Exercicis sobre REST

En el cas de ser un desenvolupador, moltes vegades necessitarem instal·lar, configurar i omplir de dades un servidor REST per fer proves de la nostra aplicació (ja que l'ús d'aquest tipus de servidors és molt habitual sobre tot si l'aplicació en qüestió és de tipus mòbil). Aquesta infraestructura prèvia sol ser bastant farragosa de preparar (¿PHP o Python?, ¿MySQL o PostgreSQL?, etc), a més de treure temps/energia de la tasca principal, que hauria de ser desenvolupar la lògica i la interfície gràfica de l'aplicació en sí. És per això que existeixen un grapat de servidors REST públics amb dades "fake" llestos per poder-se utilitzar lliure i gratuïtament (sota certs límits). Tant si es vol fer una "demo", o compartir exemples durant un "workshop" o redactar un tutorial o simplement fer tests, aquesta possibilitat ens estalviarà molta feina ingrata. En aquest sentit, el proper exercici demanarà provar el servei REST públic https://jsonplaceholder.typicode.com) , el qual està orientat a servir recursos només de tipus JSON.

> NOTA: Aquest servei REST és open-source, així que si volguéssim, el podríem implementar alguna màquina local per no dependre d'Internet ni de tercers. Concretament, el seu codi es pot descarregar a https://github.com/typicode/json-server (el qual necessita també https://github.com/typicode/lowdb , que representa el "backend" que fa de base de dades)
> NOTA: La mateixa gent que desenvolupa aquest servei REST també ofereix el servei https://my-json-server.typicode.com gratuïtament, el qual permet utilitzar el seu servei online (és a dir, no haver d'instal.lar res a la màquina local) però amb dades pròpies (emmagatzemades en un repositori GitHub personal). D'aquesta manera, s'obtenen tots els beneficis de no haver de "muntar" res i a més treballar amb dades amb format personalitzat, no amb dades "fake" que no es poden canviar.
> NOTA: El codi font d'un altre servei REST similar es pot trobar a https://github.com/lhorie/rem . Un detall important és que aquest servei no utilitza cap "backend" de base de dades sinó que aquestes les emmagatzema (i les transmet d'extrem a extrem) via "cookies.

1.-a) Realitza, des de la màquina real, una petició GET amb *curl* als següents "endpoints" i digues què n'obtens en cada cas:

https://jsonplaceholder.typicode.com/users
https://jsonplaceholder.typicode.com/users/9
https://jsonplaceholder.typicode.com/posts
https://jsonplaceholder.typicode.com/posts/47
https://jsonplaceholder.typicode.com/comments
https://jsonplaceholder.typicode.com/comments/5
https://jsonplaceholder.typicode.com/albums
https://jsonplaceholder.typicode.com/albums/3
https://jsonplaceholder.typicode.com/photos
https://jsonplaceholder.typicode.com/photos/1

b) ¿Què fa la següent petició POST (en realitat és una simulació perquè no es fa realment) i què és el que representa que veus per pantalla (compte amb el salt de línia si copies-pegues!)?

curl -X POST https://jsonplaceholder.typicode.com/posts -H "Content-Type:application/json"
-d '{"title":"Hola","body":"Bonica flor", "userId":"4"}'

c) ¿Quina diferència hi ha entre la petició POST anterior i la petició PUT següent? (observa la negreta i el que veus per pantalla)

*curl -X **PUT** https://jsonplaceholder.typicode.com/posts/**12** -H "Content-Type:application/json"*
-d '{"title":"Hola","body":"Bonica flor", "userId":"4"}'

cII) ¿Quina diferència hi ha entre la petició PUT anterior i la petició PATCH següent? (observa el que veus per pantalla)

*curl -X **PATCH** https://jsonplaceholder.typicode.com/posts/**12** -H "Content-Type:application/json"*
-d '{"title":"Adéu"}'

d) ¿Què creus que simula fer la següent petició DELETE?

curl -X DELETE https://jsonplaceholder.typicode.com/posts/63

Tal com haureu pogut comprovar si feu una petició GET després d'una POST/PUT/PATCH/DELETE al servidor REST utilitzat a l'exercici anterior, cap modificació es realitza realment: tot és mentida. En el cas, però, de què volguem enviar dades de veritat (mitjançant POST i no necessàriament en format JSON) a un servidor online de forma fàcil i gratuïta, ja coneixem d'exercicis anteriors una possibilitat: http://httpbin.org A l'exercici següent repetirem el mateix que ja vam veure en el seu moment però fent servir ara un altre servidor online molt semblant a "Httpbin", en aquest cas accesible a través de https://public.requestbin.com/r

2.-Vés, utilitzant un navegador qualsevol, al darrer enllaç indicat al paràgraf blau anterior. Veuràs que aquest enllaç et generarà dinàmicament un servidor propi amb un nom similar a https://xxxx.y.pipedream.net (on "xxxx.y" és una cadena aleatòria); aquest nom serà el que hagis de fer servir a les peticions que realitzaràs als propers apartats d'aquest exercici

> NOTA: Pots observar com a la part dreta de la web mostrada apareixen diferents exemples de com interaccionar via POST amb aquest servidor, ja sigui mitjançant Curl (que és el que farem nosaltres), o a través de diferents llenguatges de programació

a) Obre un terminal per enviar una petició POST següent:

curl -X POST https://xxxx.y.pipedream.net -d '{"nom":"ana","edat":26}' -H "Content-Type:application/json"

b) Ara realitza la petició POST següent. ¿Quina diferència hi ha amb l'anterior? ¿Quin és el valor de la capçalera "Content-Type" en aquest cas?

curl -X POST https://xxxx.y.pipedream.net -d "nom=ana&edat=26"

> NOTA: Recorda que si les dades (ja siguin en format JSON o "url-encoded") estan guardades en un fitxer, llavors hauríem d'utilitzar el paràmetre *-d* així: ***-d @/ruta/fitxer.txt*** I si provinguessin d'una canonada des d'una comanda prèvia (és a dir, de "xtdin"), caldria utilitzar *-d* així: **-d @-**

c) Comprova finalment que tant les dades enviades amb la primera petició POST com amb la segona hagin arribat bé anant amb el navegador a la pàgina https://xxxx.y.pipedream.net (concretament, fixa't en la seva part esquerra, on apareix el llistat de les peticions rebudes i on pots clicar a sobre de cadascuna d'elles per conèixer els seus detalls -com per exemple, les seves capçaleres i, el que més ens importa, les dades enviades, les quals es presenten de forma maca /"structured" o a pèl/"raw" -)

> NOTA: Existeixen més serveis gratuïts online similars al vist en aquest exercici. Exemples són:
> https://restful-api.dev ; https://reqbin.com ; https://reqres.in o
> https://github.com/mxcxvn/requestbin.net (aquest no és un servei online sinò el codi font per implementar-ne un de propi)

3.-a) Executa la comanda *curl -s "https://api.openweathermap.org/data/2.5/weather?q=Badalona&units=metric&APPID=5b305e4672d3193b7a9c65ae36749356" | jq ".main.temp"* i esbrina el significat del valor obtingut consultant la documentació oficial de l'API usada (disponible a https://openweathermap.org/current)

> NOTA: Les cometes envoltant la URL indicada a la comanda curl són importants perquè si no el terminal Bash interpretaria el símbol "&" (com a indicador de posar un procés en segon pla), cosa que òbviament no volem

aII) ¿Què significa la cadena "5b305e46..." anterior?¿Per a què serveix el paràmetre "units=metric"? ¿Quina funció té la comanda *jq* en aquest exemple?

> NOTA: A l'apartat anterior s'ha vist un mètode per restringir l'accés a una API consistent en passar una clau API com a cadena de consulta a l'URL (per exemple així, https://example.com/api/v1/widget?apikey=mysecretkey). És una opció molt fàcil però no és la més segura, perquè és molt fàcil filtrar accidentalment una clau enganxant un URL al lloc equivocat. Altres mètodes que sovint també s'utilitzen per passar la clau del client al servidor són
> * Passar la clau API a la capçalera de sol·licitud HTTP "Basic Authentication". És una opció correcta però cal decidir si es passa la clau com a nom d'usuari o com a contrasenya. ¿Què fem llavors si volem distingir als usuaris ja autenticats?
> * Passar la clau API a la capçalera de sol·licitud HTTP "Authorization" és probablement la millor manera (per exemple així, *Authorization: ApiKey mysecretkey*)
> * Una altra opció és implementar una capçalera personalitzada (per exemple, "X-MyCo-ApiKey"), però no és estàndard i no ofereix cap avantatge respecte a la capçalera "Authorization" estàndard.

aIII) Digues, d'entre les diferents API oferides pel servei OpenWeatherMap (llistades a https://openweathermap.org/price), si l'API que ofereix dades històriques ("Historical map") és de lliure accés o si és de pagament

b) ¿Què fan les següents comandes (*curl* i *jq*): *curl -s https://api.opensource.org/licenses/ | jq ".[].name"* ? Pista: Prova primer la comanda *curl* només i, a partir del resultat obtingut, dedueix el que filtra *jq*

NOTA: Els claudàtors ("[]") s'han d'indicar en el filtre de *jq* quan l'element en qüestió (en aquest cas, el ".") sigui un array

bII) Llegeix la documentació de l'API (https://github.com/OpenSourceOrg/api/blob/master/doc/endpoints.md) per tal d'esbrinar el significat del resultat de la comanda: *curl -s https://api.opensource.org/licenses/copyleft*

c) ¿Què fan les següents comandes (*curl* i *jq*)? Pista: Prova primer la comanda *curl* només i, a partir del resultat obtingut, dedueix el que filtra la comanda *jq*

curl -s "https://api.mymemory.translated.net/get?q=Bocadillo&langpair=es|en" | jq ".responseData.translatedText"

d) ¿Què fan les següents comandes (*curl* i *jq*)? Pista: Després de fer-li una ullada a la documentació del servei URLParse (disponible a https://urlparse.com), prova la comanda *curl* només i, a partir del resultat obtingut, dedueix el que filtra la comanda *jq* ¿Què significa que alguns dels camps del JSON mostrat valguin "null"?

curl -s "https://api.urlparse.com/v1/query?url=https://elpuig.xeill.net/Members/q2dg" | jq "{scheme,host,port,path,query}"

dII) ¿I aquestes comandes *curl* i *jq*, què fan? ¿Quina diferència hi ha amb les comandes de l'apartat anterior?

curl -s -X POST -H "Content-Type:application/json" -d '{"url":"https://elpuig.xeill.net/Members/q2dg"}' "https://api.urlparse.com/v1/query" | jq "{scheme,host,port,path,query}"

e) Podeu trobar molts més exemples d'API REST públiques a https://github.com/public-apis/public-apis , https://github.com/ripienaar/free-for-dev, https://the-api-collective.com, https://public-apis.xyz o https://m3o.com . Escull d'alguna de les llistes anteriors una que no necessiti cap mena d'autenticació i escriu un exemple d'ús al terminal.

4.-OPCIONAL a) Inicia una màquina virtual qualsevol; allà hi implementarem un servidor REST molt senzill mitjançant el mòdul "Flask" de Python, el qual aportarà la funcionalitat de servidor HTTP al nostre codi. Primer instal.la'l així: *sudo pip3 install flask* (si no tens la comanda "pip3" instal·lada, instal·la-la abans executant *apt install python3-pip*) i, tot seguit, crea un fitxer anomenat "servidor.py" amb aquest contingut:

```
from flask import Flask
app=Flask(__name__)
#Defineixo la ruta "/pepe"
@app.route("/pepe")
#Defineixo el que passarà quan el client vagi a la ruta "/pepe"
def pepe():
   return "Hola pepe \n"
#Em poso a escoltar al port 3000 a totes les IPs del meu sistema
app.run(host="0.0.0.0",port=3000)
```

NOTA: En el cas de voler retornar cadenes amb colors, recomano llegir el següent article per saber com fer-ho: http://www.lihaoyi.com/post/BuildyourownCommandLinewithANSIescapecodes.html

aII) Executa el codi anterior amb la comanda *python3 servidor.py* (procura abans que el tallafocs del sistema no interfereixi amb el port 3000!!) i, en un altre terminal, executa la comanda *curl http://127.0.0.1:3000* ¿Què veus als dos terminals? ¿Per què? ¿I si fas *curl http://127.0.0.1:3000/pepe* ?

aIII) Modifica ara el codi de servidor.py per a què quedi així (en negrita estan els afegits):

```
from flask import Flask,request
app=Flask(__name__)
@app.route("/pepe")
def pepe():
   return "Hola pepe"
@app.route("/login")
def login():
   usu=request.args.get("usuari")
   contra=request.args.get("contrasena")
   return "Login correcte per %s:%s \n" %(usu,contra)
app.run(host="0.0.0.0",port=3000)
```

aIV) Executa el codi anterior i, en un altre terminal, executa la comanda *curl "http://127.0.0.1:3000/login?usuari=manolo&contrasena=1234"* ¿Què veus?

b) Modifica ara el codi de servidor.py per a què quedi així (en negrita estan els afegits):

```
from flask import Flask,request
app=Flask(__name__)
@app.route("/pepe")
def pepe():
   return "Hola pepe"
@app.route("/login", methods=["POST"])
def login():
   usu=request.form.get("usuari")
   contra=request.form.get("contrasena")
   return "Login correcte per %s:%s \n" %(usu,contra)
app.run(host="0.0.0.0",port=3000)
```

bII) Executa el codi anterior i, en un altre terminal, executa la comanda *curl -X POST http://127.0.0.1:3000/login -d "usuari=manolo&contrasena=1234"* ¿Què veus? ¿Què passa si, en canvi, executes la comanda *curl "http://127.0.0.1:3000/login?usuari=manolo&contrasena=1234"* (i per què)?

bIII) ¿Què fa aquest script si l'executes contra el mateix codi Flask que tenies funcionant a l'apartat b)?

```
#!/bin/bash
BODY="usuari=$1&contrasena=$2"
BODY_LEN=$(echo -n ${BODY} | wc -c )
echo -ne "POST /login HTTP/1.1\r\nHost: 127.0.0.1\r\nContent-Type: application/x-www-form-urlencoded
        \r\nContent-Length: ${BODY_LEN}\r\n\r\n${BODY}" | nc -i 3 127.0.0.1 3000
```

NOTA: El paràmetre -i de *nc* és important per deixar-li temps a rebre la resposta abans de que tanqui la connexió

c) Modifica ara el codi de servidor.py per a què quedi així (en negreta estan els canvis):

```
from flask import Flask,request
app=Flask(__name__)
@app.route("/pepe")
def pepe():
   return "Hola pepe"
@app.route("/login", methods=["POST"])
def login():
   usu=request.json.get("usuari")
   contra=request.json.get("contrasena")
   return " Login correcte per %s:%d \n" %(usu,contra)
app.run(host="0.0.0.0",port=3000)
```

cII) Executa el codi anterior i, en un altre terminal, executa la comanda *curl -X POST http://127.0.0.1:3000/login -d '{"contrasena":1234,"usuari":"manolo"}' -H "Content-Type:application/json"* ¿Què veus?

NOTA: Per veure un servidor REST fet amb Flask molt més robut i complet, us podeu fixar en el codi d'exemple mostrat en aquest article: https://sysadmins.co.za/basic-restful-api-server-with-python-flask. Si voleu més informació, podeu consultar https://flask-restful.readthedocs.io/en/latest/

NOTA: D'altra banda, per veure com es pot fer un servidor Flask que utilitzi plantilles HTML i que connecti amb una base de dades SQLite per tal de consultar, modificar, afegir o eliminar dades a través de pàgines/formularis HTML, es pot consultar https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3

NOTA: Altres articles complementaris que il·lustren la utilització de Flask per implementar un servidor REST que poden ser interessants són: https://www.kite.com/blog/python/flask-restful-api-tutorial ,
https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask ,
https://medium.com/analytics-vidhya/understanding-restful-api-with-flask-59421f01f648 ,
https://www.digitalocean.com/community/tutorials/processing-incoming-request-data-in-flask ,
https://www.bogotobogo.com/python/python-REST-API-Http-Requests-for-Humans-with-Flask.php

NOTA: En el cas de voler fer servir el servidor web Apache i el llenguatge PHP per implementar un servidor REST, es pot seguir algun dels següents tutorials: https://diego.com.es/introduccion-a-rest-en-php
https://codeofaninja.com/create-simple-rest-api-in-php ,
https://github.com/crmcmullen/medium-php-api-starter

Cal saber que existeixen diversos clients REST que són gràfics i, per tant, més còmodes i útils que no pas Curl, sobre tot per desenvolupadors que han de fer proves d'APIs pròpies constantment. Alguns són:

Postman (https://www.postman.com)
Insomnia (https://insomnia.rest)
HTTPToolkit (https://httptoolkit.tech)
HoppScotch (https://github.com/hoppscotch/hoppscotch)
Kreya (https://kreya.app)
RestClient (https://addons.mozilla.org/en-US/firefox/addon/restclient)

NOTA: Un bon tutorial de les capacitats de Postman es pot trobar aquí:
https://sunnynetwork.wordpress.com/2017/03/11/lab-44-rest-api-testing-using-postman