# Curso de API REST

No se puede hablar de REST sin hablar de HTTP, porque es la base de todo, incluso de REST, ya que se apoya fuertemente en él.

¿Qué es HTTP?   
HyperText transference Protocol

Protocolo de comunicación entre aplicaciones. Basado en el intercambio de texto.

HTTP es la manera de comunicarse y por debajo va TCP IP que es lo que posibilita que los bytes salgan de la computadora y lleguen a otra.

¿Qué es un protocolo?

Es un concepto similar al de API, se trata de un conjunto de reglas que definen como se hará la comunicación entre dos entidades; en el caso de HTTP, será entre computadoras.

### ¿Qué es un hipertexto?

Es sencillamente un texto que contiene referencias a otros textos distintos (enlaces o links).

### ¿Cómo funciona HTTP?

Cuando ingresas a cualquier sitio, tu computadora envía un mensaje, siguiendo las reglas de HTTP, este mensaje se transmite a través de internet y llega a otra computadora que es un servidor, su objetivo es interpretar ese HTTP, procesarlo, generar una respuesta para finalmente enviarla a través de internet siempre siguiendo las reglas de HTTP. Lo que hará la computadora, será mediante el navegador interpretar el HTTP, lo cual posiblemente será un HTML/texto.

## ¿Qué es REST?

Estilo de arquitectura de software enfocado en el intercambio de recursos y basado en HTTP.

Es un acrónimo de REPRESENTATIONAL STATE TRANSFER. Es como una pequeña capa muy fina de complejidad agregada a HTTP.

REST es un protocolo se basa en el intercambio de recursos, mientras que http se basa en el intercambio de archivos.

### ¿Qué es una API RESTful?

Es una API que está diseñada alrededor de los conceptos de REST.

* Recursos
* URI = Universal Resources Identifier
* Acción

De manera que una API REST se basa en decir cuál es el RECURSO sobre el que queremos realizar una acción, cual es la acción que queremos realizar y el URI es el que nos permite identificar exactamente cual es el recurso sobre el que vamos a actuar.

Así entonces una petición REST completa se basa en una URL, que, a diferencia de una URI, incluye el dominio, el protocolo, etc. y un Verbo HTTP (GET, PUT, DELETE, POST) que mapean a las acciones más básicas, obtener un recurso, crear un recurso nuevo, eliminar un recurso o modificar un recurso existente en el servidor.

El recurso puede ser cualquier cosa, todo depende de la lógica de la aplicación.

Ejemplos REST:

* GET /books/1 = Queremos obtener la información del libro que se identifica con el número 1.
* DELETE /books/50 = Solicitamos al servidor que elimine de su base de datos, el libro identificado con el número 50.

### ¿Cómo funciona el flujo de una petición REST?

Es bastante similar a una petición HTTP, con cierta complejidad agregada. Un cliente realiza una petición HTTP a un servidor, esta petición viaja a través de internet y llega al servidor. El servidor empieza a interpretar esta petición y a generar una respuesta y en algún momento determina que ciertos recursos no le son propios sino que debe solicitárselos a otro servidor más, entonces genera una petición HTTP que viaja a través de internet, finalmente llegando a un servidor auxiliar el cual responde con HTTP que vuelve al servidor original. Una vez que el servidor tiene los datos que vienen del servidor auxiliar, continua el procesamiento de su petición original, agregando esta nueva información, genera su respuesta HTTP y esta respuesta viaja nuevamente a través del internet hacia el cliente.

### ¿En qué casos conviene realmente utilizar REST?

Debido a que RESTful no es la única forma que se puede utilizar para el desarrollo de servicios web hay que tener muy en cuenta que existen otras. Usualmente conviene cuando las interacciones son simples, es decir, cuando lo que queremos es usar las operaciones básicas (GET, POST, DELETE, PUT) y también vienen muy bien cuando los recursos de Hardware son limitados porque como el protocolo está tan cercano al HTTP no es mucho lo que se necesita instalar para que todo esto funcione.

No conviene cuando las interacciones son un poco más complejas, si no nos es suficiente con solo intercambiar recursos, sino que necesitamos que el servidor aporte más lógica. Por ejemplo, la validación de algún documento, cuando necesitamos que el servidor aporte la lógica de validación, ahí ya RESTful se va quedando corto.

## Peticiones REST

### Consumo vía REST

Las responsabilidades que le corresponde al cliente en definitiva se trata de realizar peticiones HTTP y luego interpretar las respuestas HTTP. Lo importante de este tipo de consumo es que no necesariamente lo dispara una acción de un Humano. Por ejemplo, puede ser una tarea programada.

Ejemplo:

* Petición: <https://xkcd.com/info.0.json>
* https = Protocolo
* xkcd.com = dominio
* info.0.json = URL final de la API que se está consultando

### Verbos HTTP

GET: Buscar algún recurso y traerlo. Cuando se usa este método, la información viaja a través de la URL, es decir que toda la información es completamente visible para cualquiera.

POST: Crear recursos en el servidor. La información en este caso viaja un poquito más oculta, es decir, es un poco más segura.

PUT: Modificar un recurso que existe en el servidor. El mismo caso de POST.

DELETE: Eliminar un recurso que existe en el servidor.

### Producir Servicios REST

Esto permitirá que otras aplicaciones puedan conectarse a la mía para obtener datos que tengo en mi servidor o inclusive para incorporar nueva información. Básicamente es desarrollador un servidor capaz de interpretar peticiones http y devolver respuestas HTTP. Dado que estamos apoyándonos en la arquitectura REST, hay mucho trabajo hecho por el propio Web Server (Toda la comunicación a bajo nivel). Entonces nuestro trabajo es interpretar correctamente lo que se nos pide y generar respuestas adecuadas. Usualmente estas respuestas, o más bien toda la comunicación se realiza a través de JSON, pero hay que recordar que HTTP es un protocolo de texto por lo cual, cualquier formato de texto puede usarse, como por ejemplo XML que es usado en algunas ocasiones.

## Autenticación

### Autenticación HTTP

* Poco Segura: Las credenciales se envían en cada request.
* Ineficiente: La autenticación se realiza en cada request.

Esta inseguridad se puede arreglar mediante conexiones segura y/o herramientas parecidas, pero no es el mecanismo más seguro que existe.

Y por otro lado es bastante ineficiente porque en cada pedido toca hacer la autentificación.

### HMAC

Código de autorización basado en hash de mensajes

#### Hash

Un hash es una función que transforma un texto en algo bastante difícil de leer por un humano.

Esta autenticación se basa en tres partes, primero una función de hash que sea difícil de romper y que sea conocida por el cliente y por el servidor.

Luego hay una palabra secreta que también es compartida solamente por el cliente y el servidor.

Y por último hay datos que viajan de forma pública.

El cliente, sabiendo la función de hash y el secreto, toma esa información pública, lo cual unido todo, se forma un mensaje que será enviado al servidor, el servidor toma la información pública y también conoce cual es la función de hash y cual es el secreto y vuelve a generar el hash y luego compara el recibido con el generado por él. Si coinciden, la autenticación se produce, si no, asume que no es un usuario no habilitado.

Es bastante más seguro que HTTP. Este mecanismo se utiliza cuando la información que está del lado de la API es segura pero todavía no es tan SEGURA.

### Access Tokens

Aquí comenzamos a usar un servidor exclusivo para la autenticación y el que tendrá los recursos a los que queremos acceder.

#### Flujo

* El cliente solicita un Token temporal al servidor de Autentificación
* Si las credenciales del cliente son válidas, el servidor de Autentificación generará un Token de acceso temporal y se lo retornará al cliente.
* El cliente podrá realizar su petición al servidor de recursos concatenando el Token de acceso temporal.
* El servidor de recursos solicitará al servidor de Autenticación verificar que este Token de acceso sea válido. De ser así, el servidor de recursos retornará la respuesta realizada por el cliente de manera exitosa.

Extremadamente seguro y de lo mejor para casos específicos de información sensible.

## Manejo de Errores

### 404

La página no está disponible

### 500

El servidor no puede responder en este momento