



# ESP32 ADMIN TOOL – API REST

Crear API REST, para conectar diferentes Clientes a Nuestra ESP32.





# Que es una API

- Es una manera de describir la forma en que los programas o los sitios webs intercambian datos con el Servidor.
- El formato de intercambio de datos normalmente es JSON o XML.





# ¿Para qué necesitamos una API?

- Ofrecer datos a aplicaciones que se ejecutan en un móvil.
- Ofrecer datos a otros desarrolladores con un formato más o menos estándar.
- Ofrecer datos a nuestra propia web/aplicación.
- Consumir datos de otras aplicaciones o sitios Web.





# Proveedores de APIs

Algunos ejemplos de sitios web que proveen de APIs son:

Twitter: acceso a datos de usuarios, estados, etc.

Google: por ejemplo para consumir un mapa de Google

Pero hay muchos más: Facebook, YouTube, Amazon, etc.

Pero todavía hay muchos más.....





# Qué significa API REST

- REST viene de, **RE**presentational **S**tate **T**ransfer.
- Es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.
- REST se compone de una lista de reglas que se deben cumplir en el diseño de la arquitectura de una API.
- Cuando hablamos de servicios web Restful, estos tienen que cumplir la arquitectura REST.
- Restful = adjetivo, Rest = Nombre





# Arquitectura REST

## Reglas de una arquitectura REST

- Interfaz uniforme.
- Peticiones sin estado.
- Cacheable.
- Separación de cliente y servidor.
- Sistema de Capas.
- Código bajo demanda (opcional).





# Interfaz Uniforme

- La interfaz se basa en recursos (por ejemplo el recurso Empleado (**Id**, **Nombre**, **Apellido**, **Puesto**, **Sueldo**)).
- El servidor mandará los datos (vía html, **json**, xml...) pero lo que tenga en su interior (la BD por ejemplo) para el cliente es transparente.
- La representación del recurso que le llega al cliente, será suficiente para poder editar/borrar el recurso:
  - Suponiendo que tenga permisos.
  - Por eso en el recurso solicitado se suele enviar un parámetro Id.





# Interfaz uniforme: mensajes descriptivos

Mensajes descriptivos:

- Usar las características del protocolo http para mejorar la semántica:
  - HTTP Verbs ( **POST, GET, DELETE, ...** )
  - HTTP Status Codes ( **200, 404, 500, ...** )
  - HTTP Authentication ( **Basic ...** )
- Procurar una API sencilla y jerárquica y con ciertas reglas: uso de nombres en plural.







## Peticiones sin estado

http es un protocolo sin estado ---> mayor rendimiento

GET mi\_url/empleados/1234

DELETE mi\_url/empleados/1234

*En la segunda petición hemos tenido que indicar el 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/empleados debe dar error, ¡falta el id y el servidor no lo conoce!





# Cacheable

- En la web los clientes pueden cachear 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.
- Si ahorramos peticiones, mejoraremos la escalabilidad de la aplicación y el rendimiento en el cliente (evitamos principalmente 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 documentación de 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 el es irrelevante y desconocido.
- Al cliente solo le preocupa que la API REST funcione como debe: no importa el COMO sino el QUE.
- El uso de capas o servidores intermedios puede servir para aumentar la escalabilidad (sistemas de balanceo de carga, cachés) o para implementar políticas de seguridad.





## Código bajo demanda (opcional)

- Los servidores pueden ser capaces de aumentar o definir cierta funcionalidad en el cliente transfiriéndole cierta lógica que pueda ejecutar:
  - Componentes compilados como applets de Java.
  - JavaScript en cliente.





# Consejos para elaborar una API REST

- Versiones del API.
- HTTP verbs.
- Nombre de los recursos.
- Códigos de estado.
- Formato de salida.





## Versiones del API

- Los cambios en el código no deberían afectar al API.
- Si hay cambios en el API se deben usar versiones para no frustrar a los desarrolladores.
- La mejor opción es añadir un prefijo a las URLs:

GET /v1/iot HTTP/1.1  
Host: api.iothost.org

GET /v2/iot HTTP/1.1  
Host: api.iothost.org





# HTTP verbs

- Si realizamos CRUD, debemos utilizar los HTTP verbs de forma adecuada para cuidar la semántica.
  - GET: Obtener datos. Ej: **GET** /v1/devices/1234
  - PUT: Actualizar datos. Ej: **PUT** /v1/devices/1234
  - POST: Crear un nuevo recurso. Ej: **POST** /v1/devices
  - DELETE: Borrar el recurso. Ej: **DELETE** /v1/devices/1234







## Nombre de los recursos

- Plural mejor que singular, para lograr uniformidad:  
Obtenemos un listado de dispositivos: GET /v1/devices  
Obtenemos un dispositivo en particular: GET /v1/devices/1234
- Url's lo más cortas posibles.
- Evita guiones y guiones bajos.
- Deben ser semánticas para el cliente.
- Utiliza nombres y no verbos.
- Estructura jerárquica para indicar la estructura: /v1/devices/1234/status/203





# Códigos de estado

- Se utilizan los códigos de estado de http.
- Si realizamos un request de POST deberemos devolver un 201.
- Se pueden producir múltiples errores en la llamada al API:
  - Falta de permisos.
  - Errores de validación.
  - O incluso un error interno de servidor.
- Siempre se debe devolver un código de estado HTTP con los requests.
- Añadir un mensaje de error si es necesario.

<https://www.webfx.com/web-development/glossary/http-status-codes/>





## Formato de salida

- En función de la petición nuestra API podría devolver uno u otro formato.
- Nos fijaremos en el ACCEPT HEADER.
- En principio utilizaremos JSON: sencillo y simple.
- XML no es nuestro amigo: schemas, namespaces...
- Si no es un requerimiento, evitaremos XML.

Ref. <https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html>



# ESP32 API REST

Obtenemos los valores para mostrar en la página del Índice.

URL	{URL}/api/index
Method ( http verb )	GET
Returns Code	200 OK
	401 Unauthorized
	404 Not Found

Nota: { URL } es la IP del Dispositivo, Ejemplo: <http://192.168.30.108>

# ESP32 API REST

Obtenemos los valores almacenados de la configuración WIFI.

URL	{URL}/api/connection/wifi
Method ( http verb )	GET
Returns Code	200 OK
	401 Unauthorized
	404 Not Found

Nota: { URL } es la IP del Dispositivo, Ejemplo: <http://192.168.30.108>

# ESP32 API REST

## Actualiza la configuración WIFI.

URL	{URL}/api/connection/wifi
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108

# ESP32 API REST

Obtenemos una lista de todas las redes WIFI cercanas.

URL	{URL}/api/connection/wifi-scan
Method ( http verb )	GET
Returns Code	200 OK
	401 Unauthorized
	404 Not Found

Nota: { URL } es la IP del Dispositivo, Ejemplo: <http://192.168.30.108>

# ESP32 API REST

Obtenemos los valores almacenados de la configuración MQTT.

URL	{URL}/api/connection/mqtt
Method ( http verb )	GET
Returns Code	200 OK
	401 Unauthorized
	404 Not Found

Nota: { URL } es la IP del Dispositivo, Ejemplo: <http://192.168.30.108>



# ESP32 API REST

Actualiza la configuración de la conexión MQTT.

URL	{URL}/api/connection/mqtt
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108

# ESP32 API REST

## Descargar la configuración del archivo settings.json

URL	{URL}/api/device/download
Method ( http verb )	GET
Returns Code	200 OK
	401 Unauthorized
	404 Not Found

Nota: { URL } es la IP del Dispositivo, Ejemplo: <http://192.168.30.108>

# ESP32 API REST

## Actualiza la configuración del archivo settings.json

URL	{URL}/api/device/upload
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108

# ESP32 API REST

## Actualiza el firmware del dispositivo

URL	{URL}/api/device/firmware
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108

# ESP32 API REST

Obtenemos los valores para mostrar los estados en el Header.

URL	{URL}/api/device/status
Method ( http verb )	GET
Returns Code	200 OK
	401 Unauthorized
	404 Not Found

Nota: { URL } es la IP del Dispositivo, Ejemplo: <http://192.168.30.108>

# ESP32 API REST

## Reiniciar el dispositivo desde la API

URL	{URL}/api/device/restart
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108

# ESP32 API REST

## Restaurar el dispositivo desde la API

URL	{URL}/api/device/restore
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108

# ESP32 API REST

## Actualizar la contraseña del servidor web.

URL	{URL}/api/perfil/user
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108



# ESP32 API REST

Configurar los estados de los Relays On/Off.

URL	{URL}/api/device/relays
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108

# ESP32 API REST

Configurar el Dimmer de 0 a 100%.

URL	{URL}/api/device/dimmer
Method ( http verb )	POST
Returns Code	200 OK
	401 Unauthorized, 400 <u>Bad Request</u>
	404 Not Found, 500 Internal Server Error

Nota: { URL } es la IP del Dispositivo, Ejemplo: http://192.168.30.108