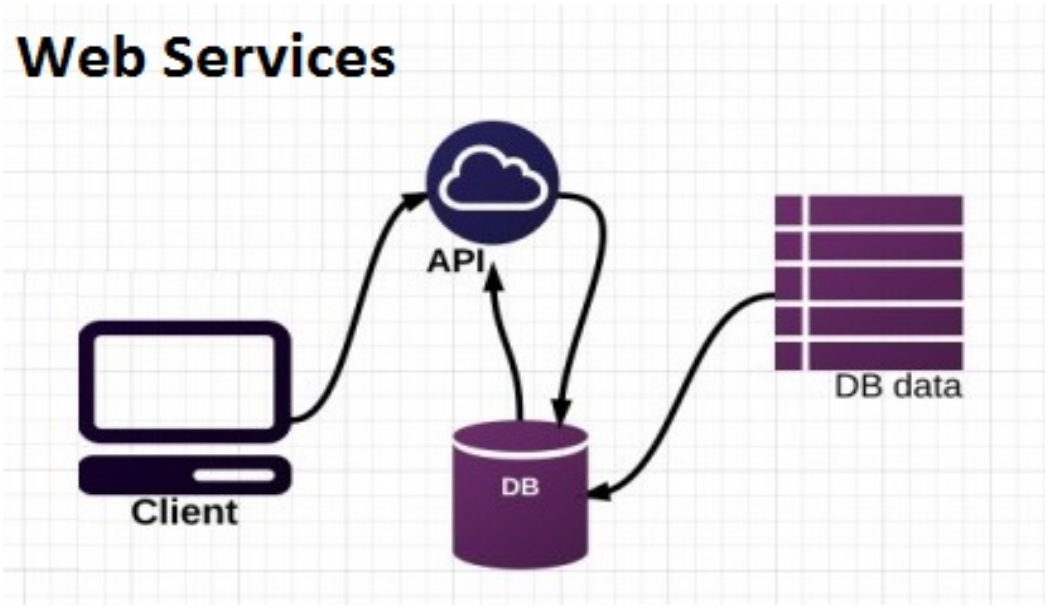

2 sesiones

Guión de Prácticas 2.2: API RESTfull

S. Alonso (zerjioi@ugr.es) y J.M. Guirao (jmguirao@ugr.es)

Entrega de las prácticas 0, 1 y 2: 16 Octubre

En esta práctica haremos API RESTfull sobre la BD de la práctica anterior. Seguimos las indicaciones de What is a REST API? , y las directrices RESTful:

- Usar los métodos de HTTP explicitamente
- Sin estado, cacheable
- Exponer urls estilo path, interface uniforme
- Responder JSON o XML

Endpoints

Lo primero será planificar que urls, con que verbos, y que respuesta tendrán. En este caso expondrán el recurso 'recipes':

- `GET /api/recipes`, devolverá un lista con todos los registros
- `GET /api/recipes?con=vodka`, devolverá un lista esta búsqueda
- `POST /api/recipes`, creará un registro nuevo a partir de los parámetros que enviemos con el

POST, y devolverá el `id` del registro creado, y sus datos

- `PUT /api/recipes/5f7b4b6204799f5cf837b1e1`, modificará el registro a partir de los parámetros enviados en el PUT y devolverá el `id` del registro modificado y sus datos
- `DELETE /api/recipes/5f7daa018ec9dfb536781afa`, borrara el registro correspondiente y devolverá el `id` del registro borrado

El código sería:

```
#./app/app.py
from flask import request, jsonify
from bson import ObjectId
...
# para devolver una lista (GET), o añadir (POST)
@app.route('/api/recipes', methods=['GET', 'POST'])
def api_1():
    if request.method == 'GET':
        lista = []
        buscados = db.recipes.find().sort('name')
        for recipe in buscados:
            recipe['_id'] = str(recipe['_id']) # casting a string (es un Obje
            lista.append(recipe)
        return jsonify(lista)

        if request.method == 'POST':
            ...

# para devolver una, modificar o borrar
@app.route('/api/recipes/<id>', methods=['GET', 'PUT', 'DELETE'])
def api_2(id):
    if request.method == 'GET':
        buscado = db.recipes.find_one({'_id':ObjectId(id)})
          if buscado:
            buscado['_id'] = str(buscado['_id']) # casting a string (es un Ob
            return jsonify(buscado)
          else:
            return jsonify({'error':'Not found'}), 404
    ...
```

¡ATENCIÓN! En este ejemplo hemos utilizado la función `jsonify` que básicamente convierte un objeto de Python y lo transforma en una cadena de tipo `json`. En la práctica anterior utilizamos la función `dumps` (que viene con `PyMongo`) para hacer exactamente lo mismo. La diferencia es que `dumps` se traga perfectamente los objeto de tipo `ObjectId`, cosa que no hace `jsonify` y por eso en el código de más arriba hemos hecho un casting manualmente del `_id`. Podéis utilizar la que prefiráis. Además en el ejemplo que hemos puesto no especificamos que lo que se devuelve es un objeto `JSON` (en la práctica anterior si). Recomendamos modificar el código para decirle al navegador que le estamos mandando datos en formato `JSON`.

En este código, falta la previsión de errores que puedan ocurrir: la BD no responde, envio de parámetros incorrectos, etc.

Para probarlo podemos usaremos la extensión REST Client de visual studio.

Hacer una segunda versión del api usando la librería flask-restfull, como en Python REST API Tutorial - Building a Flask REST API.

Incluir en la entrega el archivo *.http* con las pruebas hechas con la extensión REST Client.

Referencias:

- collection – Collection level operations