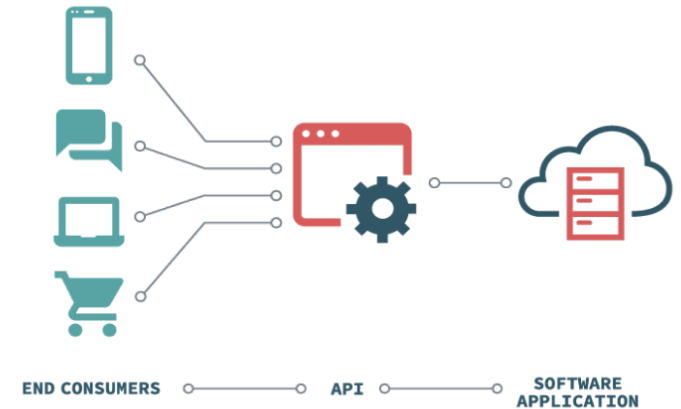
JNDI Y TIPOS DE EJBS

API REST

Una Interfaz de Programación de Aplicaciones (API) es un conjunto de reglas que determinan cómo las aplicaciones o dispositivos pueden conectarse entre sí para poder intercambiar datos entre sí mismos. De esta manera podemos decir que una API es una interfaz pues es una forma de que una cosa interactúe con otra.



Una API permite que sus productos y servicios se comuniquen con otros sin necesidad de saber cómo están implementados. Gracias a esto se simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones.

TIPOS DE API

* Locales. Cuando la aplicación se comunica con las API que se encuentran en el mismo ambiente o dispositivo.
* Remotas. Cuando la aplicación necesita consumir una API en un punto diferente.

Las APIs remotas pueden utilizar servicios web SOAP y REST. Aunque SOAP es un protocolo que ya dejó de ser usado, muchos sistemas aún funcionan con él.

REST

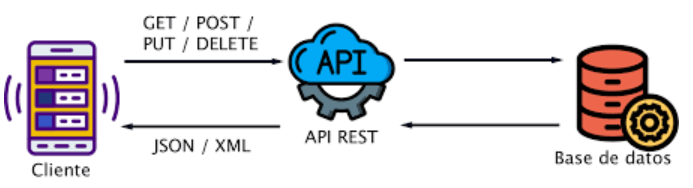
Representational State Transfer (REST) es un estilo de arquitectura cuando se realiza una comunicación entre cliente y servidor. Es la que más se utiliza en las API.

Para que una API pueda ser considera API REST o usualmente denominada API RESTful debe ajustarse a los seis principios de diseño REST:

1. Interfaz uniforme. La API REST debe asegurarse de que el mismo dato pertenezca a un único identificador uniforme de recurso (URI). Los recursos no deben ser muy grandes y deben contener toda la información que el cliente pueda necesitar.
2. Separación entre cliente y servidor. Las aplicaciones de cliente y servidor deben ser completamente independientes entre sí. La única información que la aplicación cliente debe conocer es el URI del recurso solicitado. De la misma manera, la aplicación servidor no debe modificar la aplicación cliente más allá de entregarle los datos solicitados.
3. Sin estado. Esto significa que cada solicitud debe incluir toda la información necesaria para procesarla, de esta manera, la API REST no requiere ninguna sesión en el lado del servidor. Recuerda, la aplicación servidor no puede almacenar ningún dato relacionado con una solicitud de cliente.
4. 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. El objetivo es mejorar el rendimiento en el lado del cliente, al mismo tiempo que aumenta la escabilidad en el lado del servidor.
5. Arquitectura de sistema de capas. En una API REST, las llamadas y respuestas pasan por diferentes capas. Es por esto que no se debe suponer que las aplicaciones cliente y servidor se conectan directamente entre sí, si no que puede existir una serie de intermediarios diferentes. Una API REST debe diseñarse para que ni el cliente ni el servidor noten si se comunican con la aplicación final o con un intermediario.
6. Código bajo demanda. Aunque la mayoría de las API REST envían recursos estáticos puede haber casos donde las respuestas pueden contener código ejecutable (como applets de Java). En estas situaciones, el código solo deberá ejecutarse bajo demanda.

COMO FUNCIONA UNA API REST

Las API REST se comunican a través de solicitudes HTTP para realizar sus funciones estándar de base de datos como crear, leer, actualizar y eliminar registros dentro de un recurso. Una API REST bien diseñada es similar a un sitio web que se ejecuta en un navegador web con funcionalidad HTTP incorporada.



Cuando se envía una solicitud del cliente a través de una API REST, ésta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información se entrega por medio de HTTP en algún formato, usualmente JSON pero también puede ser HTML, XLT, Python, PHP o texto sin formato.

Las cabeceras y parámetros de la solicitud también son importantes en las llamadas a la API, pues incluyen información de identificador importante como metadatos, autorizaciones, identificadores de recursos uniformes (URI), almacenamiento en memoria caché, cookies, etc.

METODOS HTTP

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs.

GET. El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

HEAD. El método HEAD pide una respuesta idéntica a la de una petición GET pero sin el cuerpo de la respuesta.

POST. El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT. El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

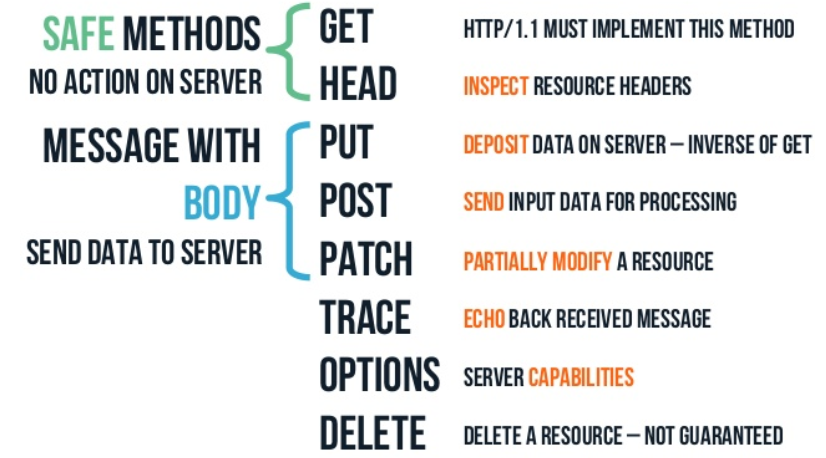
DELETE. El método DELETE borra un recurso en específico.

CONNECT. El método CONNECT establece un túnel hacia el servidor identificado por el recurso.

OPTIONS. El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.

TRACE. El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

PATCH. El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.



API REST Y METODOS HTTP

Listar (GET)

Las llamadas al API o listado se implementan como peticiones HTTP donde:

* La URL representa el recurso.
* El método o verbo HTTP representa la operación: GET http://google.com/images
* Las posibles respuestas de código HTTPS:
  + 200 OK
  + 404 NOT FOUND

Crear (POST)

* La URL estará "abierta", es decir, el recurso aún no existe y por tanto no tiene id.
* El método debe ser post.
* Algunos códigos HTML que se pueden obtener de esta acción son:
  + 201: Recurso creado correctamente.
  + 403: Acceso prohibido.
  + 400: Petición incorrecta (falta un campo o valor no válido).
  + 500: Error del lado del servidor al crear el recursos.

Actualizar (PUT)

Aunque esto significa cambiar todos los datos existe un nuevo método estándar HTTP pensado para cambiar sólo algunos datos: PATCH.

Entre los resultados posibles:

* 201: Recurso creado cuando se le pasa el id deseado al servidor.
* 200: Recurso modificado correctamente.

Eliminar (DELETE)

Se hace mediante el método HTTP delete, tras ejecutarlo con éxito, las siguientes peticiones GET al recurso deberían devolver 404.

Los resultados posibles son:

* 200: Recurso borrado correctamente.
* 404: Not found.
* 500: Server error.

