intercambio de datos para crear una nueva funcionalidad donde todo el servicio lo realiza otra empresa (la proveedora de telefonía).

¿Para qué sirve una API?

Las API se pueden utilizar de diferentes formas: integrando diferentes sistemas para una mayor eficiencia a la hora de utilizarlas, y tienen un papel estratégico en la rutina de las empresas. Después de todo, hay varios sistemas y aplicaciones que se utilizan en una empresa, y todas estas funciones interactúan con otro software, a través de la API.

El ejemplo más común de uso de API es el de un empleado que necesita emitir facturas para completar el pedido de un cliente. En este caso, la API puede conectar el sistema de gestión de la empresa al sistema de generación de comprobantes bancarios del banco y al sistema de emisión de facturas. Por lo tanto, el empleado solo necesita ingresar los datos una vez y finalizar el proceso con unos pocos clics.

Otra aplicación de la API es que puede ayudar a reducir la tasa de incumplimiento, por ejemplo. La API conectaría los datos de las cuentas por cobrar de la empresa o del sector CRM a una plataforma de chatbot. Los clientes morosos recibirían un mensaje con los datos de facturación de forma automática, para que puedan regular la situación en poco tiempo.

Los sistemas conversacionales también podrían actualizar los datos de registro de clientes utilizando una API para comunicarse con las personas y consultar la información.

Ejemplos de API

A continuación, conocerás ahora algunos ejemplos de API, para que descubras cómo pueden ayudarte.

API para WhatsApp

La API de WhatsApp le permite a la empresa comunicarse a gran escala en la plataforma. Recordando que la aplicación es utilizada por 1.500 millones de personas en todo el mundo. Con la integración vía API, es posible enviar recordatorios, tener múltiples asistentes, ofrecer soporte 24h con mensajes programados e incluso crear un chatbot para WhatsApp.

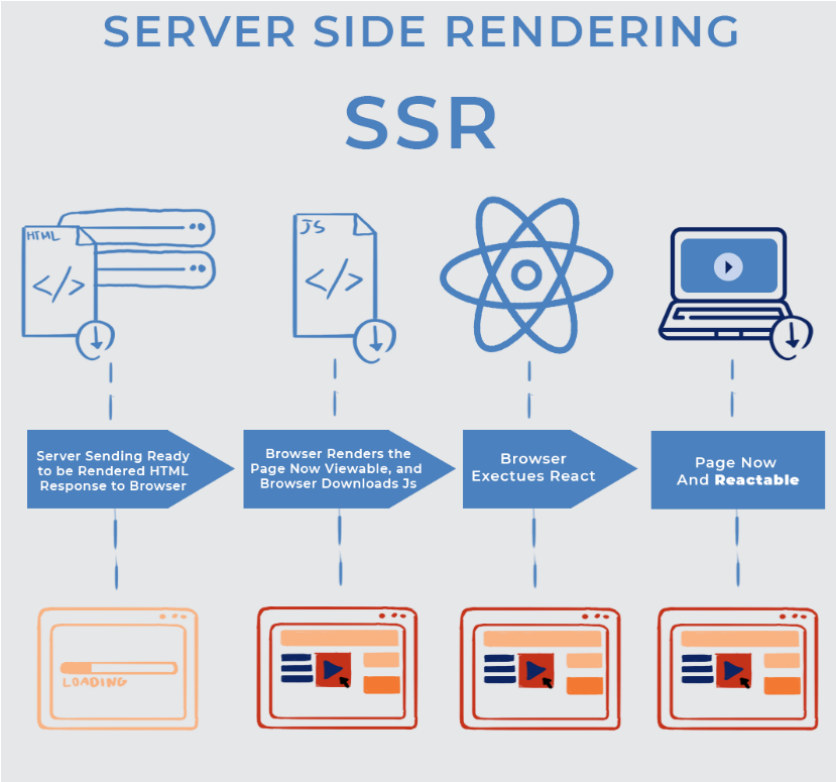
De este modo, podrás utilizar tu cuenta comercial para responder preguntas, enviar cobros de pago o informar sobre el estado de un pedido, aprovechando el potencial de la plataforma de mensajería más utilizada.

API de Mercado Libre

La API de MercadoLibre es una poderosa herramienta con la se pueden integrar en tiempo real todos los datos de artículos publicados en MercadoLibre y nuestra Web. Entre los distintos datos que la API de MercadoLibre proporciona, tenemos: los nombres de los artículos, precios, cantidades disponibles, visitas, fotos y otros datos que permiten hacer que las ofertas sean más atractivas para los usuarios.

Flujo de comunicación entre usuarios, frontend y backend

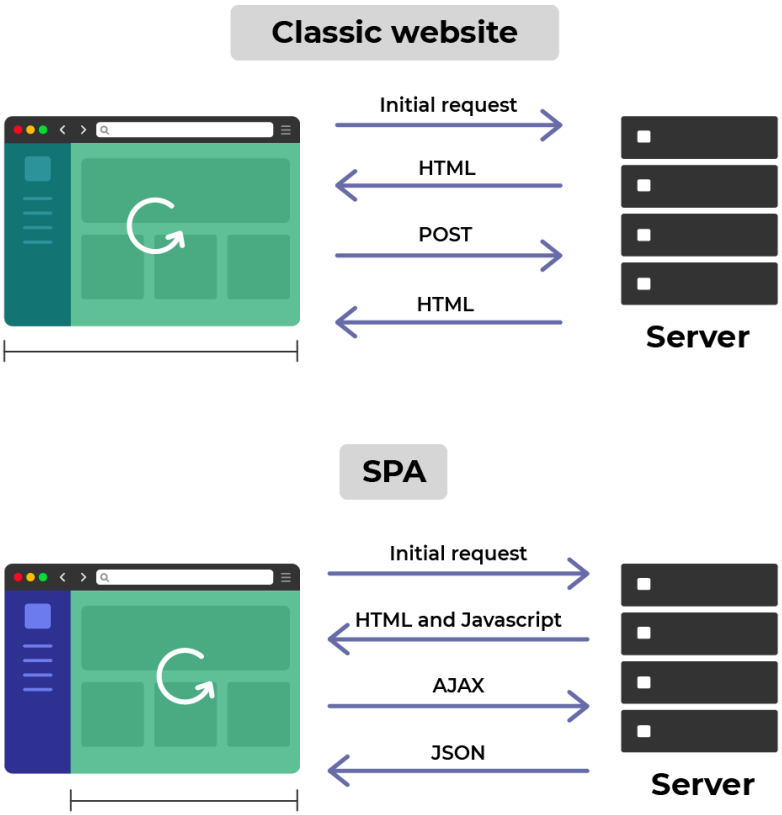
La representación del lado del servidor se refiere a la capacidad de una aplicación para mostrar la página web en el servidor en lugar de mostrarla en el navegador. Cuando el JavaScript de un sitio web se procesa en el servidor del sitio web, se envía una página completamente procesada al cliente y el paquete de JavaScript del cliente se activa y permite que funcione el marco de la aplicación de una sola página.



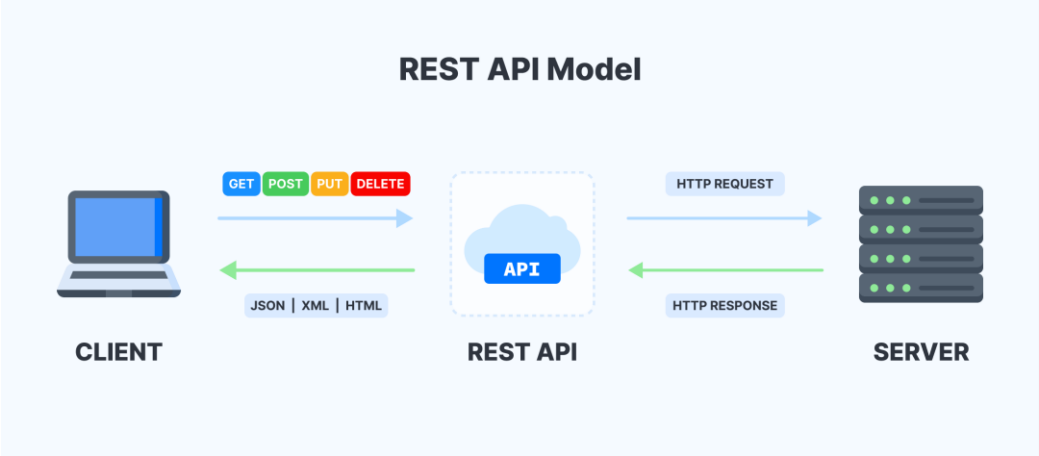
SPA (aplicación de una sola página)

Una SPA (aplicación de una sola página) es una implementación de aplicación web que carga solo un único documento web y luego actualiza el contenido del cuerpo de ese documento único a través de las API de JavaScript, como XMLHttpRequest o fetch y se muestra contenido diferente.

Por lo tanto, esto permite a los usuarios usar sitios web sin cargar páginas completamente nuevas desde el servidor, lo que puede generar mejoras en el rendimiento y una experiencia más dinámica.


Classic website
Initial request
HTML
POST
HTML
Server
SPA
Initial request
HTML and Javascript
AJAX
JSON
Server

API REST



Las API REST proporcionan una forma flexible y ligera de integrar aplicaciones y han surgido como el método más común para conectar componentes en la arquitectura de microservicios.

¿Qué es una API REST?

Una API, o interfaz de programación de aplicaciones, es un conjunto de reglas que determinan cómo las aplicaciones o los dispositivos pueden conectarse y comunicarse entre sí. Una API REST es una API que se ajusta a los principios de diseño de REST, un estilo de arquitectura también denominado transferencia de estado representacional. Por este motivo, las API REST son a veces denominadas API RESTful.

REST, definido por primera vez en el año 2000 por el científico de la computación Dr. Roy Fielding en su tesis doctoral, proporciona un nivel relativamente alto de flexibilidad y libertad para los desarrolladores. Dicha flexibilidad es solo una razón por la que las API REST han surgido como un método común para conectar componentes y aplicaciones en una arquitectura de microservicios.

Principios de diseño de REST

En el nivel más básico, una API es un mecanismo que permite a una aplicación o servicio acceder a un recurso dentro de otra aplicación o servicio. La aplicación o servicio que realiza el acceso se denomina cliente y la aplicación o servicio que contiene el recurso se denomina servidor.

Algunas API, como SOAP o XML-RPC, imponen una estructura estricta a los desarrolladores. Sin embargo, las API REST se pueden desarrollar mediante el uso de prácticamente cualquier lenguaje de programación y son compatibles con una variedad de formatos de datos. El único requisito es que se ajusten a los siguientes seis principios de diseño de REST, también conocidos como restricciones de arquitectura:

Interfaz uniforme: Todas las solicitudes de API para el mismo recurso deben ser iguales, independientemente de la procedencia de la solicitud. La API REST debe asegurarse de que el mismo dato, como el nombre o la dirección de e-mail de un usuario, pertenezca a un único identificador uniforme de recurso (URI). Los recursos no deben ser demasiado grandes, sin embargo, deben contener toda la información que el cliente pueda necesitar.

Separación entre cliente y servidor: En el diseño de API REST, las aplicaciones de cliente y de servidor deben ser completamente independientes entre sí. La única información que la aplicación de cliente debe conocer es el URI del recurso solicitado. No puede interactuar con la aplicación de servidor de ninguna otra forma. Del mismo modo, una aplicación de servidor no debe modificar la aplicación de cliente más allá de entregarle los datos solicitados vía HTTP.

Sin estado. Las API REST son sin estado, lo que significa que cada solicitud debe incluir toda la información necesaria para procesarla. En otras palabras, las API REST no requieren ninguna sesión en el lado del servidor. Las aplicaciones de servidor no pueden almacenar ningún dato relacionado con una solicitud de cliente.

Capacidad de almacenamiento en memoria caché. Siempre que sea posible, los recursos deben poder almacenarse en la memoria caché en el lado del cliente o el servidor. Las respuestas de servidor también necesitan contener información de si el almacenamiento en memoria caché está permitido para el recurso entregado. El objetivo es mejorar el rendimiento en el lado del cliente, al mismo tiempo que aumenta la escalabilidad en el lado del servidor.

Arquitectura de sistema de capas. En las API REST, las llamadas y respuestas pasan por diferentes capas. Como regla general, no debe suponer que las aplicaciones de cliente y de servidor se conectan directamente entre sí. Puede haber una serie de intermediarios diferentes en el bucle de comunicación. Las API REST deben diseñarse para que ni el cliente ni el servidor puedan notar si se comunican con la aplicación final o con un intermediario.

Funcionamiento de las API REST

Las API REST se comunican a través de solicitudes HTTP para realizar funciones estándar de base de datos, como crear, leer, actualizar y suprimir registros (también conocidos como CRUD) dentro de un recurso. Por ejemplo, una API REST utilizará una solicitud GET para recuperar un registro, una solicitud POST para crearlo, una solicitud PUT para actualizarlo y una solicitud DELETE para suprimirlo. Todos los métodos HTTP se pueden utilizar en llamadas API. Una API REST bien diseñada es similar a un sitio web que se ejecuta en un navegador web con funcionalidad HTTP incorporada.

El estado de un recurso en un instante específico, o en una indicación de fecha y hora, se conoce como la representación de recursos. Esta información se puede entregar a un cliente en prácticamente cualquier formato, entre ellos JavaScript Object Notation (JSON), HTML, XLT, Python, PHP o un texto sin formato. JSON es popular debido a que es legible tanto por los seres humanos como por las máquinas y debido a que es independiente de un lenguaje de programación.

Las cabeceras y parámetros de solicitud también son importantes en las llamadas de API REST, puesto que incluyen información de identificador importante, como metadatos, autorizaciones, identificadores de recursos uniformes (URI), almacenamiento en memoria caché, cookies y más. Las cabeceras de solicitud y las cabeceras de respuesta, junto con los códigos de estado HTTP convencionales, se utilizan en las API REST bien diseñadas.

Peticiones HTTP con fetch

Fetch es el nombre de una nueva API para Javascript con la cuál podemos realizar peticiones HTTP asíncronas utilizando promesas y de forma que el código sea un poco más sencillo y menos verbose. La forma de realizar una petición es muy sencilla, básicamente se trata de llamar a fetch y pasarle por parámetro la URL de la petición a realizar:

```
// Realizamos la petición y guardamos la promesa
const request = fetch("/robots.txt");
// Si es resuelta, entonces ejecuta esta función...
request.then(function(response) { ... });
```

El fetch() devolverá una que será aceptada cuando reciba una respuesta y sólo será rechazada si hay un fallo de red o si por alguna razón no se pudo completar la petición. El modo más habitual de manejar las promesas es utilizando .then(). Esto se suele reescribir de la siguiente forma, que queda mucho más simple:

```
fetch("/robots.txt")
  .then(function(response) {
    /** Código que procesa la respuesta **/
  });
```

Al método .then() se le pasa una función callback donde su parámetro response es el objeto de respuesta de la petición que hemos realizado. En su interior realizaremos la lógica que queramos hacer con la respuesta a nuestra petición. A la función fetch(url, options) se le pasa por parámetro la url de la petición y, de forma opcional, un objeto options con opciones de la petición HTTP.

Vamos a examinar un código donde veamos un poco mejor como hacer la petición con fetch:

```
// Opciones de la petición (valores por defecto)
const options = {
  method: "GET"
};
// Petición HTTP
fetch("/robots.txt", options)
  .then(response => response.text())
  .then(data => {
    /** Procesar los datos **/
```

});

Un poco más adelante, veremos cómo trabajar con la respuesta response, pero vamos a centrarnos ahora en el parámetro opcional ***options*** de la petición HTTP. En este objeto podemos definir varios detalles:

method **:** Método HTTP de la petición. Por defecto, GET. Otras opciones: HEAD, POST, etc...

body: Cuerpo de la petición HTTP. Puede ser de varios tipos: String, FormData, Blob, etc...

headers: Cabeceras HTTP. Por defecto, {}.

Credentials: Modo de credenciales. Por defecto, omit. Otras opciones: same-origin e include.

Lo primero, y más habitual, suele ser indicar el método HTTP a realizar en la petición. Por defecto, se realizará un GET, pero podemos cambiarlos a HEAD, POST, PUT o cualquier otro tipo de método. En segundo lugar, podemos indicar objetos para enviar en el body de la petición, así como modificar las cabeceras en el campo headers:

```
const options = {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify(jsonData)
};
```

Por último, el campo credentials permite modificar el modo en el que se realiza la petición. Por defecto, el valor omit hace que no se incluyan credenciales en la petición, pero es posible indicar los valores same-origin, que incluye las credenciales si estamos sobre el mismo dominio, o include que incluye las credenciales incluso en peticiones a otros dominios.