Lección 17: Peticiones HTTP y consumo de APIs

Índice

Anterior

Lo básico de HTTP

HTTP, de sus siglas en inglés: *"Hypertext Transfer Protocol"*, es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos en internet, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor. Así funcionan los navegadores web.

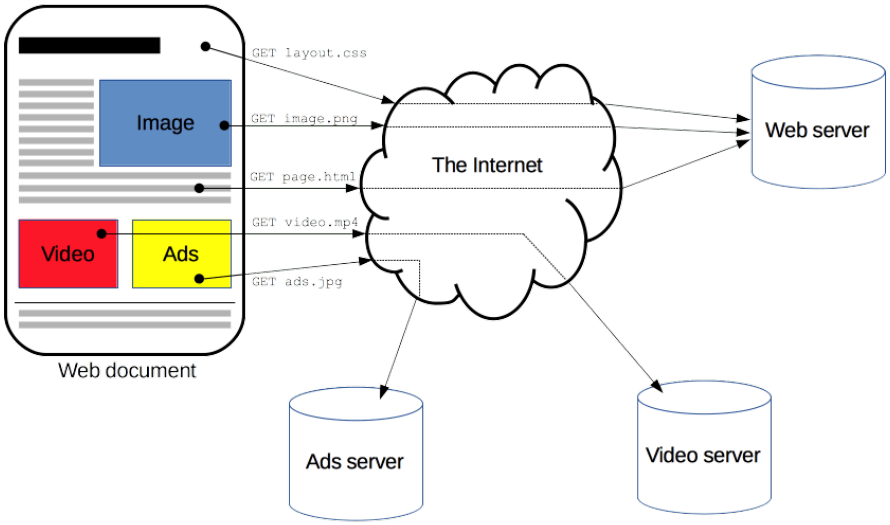

Figure 1: Esquema básico de HTTP

La comunicación entre un cliente(navegador web, App móvil o un programa de python) y un servidor en internet se produce mediante una serie de *peticiones* y *respuestas*. El cliente inicia el proceso haciendo una petición al servidor web y recibe una respuesta del servidor web. Con sucesivas peticiones y respuestas se completa la comunicación.

Las peticiones y respuestas HTTP, comparten una estructura similar, compuesta de: 1. Una línea de inicio ('start-line' en inglés) describiendo la petición a ser implementada, o su estado, sea de éxito o fracaso. Esta línea de comienzo, es siempre una única línea. 2. Un grupo opcional de cabeceras HTTP, indicando la petición o describiendo el cuerpo ('body' en inglés) que se incluye en el mensaje. 3. Una línea vacía ('empty-line' en inglés) indicando toda la meta-información

ha sido enviada. 4. Un campo de cuerpo de mensaje opcional ('body' en inglés) que lleva los datos asociados con la petición (como contenido de un formulario HTML), o los archivos o documentos asociados a una respuesta (como una página HTML, o un archivo de audio, vídeo …) . La presencia del cuerpo y su tamaño es indicada en la línea de inicio y las cabeceras HTTP.

La línea de inicio y las cabeceras HTTP, del mensaje, son conocidas como la **cabeza de la peticiones** o **headers**, mientras que su contenido en datos se conoce como el **cuerpo del mensaje** o **payload**.

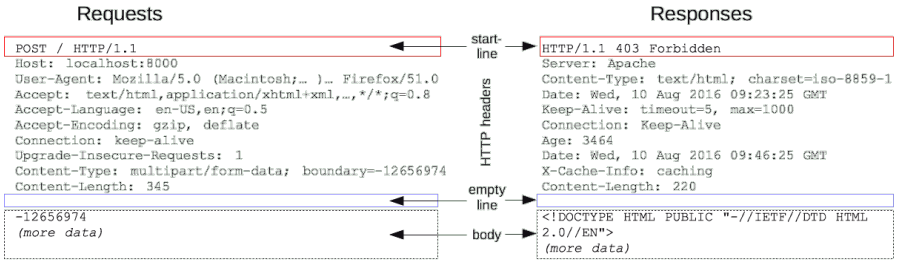

Figure 2: Estructura mensajes HTTP

Peticiones HTTP - Requests

Línea de inicio

La línea de inicio está formada por tres elementos:

1. Un método HTTP (GET, POST u otros), que describan la acción que se pide sea realizada. Por ejemplo, GET indica que un archivo ha de ser enviado hacia el cliente, o POST indica que hay datos que van a ser enviados hacia el servidor (creando o modificando un recurso, o generando un documento temporal para ser enviado).
2. El destino de la petición, normalmente es una URL, o la dirección completa del protocolo, puerto y dominio también suelen ser especificados por el contexto de la petición. El formato del objetivo de la petición varia según los distintos métodos HTTP.
3. La versión de HTTP, la cual define la estructura de los mensajes, actuando como indicador, de la versión que espera que se use para la respuesta.

Ejemplo de una línea de inicio: GET http://developer.mozilla.org/es/docs/Web/HTTP/Messages HTTP/1.1

Cabeceras - Headers

Las cabeceras (en inglés headers) HTTP permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta. Una cabecera de petición esta compuesta por su nombre (no sensible a las mayusculas) seguido de dos puntos ':', y a continuación su valor (sin saltos de línea).

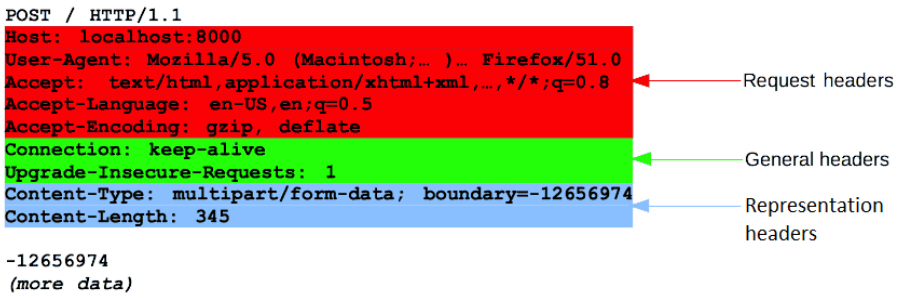

Figure 3: Cabeceras de petición HTTP

En las cabeceras van también las cookies, si las hubiera para el dominio al que se envía la petición.

Cuerpo - Payload

La parte final de la petición el el cuerpo, es el contenido de la petición. No todas las peticiones llevan uno: las peticiones que reclaman datos, como GET, normalmente no necesitan ningún cuerpo. Las peticiones que envían datos, como POST, normalmente siempre llevan un cuerpo. El cuerpo de la petición puede ser datos en formato JSON, puede ser una imágen, cualquier tipo de archivo, …

Respuestas HTTP - Responses

Linea de estado

La línea de inicio de una respuesta HTTP, se llama la línea de estado, y contienen la siguiente información:

1. La versión del protocolo, normalmente HTTP/1.1.
2. Un código de estado, indicando el éxito o fracaso de la petición. Códigos de estado muy comunes son: 200, 404, o 302
3. Un texto de estado, que es una breve descripción, en texto, a modo informativo, de lo que significa el código de estado, con el fin de que una persona pueda interpretar el mensaje HTTP.

Una línea de estado típica es por ejemplo: HTTP/1.1 404 Not Found.

Cabeceras - Headers

Contiene las cabeceras de la respuesta con la misma estructura que las cabezas de la petición. En las cabeceras van también las cookies que el servidor envía al cliente.

Cuerpo - Payload

La última parte del mensaje de respuesta el es 'cuerpo'. No todas las respuestas tienen uno, respuestas con un código de estado como 201 o 204 (en-US) normalmente prescinden de él.

Códigos de respuesta

- 200: OK
- 301: Moved Permanently
- 400: Bad Request
- 401: unauthorized
- 403: Forbidden
- 404: Not Found
- 418: I'm a teapot

Requests en Python

Requests permite enviar peticiones HTTP/1.1 de forma extremadamente sencilla. No es necesario añadir manualmente cadenas de consulta a las URL ni codificar los datos POST. Keep-alive y HTTP connection pooling son 100% automáticos, gracias a urllib3.

Lo primero que hay que hacer para trabajar con requests es importarlo:

```
import requests
```

Hacer una petición GET:

```
r = requests.get('https://www.aepd.es')
print(r.status_code)
print(r.request.headers)
print(r.headers)
if r.status_code == requests.codes.ok:
    print(r.text)
```

API REST

API es el acrónimo de interfaz de programación de aplicaciones (application programming interface en inglés). Es un conjunto de reglas bien definidas que se utilizan para especificar formalmente la comunicación entre dos componentes de software.

Una API REST es una interfaz de comunicación entre sistemas de información que usa el protocolo de transferencia de hipertexto (hypertext transfer protocol o HTTP, por su siglas en inglés) para obtener datos o ejecutar operaciones sobre dichos datos en diversos formatos, como pueden ser XML o JSON. En su mayoría las API REST actuales utilizan JSON.

Para utilizar una API, debe realizar una petición a un servidor web remoto y recuperar los datos de respuesta en JSON.

Vamos a utilizar una API de una tienda online fake para pruebas, porque es una API sencilla y gratuita que no requiere autenticación.

Métodos HTTP y códigos de en API REST

Los métodos HTTP le dicen a la API qué queremos hacer:

Método	Acción
GET	Pedir datos
POST	Añadir datos
PUT	Actualizar datos existentes
DELETE	Eliminar datos

Una vez que una API REST recibe y procesa una solicitud HTTP, devuelve una respuesta con un código de estado HTTP. Este código de estado proporciona información sobre la respuesta y ayuda a la aplicación cliente a saber de qué tipo de respuesta se trata.

Rango	Información
1xx	Respuesta informativa
2xx	Operación ejecutada con éxito
3xx	Redirección
4xx	Error en el cliente/peticion
5xx	Error en el servidor

Documentación de una API

Normalmente cualquier API permitirá consultar su documentación para saber a qué **endpoints** se puede pueden enviar peticiones, qué tipo de peticiones se pueden enviar, con qué parámetros y funcionalidad tienen.

Los **endpoints** de la API son las URL públicas expuestas por el servidor que una API y que el cliente utiliza para acceder a recursos y datos.

HTTP method	API endpoint	Description
GET	/products	Get a list of products.
GET	/products?limit=x	Get only x products.
GET	/products/	Get a single product.
POST	/products	Create a new product.
PUT	/products/	Update a product.
PATCH	/products/	Partially update a product.
DELETE	/products/	Delete a product.

La URL de la API es https://fakestoreapi.com

Haciendo peticiones GET a una API

Si se hace una primera petición a un recurso inexistente de una API, se obtendrá una respuesta con un código de estado error. En el siguiente ejemplo el código de estado obtenido será ERROR 404, cuyo significado es *Not Found* (recurso no encontrado).

```
r = requests.get('https://fakestoreapi.com/este-endpoint-no-existe')
print(r.status_code) # 404 Error porque el endpoint no existe
```

Si se hace una petición a un **endpoint** existente y siguiendo la estructura de la documentación, debería obtenerse un código de estado SUCCESS, por ejemplo 200 OK.

```
r = requests.get('https://fakestoreapi.com/products')
if r.status_code == requests.codes.ok: # 200 OK
    print(type(r.json()))
    print(json.dumps(r.json()[0:2],indent=4)) # Imprime los dos primeros productos que ha d
```

Peticiones GET con parámetros

Según la documentación de la API, el **endpoint** *products* admite un parámetro de url *limit* para limitar el número de productos que envía el servidor.

```
params = {'limit': 2}
r = requests.get('https://fakestoreapi.com/products', params=params)
print(r.url)  # https://fakestoreapi.com/products?limit=2
if r.status_code == requests.codes.ok:
    print(json.dumps(r.json(),indent=4)) # La API devuelve 2 productos
```

El mismo **endpoint** también admite recuperar un único producto especificando el *product_id* en la petición. En este caso, el parámetro *product_id* formará parte del *path* de la URL y no se envía como un parámetro.

```
URL = 'https://fakestoreapi.com/'
ENDPOINT = 'products/'
product_id = 15

url = URL + ENDPOINT + str(product_id)

print(url) # https://fakestoreapi.com/products/15

r=requests.get(url)
if r.status_code == requests.codes.ok:
    print(r.json())
    for key,value in r.json().items():
        print(f'{key}: {value}')
```

Peticiones POST a una API

Según la documentación de la API, el **endpoint** *products* admite peticiones POST para crear nuevos productos. En este caso, vamos a añadir un nuevo producto de prueba:

```
new_product = {
    "title": 'test product',
    "price": 13.5,
    "description": 'This is just a test product ',
    "image": 'https://i.pravatar.cc',
    "category": 'electronic'
}

URL = 'https://fakestoreapi.com/'
ENDPOINT = 'products/'

url = URL + ENDPOINT

r=requests.post(url, data=new_product)
if r.status_code == requests.codes.ok:
    print(r.json()) # Devuelve el nuevo producto, incluyendo el product_id asignado
    for key,value in r.json().items():
        print(f'{key}: {value}')
```

Peticiones PUT y DELETE

El mismo **endopoint** admite peticiones PUT para modificar productos existentes y peticiones DELETE para eliminar productos. A continuación se muestra un ejemplo para modificar el producto que acabamos de crar y otro ejemplo para eliminarlo de la lista de productos.

```
URL = 'https://fakestoreapi.com/'
ENDPOINT = 'products/'

url = URL + ENDPOINT + str(new_product_id)

product_changes = {
    'price': '20.5',
    'category': 'photo'
}

r=requests.put(url, data=product_changes)
if r.status_code == requests.codes.ok:
    print(r.json()) # Devuelve los campos modificados

r=requests.delete(url)
if r.status_code == requests.codes.ok:
```

```
    print(r.json()) # Elimina el producto
```

Enviar un archivo en una petición con requests

```
ficheros = {'file1': open('./nombre_fichero.pdf', 'rb')}
r = requests.post('https://mipagina.xyz/form/', files=ficheros)
```

Descargar un archivo con requests

```
url = '' # URL del archivo descargar (en este caso un binario como puede ser un pfd o una i
filepath = './descargado.pdf'  # Path dónde guardar el archivo
with requests.get(url, stream = True) as r:
    with open(filepath, 'wb') as f:
        for chunk in r.iter_content(chunk_size=1024):
            f.write(chunk)
```

Proyecto de Ejemplo: Consumiendo una API real con autenticación

urlscan.io es un servicio de internet para escanear páginas web, que ofrece una API que permite enviar URL para su escaneado y recuperar los resultados una vez finalizado el escaneado. Además, se puede utilizar la API para buscar análisis existentes por atributos como dominios, IP, números de sistema autónomo (AS), hashes, etc. números, hashes, etc.

Para utilizar las API, se debe crear una cuenta de usuario, crear una API key e incluirla en las llamadas a la API.

El **objetivo del proyecto** es crear un **módulo de python** con funciones que puedan reutilizarse desde otros proyectos y que el módulo pueda llamarse por sí mismo para hacer consultas a la API.

El módulo se llamará urlscan.py y contendrá tres funciones:

1. Una función *submit* para enviar urls a analizar. Recibe como parámetro de entrada la API key y la url a analizar. Devuelve un diccionario/json con la respuesta obtenida de la API.
2. Una función *result* para consultar un análisis a partir de su *uuid*. Recibe como parámetro de entrada la *API key* y el *uuid*. Devuelve un diccionario/json con la respuesta obtenida de la API.
3. Una función *search* para buscar análisis previos sobre algún dominio. Recibe como parámetro de entrada la *API key* y el *dominio* a buscar. Devuelve un diccionario/json con la respuesta obtenida de la API.
4. [Opcional] Una función *report* para gerenar un informe en .docx de un análisis. Recibe como parámetro de entrada la *API key* y el *uuid*. Crea un documento report.docx con el informe del análisis.

El módulo se podrá llamar desde línea de comando especificando 3 parámetros: API key, comando (submit, result, search, report), parámetro (url, uuid, dominio).

> **Warning:** Hay que evitar que la API key se quede hardcodeada en el código y pueda acabar publicada.

De forma no exhaustiva, los pasos a seguir son los siguientes:

1. Consultar la (documentación de la API)[https://urlscan.io/docs/api/] para ver qué **endpoints** están disponibles, cómo hay que llamarlos y qué resultados devuelven.
2. Crear un usuario en (urlscan.io)[https://urlscan.io/user/signup], hacer login y crear un nuevo API key y copialo para poder hacer primeras peticiones de prueba.
3. Crear una nueva carpeta para el proyecto y un entorno virtual dentro del proyecto.
4. Activar el entorno virtual e instalar los modulos/paquetes necesarios con pip: *requests* para las peticiones a la API y *docxtpl* para los informes en docx.
5. Crear el fichero requiements.txt con *pip freeze > requirements.txt*
6. Crear un archivo urlscan.py y comprobar que se pueden hacer peticiones a la API para enviar urls, recuperar resultados de análisis y buscar dominios.
7. Dar forma las funciones en el módulo, que se podrán llamar desde otros módulos cuando se importe urlscan.
8. Completar con la parte del código que se ejecutará cuando el módulo se llame independientemente desde línea de comandos.

Solución: Encuentra una posible solución en (urlscan)[https://github.com/mercaderd/curso_python/tree/main/17_HTTP/urlscan]

Siguiente