TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- APIs Restful
- Qué significa REST y para qué sirve
- Características principales
- Ventajas y desventajas
- Reglas de una arquitectura REST
- Buenas prácticas en la creación de una API REST o Versionamiento de la API
- Verbos HTTP
- Búsqueda y filtrado
- HATEOAS

QUÉ SIGNIFICA REST Y PARA QUÉ SIRVE

Las API REST son una forma estándar de la industria para que los servicios web envíen y reciban datos. Utilizan métodos de solicitud HTTP para facilitar el ciclo de solicitud-respuesta y, por lo general, transfieren datos mediante JSON y, más raramente, HTML, XML u otros formatos.

REST (Representational State Transfer, Transferencia de estado representacional), es una arquitectura estándar para construir y comunicarse con servicios web. Por lo general, exige que los recursos en la web se representan en un formato de texto (como JSON, HTML o XML) y se puede acceder a ellos o modificarlos mediante un conjunto predeterminado de operaciones. Dado que normalmente construimos API REST para aprovechar HTTP en lugar de otros protocolos, estas operaciones corresponden a métodos HTTP como **GET**, **POST** o **PUT**.

Aunque muchas veces se usan los términos Rest y Restful como sinónimos, no lo son. Rest, es un modelo de arquitectura web basado en el protocolo HTTP para mejorar las comunicaciones cliente-servidor, mientras que Restful Web Service o Restful API son programas basados en REST.

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful

CARACTERÍSTICAS PRINCIPALES

Los objetos REST son manipulados a través de una URI (Uniform Resource Identifier) Esta URI (endpoint) hace de identificador único de cada recurso del sistema REST, por lo que no puede ser compartida por más de un recurso. La estructura básica de una URI es la siguiente:

```
{protocolo}://{hostname}:{puerto}/{ruta del recurso}?{parámetros de filtrado (opcional)}
```

El nombre de la URI no debe contener palabras que impliquen acciones, por lo que deben evitarse los verbos en su construcción. Además, las URI siguen una jerarquía lógica de capas que permite ordenar los recursos y englobar las distintas funcionalidades entre sí. Por ejemplo:

```
https://zoo-animal-api.herokuapp.com/animals/rand
```

O bien agregando un cuerpo a la llamada REST en cualquier tipo de formato, siendo los más usados JSON y XML.

TIPOS DE SALIDAS

Como se indicó anteriormente, las API pueden generar datos en formatos JSON, XML e incluso HTML, aunque este último no es tan común como los dos primeros. El siguiente es el resultado de llamar a una API que solo devuelve datos en formato JSON.

```
// 20220119144018
// https://zoo-animal-api.herokuapp.com/animals/rand

{
    "name": "Yellow Rat Snake",
    "latin_name": "Pantherophis alleghaniensis",
    "animal_type": "Reptile",
    "active_time": "Nocturnal",
    "length_min": "3",
    "length_max":  "6",
    "weight_min":"1.2",
    "weight_max": "1.8",
    "lifespan": "20",
    "habitat": "Woods, fields, and swamps",
    "diet": "Rodents, birds, eggs, lizards, frogs, and insects",
    "geo_range": "Southeastern United States",
    "image_link":
"https://upload.wikimedia.org/wikipedia/commons/d/de/Pantherophis_allegha
niensis_ssp._quadrivittata_03.JPG",
    "id":195
}
```

```
22
23
```

Si la misma API pudiera devolver información en formato XML, la salida se vería así:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <active_time>Nocturnal</active_time>
   <animal_type>Reptile</animal_type>
   <diet>Rodents, birds, eggs, lizards, frogs, and insects</diet>
   <geo_range>Southeastern United States</geo_range>
   <habitat>Woods, fields, and swamps</habitat>
   <id>195</id>
   <image_link>https://upload.wikimedia.org/wikipedia/commons/d/de/
Pantherophis_alleghaniensis_ssp._quadrivittata_03.JPG</image_link>
   <latin_name>Pantherophis alleghaniensis</latin_name>
   <length_max>6</length_max>
   <length_min>3</length_min>
   <lifespan>20</lifespan>
   <name>Yellow Rat Snake</name>
   <weight_max>1.8</weight_max>
   <weight_min>1.2</weight_min>
</root>
```

Para el desarrollo de una API REST es necesario un conocimiento profundo de la especificación HTTP, sobre todo en lo referente a métodos permitidos, códigos de estado y aceptación de tipos de contenido.

Los métodos son usados para manipular los diferentes recursos que conforman la API. Los principales métodos soportados por HTTP y por ello usados por una API REST son:

POST: crear un recurso nuevo.
PUT: modificar un recurso existente.
GET: consultar información de un recurso.
DELETE: eliminar un recurso determinado.
PATCH: modificar solamente un atributo de un recurso.

VENTAJAS Y DESVENTAJAS

- **Separación de un recurso de su representación:** Rest puede tener múltiples representaciones. No tiene estado, por lo tanto, hay que especificar en el encabezado de la petición HTTP el tipo de

contenido que se desea obtener. Tras este paso, será la aplicación del servidor la encargada de manejar la representación y devolver el estado HTTP apropiado, al cual finalmente deberíamos enviarle al servidor qué es lo que esperamos recibir.

- **Visibilidad:** Rest está diseñado para ser visible y simple, lo que significa que cada aspecto del servicio debe ser auto descriptivo siguiendo las normas HTTP.

- **Seguridad:** Al utilizar Rest garantizamos que los métodos HTTP son seguros, lo que significa que, al solicitar un recurso, este requerimiento no modifica o causa ningún tipo de cambio en su estado.

- **Escalabilidad:** Si el aumento de la demanda exige aumentar el número de servidores, esto puede hacerse sin preocuparse por la sincronización entre los mismos, puesto que no hay que estar pendiente del estado de los recursos.

- **Rendimiento:** La escalabilidad no debe ser confundida con el rendimiento. El rendimiento se mide por el tiempo necesario para que una única petición sea procesada, mientras que la escalabilidad depende del número total de peticiones que la aplicación puede manejar.

- **Tiempo de respuesta mínimo:** Al ser pequeños servicios, el tiempo de respuesta es 0 o con tendencia a 0. Por lo tanto, con esta arquitectura estamos aprovechando el procesamiento de las máquinas cliente para usar su poder de procesamiento.

- **Separación entre el cliente y el servidor:** API REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

- **La API REST siempre es independiente del tipo de plataformas o lenguajes:** Se adapta al tipo de sintaxis o plataformas con las que se esté trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente JSON o XML.

Entre algunas de las desventajas que podemos destacar de API para tener una buena presencia en el desarrollo de proyectos:

- El sistema de API REST, al poder disponer de más de un servidor, no sabes de forma concreta cuál atenderá tu demanda al sistema y cómo solucionará tu solicitud que previamente le has confirmado.

- API REST no sustenta el estado de la aplicación, implicando que el cliente tenga que establecer una estructura propia para mantener el conjunto de la aplicación. Para ello será necesario enviarle al servidor un token para que pueda identificarnos e indicarle a la misma vez la función desempeñada en la aplicación.

- Para poder desarrollar la estructura de debe invertir tiempo en la construcción del API. Este desarrollo requiere una serie de procedimientos de larga duración y formación para poner en marcha estableciendo un nuevo lenguaje, protocolo HTML y bases de datos para lograr la creación de esta nueva estructura propia.

- Requiere un mayor esfuerzo, debido a que es conveniente la realización de pruebas para poder comprobar que todo funciona correctamente.

REGLAS DE UNA ARQUITECTURA API REST

INTERFAZ UNIFORME

- La interfaz se basa en recursos, por ejemplo, el recurso Empleado (Id, Nombre, Apellido, Sueldo, Carnet).
- El servidor enviará los datos (vía json, xml, html...) y para el cliente es transparente.
- La representación del recurso que le llega al cliente será suficiente para poder cambiar/borrar el recurso.
- Interfaz uniforme con mensajes descriptivos
    - Usar las características del protocolo http para mejorar la semántica:
    - HTTP Verbs()
    - HTTP Status Codes

    - HTTP Authentication
- Procurar una API sencilla y jerárquica y con ciertas reglas: uso de nombres en plural

PETICIONES SIN ESTADO

Http es un protocolo sin estado, logrando de este modo mayor rendimiento, por ejemplo:

```
1 GET mi_url/empleados/1234
2 DELETE mi_url/empleados/1234
```

- En la segunda petición colocamos un identificador del recurso que queremos borrar.
- El servidor no guardaba los datos de la consulta previa que tenía el cliente en particular.
- Una petición del tipo DELETE mi_url/empleado debe dar error, falta el identificador y el servidor no lo conoce.

CACHEABLE

- En la web los clientes pueden almacenar el caché de las respuestas del servidor
- Las respuestas se deben marcar de forma implícita o explícita como cacheables o no
- En futuras peticiones, el cliente sabrá si puede reutilizar o no los datos que ya ha obtenido.
- Ahorramos las peticiones, mejoramos la escabilidad de la aplicación y el rendimiento en el cliente evitando la latencia

SEPARACIÓN DE CLIENTE Y SERVIDOR

- El cliente y servidor están separados, su unión es mediante la interfaz uniforme
- Los desarrollos en frontend y backend se hacen por separado, teniendo en cuenta la API.
- Mientras la interfaz no cambie, podremos cambiar el cliente o el servidor sin problemas.

SISTEMA DE CAPAS

- El cliente puede estar conectado mediante la interfaz al servidor o a un intermediario, para él es irrelevante y desconocido.
- Al cliente solo le preocupa que la API REST funcione como debe: no importa el CÓMO sino el QUE
- El uso de capas o servidores intermedios puede servir para aumentar la escalabilidad (sistema de balanceo de carga, cachés) o para implementar políticas de seguridad

CÓDIGO BAJO DEMANDA

- Los servidores pueden ser capaces de aumentar o definir cierta funcionalidad en el cliente transfiriendo cierta lógica que pueda ejecutar, por ejemplo: componentes compilados como applets de Java, JavaScript en cliente.

FORMATO DE SALIDA

- El formato de salida predeterminado para las API RESTful son archivos HTML navegables y legibles por humanos, pero este formato no siempre es el más conveniente de usar para los desarrolladores web, por lo que hay otros formatos que las API pueden generar que facilitan mucho la transmisión de datos. Los siguientes son algunos ejemplos de los diferentes formatos de salida de las API:
    - **CSV:** Un archivo de valores separados por comas es un archivo de texto delimitado que usa una coma para separar valores.
    - **HTML:** Salida de la base de datos visible en formato html (predeterminado)
    - **JSON:** Es un formato estándar basado en texto para representar datos estructurados según la sintaxis de objetos de JavaScript.
    - **XLS/XLSX:** Los datos se envían en formato de hoja de cálculo de Excel, probablemente lo mejor para el análisis posterior por parte de los usuarios finales.
    - **XML:** Es un archivo utilizado para almacenar datos en forma de elementos jerárquicos. Los datos almacenados en archivos XML pueden ser leídos por programas informáticos con la ayuda de etiquetas personalizadas, que indican el tipo de elemento.

BUENAS PRÁCTICAS EN LA CREACIÓN DE UNA API REST

EVITA DEVOLVER SIEMPRE EL MISMO CÓDIGO DE ESTADO HTTP

En el diseño de una API RESTful se debe evitar enviar un solo código HTTP estándar, por ejemplo no solo el código 200 cuando la respuesta es exitosa o 400 cuando la petición es incorrecta o error por parte del cliente. En este sentido se debe tener en cuenta que por lado del cliente todo debe ser transparente, y el no debe en sí conocer la lógica de la API del servidor,

La recomendación es utilizar códigos de estado HTTP para las respuestas del servidor. Puede encontrar los siguientes códigos de estado según RFC7321 o en https://restfulapi.net/http-status-codes/, por ejemplo algunos:

1xx Códigos de estado informativo

Código de estado	Descripción
100	Continuar
101	Cambio de protocolos
102	Procesamiento

2xx Códigos de estado de respuesta exitosa

Código de estado	Descripción
200	ok
201	Creado
202	Aceptado

3xx Códigos de estado de redirección

Código de estado	Descripción
300	Opciones múltiple
301	Movido permanentemente
302	Encontrado

4xx Códigos de estado de error del cliente

Código de estado	Descripción
400	Petición incorrecta
401	No autorizado
403	Prohibido
404	No encontrado

5xx Códigos de estado de error del servidor

Código de estado	Descripción
500	Error interno del servidor
501	No implementado
502	Puerta de enlace no válida
503	Servicio no disponible

NO UTILIZAR FORMATOS DE RESPUESTA DE ERROR DIFERENTES

Se debe mantener las respuestas de error en el mismo formato, esto facilitará la integración de los clientes con las API

UTILIZAR LOS MÉTODOS ESTANDARIZADOS DE HTTP

- `GET:` transfiere una representación actual del recurso de destino.
- `POST:` realiza un procesamiento específico de recursos en la carga útil de la solicitud.
- `PUT:` Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la solicitud.
- `DELETE:` Elimina todas las representaciones actuales del recurso de destino.

CONTROL DE VERSIONES DEL API REST

Las API solo deben actualizarse cuando se realiza un cambio importante. Los cambios importantes incluyen:

- Un cambio en el formato de los datos de respuesta para una o más llamadas
- Un cambio en el tipo de solicitud o respuesta (es decir, cambiar un número entero a un flotante)
- Eliminando cualquier parte de la API.

Los cambios importantes siempre deben dar lugar a un cambio en el número de versión principal de una API o tipo de respuesta de contenido .

Los cambios no importantes, como agregar nuevos puntos finales o nuevos parámetros de respuesta, no requieren un cambio en el número de versión principal.

REST no proporciona ninguna guía de control de versiones específica, pero los enfoques más utilizados se dividen en tres categorías:

- Control de versiones de URI

```
http://api.example.com/v1
http://apiv1.example.com
```

- Control de versiones mediante encabezado de solicitud personalizado

```
Accept-version: v1
Accept-version: v2
```

- Control de versiones usando el encabezado "Aceptar"

```
Accept: application/vnd.example.v1+json
Accept: application/vnd.example+json;version=1.0
```

INCLUIR TOKEN DE SESIÓN EN ENCABEZADOS

La mayoría de las API tienen solicitudes privadas, solo se puede acceder a las solicitudes si hay una sesión activa. A veces, los desarrolladores incluyen el token de sesión como parte de las URL utilizando un parámetro de consulta o implementando todas las solicitudes con el método `POST` para enviar el token de sesión como un parámetro en el cuerpo.

INTERNACIONALIZA Y DOCUMENTACIÓN DE LA API

Es muy común devolver mensajes de error, exitosos o informativos en nuestra API. Por lo general el cliente no debe contener ninguna lógica de servidor, ni siquiera para mensajes. A veces, necesitamos mostrar esos mensajes al cliente. Luego, debemos enviar internacionalizados según el dispositivo del cliente.

PRUEBAS AUTOMATIZADAS EN EL BACKEND

Las pruebas deberían ser obligatorias en todos los proyectos pero aún más con la parte que tiene toda la lógica del producto. Ayudarán a no estropear o incluir nuevos errores en las funciones que están funcionando actualmente. Es recomendable realizar algunas pruebas de integración y unitarias.

UTILICE HTTPS SOBRE HTTP Y CERTIFICADOS VÁLIDOS SEGÚN SUS CLIENTES

Si desea tener una API segura, primero deberá ejecutarla con un esquema HTTPS, este hecho es conocido por la mayoría de las personas.

HATEOAS

HATEOAS (Hypermedia as the Engine of Application State) es una restricción de la arquitectura de aplicaciones REST. HATEOAS mantiene la arquitectura de estilo REST única de la mayoría de las otras arquitecturas de aplicaciones de red.

El término " hipermedia " se refiere a cualquier contenido que contenga enlaces a otras formas de medios, como imágenes, películas y texto.

El estilo arquitectónico REST nos permite usar los enlaces hipermedia en los contenidos de respuesta de la API. Permite que el cliente navegue dinámicamente con los recursos apropiados atravesando los enlaces hipermedia.

Navegar por enlaces hipermedia es conceptualmente lo mismo que navegar por páginas web haciendo clic en los hipervínculos relevantes para lograr un objetivo final.

Por ejemplo, la respuesta JSON dada a continuación puede ser de una API como HTTP `GET` http://api.domain.com/management/departments/10

```
{
    "departmentId": 10,
    "departmentName": "Administration",
    "locationId": 1700,
    "managerId": 200,
    "links": [
        {
            "href": "10/employees",
            "rel": "employees",
            "type" : "GET"
        }
    ]
}
```

En el ejemplo anterior, la respuesta devuelta por el servidor contiene enlaces hipermedia a los recursos de los empleados `10/employees` que el cliente puede atravesar para leer los empleados que pertenecen al departamento.

La ventaja del enfoque anterior es que los enlaces hipermedia devueltos desde el servidor controlan el estado de la aplicación y no al revés.

JSON no tiene ningún formato universalmente aceptado para representar enlaces entre dos recursos. Podemos optar por enviar el cuerpo de la respuesta o decidir enviar enlaces en encabezados de respuesta HTTP.