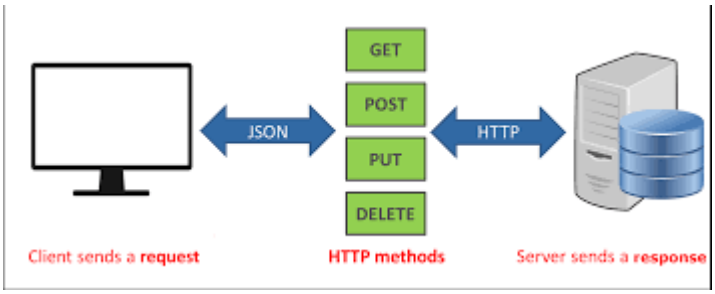
Universidad Israel

DEPARTAMENTO: Ciencias de la Ingeniería **PROFESOR:** Mg. Luis Fernando Aguas B.
CARRERA: Sistemas de Información **ESTUDIANTE:** Marco Antonio Ayala Lituma
CURSO: Octavo **PARALELO:** "A" **DESCRIPCIÓN:** Tarea Semana 3
ASIGNATURA: Plataformas de Desarrollo 2

TEMA: REST

DESARROLLO:

En los últimos años, se ha vuelto claro que HTTP no es solo para proporcionar páginas HTML. También es una plataforma eficaz para la creación de API Web, con una serie de verbos (GET, POST, etc.) además de algunos conceptos simples, como los URI y los encabezados. ASP.NET Web API es un conjunto de componentes que simplifican la programación HTTP. Dado que se basa en el tiempo de ejecución de ASP.NET MVC, Web API controla automáticamente los detalles de transporte de bajo nivel de HTTP. Al mismo tiempo, la API Web expone de forma natural el modelo de programación HTTP. De hecho, un objetivo de Web API es no abstraer la realidad de http. Como resultado, la API Web es flexible y fácil de ampliar. El estilo arquitectónico REST ha demostrado ser una manera eficaz de aprovechar HTTP, aunque ciertamente no es el único enfoque válido para HTTP. El administrador de contactos expondrá el RESTful para enumerar, agregar y quitar contactos, entre otros.



Definiciones de RESTful serían:

Cliente-servidor: El servidor se encarga de controlar los datos mientras que el cliente se encarga de manejar las interacciones del usuario. Esta restricción mantiene al cliente y al servidor débilmente acoplados (el cliente no necesita conocer los detalles de implementación del servidor y el servidor se "despreocupa" de cómo son usados los datos que envía al cliente).
Sin estado: aquí decimos que cada petición que recibe el servidor debería ser independiente y contener todo lo necesario para ser procesada.

Cacheable: debe admitir un sistema de almacenamiento en caché. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.
Interfaz uniforme: define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección o "URI".

Sistema de capas: el servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.
Hoy en día la mayoría de las empresas utilizan API REST para crear servicios web. Esto se debe a que es un estándar lógico y eficiente. Por poner algún ejemplo tenemos los sistemas de identificación de Facebook o también la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, Google Maps, entre otras).

Métodos en una API REST
Las operaciones más importantes que nos permitirán manipular los recursos son:

GET es usado para recuperar un recurso.

POST se usa la mayoría de las veces para crear un nuevo recurso. También puede usarse para enviar datos a un recurso que ya existe para su procesamiento. En este segundo caso, no se crearía ningún recurso nuevo.

PUT es útil para crear o editar un recurso. En el cuerpo de la petición irá la representación completa del recurso. En caso de existir, se reemplaza, de lo contrario se crea el nuevo recurso. PATCH realiza actualizaciones parciales. En el cuerpo de la petición se incluirán los cambios a realizar en el recurso. Puede ser más eficiente en el uso de la red que PUT ya que no envía el recurso completo.

DELETE se usa para eliminar un recurso.
Otras operaciones menos comunes pero también destacables son:

HEAD funciona igual que GET pero no recupera el recurso. Se usa sobre todo para testear si existe el recurso antes de hacer la petición GET para obtenerlo (un ejemplo de su utilidad sería comprobar si existe un fichero o recurso de gran tamaño y saber la respuesta que obtendríamos de la API REST antes de proceder a la descarga del recurso).
OPTIONS permite al cliente conocer las opciones o requerimientos asociados a un recurso antes de iniciar cualquier petición sobre el mismo.
Dos términos a tener en cuenta son los de métodos seguros y métodos idempotentes. Se dice que los métodos seguros son aquellos que no modifican recursos (serían GET, HEAD y OPTIONS), mientras que los métodos idempotentes serían aquellos que se pueden llamar varias veces obteniendo el mismo resultado (GET, PUT, DELETE, HEAD y OPTIONS).

La idempotencia es de mucha importancia ya que permite que la API sea tolerante a fallos. Por ejemplo, en una API REST bien diseñada podremos realizar una operación PUT para editar un recurso. En el caso de que obtuviésemos un fallo de Timeout (se ha superado el tiempo de espera de respuesta a la petición), no sabríamos si el recurso fue creado o actualizado. Como PUT es idempotente, no tenemos que preocuparnos de comprobar su estado ya que, en el caso de que se haya creado el recurso, una nueva petición PUT no crearía otro más, algo que sí habría podido pasar con una operación como POST.

Algunas características de una API REST
El uso de hipermedios (procedimientos para crear contenidos que contengan texto, imagen, vídeo, audio y otros métodos de información) para permitir al usuario navegar por los distintos recursos de una API REST a través de enlaces HTML (principio HATEOAS, Hypermedia as the engine of application state o hipermedia como el motor del estado de la aplicación).

Independencia de lenguajes. La separación en capas de la API permite que el cliente se despreocupe del lenguaje en que esté implementado el servidor. Basta a ambos con saber que las respuestas se recibirán en el lenguaje de intercambio usado (que será XML o JSON).
Los recursos en una API REST se identifican por medio de URI. Será esa misma URI la que permitirá acceder al recurso o realizar cualquier operación de modificación sobre el mismo.
Las APIs deben manejar cualquier error que se produzca, devolviendo la información de error adecuada al cliente. Por ejemplo, en el caso de que se haga una petición GET sobre un recurso inexistente, la API devolvería un código de error HTTP 404.

Algunos frameworks con los que podremos implementar nuestras APIs son: JAX-RS y Spring Boot para Java, Django REST framework para Python, Laravel para PHP, Rails para Ruby o Restify para Node.js.

COMENTARIO:

Aunque la evolución de comunicación de sistemas ha evolucionado acorde a las tecnologías existen varias herramientas como se lo anteriormente con métodos SOAP, y estas trabajaban mediante una arquitectura que para la actualidad la demanda de consumo de información es mas demandante por grandes cantidades de información y que esta se pueda mapear, REST es una de las métodos mas comunes hoy en día ya que nos da gran facilidad en las definiciones anteriores mencionadas y por su comunicación mas limpia al trabajar con arquitectura json, porque al futuro también salen nuevas herramientas como lo son graph que es una evolución super mas fuerte q REST con el mismo principio.

BIBLIOGRAFÍA:

Compilación de API de RESTful con ASP.NET Web API recuperado en: Cree API de RESTful con ASP.NET Web API ASP.NET 4. x | Microsoft Docs
Que es REST, recuperado en: ¿Qué es REST? Conoce su potencia | OpenWebinars
Diferencias entre REST y SOAP, recuperado en: Diferencias entre REST y SOAP (redhat.com)