

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 menținerea listei de filme.

Pregătirea pentru laborator

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

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

Pe un sistem cu Debian/Ubuntu, instalarea Python3 și 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 obișnuit, aplicațiile 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 aplicația de mai sus adăugăm o funcție 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/dorinel Filip/f3129e557acbcd57ad8ee2ed9c26846c/raw/42b1859da867a556e4c48b008113850b24b032fe/main.py>

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

Rutele din Flask pot conține și parametrii, care vor fi oferiți automat funcției care asigură ruta respectivă. În exemplul de mai jos, funcția 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': 'ok'}), 201
```

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

Totuși, cea mai bună sursă de informații este documentația oficială:

<https://flask.palletsprojects.com>

Dacă doriți să rezolvați laboratorul în **NodeJS**, puteți urmări exemplul din repository-ul următor: <https://gitlab.com/mobycloudcomputing/exempluserverexpress.git>

Cerință

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) și 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 operațiile permise.

Mai jos sunt rutele pe care trebuie să le creați, alături de specificațiile 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

Primește ca payload un obiect ce conține un singur câmp *nume* și 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 conține câmpul nume).

Pentru compatibilitate cu checker-ul, ID-urile vor fi numerice, începând cu 1. Bineînțeles, obiectele sunt unice, astfel la ștergerea unui film ID-urile celorlalte **NU** trebuie să se modifice.

- PUT /movie/{id}

Primește un payload similar metodei **POST** /movie și 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 conține 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}

Șterge filmul cu ID-ul dat și întoarce codul 200, în caz de succes sau 404 dacă filmul nu există.

La ștergerea unui film ID-urile celorlalte **NU** trebuie să se modifice. Ar fi greșit ca ID-ul unic al unui element să se modifice.

- DELETE /reset

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

Observație: Pentru ca testele să treacă și la rulări repetate, ID-urile asignare trebuie să se reseteze și ele (la 1), când faci /reset.

Testare și notare

Pentru a genera, mai ușor decât în laboratorul 1, request-uri către soluția voastră, puteți folosi utilitarul Postman (<https://www.getpostman.com>).

Pentru a se considera validă, o implementare trebuie să treacă toate testele din colecția următoare: https://static.dfilip.xyz/sprc_lab3.json. Aceasta trebuie importată în Postman.

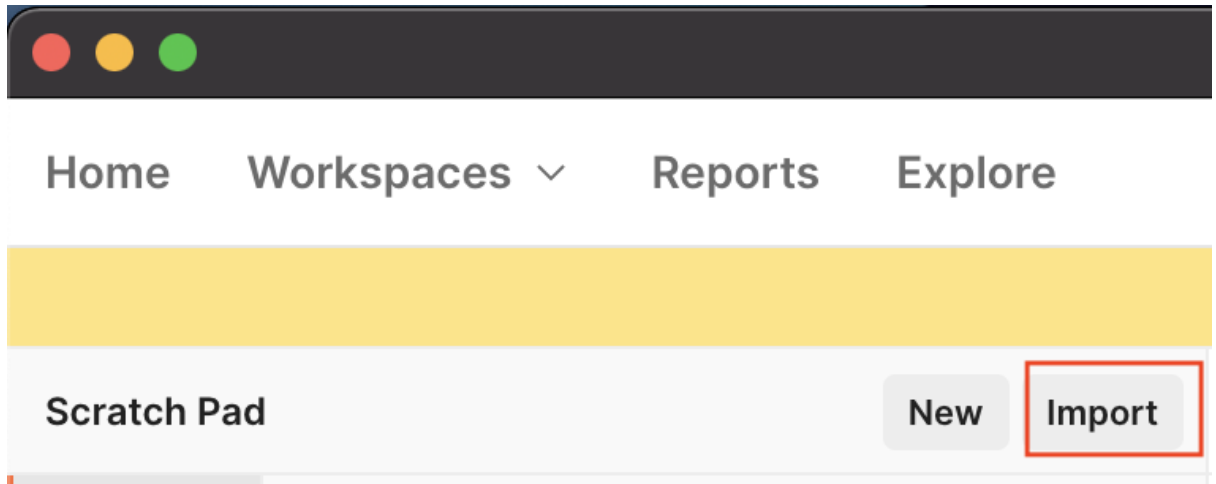
Observație: Implicit, `{{BASE_URL}}` din teste este setat la **localhost:5000**. Dacă este diferit, în puteți schimba din scriptul **Pre-req.** al primului request din colecție.

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

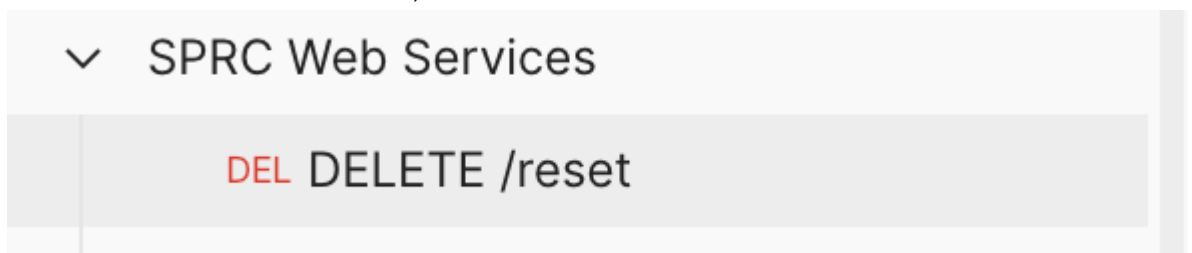
FAQ:

Cum rulez testele din POSTMAN?

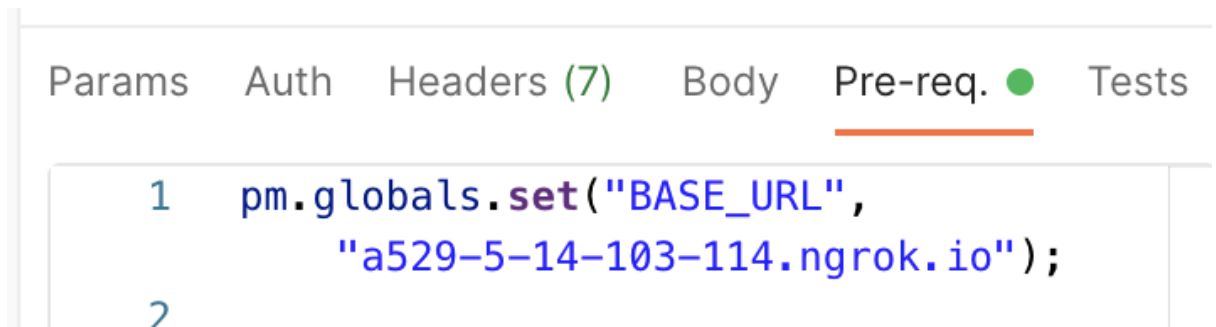
1. Adaugi colecția https://static.dfilip.xyz/sprc_lab3.json în PostMan



2. Intri pe primul request din colecția nou creată



3. La din Pre Req Script modifici valoarea Base URL



Valoarea de aici poate să fie adresa pe care te anunță Flask că rulează (ex. **localhost:5000**)

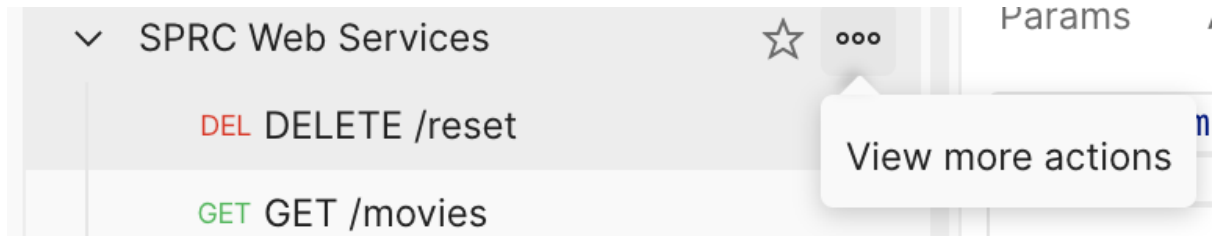
De exemplu, dacă Flask zicea asta:

```
* Serving Flask app 'main' (lazy loading)  
* Environment: production  
  WARNING: This is a development server. Do not use it in a production deployment.  
  Use a production WSGI server instead.  
* Debug mode: on  
* Running on all addresses.  
  WARNING: This is a development server. Do not use it in a production deployment.  
* Running on http://192.168.8.111:7070/ (Press CTRL+C to quit)  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 116-243-024
```

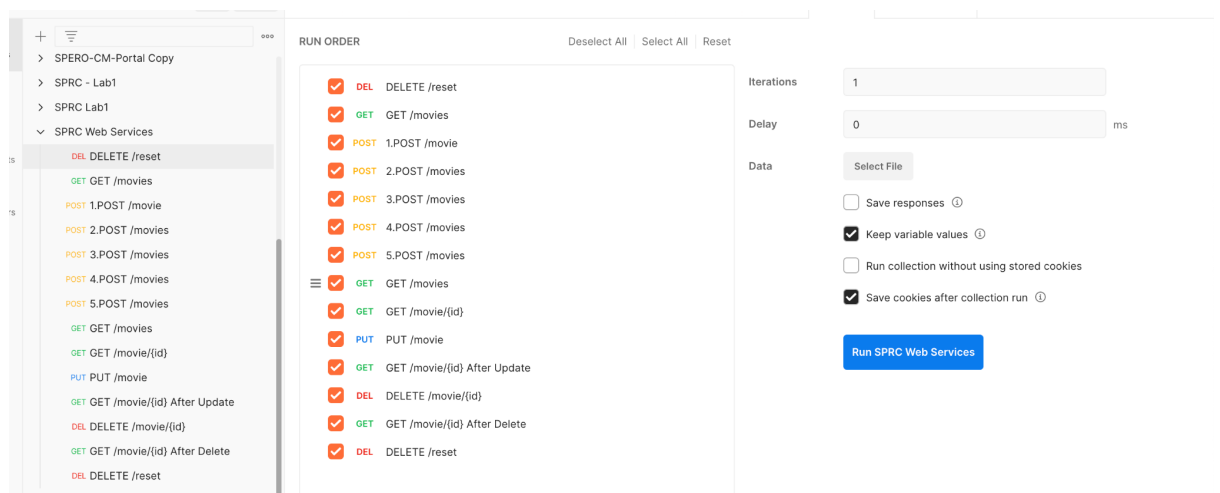
Atunci pot pune variabila la **192.168.8.111:7878**



4. Dai Run Collection și vezi ce teste îți trec
- Click pe More actions din dreptul colecției



- Click pe Run SPRC Web Services



Ordinea testelor ar trebui să fie implicit ok, dar nu există această garanție cu toate variantele de PostMan. Verificați să fie ca în imagine dacă apar probleme dubioase.

Q: Nu merge să fac request către localhost din Postman

Eroare de tipul „Blocked IP address”?

Dacă da, NU folosi Postman din browser. Descarcă aplicația standalone de aici:
<https://www.postman.com/downloads/>

Q: Ce comandă dădeam pentru tunelul cu ngrok?

ngrok http PORT

Ex. :ngrok http 5000