***Notas del Curso.***

**Qué es una API (A**pplication **P**rogramming **I**nterface**)**

En español, interfaz de programación de aplicaciones, se trata de reglas que definen como interactúan dos aplicaciones y es un concepto que existe hace tiempo en el mundo de la programación. Ejemplo: servicios web, clases públicas de POO. (El conjunto de todos los miembros públicos que ofrece un clase)





# Qué es y cómo funciona el protocolo HTTP

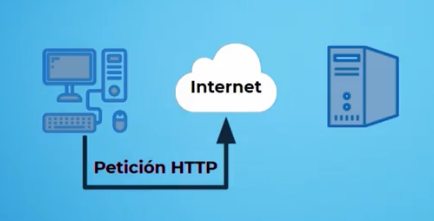
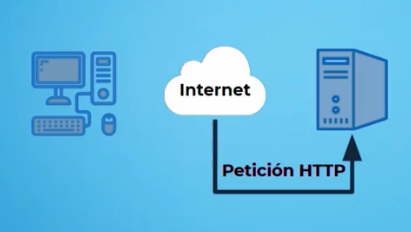
HTTP son las siglas de HyperText Transfer Protocol o protocolo de transferencia de hipertexto, es el conjunto de reglas en las que se van a comunicar dos entidades, en este caso dos computadoras.

Así como nosotros nos comunicamos en español gracias a poder mover las cuerdas vocales, producir y escuchar sonidos, las computadoras se pueden comunicar a través de HTTP gracias al modelo de TCP/IP.

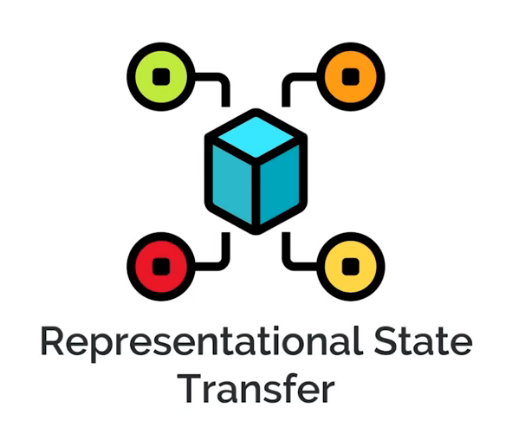
Como funciona el protocolo:  
1.El cliente envía una petición, se transmite por internet  
2.El servidor recibe la petición y genera una respuesta, la envía  
3.El cliente recibe la respuesta (y si estamos en el navegador, interpreta lo recibido )

Podemos hacer una petición directamente desde la terminal:





**Qué significa REST (Re**presentational **St**ate Transfer)

REST es un acrónimo de Representational State Transfer o transferencia de estado representacional, le agrega una capa muy delgada de complejidad y abstracción a HTTP. Mientras que HTTP es transferencia de archivos, REST se basa en la **transferencia de recursos**.

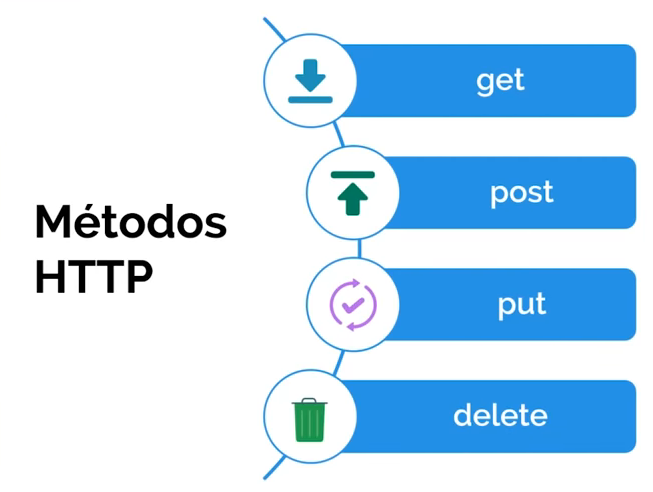
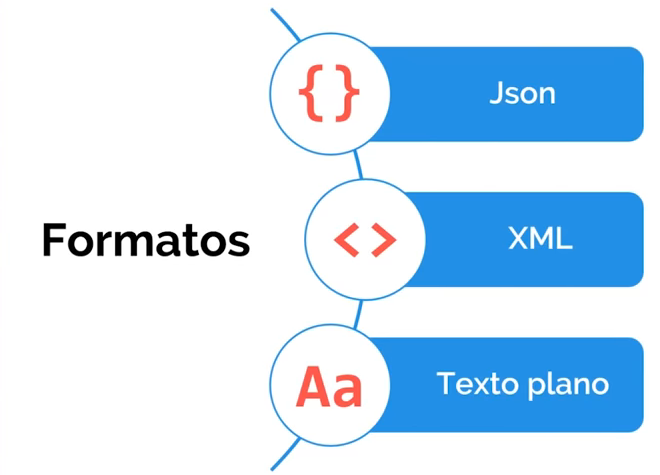
Una API RESTful es una API diseñada con los conceptos de REST:

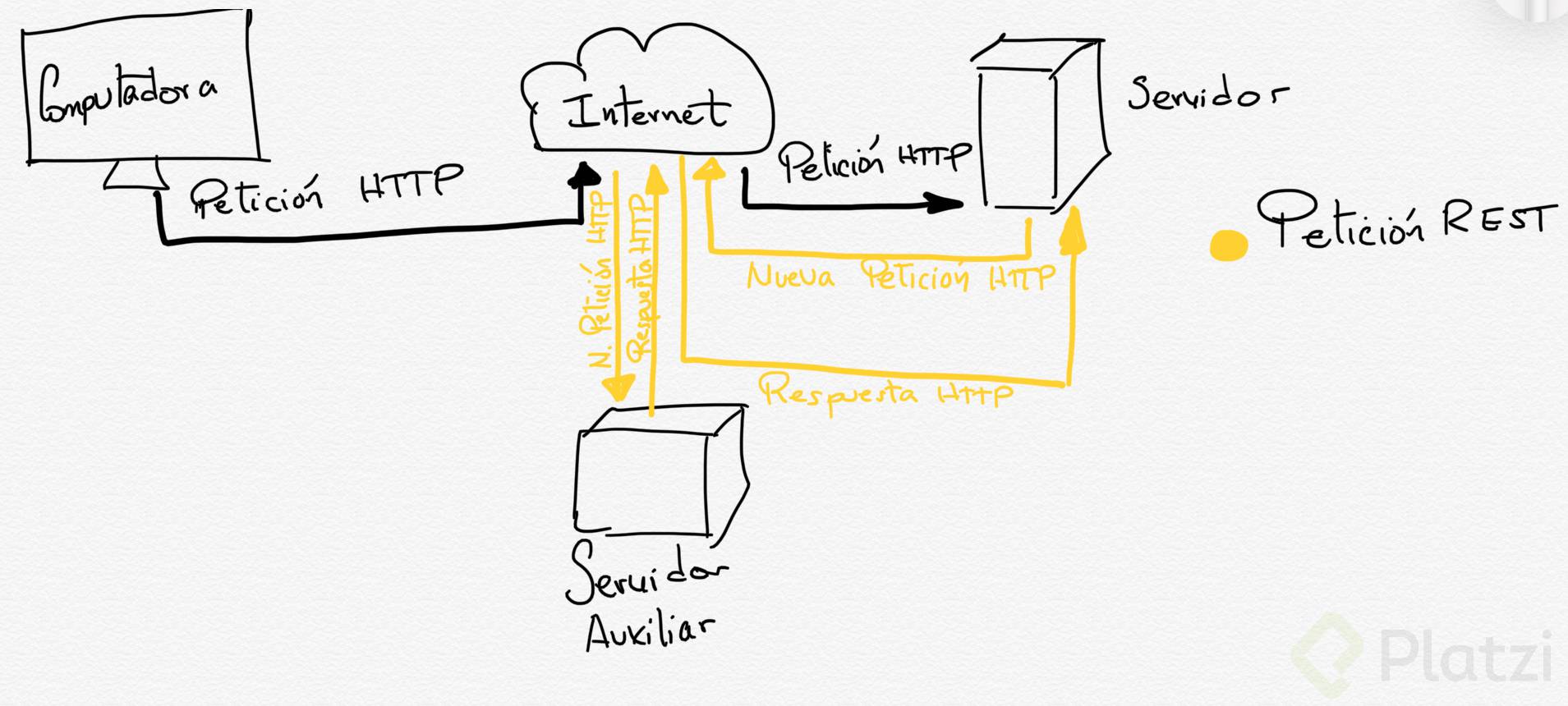
* Recurso: todo dentro de una API RESTful debe ser un recurso.
* URI: los recursos en REST siempre se manipulan a partir de la URI, identificadores universales de recursos.
* Acción: todas las peticiones a tu API RESTful deben estar asociadas a uno de los verbos de HTTP: GET para obtener un recurso, POST para escribir un recurso, PUT para modificar un recurso y DELETE para borrarlo.

REST es muy útil cuando:

* Las interacciones son simples.
* Los recursos de tu hardware son limitados.

No conviene cuando las interacciones son muy complejas.





Cuando hablamos de interacciones complejas nos referimos a operaciones *compuestas* (Operaciones que no pueden ser resueltas con un simple ***CRUD***).

Piensa en algún cálculo que involucre mucha lógica.

Siempre es posible forzar una arquitectura REST (Un ejemplo podría ser la creación de tokens de acceso) pero existen escenarios en los que es más fácil usar otras como SOAP.

Una ventaja interesante de SOAP es el uso de WSDLs (Descripciones formales de la API que pueden ser usadas para hacer más fáciles las validaciones)

***REST*** tiene unas operaciones limitadas (**GET, POST, UPDATE, ETC**) que se pueden realizar en el servidor. Ya si por lógica del negocio se requiere otro tipo de operaciones que se escapa del alcance de REST, no conviene usarlo.

Recuerda que GraphQL es el siguiente paso o, más bien, la evolución de las APIs REST con JSON.

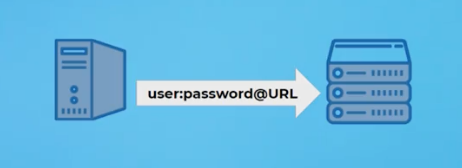
**Web Service** o [Servicio Web](https://es.wikipedia.org/wiki/Servicio_web) es un servidor que pueden intercambiar datos con un cliente por medio de **requests** en un formato que ambos entienden. Estos pueden ser XML, SOAP, WSDL, REST etc.

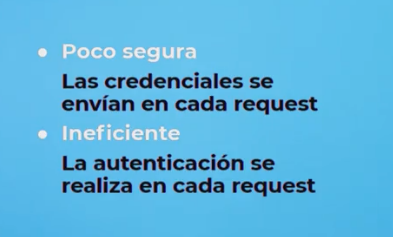
Una **API REST** es un tipo de Web Service que usa el protocolo HTTP para proporcionar una API (Application Programming Interface) que cumple con las reglas de la arquitectura [REST](https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional) usando los “verbos” POST, GET, PUT, DELETE, PATCH etc.

**Autenticación vía HTTP**

La autenticación vía HTTP tiene dos problemas:

* Es poco segura: las credenciales se envían en cada request anteponiendo el usuario y contraseña en la url, por ejemplo: user:password@platzi.com.
* Es ineficiente: la autenticación se debe realizar en cada request.



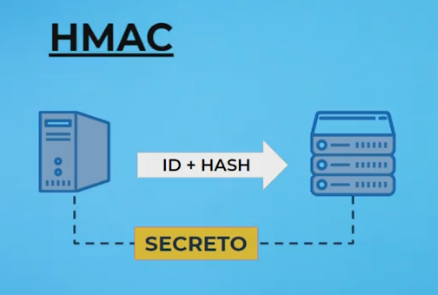


**Autenticación vía HMAC**

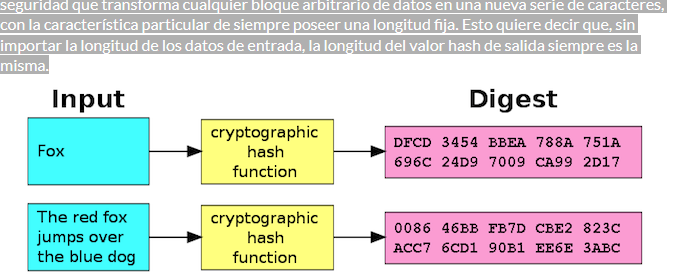
Para esta autenticación necesitamos 3 elementos:

* Función Hash: Difícil de romper, que sea conocida por el cliente y servidor.
* Clave secreta: Solamente la pueden saber el cliente y el servidor, será utilizada para corroborar el hash.
* UID: El id del usuario, será utilizado dentro de la función hash junto con la clave secreta y un timestamp.

Es mucho más segura que la autenticación vía HTTP, por ello la información que se envía a través de este método no es muy sensible.



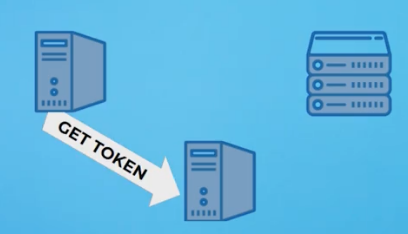
**¿Qué es un HASH?**  
Una función criptográfica hash- usualmente conocida como “hash”- es un algoritmo matemático de seguridad que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres, con la característica particular de siempre poseer una longitud fija. Esto quiere decir que, sin importar la longitud de los datos de entrada, la longitud del valor hash de salida siempre es la misma.



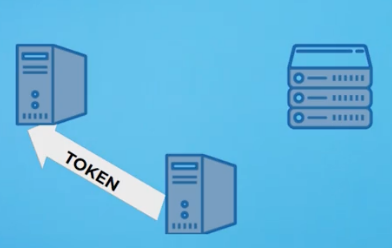
**Autenticación vía Access Tokens**

Está forma es la más compleja de todas, pero también es la forma más segura utilizada para información muy sensible. El servidor al que le van a hacer las consultas se va a partir en dos:

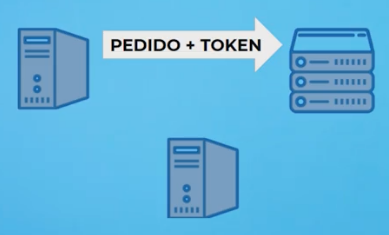
* Uno se va a encargar específicamente de la autenticación.
* El otro se va a encargar de desplegar los recursos de la API.

El flujo de la petición es la siguiente:

1. Nuestro usuario hace una petición al servidor de autenticación para pedir un token.



1. El servidor le devuelve el token.



1. El usuario hace una petición al servidor para pedir recursos de la API.



1. El servidor con los recursos hace una petición al servidor de autenticación para verificar que el token sea válido.



1. Una vez verificado el token, el servidor le devuelve los recursos al cliente.

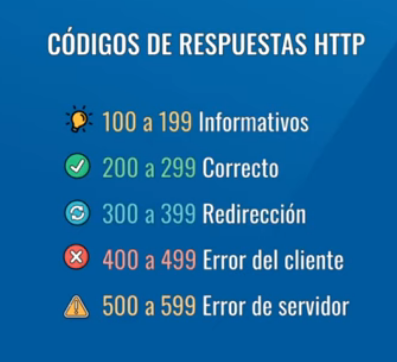
**Manejo de errores de un servicio REST**

De momento nuestra API no nos indica que haya ocurrido un error, solamente nos regresa un código 200 de HTTP que significa que la petición se realizó sin problemas.

Para mejorar nuestra API debemos añadir respuestas con los códigos HTTP más comunes:

* 400 Bad Request: indica que el servidor no puede o no procesa la petición debido a algo que es percibido como un error del cliente
* 404 Not Found: el servidor no encuentra el recurso solicitado.
* 500 Internal Server Error: la petición no pudo procesarse por un error del servidor.

Los códigos de estado en HTTP se clasifican en varios tipos:



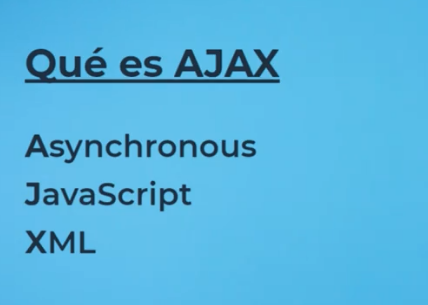
**Pagina para validar los códigos de estado.**

[**https://httpstatuses.com/**](https://httpstatuses.com/)

# Introducción a Ajax.

Es muy común tener comunicaciones con API REST al momento de tener una aplicación de una sola página o SPA (Single Page Application), ya sea para obtener o guardar datos. Esta comunicación se realiza a través de AJAX, Asynchronous JavaScript XML. la clave es la parte de asincronismo pues el cliente no se queda bloqueado.

Ajax es una tecnología y una manera de realizar peticiones asíncronas desde los navegadores web utilizando el objeto XMLHttpRequest.

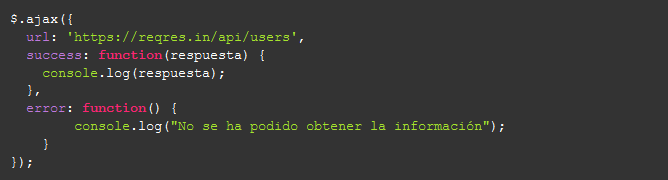
El cliente hace una petición, y recibe como respuesta HTML y JS. En ese momento, es la responsabilidad del navegador interpretar ese HTML dibujar la página y mediante JS reacciona a los estímulos (***eventos***) del sitio. Luego se dispara un nuevo pedido al servidor, este efectivamente es el que viaja vía AJAX, y lo que se supone que hay del otro lado es justamente un servidor de tipo RESTful que responderá usualmente con información JSON.



# Get vía Ajax.

# Para ello necesitaremos agregar la librería de JQuery a nuestro proyecto web. Cuando usemos el método *$.ajax()* lo que nos recibirá será un objeto expresado como JSON que contiene toda la configuración de la llamada que vamos a realizar.

# Ejemplo:



# Post vía Ajax.

# 

# 

**7 Buenas prácticas del diseño de APIs RESTful**

Felicitaciones has llegado al final del curso, ahora conoces todos los tecnicismos para generar una API RESTful. Recuerda seguir las buenas prácticas:

1. Siempre utiliza **sustantivos para nombrar tus recursos**.
2. Añade los **nombres en plural** para las urls.
3. Las modificaciones a recursos deben hacerse con su verbo HTTP correspondiente: POST, PUT o DELETE.
4. Para devolver recursos asociados a otro recurso utiliza url que incorporen subrecursos: /Autos/1/Choferes.
5. Navegabilidad vía vínculos.
6. Cuando devuelvas colecciones deben ser filtrables, ordenables y paginables.
7. Versiona tu API, añade el número de versión en la url: v1/Autos.



***Important Questions and Answers.***

