# Python

Módulo 06

# Interacción entre Requests y el servicio

# Interacción entre Requests y el servicio

Nuestro servicio de prueba corre en la dirección http://localhost:7001/ y expone el recurso (según lo que acabamos de decir sobre la arquitectura REST) http://localhost:7001/alumno. Sabido esto, creemos un nuevo archivo de Python para interactuar con el servicio vía la librería ***Requests.***

Pero primero veamos cómo funciona este paquete.

Observemos el siguiente código.

```
import requests

r = requests.get("https://google.com/")
print(r.status_code)
```

La primera línea de código importa el paquete según hemos visto anteriormente. La segunda realiza una petición vía el método ***GET*** a la dirección de URL https://google.com/. El valor de retorno de la función ***get()*** es la respuesta del servidor que opcionalmente tendrá algún contenido (un código HTML en este caso por tratarse de un sitio web), pero obligatoriamente habrá un código de estado *(status code)* que indicará si la operación se ejecutó correctamente o no.

Si estamos conectados a internet, el código anterior debe imprimir lo siguiente.

```
200
```

Es decir, el código de estado *(r.status_code)* de la respuesta del servidor es *200.* El valor 200 (y también los valores 201, 202 y 204) indica que la operación se ejecutó correctamente.

Ahora bien, vayamos a lo que nos interesa que es interactuar con un servicio web. El servicio que tenemos de prueba expone el siguiente recurso con la arquitectura **REST**:

http://localhost:7001/student

Por lo general los recursos de los servicios web estarán escritos en inglés, así que podemos ir acostumbrándonos desde el comienzo. Este recurso permite obtener, agregar, modificar y eliminar alumnos *(students)*. Cada alumno del recurso tiene un nombre *(name)* y una cantidad de cursos realizados *(courses)*. Intentemos hacer una petición GET a este recurso a ver cuál es el resultado.

```
import requests

r = requests.get("http://localhost:7001/student")
print("Código de estado:", r.status_code)
print("Contenido de la respuesta:", r.json())
```

Aquí lo que hemos agregado es una cuarta línea en donde se llama a ***r.json()***, que lee el contenido de la respuesta del servidor en formato JSON y lo convierte a las estructuras de Python correspondientes.

El texto impreso en pantalla es el siguiente:

```
Código de estado: 200
Contenido de la respuesta: {'students': []}
```

Vemos que *r.status_code == 200*, por lo que la operación se ejecutó correctamente. Recordemos que el método GET, en la arquitectura REST, quiere decir *"obtener una lista de los registros de un recurso"*. Si nuestro recurso es */student*, la operación *GET /student* debe responder con una lista de alumnos. La respuesta la obtenemos vía ***r.json()***, como vimos recién.

Para trabajar mejor con ella, asignémosla a una variable:

```
r = requests.get("http://localhost:7001/student")
print("Código de estado:", r.status_code)
respuesta = r.json()
print("Contenido de la respuesta:", respuesta)
```

Según lo impreso en pantalla, r*espuesta* == *{'students': []}*, es decir, es un diccionario con un único elemento, cuya clave es *"students"* y su valor, una lista vacía. Lo cual quiere decir que aún no hay alumnos cargados en el recurso */student*. Pero por el momento ya sabemos que, luego de hacer esta petición, la lista de alumnos estará en *respuesta["students"]*.

Ya vimos que el método HTTP para agregar información a un recurso es ***POST.*** En el caso de nuestro servicio de prueba, lo estaremos usando para insertar nuevos alumnos. Dijimos que cada alumno tiene un nombre y una cantidad de cursos. Entonces, el código para insertar un nuevo alumno se vería más o menos así:

```
r =
requests.post("http://localhost:7001/student",
              json={"name": "Lautaro", "courses":
3})
print("Código de estado:", r.status_code)
print("Contenido de la respuesta:", r.json())
```

Esto imprime:

```
Código de estado: 201
Contenido de la respuesta: {'id': 1}
```

¡Perfecto! Dijimos que los códigos de estado 200, 201, 202 y 204 indican que la operación se ejecutó correctamente. En los servicios con arquitectura REST, el código 201 indica que se insertó satisfactoriamente un nuevo registro a un recurso. También tenemos una respuesta, el diccionario *{"id": 1}* que nos indica que el servicio le asignó el número 1 al alumno que acabamos de crear. Este identificador nos será útil luego para hacer modificaciones sobre ese alumno e incluso eliminarlo.

Agreguemos algunos alumnos más para tener más de un registro con el cual seguir trabajando:

```
alumnos = (
   ("Juan", 1),
   ("Sofía", 5),
   ("Martín", 2)
)
for nombre, cursos in alumnos:
   r =
requests.post("http://localhost:7001/student",
                  json={"name": nombre, "courses":
cursos})
   print("Código de estado:", r.status_code)
   print("Contenido de la respuesta:", r.json())
```

Si todo salió bien, al ejecutarse esto debe imprimir:

```
Código de estado: 201
Contenido de la respuesta: {'id': 2}
Código de estado: 201
Contenido de la respuesta: {'id': 3}
Código de estado: 201
Contenido de la respuesta: {'id': 4}
```

Ahora que tenemos algunos alumnos, si volvemos a ejecutar nuestro primer código...

```
r = requests.get("http://localhost:7001/student")
print("Código de estado:", r.status_code)
respuesta = r.json()
print("Contenido de la respuesta:", respuesta)
```

...obtendremos lo siguiente:

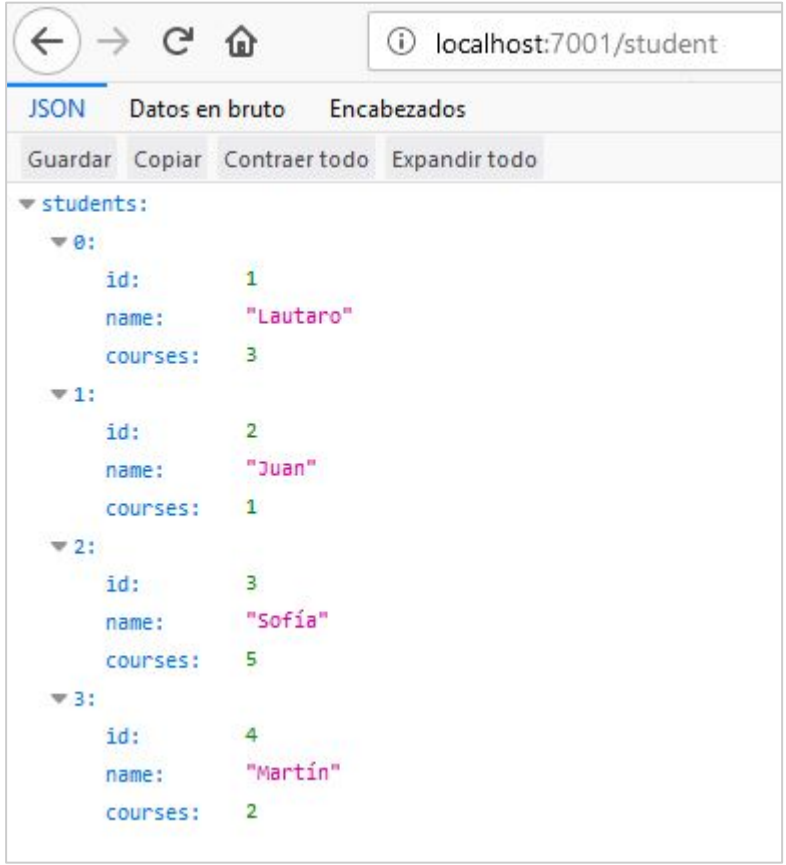
```
Código de estado: 200
Contenido de la respuesta: {'students': [{'id': 1,
'name': 'Lautaro', 'courses': 3}, {'id': 2, 'name':
'Juan', 'courses': 1}, {'id': 3, 'name': 'Sofía',
'courses': 5}, {'id': 4, 'name': 'Martín', 'courses':
2}]}
```

Ahora el contenido de *respuesta["students"]* ya no es una lista vacía. Cada uno de sus elementos representa a un alumno como un diccionario. Cada alumno tiene tres claves: un identificador numérico ("id"), un nombre ("name") y una cantidad de cursos ("courses").

Podemos recorrer cada uno de los alumnos usando un bucle “for” así:

```
r = requests.get("http://localhost:7001/student")
if r.status_code == 200:
    respuesta = r.json()
    for alumno in respuesta["students"]:
        print("Alumno", alumno["id"])
        print("Nombre:", alumno["name"])
        print("Cursos:", alumno["courses"])
else:
    print("Ocurrió un error.")
```

Si ingresamos a http://localhost:7001/student desde el navegador también será capaz de mostrarnos la lista de alumnos, puesto que cada vez que visitamos una dirección de URL desde un navegador web se realiza una petición del tipo GET.



Ahora bien, usando ***GET*** también podemos obtener la información de algún alumno en particular indicando su identificador numérico. Por ejemplo, *GET /student/3* debe retornar únicamente la información de Sofía. Comprobémoslo:

```
r =
requests.get("http://localhost:7001/student/3")
print("Código de estado:", r.status_code)
print("Contenido de la respuesta:", r.json())
```

En efecto, el resultado es:

```
Código de estado: 200
Contenido de la respuesta: {'student': {'name':
'Sofía', 'courses': 5}}
```

Nótese que en este caso la información del alumno está dentro de la clave *"student"*, por lo que habremos de acceder a ella más o menos así:

```
r =
requests.get("http://localhost:7001/student/3")
if r.status_code == 200:
    alumno = r.json()["student"]
    print("Nombre:", alumno["name"])
    print("Cursos:", alumno["courses"])
else:
    print("Ocurrió un error.")
```

¿Cómo proceder si queremos modificar algún dato (nombre o cantidad de cursos) del alumno 3 (Sofía)? Según lo que vimos más arriba, debemos usar el método ***PUT.*** Esto en Python se traduce como la función ***requests.put()***, y vamos a pasar como argumento un diccionario con las claves que queremos modificar y sus respectivos valores, que luego serán convertidas a formato JSON automáticamente por la librería.

El siguiente código cambia la cantidad de cursos de Sofía a 6:

```
datos = {"courses": 6}
r = requests.put("http://localhost:7001/student/3", json=datos)
print("Código de estado:", r.status_code)
```

El resultado en pantalla es:

```
Código de estado: 204
```

El código de estado 204 indica que la petición se ejecutó correctamente, pero a diferencia de los estados anteriores, este no retorna ningún contenido. Por eso no llamamos a ***r.json()***.

Si la petición fallara por alguna razón, su código de estado sería otro que 204. Por ejemplo:

```
datos = {"courses": 6}
r = requests.put("http://localhost:7001/student/10", json=datos)
print("Código de estado:", r.status_code)
```

Este código imprime:

```
Código de estado: 404
```

El código de estado 404 indica que no se encontró el recurso sobre el cual se quiere ejecutar la operación. En este caso, efectivamente, no hay ningún alumno cuyo identificador sea el número 10.

Al invocar la función ***put()***, es posible modificar varias claves simultáneamente:

```
# Modifica el nombre y la cantidad de cursos del alumno 3.
datos = {"name": "Josefina", "courses": 6}
r = requests.put("http://localhost:7001/student/3", json=datos)
print("Código de estado:", r.status_code)
```

Por último, para eliminar un registro en un recurso usamos el método ***DELETE*** del protocolo HTTP: esto es, en Python, ***requests.delete()***.

```
# Elimina el alumno 3.
r = requests.delete("http://localhost:7001/student/3")
print("Código de estado:", r.status_code)
```

El código de estado es el mismo que para las operaciones PUT:

```
Código de estado: 204
```

# ¡Muchas gracias!

¡Sigamos trabajando!