



PYTHON

Working with RESTful APIs

Working with RESTful APIs

WORKING WITH RESTFUL APIS

CONCEPTOS BÁSICOS DE LA PROGRAMACIÓN EN RED

WORKING WITH RESTFUL APIS

- **REST:**

- Es una arquitectura/API de servicios web.
- Se basa en el protocolo **HTTP**.
- **No guarda el estado**. Cada comunicación es independiente.
- Representa los datos de intercambio como **texto plano**.
Generalmente JSON o XML.
- **Acrónimo de**
 - **RE**presentational.
 - **State**.
 - **Transfer**.

WORKING WITH RESTFUL APIS

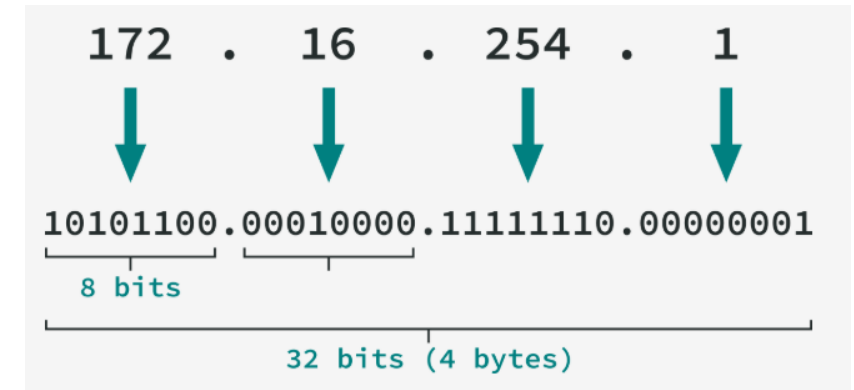
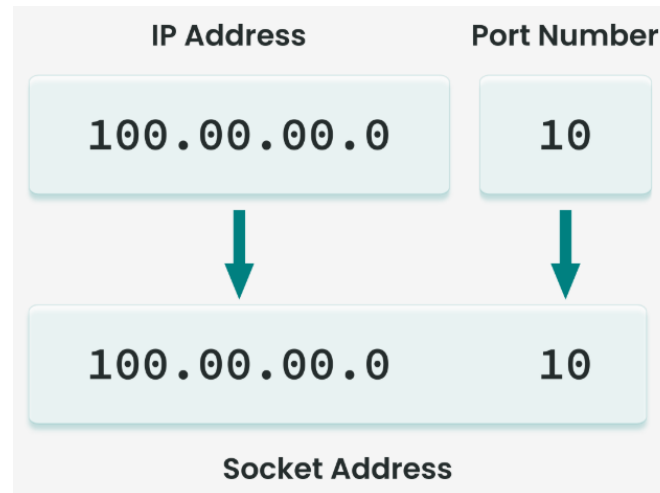
- Sockets BSD:

- Un socket es un *end-point* desde el que se pueden obtener o transmitir datos a través de una red.
 - Nacen en 1983 en la Universidad de California en Berkeley, donde se diseñó y se llevó a cabo la primera implementación.
 - Se adoptaron por POSIX (estándar de Unix) como *POSIX Sockets*.
 - La mayoría de los sistemas operativos modernos implementan Sockets BSD (Windows los reimplementa como WinSock, pero son compatibles).
-
- Nota. BSD → *Berkeley Software Distribution* → También conocido como *Berkeley Unix* o *BSD Unix* fue una distribución de Unix creada por el *Computer Systems Research Group (CSRG)* de la Univesidad de Berkeley en 1978.

WORKING WITH RESTFUL APIS

■ Sockets BSD

- Inicialmente los sockets BSD se diseñaron para realizar comunicaciones en dos dominios distintos:
 - **Unix domain (Unix).** Comunicar programas en el mismo ordenador.
 - **Internet domain (INET).** Comunicar programas en distintos ordenadores.
- Dirección del socket.
 - Los sockets del dominio INET necesita:
 - Dirección IP
 - Puerto
- Direcciones IP:
 - V4 – 32 bits.
 - V6 – 128 bits.



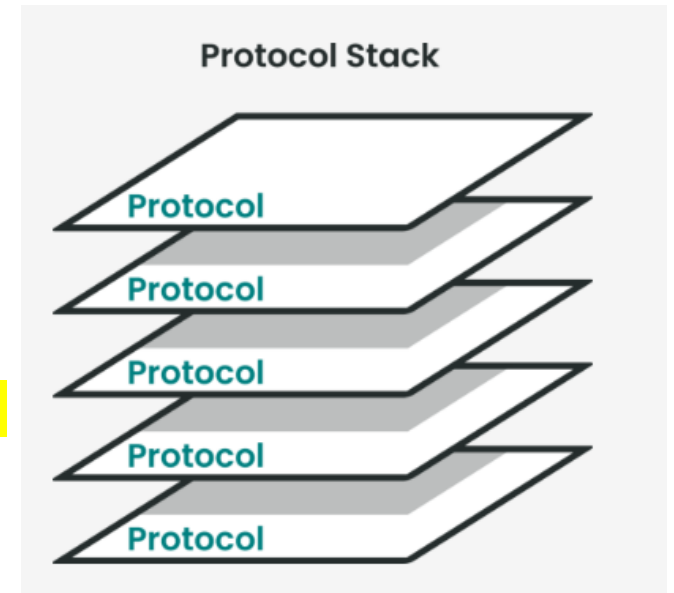
WORKING WITH RESTFUL APIS

■ Sockets BSD

- Número del socket/servicio: número entero de 16 bits que identifica un socket en un sistema.
- Protocolo (en comunicaciones de red): conjunto estandarizado de reglas que permite la comunicación entre procesos.

■ Pila de protocolos (Protocol Stack).

- Es un conjunto multicapa de protocolos cooperativos que proporcionan un repertorio unificado de servicios. La pila TCP/IP está diseñada para trabajar en redes basadas en el protocolo IP.
- En el modelo conceptual de los servicios de red los servicios más elementales y básicos están en la parte baja de la pila y los más avanzados en la parte alta.



WORKING WITH RESTFUL APIS

- Protocolo IP
 - El protocolo **IP (*Internet Protocol*)** se encuentra en parte baja de la pila TCP/IP, en la capa de RED.
 - Se encarga del envío paquetes de datos (datagramas) entre dos nodos.
 - **No es fiable:**
 - No garantiza la entrega de todos los paquetes.
 - No garantiza la integridad de los paquetes.
 - No garantiza el orden de entrega de los paquetes.

WORKING WITH RESTFUL APIS

- Protocolo TCP
 - TCP (*Transmission Control Protocol*) es la parte superior de la pila de protocolos TCP/IP (capa de transporte). Utiliza ¿datagramas? (proporcionados por las capas inferiores, encapsulados en segmentos) y protocolos de enlace (un proceso automatizado de sincronización del flujo de datos) para construir un canal de comunicación fiable capaz de transmitir y recibir caracteres individuales.
 - Garantiza:
 - Entrega de paquetes.
 - Integridad de paquetes.
 - Orden de entrega de los paquetes.

WORKING WITH RESTFUL APIS

- Protocolo UDP
 - **UDP (*User Datagram Protocol*)** es la parte superior de la pila de protocolos TCP/IP (capa de transporte) pero en una posición inferior a la de TCP.
 - **No utiliza protocolos de enlace**, lo que tiene dos consecuencias graves:
 - **Es más rápido que TCP** (debido a una menor sobrecarga)
 - **Es menos fiable que TCP.**
 - TCP es el protocolo de primera elección para aplicaciones donde la seguridad de los datos es más importante que la eficiencia (p. ej., WWW, REST, transferencia de correo, etc.).
 - UDP es más adecuado para aplicaciones donde el tiempo de respuesta es crucial (DNS, DHCP, etc.).

WORKING WITH RESTFUL APIS

- **Comunicaciones orientadas a conexión vs sin conexión**
 - En una comunicación orientada a conexión:
 - Se deben realizar un paso previo de establecimiento de la conexión y otro posterior de finalización.
 - Las dos partes de la comunicación son asimétricas. Cada elemento de la comunicación tiene un rol.
 - Cada parte sabe si la otra está conectada.
 - En una comunicación NO orientada a conexión:
 - Cualquiera pueda iniciar la comunicación.
 - Nadie sabe el estado de los otros elementos de la comunicación.
- **TCP es orientado a conexión. UDP no.**
- En TCP/IP se denomina **cliente** al elemento iniciador de la comunicación y **servidor** al elemento llamado.

WORKING WITH RESTFUL APIS

SOCKETS EN PYTHON

WORKING WITH RESTFUL APIS

- Pasos para crear establecer una comunicación con sockets TCP.
 - **Crear un nuevo socket** capaz de gestionar transmisiones orientadas a conexión basadas en TCP. **CLASE socket.**
 - **Conectar el socket** al servidor de una dirección dada. **MÉTODO connect**
 - **Enviar una solicitud** al servidor (el servidor desea saber qué queremos). **MÉTODO send.**
 - **Recibir la respuesta** del servidor. **MÉTODO recv.**
 - **Cerrar el socket** (finalizar la conexión). **MÉTODOS shutdown** (notificación de fin de la comunicación) **y close** (cierre).

WORKING WITH RESTFUL APIS

■ Ejemplo.

- Establecer un socket con un servidor web y realizar una petición.
- Levantar un servidor local ejecutando: **python -m http.server**
- La carpeta desde la que se arranque el servidor será la carpeta root del servidor. Ubicar un archivo index.html y probar desde un navegador accediendo a **http://localhost:8000**

El puerto usa 16 bits

```
import socket
```

```
server_addr = input("¿A qué servidor te quieres conectar?")  
server_port = int(input("¿A qué puerto?"))
```

```
# 1. Creación del socket  
# socket.AF_INET -> Dirección compuesta por IP + PORT  
# socket.SOCK_STREAM -> TCP (socket.SOCK_DGRAM -> UDP)  
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

WORKING WITH RESTFUL APIS

■ Ejemplo.

Atención a la sintaxis: el argumento es una tupla

```
# 2. Conectar el socket
sock.connect((server_addr, server_port))
# sock.connect((server_addr, 80)) # Servidor web http
# sock.connect((server_addr, 443)) # Servidor web https

# 3. Enviar la solicitud (hay que transformar a bytes la cadena)
sock.send(b"GET / HTTP/1.1\r\nHost: " +
          bytes(server_addr, "utf8") +
          b"\r\nConnection: close\r\n\r\n")

# 4. Recibir la respuesta
reply = sock.recv(10000) # 10000 es el número de byte máximo

# 5. Cerrar el socket
# socket.SHUT_RD - Notifica que deja de leer
# socket.SHUT_WR - Notifica que deja de escribir
# socket.SHUT_RDWR - Notifica que deja de leer y escribir
sock.shutdown(socket.SHUT_RDWR)
sock.close()

# Tratar los datos
print(repr(reply))
```

WORKING WITH RESTFUL APIS

- Excepción **socket.gaierror**.

- Normalmente es lanzada por la función del kernel del sistema **getaddrinfo()** cuando se ejecuta el método **connect()**, no por Python.
- Se puede producir por:
 - Dirección inexistente.
 - Dirección mal formada.

- Exception **ConnectionRefusedError**.

- Se produce cuando el servidor rechaza la petición (por ejemplo, por una llamada a un puerto que no está habilitado)

- Exception **socket.timeout**.

- Se ha excedido el tiempo de espera. Este tiempo se puede modificar con el método **settimeout** de **socket**.

WORKING WITH RESTFUL APIS

- Ejemplo Servidor:

```
import socket

host = '127.0.0.1' #Dirección IP del servidor (localhost)
puerto = 8021 #Puerto

socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_servidor.bind((host, puerto))
socket_servidor.listen()
print(f"Servidor arrancado. Host: {host}. IP: {puerto}")
socket_cliente, direccion_cliente = socket_servidor.accept()
print(f"Se ha recibido una petición desde {direccion_cliente}")
data = socket_cliente.recv(1024)
print(f"Datos recibidos del cliente: {data.decode('utf-8')}")
mensaje = "Hola, cliente. Todo bien por aquí."
socket_cliente.sendall(mensaje.encode('utf-8'))
print(f"Datos enviados al cliente: {mensaje}")
socket_cliente.close()
socket_servidor.close()
```


WORKING WITH RESTFUL APIS

- Ejemplo Cliente:

```
import socket

host = '127.0.0.1' #Dirección IP del servidor (localhost)
puerto = 8021 #Puerto

socket_cliente = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
socket_cliente.connect((host, puerto))
print(f"Conectado al servidor en {host}:{puerto}")
mensaje = "Hola, servidor. ¿Cómo estás?"
socket_cliente.sendall(mensaje.encode('utf-8'))
print(f"Datos enviados al servidor: {mensaje}")
datos = socket_cliente.recv(1024)
print(f"Datos recibidos del servidor: {datos.decode('utf-8')}")
socket_cliente.close()
```

WORKING WITH RESTFUL APIS

INTRODUCCIÓN A JSON

WORKING WITH RESTFUL APIS

- JSON → JavaScript Object Notation

- Lenguaje de marcas para la representación de estructuras de datos.
- Uso universal e independiente de la plataforma.
- Utiliza UTF-8.
- Sintaxis muy sencilla.
- Más información en: <https://ecma-international.org/wp-content/uploads/ECMA-404.pdf>

```
{
  "películas": [
    {
      "titulo": "Blade Runner 2049",
      "director": "Denis Villeneuve",
      "año": 2017,
      "duracion_min": 164,
      "pais": "Estados Unidos",
      "genero": ["Ciencia ficción", "Neo-noir"]
    },
    {
      "titulo": "Interstellar",
      "director": "Christopher Nolan",
      "año": 2014,
```

WORKING WITH RESTFUL APIS

- JSON → JavaScript Object Notation
 - Los **arrays** se delimitan con **corchetes**.
 - Los **objetos** se delimitan con **llaves**.
 - Las propiedades son pares **clave : valor**.
 - Las propiedades **se separan entre sí por comas**.
 - **Una propiedad puede contener un array o un objeto.**
 - Las **claves** se entrecomillan con **comillas dobles**.
 - Los nombres de las claves **no tienen restricciones**.
 - Las **claves se separan de sus valores por dos puntos**.

WORKING WITH RESTFUL APIS

- JSON → JavaScript Object Notation

- Las cadenas de caracteres se delimitan con **comillas dobles**.
- Las cadenas **no se pueden partir**, deben estar en una única línea.
- Reconoce **números enteros y decimales** (usando punto). No utilizar ceros a la izquierda los números.
- **No reconoce número en bases que no sean la decimal**. No reconoce:
 - 0x10
 - 0o10
 - 0b10
- Representa los números reales igual que Python:
 - 3.141592653589
 - 3.0857E16
 - -1.6021766208E-19
- Valores booleanos válidos son **true y false**.
- Valores nulos se representan con **null**.

WORKING WITH RESTFUL APIS

- JSON → JavaScript Object Notation
 - Admite caracteres de escape (además de las comillas dobles \").

digraph	effective meaning
\\	\
\/	/
\b	backspace
\f	form feed
\n	line feed
\r	carriage return
\t	tabulation
\uxxxx \Uxxxx	UNICODE codepoint

WORKING WITH RESTFUL APIS

- Módulo **json**

- Función **dumps** → Convierte un objeto a formato json (no siempre, por ejemplo, una variable de tipo básico).

```
titulo = "Nunca me dijo \"jamás\""
json.dumps(titulo)
'"Nunca me dijo \\'jam\\u00e1s\\'"'
```

Atención a la sintaxis de las comillas internas

- Función **dump** -> Convierte un objeto a formato json y lo almacena en un fichero (no siempre).

WORKING WITH RESTFUL APIS

- Módulo **json**
 - Mapeo de tipos Python a JSON (los objetos de clases propias no son directamente convertibles)

Python data	JSON element
dict	object
list or tuple	array
string	string
int or float	number
True/False	true/false
None	null

WORKING WITH RESTFUL APIS

■ Módulo **json**

■ Mapeo de objetos a JSON.

- Un intento de convertir un objeto de una clase propia produce un **TypeError**.

■ Soluciones:

- **0.** Usar `__dict__` (muy básico)
- **1.** Escribir la propia función equivalente a **`dumps`** y utilizarla como alternativa a través del atributo **`default`**, sobre escribiendo el método por defecto (debe lanzar `TypeError`):

```
import json

class Who:
    def __init__(self, name, age):
        self.name = name
        self.age = age

def encode_who(w):
    if isinstance(w, Who):
        return w.__dict__
    else:
        raise TypeError(w.__class__.__name__ + ' is not JSON serializable')

some_man = Who('John Doe', 42)
print(json.dumps(some_man, default=encode_who))
```

WORKING WITH RESTFUL APIS

- Módulo **json**

- Mapeo de objetos a JSON.

- Soluciones:

- **2.** Escribir una versión específica de **JSONEncoder** que sobrescriba el método **default** y pasarle la referencia a la clase en la función **dumps** (no debe lanzar **TypeError**, ya que lo lanza la clase base):

```
import json

class Who:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class MyEncoder(json.JSONEncoder):
    def default(self, w):
        if isinstance(w, Who):
            return w.__dict__
        else:
            return super().default(self)

some_man = Who('John Doe', 42)
print(json.dumps(some_man, cls=MyEncoder))
```

En el curso de Edube hay una errata en esta llamada (2025-03)

WORKING WITH RESTFUL APIS

■ Módulo **json**

■ Conversión de json a tipos Python.

- Función **loads**. Convierte una cadena json a datos Python.
- Función **load**. Lee un fichero con json y convierte el contenido a datos Python.

```
import json
```

```
entrada = '3.1416'  
dato = json.loads(entrada)  
print(type(dato)) # <class 'float'>  
print(dato)
```

```
entrada = '24'  
dato = json.loads(entrada)  
print(type(dato)) # <class 'int'>  
print(dato)
```

```
entrada = 'true'  
dato = json.loads(entrada)  
print(type(dato)) # <class 'bool'>  
print(dato)
```

```
entrada = '"Lo que sea"'  
dato = json.loads(entrada)  
print(type(dato)) # <class 'str'>  
print(dato)
```

```
entrada = '[1, 2, 3]'  
dato = json.loads(entrada)  
print(type(dato)) # <class 'list'>  
print(dato)
```

```
entrada = '{"nombre": "Python", "tipo": "Multiparadigma"}'  
dato = json.loads(entrada)  
print(type(dato)) # <class 'dict'>  
print(dato)
```

WORKING WITH RESTFUL APIS

■ Módulo **json**

- Conversión de json a objetos propios. El atributo **object_hook**
 - Permite la instancia de un objeto almacenado como json (diccionario).
 - Las claves deben coincidir con los parámetros del `__init__`.

ALTERNATIVA 1
Basada en
funciones

```
import json

class Who:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def saludar(self):
        print(f'Hola, soy {self.name}')

def encode_who(w):
    if isinstance(w, Who):
        return w.__dict__
    else:
        raise TypeError(w.__class__.__name__ + 'is not JSON serializable')

def decode_who(w):
    return Who(w['name'], w['age'])

old_man = Who("Jane Doe", 23)
json_str = json.dumps(old_man, default=encode_who)
new_man = json.loads(json_str, object_hook=decode_who)
print(type(new_man))
print(new_man.__dict__)
new_man.saludar()
```

WORKING WITH RESTFUL APIS

■ Módulo **json**

- Conversión de json a objetos propios. El atributo **object_hook**
 - Permite la instancia de un objeto almacenado como json (diccionario).
 - Las claves deben coincidir con los parámetros del `__init__`.

ALTERNATIVA 2
(basada en clases)

```
import json
```

```
class Who:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

```
class MyEncoder(json.JSONEncoder):  
    def default(self, w):  
        if isinstance(w, Who):  
            return w.__dict__  
        else:  
            return super().default(self, z)
```

```
class MyDecoder(json.JSONDecoder):  
    def __init__(self):  
        json.JSONDecoder.__init__(self, object_hook=self.decode_who)
```

```
    def decode_who(self, d):  
        return Who(**d)
```

```
some_man = Who('Jane Doe', 23)  
json_str = json.dumps(some_man, cls=MyEncoder)  
new_man = json.loads(json_str, cls=MyDecoder)
```

```
print(type(new_man))  
print(new_man.__dict__)
```

WORKING WITH RESTFUL APIS

INTRODUCCIÓN A XML

WORKING WITH RESTFUL APIS

■ XML

- eXtensible Markup Language
- Sintaxis:
 - Declaración (primera línea):
 - `<?xml version = "1.0" encoding = "utf-8"?>`
 - Comentarios:
 - `<!-- Texto del comentario -->`
 - Asociación de un DTD (opcional):
 - `<!DOCTYPE nombre_nodo_raiz SYSTEM "nombre_fichero.dtd">`
 - Nombre: nombre_nodo
 - Especificador de tipo: SYSTEM o PUBLIC
 - URI: identificador del recurso (el fichero dtd).
 - Aperturas y cierres de elementos:
 - `<etiqueta></etiqueta>`
 - `<etiqueta/>`

WORKING WITH RESTFUL APIS

■ XML. Ejemplo.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- cars.xml - List of cars ready to sell -->
<!DOCTYPE cars_for_sale SYSTEM "cars.dtd">
<cars_for_sale>
  <car>
    <id>1</id>
    <brand>Ford</brand>
    <model>Mustang</model>
    <production_year>1972</production_year>
    <price currency="USD">35900</price>
  </car>
  <car>
    <id>2</id>
    <brand>Aston Martin</brand>
    <model>Rapide</model>
    <production_year>2010</production_year>
    <price currency="GBP">32000</price>
  </car>
</cars_for_sale>
```


WORKING WITH RESTFUL APIS

- **XML. Paquete xml**
 - Contiene 4 subpaquetes:
 - dom. Document Object Model.
 - parsers. Wrappers para XML.
 - sax. Simple Api for XML
 - etree. La librería de XML ElementTree

WORKING WITH RESTFUL APIS

- **XML. Paquete xml**
 - Contiene 4 subpaquetes:
 - dom. Document Object Model.
 - parsers. Wrappers para XML.
 - sax. Simple Api for XML
 - etree. La librería de XML ElementTree

WORKING WITH RESTFUL APIS

- **XML. Paquete xml.etree. Procesado de documento.**

```
import xml.etree.ElementTree
```

```
cars_for_sale =  
xml.etree.ElementTree.parse('cars.xml').getroot()  
print(cars_for_sale.tag)  
for car in cars_for_sale.findall('car'):  
    print('\t', car.tag)  
    for prop in car:  
        if prop.tag == 'price':  
            print('\t\t', prop.tag, end='')  
            print(prop.attrib, end='')  
            print(' =', prop.text)  
        else:  
            print('\t\t', prop.tag, end='')  
            print(' =', prop.text)
```

El ejemplo equivalente del curso
de EDUBE es erróneo

```
cars_for_sale  
car  
    id = 1  
    brand = Ford  
    model = Mustang  
    production_year = 1972  
    price{'currency': 'USD'} = 35900  
car  
    id = 2  
    brand = Aston Martin  
    model = Rapide  
    production_year = 2010  
    price{'currency': 'GBP'} = 32000
```

WORKING WITH RESTFUL APIS

- **XML. Paquete xml.etree. Modificación de documento.**

```
import xml.etree.ElementTree

tree = xml.etree.ElementTree.parse('cars.xml')
cars_for_sale = tree.getroot()
for car in cars_for_sale.findall('car'):
    if car.find('brand').text == 'Ford' and car.find('model').text == 'Mustang':
        cars_for_sale.remove(car)
        break
new_car = xml.etree.ElementTree.Element('car')
xml.etree.ElementTree.SubElement(new_car, 'id').text = '4'
xml.etree.ElementTree.SubElement(new_car, 'brand').text = 'Maserati'
xml.etree.ElementTree.SubElement(new_car, 'model').text = 'Mexico'
xml.etree.ElementTree.SubElement(new_car, 'production_year').text = '1970'
xml.etree.ElementTree.SubElement(new_car, 'price', {'currency': 'EUR'}).text = '61800'
cars_for_sale.append(new_car)
tree.write('newcars.xml', method='')
```

WORKING WITH RESTFUL APIS

EL MÓDULO REQUEST

WORKING WITH RESTFUL APIS

- Método principales HTTP:
 - El método **GET**
 - Obtener un recurso del servidor.
 - El método **POST**
 - Enviar un nuevo recurso al servidor.
 - El método **PUT**
 - Enviar un recurso existente al servidor para actualizarlo completamente.
 - El método **DELETE**
 - Eliminar un nuevo recurso al servidor.
 - El método **PATCH**
 - Enviar un recurso existente al servidor para actualizarlo parcialmente.

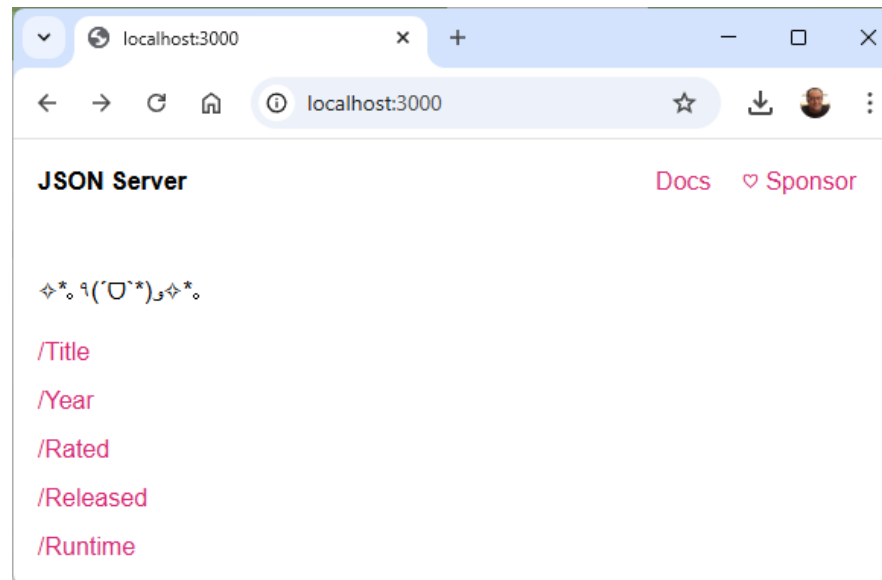
WORKING WITH RESTFUL APIS

- Otros métodos de HTTP:
 - El método **HEAD**
 - Igual que **GET** pero sólo devuelve las cabeceras, no el cuerpo.
 - El método **CONNECT**
 - Establece una conexión con el servidor.
 - El método **OPTIONS**
 - Devuelve los métodos HTTP permitidos por el servidor.
 - El método **TRACE**
 - Devuelve los datos recibidos por el servidor en la petición (útil para depurar).

WORKING WITH RESTFUL APIS

- Para hacer pruebas:
 - Instalar node
 - Instalar **json-server**:
 - `npm install -g json-server`
 - Arrancar json-server indicando el fichero json a proporcionar:
 - `json-server --watch pelicula.json`
 - Al acceder desde el navegador se obtendrá algo similar a la siguiente captura:

El comportamiento del servidor cambia si se crea una carpeta **public** con un fichero **index.html**



WORKING WITH RESTFUL APIS

- El módulo **requests**
 - El método **GET**
 - Primer ejemplo: petición **get** y obtención del **status_code**:

```
import requests
```

```
reply = requests.get('http://localhost:3000')  
print(reply.status_code)
```

WORKING WITH RESTFUL APIS

- El módulo **requests**
 - La evaluación del **status_code** se puede realizar a través de **request.codes**, que admite varias alternativas (ver el código de **codes**):
 - **requests.codes.OK** → 200
 - **requests.codes.ok** → 200
 - **requests.codes.okay** → 200
 - La respuesta es un objeto de la clase **Response**. Se divide en dos partes:
 - Encabezado (**respuesta.header**), un diccionario.
 - Contenido (**respuesta.text**), un texto plano

WORKING WITH RESTFUL APIS

- El módulo **requests**. Excepciones.
 - Cuando la URL está mal formada, se lanza una excepción **requests.exceptions.InvalidURL**
 - Cuando se produce un error de conexión (no hay acceso a la red, el servidor está caído) se lanza una excepción **requests.exceptions.ConnectionError**.
 - Cuando se hace una petición, se puede especificar un tiempo de espera máximo a través del parámetro **timeout**, expresado en **segundos**. Si se supera, se lanza una excepción **requests.exceptions.Timeout**.

```
import requests

try:
    reply = requests.get('http://localhost:3000', timeout=1)
except requests.exceptions.Timeout:
    print('Se me ha terminado la paciencia')
else:
    print('Petición finalizada')
```

WORKING WITH RESTFUL APIS

- El módulo **requests**. Excepciones.

```
RequestException
|__ HTTPError
|__ ConnectionError
|   |__ ProxyError
|   |__ SSLError
|__ Timeout
|   |__ ConnectTimeout
|   |__ ReadTimeout
|__ URLRequired
|__ TooManyRedirects
|__ MissingSchema
|__ InvalidSchema
|__ InvalidURL
|   |__ InvalidProxyURL
|__ InvalidHeader
|__ ChunkedEncodingError
|__ ContentDecodingError
|__ StreamConsumedError
|__ RetryError
|__ UnrewindableBodyError
```

WORKING WITH RESTFUL APIS

CONSTRUCCIÓN DE UN CLIENTE REST SIMPLE

WORKING WITH RESTFUL APIS

- Utilizando el fichero **`cars.json`**, arrancar un servidor web con:
 - `json-server --watch cars.json`
- Petición básica:

```
# Utilizando el fichero cars.json, arrancar un servidor web con:  
# json-server --watch cars.json
```

```
import requests  
  
try:  
    reply = requests.get("http://localhost:3000/cars")  
except requests.RequestException:  
    print("Communication error")  
else:  
    if reply.status_code == requests.codes.ok:  
        print(reply.text)  
    else:  
        print("Server error")
```

WORKING WITH RESTFUL APIS

- Recibiendo json:
 - Atributo **Content-type** de **headers** nos indica el tipo de contenido recibido.
 - Método **json()** de **Response** proporciona un objeto con los datos recibidos en un documento **json**.

```
import requests

try:
    reply = requests.get("http://localhost:3000/cars")
except:
    print("Communication error")
else:
    if reply.status_code == requests.codes.ok:
        print(reply.headers['Content-Type']) # application/json
        print(reply.json()) # [{'id': '2', 'brand': 'Chevrolet', 'model':
'Camaro'...
    else:
        print("Server error")
```

WORKING WITH RESTFUL APIS

■ Estado de la conexión:

- El atributo **Connection** tiene información sobre el estado de la conexión.
- La conexión en HTTP1.1 se mantiene abierta durante un tiempo (la cierra el servidor). El estado es **keep-alive** por defecto.
- En HTTP1.0, el estado por defecto es **close**, que indica que la conexión se cierra inmediatamente después de transmitirse el resultado de la petición.

```
try:
    reply = requests.get('http://localhost:3000/cars')
except requests.RequestException:
    print('Communication error')
else:
    print('Connection=' + reply.headers['Connection']) # Connection=keep-alive
    if reply.status_code == requests.codes.ok:
        show(reply.json())
    elif reply.status_code == requests.codes.not_found:
        print("Resource not found")
    else:
        print('Server error')
```


WORKING WITH RESTFUL APIS

- Estado de la conexión:

- Se puede indicar cómo se quiere que sea el estado de la conexión creando una cabecera y asignándola en la petición:

```
headers = {  
    'Connection': 'close'  
}  
  
try:  
    reply = requests.get('http://localhost:3000/cars',  
headers=headers)  
except requests.RequestException:  
    print('Communication error')  
else:  
    print('Connection=' + reply.headers['Connection']) #  
Connection=close
```

WORKING WITH RESTFUL APIS

■ CRUD: Read all

```
import requests
```

```
key_names = ["id", "brand", "model", "production_year", "convertible"]  
key_widths = [10, 15, 10, 20, 15]
```

```
def show_head():  
    for (n, w) in zip(key_names, key_widths):  
        print(n.ljust(w), end='| ')  
    print()
```

```
def show_car(car):  
    for (n, w) in zip(key_names, key_widths):  
        print(str(car[n]).ljust(w), end='| ')  
    print()
```

```
def show(json):  
    show_head()  
    for car in json:  
        show_car(car)
```

```
try:  
    reply = requests.get('http://localhost:3000/cars')  
except requests.RequestException:  
    print('Communication error')  
else:  
    if reply.status_code == requests.codes.ok:  
        show(reply.json())  
    else:  
        print('Server error')
```



id	brand	model	production_year	convertible
2	Chevrolet	Camaro	1988	True
3	Aston Martin	Rapide	2010	False
4	Maserati	Mexico	1970	False
5	Nissan	Fairlady	1974	False
6	Mercedes Benz	300SL	1967	True
7	Porsche	911	1963	False

URL CON ARGUMENTO DE ORDENACIÓN:

```
reply = requests.get('http://localhost:3000/cars?_sort=production_year')
```

WORKING WITH RESTFUL APIS

■ CRUD: Read one

```
import requests

key_names = ["id", "brand", "model", "production_year",
             "convertible"]
key_widths = [10, 15, 10, 20, 15]

def show_head():
    for (n, w) in zip(key_names, key_widths):
        print(n.ljust(w), end='| ')
    print()

def show_empty():
    for w in key_widths:
        print(' '.ljust(w), end='| ')
    print()

def show_car(car):
    for (n, w) in zip(key_names, key_widths):
        print(str(car[n]).ljust(w), end='| ')
    print()
```

```
def show(json):
    show_head()
    if type(json) is list:
        for car in json:
            show_car(car)
    elif type(json) is dict:
        if json:
            show_car(json)
        else:
            show_empty()

try:
    reply = requests.get('http://localhost:3000/cars/2')
except requests.RequestException:
    print('Communication error')
else:
    if reply.status_code == requests.codes.ok:
        show(reply.json())
    elif reply.status_code == requests.codes.not_found:
        print("Resource not found")
    else:
        print('Server error')
```



id	brand	model	production_year	convertible
2	Chevrolet	Camaro	1988	True

WORKING WITH RESTFUL APIS

■ CRUD: Delete

```
headers = {'Connection': 'Close'}
try:
    reply = requests.delete('http://localhost:3000/cars/1')
    print("res=" + str(reply.status_code))
    reply = requests.get('http://localhost:3000/cars/', headers=headers)
except requests.RequestException:
    print('Communication error')
else:
    print('Connection=' + reply.headers['Connection'])
    if reply.status_code == requests.codes.ok:
        show(reply.json())
    elif reply.status_code == requests.codes.not_found:
        print("Resource not found")
    else:
        print('Server error')
```

WORKING WITH RESTFUL APIS

■ CRUD: Create

```
import json

h_close = {'Connection': 'Close'}
h_content = {'Content-Type': 'application/json'}
new_car = {'id': 8,
           'brand': 'Seat',
           'model': 'Panda',
           'production_year': 1980,
           'convertible': False}
print(json.dumps(new_car))
try:
    reply = requests.post('http://localhost:3000/cars', headers=h_content, data=json.dumps(new_car))
    print("reply=" + str(reply.status_code))
    reply = requests.get('http://localhost:3000/cars/', headers=h_close)
except requests.RequestException:
    print('Communication error')
else:
    print('Connection=' + reply.headers['Connection'])
    if reply.status_code == requests.codes.ok:
        show(reply.json())
    elif reply.status_code == requests.codes.not_found:
        print("Resource not found")
    else:
        print('Server error')
```

WORKING WITH RESTFUL APIS

■ CRUD: Update

```
import json

h_close = {'Connection': 'Close'}
h_content = {'Content-Type': 'application/json'}
car = {'id': 6,
       'brand': 'Mercedes PRO',
       'model': '300SL',
       'production_year': 1967,
       'convertible': True}

try:
    reply = requests.put('http://localhost:3000/cars/6', headers=h_content, data=json.dumps(car))
    print("res=" + str(reply.status_code))
    reply = requests.get('http://localhost:3000/cars/', headers=h_close)
except requests.RequestException:
    print('Communication error')
else:
    print('Connection=' + reply.headers['Connection'])
    if reply.status_code == requests.codes.ok:
        show(reply.json())
    elif reply.status_code == requests.codes.not_found:
        print("Resource not found")
    else:
        print('Server error')
```