SPRC - Laboratorul 3

Servicii Web

Responsabili: Adrian Ilie, Dorinel Filip

Obiectivul laboratorului

În cadrul acestui laborator, ne propunem construirea unui Serviciu Web **(componenta server)** potrivit pentru a fi folosit pentru managementul unei liste partajate de filme. Considerăm acest scenariu pentru un serviciu ce va fi folosit de un grup de oameni de încredere, deci nu ne vom preocupa de autentificare/autorizare, ci doar de menţinerea listei de filme.

Pregătirea pentru laborator

La acest laborator puteţi folosi **orice** limbaj de programare doriţi, alături de un framework pentru crearea unui server web.

Echipa de asistenţi va oferi suport pentru folosirea **Python 3.x**, alături de Framework-ul **Flask**.

Pe un sistem cu Debian/Ubuntu, instalarea Python3 şi Flask se poate face folosind următoarele comenzi:

```
sudo apt install python3 python3-pip
sudo pip3 install Flask
```

Un exemplu de serviciu web foarte simplu, în Flask, este următorul:

```
from flask import Flask

app = Flask(__name__)

@app.route("/ruta1", methods=[“GET”])
def hello():
    return "Hello, World!"
```

```
@app.route("/ruta2")
def salut():
    return "Salut, Lume!"
```

În mod obişnuit, aplicaţiile Flask se rulează alături de un Server Web (cum ar fi Apache sau Nginx), însă pentru laborator este suficient să folosim serverul de dezvoltare inclus în Flask. În acest caz, la aplicaţia de mai sus adăugăm o funcţie main, care să facă rularea serverului:

```
if __name__ == '__main__':
   app.run('0.0.0.0', debug=True)
```

Exemplul construit din exemplele de mai sus poate fi descărcat de aici: https://gist.githubusercontent.com/dorinelfilip/f3129e557acbcd57ad8ee2ed9c26846c/raw/42b1859da867a556e4c48b008113850b24b032fe/main.py

Pentru a rula serverul de dezvoltare, putem folosi comanda **python3 main.py**.

Rutele din Flask pot conţine şi parametrii, care vor fi oferiţi automat funcţiei care asigură ruta respectivă. În exemplul de mai jos, funcţia are un parametru de tip **int**.

```
@app.route('/salut/<int:numar>')
def salut_n(numar):
    return "Hello " + str(numar)
```

Prin intermediul modulului request, putem vedea detaliile request-ului primit. În exemplul de mai jos, preluăm payload-ul (în format JSON) al request-ului. De asemenea, răspunsul este tot un json cu codul (HTTP) 404.

```
from flask import request, jsonify, Response
@app.route('/call_me')
def call_me():
    payload = request.get_json(silent=True)
    if not payload:
        # Error handling
        return Response(status=400)

    return jsonify({'status': 'not found'}), 404
```

Alte răspunsuri, care nu sunt JSON, pot fi generate prin intermediul clasei Response (https://flask.palletsprojects.com/en/1.1.x/api/#response-objects).

Totuşi, cea mai bună sursă de informaţii este documentaţia oficială: https://flask.palletsprojects.com

Dacă doriţi să rezolvaţi laboratorul în **NodeJS**, puteţi urmări exemplul din repository-ul următor: ***https://gitlab.com/mobycloudcomputing/exempluserverexpress.git***

Cerinţă

Pentru acest laborator, se cere să se creeze un serviciu web care să permită unui grup de prieteni, care sunt de încredere, să partajeze o listă de filme.

Fiecare film va fi identificat printr-un **ID unic** (asignat de server) şi un **nume.**

Astfel, în JSON, fiecare film va avea structura de mai jos. Vom numi acesta **tipul film**.

```
{
  "id": 1,
  "nume": "Star Wars 1 - Yoda Version"
}
```

În continuare, vom presupune că nimeni nu va încerca abuzarea sistemului, deci oricine este liber, din punctul de vedere al serverului, să facă oricare din operaţiile permise.

Mai jos sunt rutele pe care trebuie să le creaţi, alături de specificaţiile lor:

- GET /movies

Întoarce, în format JSON, **o listă** cu toate filmele din sistem, fiecare film fiind un obiect de tipul film.
Această metodă va întoarce mereu codul 200.

- **POST /movies**

Primeşte ca payload un obiect ce conţine un singur câmp *nume* şi adaugă respectivul film în listă.

Un exemplu de payload ar putea fi:

```
{
  "nume": "Star Wars 1 - Yoda Version"
}
```

Această metodă va întoarce codul **201 CREATED**, în caz de succes, sau **400** (Bad Request), dacă JSON-ul primit nu este potrivit (nu este JSON sau nu conţine câmpul nume).

- **PUT /movie/{id}**

  Primeşte un payload similar metodei **POST** /movie şi actualizează numele filmului cu ID-ul dat.

  Dacă filmul nu există în baza de date, atunci serverul va răspunde cu status code 404.

  Dacă JSON-ul primit nu este potrivit (nu este JSON sau nu conţine câmpul nume) întoarce **400** (Bad Request).

- **GET /movie/{id}**

  Întoarce (în format JSON) filmul cu ID-ul dat.

  Dacă filmul nu există în baza de date, atunci serverul va răspunde cu status code 404.

- **DELETE** /movie/{id}

  Şterge filmul cu ID-ul dat şi întoarce codul 200, în caz de succes au 404 dacă filmul nu există.

- **DELETE** /reset

  Şterge întreaga bază de date. Întoarce întotdeauna, ca răspuns, 200 OK.

  Observaţie: Pentru ca testele să treacă şi la rulări repetate, ID-urile asignate trebuie să se reseteze şi ele (la 1), când faceţi /reset.

Testare şi notare

Pentru a genera, mai uşor decât în laboratorul 1, request-uri către soluţia voastră, puteţi folosi utilitarul Postman (https://www.getpostman.com).

Pentru a se considera validă, o implementare trebuie să treacă toate testele din colecţia următoare: https://static.dfilip.xyz/sprc.json. Aceasta trebuie importată în Postman.

Observaţie: Implicit, {{BASE_URL}} din teste este setat la **localhost:5000**. Dacă este diferit, în puteţi schimba din scriptul **Pre-req.** al primului request din colecţie.

Pentru a expune API-ul vostru către asistent vă recomandăm folosirea https://ngrok.com.