Aplicaciones RESTful:
Conceptos Generales

Teoría

Existen diversos tipos de aplicaciones. Uno de los tipos más nombrados en la actualidad es este. Cuando hablamos de aplicaciones RESTful, nos referimos a aplicaciones que operan en forma de servicios web, respondiendo consultas a otros sistemas a través de internet, y lo hacen respetando algunas reglas y convenciones que detallaremos a lo largo de este documento.

API

Una API (sigla de Application Programming Interface) es un conjunto de reglas y especificaciones que describen la manera en que un sistema puede comunicarse con otros. Definir una API en forma clara y explícita, habilita y facilita el intercambio de mensajes entre sistemas, y permite la colaboración e interoperabilidad entre los mismos, aún cuando éstos hayan sido desarrollados para distintas plataformas e incluso en distintos lenguajes.

Algunas APIs cuentan con una interfaz gráfica que puede ser embebida en un sitio web y directamente usada por un usuario, mientras que otras sólo son de uso interno, es decir que un usuario nunca accederá directamente a ella, sino que será algún programa quien la utilice (internamente).

Por lo dicho, es importante al momento de crear una API también generar la documentación detallada que acompañe, en donde se especifique con precisión cómo se debe interactuar con la misma (esto es, qué tipo de mensajes puede recibir, y qué clase de respuestas se puede esperar de la misma).

Como ejemplo de API con interfaz gráfica accesible directamente por usuarios podemos nombrar la API de Google Maps, que nos permite embeber en nuestros sitios y aplicaciones mapas con toda la información actualizada de google, sin necesidad de haberlos programado. O también, las APIs de redes sociales como Facebook o Twitter, que nos permiten compartir artículos y publicaciones en las redes desde el mismo sitio que estamos mirando, sin necesidad de salir del mismo, vinculándose automáticamente con nuestra cuenta (previo login), y realizando la publicación del contenido en cuestión.

REST

REST viene del inglés “REpresentational State Transfer” (o en español: Transferencia de Estado Representacional).
Por Representación nos referimos a un modelo o estructura con la que representamos algo.

Por Estado de una representación, hablamos de los datos que contiene ese modelo estructura. Transferir un Estado de Representación implica el envío de datos (con una determinada estructura) entre dos partes.

Los dos formatos más utilizados para este tipo de transferencias de datos son XML y JSON. Ambos formatos permiten asociar valores con identificadores, así como también generar estructuras anidadas.

Ejemplo XML

```xml
<factura>
  <cliente>Gomez</cliente>
  <emisor>Perez S.A.</emisor>
  <tipo>A</tipo>
  <items>
    <item>Producto 1</item>
    <item>Producto 2</item>
    <item>Producto 3</item>
  </items>
</factura>
```

Ejemplo JSON

```json
{
  "cliente":"Gomez",
  "emisor":"Perez S.A.",
  "tipo":"A",
  "items": [
    "Producto 1",
    "Producto 2",
    "Producto 3"
  ]
}
```

HTTP

HTTP (*Hypertext Transfer Protocol* o Protocolo de Transferencia de HiperTexto) es, como su nombre lo dice un protocolo (conjunto de reglas y especificaciones) que se utiliza a la hora de intercambiar datos a través de internet.

El funcionamiento del protocolo se basa en un esquema de petición-respuesta entre un cliente que realiza la solicitud de transmisión de datos, y un servidor que atiende la petición. Un cliente puede ser, por ejemplo, un navegador cuando intentamos abrir una página web, o una

aplicación de mensajería que precisa enviar mensajes a otros usuarios. A ellos el servidor brinda una respuesta estructurada de modo puntual y dotada de una serie de metadatos, que establecen las pautas para el inicio, desarrollo y cierre de la transmisión de la información.
Http establece varios tipos de peticiones, siendo las principales: POST, GET, PUT, y DELETE. Cada tipo de petición tiene asociada una operación, definida por el servidor, pero siguiendo las convenciones actuales.

Otro aspecto que define el protocolo es la forma de comunicar el resultado de las peticiones HTTP.
Cada vez que se realiza una petición, el servidor deberá responder con algún mensaje. Cada mensaje de respuesta debe tener un código de estado numérico de tres cifras.
El formato de los códigos de estado es el siguiente:

1xx (Informativo): La petición fue recibida, y continúa su procesamiento.
2xx (Éxito): La petición fue recibida con éxito, comprendida y procesada.
3xx (Redirección): Más acciones son requeridas para completar la petición.
4xx (Error del cliente): La petición tiene algún error, y no puede ser procesada.
5xx (Error del servidor): El servidor falló al intentar procesar una petición aparentemente válida.

Algunos de los Códigos de Estado más comunes (y más genéricos) son:

200	OK	Todo salió como lo esperado
400	Bad Request	La petición no cumple con lo esperado
404	Not Found	El recurso buscado no existe (URI inválido)
500	Internal Server Error	Error genérico del servidor al procesar una petición válida

API REST

Este tipo de API tiene como particularidad que no posee interfaz gráfica, y se utiliza exclusivamente para comunicación entre sistemas, mediante el protocolo HTTP.
Para que una API se considere REST, debe cumplir con las siguiente características:

Arquitectura Cliente / Servidor sin estado

Cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Esta restricción mantiene al cliente y al servidor débilmente acoplados. Esto quiere decir que 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.

Cacheable

Debe admitir un sistema de almacenamiento en caché. La infraestructura de red debe soportar una caché de varios niveles. Este almacenamiento evita repetir varias conexiones entre el servidor y el cliente, en casos en que peticiones idénticas fueran a generar la misma respuesta.

Operaciones comunes

Todos los recursos detrás de nuestra API deben poder ser consumidos mediante peticiones HTTP, preferentemente sus principales (POST, GET, PUT y DELETE).
Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (en inglés: Create, Read, Update, Delete, en español: Alta, Lectura, Modificación, y Baja).
Al tratarse de peticiones HTTP, éstas deberán devolver con sus respuestas los correspondientes códigos de estado, informando el resultado de las mismas.

Interfaz uniforme

En un sistema REST, cada acción (más correctamente, cada recurso - ver próximo punto) debe contar con una URI (Uniform Resource Identifier), un identificador único. Ésta nos facilita el acceso a la información, tanto para consultarla, como para modificarla o eliminarla, pero también para compartir su ubicación exacta a terceros.

Utilización de hipermedios

Cada vez que se hace una petición al servidor y este devuelve una respuesta, parte de la información devuelta pueden ser también hipervínculos de navegación asociada a otros recursos del cliente. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

Recursos en lugar de funcionalidades

Una aplicación RESTful requiere un enfoque de diseño diferente al que estamos a la forma típica de pensar en un sistema, con llamados a funciones, usualmente en forma de acciones. Existe una forma de trabajo que sigue este último modelo, basado en RPC (Remote Procedure Calls, llamadas a procedimientos remotos). En RPC, se pone el énfasis en la diversidad de operaciones que puede realizar el sistema (acciones, usualmente verbos).
Por ejemplo, una aplicación RPC podría definir operaciones como:

- getUsuario()
- addUsuario()
- removeUsuario()
- updateUsuario()

- listUsuarios()
- findUsuario()

En REST, al contrario, el énfasis se pone en los **recursos** (usualmente sustantivos), especialmente en los nombres que se le asigna a cada tipo de recurso.
Por ejemplo, una aplicación REST con los mismos fines que la anterior podría definir el siguiente recurso:

- Usuarios

Luego, cada funcionalidad relacionada con este recurso tendría sus propios identificadores:

Listar usuarios:
Petición HTTP de tipo GET a la URL: *http://servicio/api/usuarios*

Agregar usuario:
Petición HTTP de tipo POST a la URL: *http://servicio/api/usuarios*
(Agregando a la petición el registro correspondiente con los datos del nuevo usuario)

Obtener al usuario 1:
En caso de querer acceder a un elemento en particular dentro de un recurso, se pueden hacerlo fácilmente si se conoce su identificador (URI):

Petición HTTP de tipo GET a la URL: *http://servicio/api/usuarios/1*

Modificar al usuario 1:
Para actualizar un dato del usuario, un cliente REST podría primero descargar el registro anterior usando GET. El cliente después modificaría el objeto para ese dato, y lo enviaría al servidor utilizando una petición HTTP de tipo PUT.

Obtener usuarios con domicilio en CABA:
Si, en cambio, es necesario realizar una búsqueda por algún criterio, se pueden enviar parámetros en una petición HTTP. Éstos se pueden añadir al final de la misma siguiendo la siguiente sintaxis:

Petición HTTP de tipo GET a la URL: *http://servicio/api/usuarios?domicilio=CABA*