Servicios Rest

Los servicios REST, o Transferencia de Estado Representacional (Representational State Transfer), son un estilo arquitectónico para el desarrollo de aplicaciones web. Este estilo se basa en los siguientes principios:

- Protocolo HTTP: Utiliza HTTP como su protocolo de comunicación, lo que lo hace ampliamente compatible con la infraestructura existente de Internet.
- Operaciones CRUD: Los servicios REST operan sobre recursos que pueden ser creados (Create), leídos (Read), actualizados (Update) y eliminados (Delete) utilizando los métodos HTTP estándar: POST, GET, PUT y DELETE, respectivamente.
- Sin estado (Stateless): Cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender y procesar la solicitud. El servidor no mantiene ningún estado de sesión entre las solicitudes.
- Interfaz uniforme: Los servicios REST utilizan una interfaz uniforme para acceder y manipular recursos a través de identificadores de recursos (URI), operaciones (métodos HTTP), representaciones de recursos (generalmente JSON o XML) y enlaces hipertextuales.
- Sistema cliente-servidor: Existe una clara separación entre el cliente y el servidor, lo que permite la independencia de la implementación y la escalabilidad.
- Cacheable (Almacenables en caché): Las respuestas a las solicitudes REST pueden ser almacenadas en caché para mejorar el rendimiento.

Un estilo de arquitectura es un conjunto coordinado de restricciones que controlan el funcionamiento y las características de los elementos de la arquitectura y permiten las relaciones de unos elementos con otros. Los elementos de una arquitectura son tres:

REST es un estilo de arquitectura, por lo tanto, es conveniente explicar este concepto:

Un estilo de arquitectura es un conjunto coordinado de restricciones que controlan el funcionamiento y las características de los elementos de la arquitectura y permiten las relaciones de unos elementos con otros.

Los elementos de una arquitectura son tres: componentes, conectores y datos:

• Componente: Es una unidad abstracta de instrucciones software y estados internos que proporciona una transformación de los datos a través de su interfaz.

• Conector: Es un mecanismo abstracto que hace posible la comunicación, coordinación y cooperación entre componentes.

• Datos: Es un elemento de información que se transfiere desde o hacia un componente a través de un conector.

Objetivos :

Cuando Roy T. Fielding creó REST su disertación, perseguía unos objetivos muy concretos que Paul Prescod resumió en un artículo :

• Escalabilidad de los componentes de interacción.
• Generalidad de las interfaces.
• Independencia en el desarrollo de componentes. • Sistemas intermedios para reducir el tiempo de interacción, mejorar la seguridad, y encapsular los sistemas de herencia.

Restricciones de Rest:

Para poder conseguir los objetivos que se han comentado en el apartado anterior, REST define un conjunto de restricciones o características que deben cumplir las arquitecturas de Servicios Web:

1. Cliente Servidor

2. Sin estado

3. Caché

4. Sistema de capas

5. Interfaz Uniforme

6. Sistema de capas

7. Código bajo demanda A continuación se va a explicar en qué consiste cada restricción.

Cliente Servidor :

La primera restricción que tiene REST está en común con el estilo Cliente-Servidor. Separar lo que es competencia del cliente y el servidor es el principio de todo. Hay que distinguir lo que concierne al interfaz del usuario del almacenamiento de datos. De esta manera se ayuda a mejorar la portabilidad de la interfaz de usuario a través de múltiples plataformas. Además también se mejora la escalabilidad porque se simplifican las componentes del

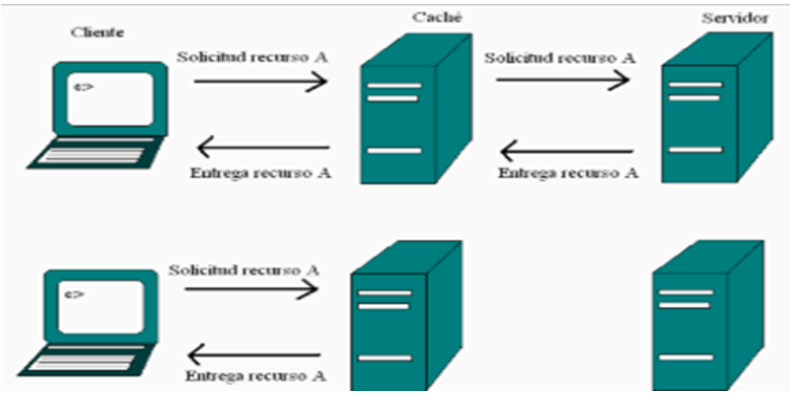
servidor al no tener que implementar las funcionalidades que van asociadas a la interfaz del usuario. Otro factor importante es que la separación permite a los componentes desarrollarse independientemente.

Sin Estado (Stateless) :

La segunda restricción está en común con el estilo client-stateless-server. Cada petición del cliente debe contener toda la información necesaria para que el servidor la comprenda y no necesite mirar ningún dato almacenado previamente sobre el contexto de la comunicación. El estado de la sesión por lo tanto se guarda íntegramente en el cliente:

Cache:

Esta tercera restricción la comparte REST con el estilo Client-CachestatelessServer Style. Las respuestas a una petición deben poder ser etiquetadas como cacheable o nocacheable. Si una respuesta es cacheable, entonces al cliente cache se le da permiso para reutilizar la respuesta más tarde si se hace una petición equivalente.



Interfaz Uniforme:

La principal característica que distingue a REST del resto de estilos de arquitecturas de red es el énfasis de usar una interfaz uniforme entre los componentes. Aplicando los principios de generalidad de la ingeniera del software a los componentes de la interfaz, se simplifica la arquitectura del sistema global y la visibilidad de interacciones se mejora. Las implementaciones se separan de los servicios que proporcionan, lo que anima al desarrollo independiente.

La desventaja de usar una interfaz uniforme es que degrada la eficiencia porque la información transferida está en una forma estandarizada y no según las necesidades que tenga la aplicación. El interfaz de REST está diseñado para ser eficiente con transferencias de datos de hipermedios (suelen ser datos voluminosos). Con esta decisión, está optimizado para la mayor parte de la Web pero no siendo así para otras formas de arquitectura de interacción. Para obtener una interfaz uniforme, REST define cuatro restricciones de interfaz: • Identificación de recursos • Manipulación de recursos a través de sus representaciones • Mensajes auto-descriptivos • Hipermedios como el motor del estado de la aplicación.

Sistema de Capas:

Para poder mejorar el comportamiento de la escalabilidad en Internet, se añade la restricción del sistema de capas. Este sistema permite tener una arquitectura compuesta por capas jerárquicas, limitando el comportamiento de los componentes porque no pueden “ver” mas allá de la capa con la que está interactuando.

Código Bajo Demanda:

La última restricción es opcional, consiste en permitir a los clientes tener la funcionalidad de descargar y ejecutar código en forma de applets y scripts. Esto simplifica el lado del cliente porque reduce el número de funcionalidades que tiene que tener implementadas al crearse. Las funcionalidades se pueden descargar posteriormente aumentando así la extensibilidad del sistema.

Los componentes que tiene REST son:

Agente Usuario, Servidor, Gateway (puerta de acceso) y Proxy. Un Agente Usuario usa un conector cliente para iniciar una petición y ser el verdadero receptor de la respuesta. El ejemplo más común es el navegador Web, proporciona Capítulo 5:

REST: Representational State Transfer: REST 68 acceso a los servicios de información y renderiza las respuestas del servicio de acuerdo con las necesidades de la aplicación. El Servidor usa un conector de servidor para gobernar el espacio de nombres cuando se le solicita un recurso. Es el que realiza la representación de sus recursos y debe ser el receptor último de todas las peticiones que solicitan modificar los valores de sus recursos. Cada servidor proporciona una interfaz genérica para sus servicios como una jerarquía de recursos. Los detalles de implementación de los recursos se ocultan detrás de la interfaz.

Los componentes de los sistemas intermedios actúan a la vez como cliente y como servidor para llevar a sus destinos las posibles transacciones, peticiones y respuestas. Un componente Proxy es un sistema intermedio seleccionado por el cliente para proporcionar la encapsulación de la interfaz de otros servicios, transporte de datos, perfeccionamiento del funcionamiento o seguridad. Los componentes de un Gateway son un sistema intermedio impuesto por la red o el servidor para proporcionar una encapsulación del interfaz de otros servicios, transmisión de datos, perfeccionamiento del funcionamiento o seguridad. La diferencia entre proxy y puerta de acceso es que el cliente es el que determina cuándo va a usar un proxy. A continuación se muestra la Tabla [5] que es una gráfica resumen de los componentes REST:

COMPONENTE	EJEMPLO DE LA WEB
Servidor Origen	Apache httpd, Microsoft IIS
Gateway	Squid, CGI, Reserve Proxy
Proxy	CERN Proxy, Netscape Proxy, Gauntlet
Usuario Agente	Netscape Navigator, Linxs, MOMspider

HTTP, URI y XML en REST

HTTP (Hypertext Transfer Protocol) es el protocolo subyacente utilizado en la World Wide Web para la transferencia de datos. Define cómo los mensajes deben ser formateados y transmitidos entre un cliente web y un servidor.

URI (Uniform Resource Identifier) es una cadena de caracteres que identifica un recurso en la web de manera única. Es un identificador de recursos genérico que puede ser utilizado para identificar recursos tanto en la web como fuera de ella.

XML (Extensible Markup Language) es un lenguaje de marcado diseñado para almacenar y transportar datos de manera legible tanto para humanos como para máquinas. Se utiliza comúnmente en servicios web REST como un formato para intercambiar datos entre clientes y servidores.

REST (Representational State Transfer) es un estilo arquitectónico para el diseño de sistemas distribuidos, como aplicaciones web. Se basa en la idea de que los recursos son accesibles a través de URLs únicas y que las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) se pueden realizar utilizando los métodos HTTP estándar (GET, POST, PUT, DELETE) sobre estos recursos.

En un servicio web REST, las URI se utilizan para identificar recursos específicos, HTTP se utiliza para definir las acciones que se realizarán en esos recursos (por ejemplo, GET para recuperar datos, POST para crear nuevos recursos, PUT para actualizar recursos existentes, DELETE para eliminar recursos) y XML se puede utilizar como un formato de intercambio de datos entre el cliente y el servidor, aunque también se pueden utilizar otros formatos como JSON (JavaScript Object Notation).

En servicios REST, los códigos de estado HTTP más comunes se utilizan para comunicar el resultado de la solicitud entre el cliente y el servidor. Aquí tienes algunos de los códigos de estado HTTP más comunes y sus significados en el contexto de servicios REST:

200 OK: Indica que la solicitud se ha completado correctamente y que el servidor ha devuelto los datos solicitados.

201 Created: Se utiliza para indicar que la solicitud se ha completado con éxito y que se ha creado un nuevo recurso en el servidor como resultado de la solicitud (por ejemplo, después de una solicitud POST).

204 No Content: Indica que la solicitud se ha procesado correctamente, pero que no hay contenido para devolver en la respuesta.

400 Bad Request: Indica que la solicitud del cliente no se pudo entender o procesar por parte del servidor debido a una sintaxis incorrecta o datos no válidos.

401 Unauthorized: Indica que se requiere autenticación para acceder al recurso solicitado, pero el cliente no ha proporcionado credenciales válidas.

403 Forbidden: Indica que el servidor ha entendido la solicitud, pero el cliente no tiene permiso para acceder al recurso solicitado.

404 Not Found: Indica que el servidor no pudo encontrar el recurso solicitado. Este es uno de los códigos de error más comunes y se utiliza cuando una URI no coincide con ningún recurso en el servidor.

405 Method Not Allowed: Indica que el método HTTP utilizado en la solicitud no está permitido para el recurso solicitado. Por ejemplo, intentar realizar un DELETE en un recurso que solo admite GET y POST.

500 Internal Server Error: Indica que ha ocurrido un error interno en el servidor que ha impedido que se pueda completar la solicitud del cliente.

503 Service Unavailable: Indica que el servidor no está disponible en este momento debido a una sobrecarga temporal o mantenimiento programado. El cliente puede intentar la solicitud nuevamente más tarde.

Estos son algunos de los códigos de estado HTTP más comunes en servicios REST, cada uno indicando una situación diferente entre el cliente y el servidor. Es importante que los clientes de una API REST estén diseñados para manejar estos códigos de estado de manera adecuada para proporcionar una experiencia de usuario robusta y amigable.

JSON Y YAML con Servicios Rest.

JSON (JavaScript Object Notation):

JSON es un formato ligero de intercambio de datos que se utiliza comúnmente en aplicaciones web y servicios REST. Se basa en la sintaxis de objetos de JavaScript, pero es independiente del lenguaje y se puede utilizar con muchos otros lenguajes de programación. JSON se compone de pares clave-valor, donde las claves son cadenas de caracteres y los valores pueden ser de varios tipos de datos, como cadenas, números, booleanos, matrices u otros objetos JSON. Este formato es fácil de leer y escribir para los humanos, y también es fácil de parsear y generar para las aplicaciones. Es ampliamente utilizado en servicios REST para intercambiar datos entre clientes y servidores debido a su simplicidad y su capacidad para representar datos estructurados de manera clara y concisa.

YAML (YAML Ain't Markup Language):

YAML es un formato de serialización de datos legible por humanos que se utiliza para representar datos de manera estructurada. Aunque inicialmente se desarrolló como un formato para archivos de configuración, YAML ha ganado popularidad en el desarrollo de aplicaciones debido a su legibilidad y facilidad de uso. En YAML, los datos se representan mediante una combinación de listas, objetos y tipos de datos simples como cadenas y números. Utiliza sangrías y espacios en blanco para definir la estructura de los datos, lo que lo hace más legible que otros formatos de serialización como JSON o XML. YAML es especialmente útil para definir estructuras de datos complejas de manera clara y concisa, lo que lo hace adecuado para describir configuraciones de aplicaciones y definiciones de servicios en el contexto de servicios REST.

API REST

Un API REST, que significa Representational State Transfer (Transferencia de Estado Representacional), es un conjunto de reglas y convenciones que permite que sistemas informáticos se comuniquen entre sí a través de la web de una manera uniforme y predecible.

En pocas palabras, un API REST proporciona una interfaz para que diferentes sistemas puedan interactuar entre sí utilizando el protocolo HTTP estándar. Esto significa que las aplicaciones pueden solicitar datos o realizar acciones en un servidor web utilizando solicitudes HTTP como GET, POST, PUT o DELETE, y el servidor responderá con datos estructurados generalmente en formato JSON o XML.

Las API REST son muy utilizadas en el desarrollo de aplicaciones web y móviles, ya que permiten una comunicación flexible y eficiente entre sistemas distribuidos en diferentes plataformas.

Usos y versiones o Especificaciones :

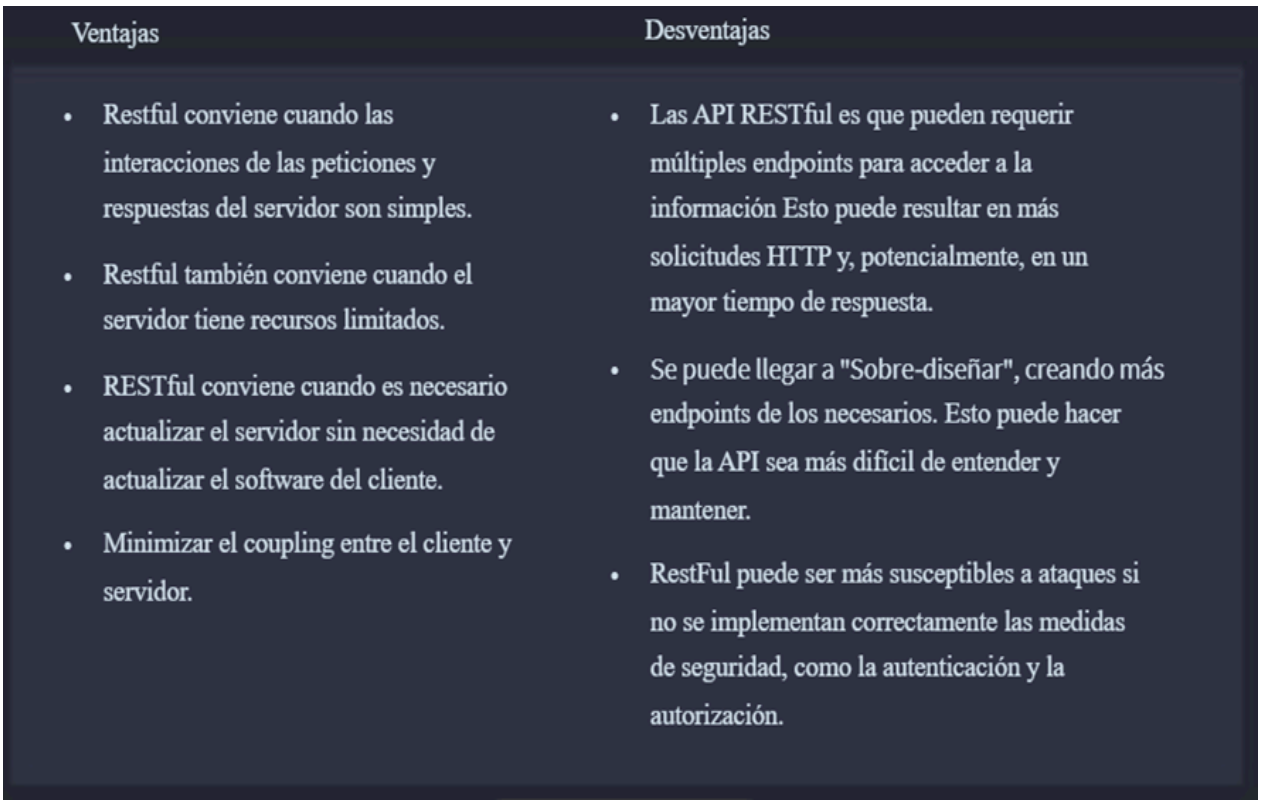
API RESTful Públicas:Son APIs que están abiertas al público y pueden ser utilizadas por cualquier persona o aplicación.

·API RESTful Privadas:Estas APIs solo están disponibles para usuarios autorizados, como los empleados de una empresa o los desarrolladores que han sido dados acceso a la API.

·API RESTful Servicio a Servicio:Estas APIs permiten la comunicación entre diferentes servicios dentro de una misma aplicación o entre diferentes aplicaciones.

·API RESTful de Datos: Estas APIs permiten el acceso a los datos de una aplicación. Un ejemplo de esto es la API de un sistema de gestión de bases de datos.

·API RESTful de Operaciones: Estas APIs permiten la realización de operaciones específicas en una aplicación.



Semántica de Recursos Rest :

Los recursos en un servicio web RESTful se intercambian entre el cliente y el servidor, que representan entidades comerciales o datos. El protocolo HTTP establece métodos o acciones para recursos como POST, GET, PUT y DELETE, lo que simplifica el diseño de la API REST al asignar estas equivalencias HTTP en estas acciones de datos:

CREATE:POST || PUT

•READ:GET

•UPDATE:PUT || PATCH

•DELETE:DELETE

Hay dos métodos usados con menos frecuencia en el contexto de las implementaciones RESTful:

•OPTIONS:este método es usado por el cliente para determinar las opciones/acciones asociadas con el recurso de destino, sin causar ninguna acción sobre el recurso o la recuperación del recurso.

•HEAD:este método es utilizado para recuperar sobre la entidad sin tener la propia entidad en la respuesta

Conclusiones :

- REST hace énfasis en una interfaz uniforme entre componentes, que abstrae la información que transmite.
- REST no prescribe ningún formato de mensaje específico para la comunicación cliente-servidor, por lo que se puede utilizar cualquier formato compatible con HTTP
- REST tiene restricciones para hacer una aplicación una aplicación REST, según Fielding.