Tema 1 - Le Stats Sportif

- **Deadline soft:** ~~7 aprilie 2025~~ **9 aprilie 2025**, ora 23:55. Veți primi o depunctare de 10% din punctajul maxim al temei pentru fiecare zi de întârziere, până la maxim 7 zile, adică până pe ~~14 aprilie 2025~~ 16 aprilie 2025, ora 23:55.
- **Deadline hard:** ~~14 aprilie 2025~~ **16 aprilie 2025**, ora 23:55.
- **Responsabili:** Andrei Ouatu [mailto:andreicatalin.ouatu@gmail.com], Eduard Stăniloiu [mailto:eduard.staniloiu@cs.pub.ro], Giorgiana Vlăsceanu [mailto:giorgiana.vlasceanu@gmail.com]
- **Autori:** Eduard Stăniloiu [mailto:eduard.staniloiu@cs.pub.ro], Giorgiana Vlăsceanu [mailto:giorgiana.vlasceanu@gmail.com]

- Dată publicare: 24 martie
- Dată actualizare deadline: 6 aprilie

Scopul temei

- Utilizarea eficientă a elementelor de sincronizare studiate la laborator
- Implementarea unei aplicații concurente utilizând o problemă clasică (client - server)
- Aprofundarea anumitor elemente din Python (clase, elemente de sintaxă, threaduri, sincronizare, precum și folosirea modulelor Python pentru lucrul cu threaduri)

Enunț

În cadrul acestei teme veți avea de implementat un server python care va gestiona o serie de requesturi plecând de la un set de date în format *csv* (comma separated values). Serverul va oferi statistici pe baza datelor din csv.

Setul de date

Setul de date [https://catalog.data.gov/dataset/nutrition-physical-activity-and-obesity-behavioral-risk-factor-surveillance-system] conține informații despre nutriție, activitatea fizică și obezitate în Statele Unite ale Americii în perioada 2011 - 2022. Datele au fost colectate de către U.S. Department of Health & Human Services. Informațiile sunt colectate per stat american (ex. California, Utah, New York) și răspund următorului **set de întrebări**:

- 'Percent of adults who engage in no leisure-time physical activity'
- 'Percent of adults aged 18 years and older who have obesity'
- 'Percent of adults aged 18 years and older who have an overweight classification'
- 'Percent of adults who achieve at least 300 minutes a week of moderate-intensity aerobic physical activity or 150 minutes a week of vigorous-intensity aerobic activity (or an equivalent combination)'
- 'Percent of adults who achieve at least 150 minutes a week of moderate-intensity aerobic physical activity or 75 minutes a week of vigorous-intensity aerobic physical activity and engage in muscle-strengthening activities on 2 or more days a week'
- 'Percent of adults who achieve at least 150 minutes a week of moderate-intensity aerobic physical activity or 75 minutes a week of vigorous-intensity aerobic activity (or an equivalent combination)'
- 'Percent of adults who engage in muscle-strengthening activities on 2 or more days a week'
- 'Percent of adults who report consuming fruit less than one time daily'
- 'Percent of adults who report consuming vegetables less than one time daily'

Valorile pe care le veți folosi în calculul diverselor statistici la care răspunde aplicația voastră se găsesc în coloana **Data_Value**.

Detalii de implementare

Aplicația server pe care o dezvoltați este una multi-threaded. Atunci când serverul este pornit, trebuie să încărcați fișierul csv și să extrageți informațiile din el a.î. să puteți calcula statisticile cerute la nivel de request.

Întrucât procesarea datelor din csv poate dura mai mult timp, modelul implementat de către server va fi următorul: * un endpoit (ex. '/api/states_mean') care primește requestul și va întoarce clientului un **job_id** (ex. "job_id_1", "job_id_2", ..., "job_id_n") * endpointul '/api/get_results/job_id' care va verifica dacă job_id-ul este valid, rezultatul calculului este gata sau nu și va returna un răspuns corespunzător (detalii mai jos)

Mecanica unui request

Asociază un job_id requestului, pune jobul (closure care încalsulează unitatea de lucru) într-o coadă de joburi care este procesată de către un **Thread pool**, incrementează job_id-ul intern și returnează clientului job_id-ul asociat.

Un thread va prelua un job din coada de joburi, va efectua operația asociată (ceea ce a fost capturat de către closure) și va scrie rezultatul calculului într-un fișier cu numele job_id-ului în directorul **results/**.

Prin scrierea rezultatelor pe disc, în directorul **results/**, simulăm interacțiunea cu o bază de date (poor man's db).

Nu rețineți rezultatul într-o structură de date în memorie, este abordarea greșită. Priviți problema din unghiul: dacă primesc 2k de requesturi de tipul fă-mi calculul x, voi rămâne fără memorie RAM înainte de a oferi un rezultat.

Dacă vreți să folosiți o bază de date, go for it. Checkerul nu va verifica asta.

Requesturile pe care trebuie să le implementați sunt

/api/states_mean

Primește o întrebare (din **setul de întrebări** de mai sus) și calculează media valorilor înregistrate (**Data_Value**) din intervalul total de timp (2011 - 2022) pentru fiecare stat, și sortează crescător după medie.

/api/state_mean

Primește o întrebare (din **setul de întrebări** de mai sus) și un stat, și calculează media valorilor înregistrate (**Data_Value**) din intervalul total de timp (2011 - 2022).

/api/best5

Primește o întrebare (din **setul de întrebări** de mai sus) și calculează media valorilor înregistrate (**Data_Value**) din intervalul total de timp (2011 - 2022) și întoarce primele 5 state.

/api/worst5

Primește o întrebare (din **setul de întrebări** de mai sus) și calculează media valorilor înregistrate (**Data_Value**) din intervalul total de timp (2011 - 2022) și întoarce ultimele 5 state.

În funcție de întrebare, primele state pot să aibă fie cel mai mic sau cel mai mare scor. De exemplu, pentru întrebarea: "Percent of adults who engage in no leisure-time physical activity", primele state (best) vor avea scorurile cele mai mici, iar worst vor avea scorurile cele mai mari. Pentru întrebarea: "Percent of adults who engage in muscle-strengthening activities on 2 or more days a week", primele state (best) vor avea scorurile cele mai mari, iar worst vor avea scorurile cele mai mici.

/api/global_mean

Primește o întrebare (din **setul de întrebări** de mai sus) și calculează media valorilor înregistrate (**Data_Value**) din intervalul total de timp (2011 - 2022) din întregul set de date.

/api/diff_from_mean

Primește o întrebare (din **setul de întrebări** de mai sus) și calculează diferența dintre global_mean și state_mean pentru toate statele.

/api/state_diff_from_mean

Primește o întrebare (din **setul de întrebări** de mai sus) și un stat, și calculează diferența dintre global_mean și state_mean pentru statul respectiv.

/api/mean_by_category

Primește o întrebare (din **setul de întrebări** de mai sus) și calculează valoarea medie pentru fiecare segment (**Stratification1**) din categoriile (**StratificationCategory1**) fiecărui stat.

/api/state_mean_by_category

Primește o întrebare (din **setul de întrebări** de mai sus) și un stat, și calculează valoarea medie pentru fiecare segment (**Stratification1**) din categoriile (**StratificationCategory1**).

/api/graceful_shutdown

Răspunde la un apel de tipul GET și va duce la notificarea Thread Poolului despre încheierea procesării. Scopul acesteia este de a închide aplicația într-un mod graceful: nu se mai acceptă requesturi noi, se termină de procesat requesturile înregistrate până în acel moment (drain mode) și apoi aplicația poate fi oprită. Endpointul **graceful_shutdown** va întoarce un JSON cu statusul **running** dacă încă sunt requesturi de procesat în coadă, sau cu statusul **done** atunci când coada este goală:

```
{
  "status: "running"
}
```

În cazul în care se mai fac requesturi de tip procesare, de exemplu **states_mean**, se va întoarce un JSON:

```
{
  "status: "error",
  "reason": "shutting down"
}
```

Requesturile de tipul **get_results**, **jobs**, **num_jobs** se acceptă și după **graceful_shutdown**.

/api/jobs

Răspunde la un apel de tipul GET cu un JSON care conține toate JOB_ID-urile de până la acel moment și statusul lor. De exemplu:

```
{
  "status": "done"
  "data": [
    { "job_id_1": "done"},
    { "job_id_2": "running"},
    { "job_id_3": "running"}
  ]
}
```

/api/num_jobs

Răspunde la un apel de tipul GET cu numărul joburilor rămase de procesat. După un **/api/graceful_shutdown** și o perioadă de timp, aceasta ar trebui să întoarcă valoarea 0, semnalând astfel că serverul flask poate fi oprit.

/api/get_results/<job_id>

Răspunde la un apel de tipul GET (job_id-ul este parte din URL). Acesta verifică dacă job_id-ul primit este valid și răspunde cu un JSON corespunzător, după cum urmează:

1. JOB_ID-ul este invalid

```
{
  "status": "error",
  "reason": "Invalid job_id"
}
```

2. JOB_ID-ul este valid, dar rezultatul procesării nu este gata

```
{
  "status": "running",
}
```

3. JOB_ID-ul este valid și rezultatul procesării este gata

```
{
  "status": "done",
  "data": <JSON_REZULTAT_PROCESARE>
}
```

Server

Implementarea serverului se face folosind framework-ul **flask** și va extinde scheletul de cod oferit. Mai multe detalii despre Flask găsiți mai jos. Deasemeni, un tutorial extensiv (pe care vi-l recomandăm) este The flask mega tutorial [https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world].

Python Flask este un micro-framework web open-source care permite dezvoltatorilor să creeze aplicații web ușor și rapid, folosind limbajul de programare Python. Flask este minimalist și flexibil, oferind un set de instrumente de bază pentru crearea unei aplicații web, cum ar fi rutele URL, gestionarea cererilor și a sesiunilor, șablonarea și gestionarea cookie-urilor. Cu Flask, dezvoltatorii pot construi rapid API-uri sau aplicații web de dimensiuni mici și medii.

Instalare și activarea mediului de lucru

Pentru a instala Flask, creați-vă un mediu virtual (pentru a nu instala pachete global, pe sistem) folosind comanda

```
$ python -m venv venv
```

Activați mediul virtual

```
$ source venv/bin/activate
```

Și instalați pachetele din fișierul **requirements.txt**

```
$ python -m pip install -r requirements.txt
```

Pașii de creare a mediului virtual și de instalare a pachetelor se regăsesc în fișierul Makefile. Astfel, pentru a vă crea spațiul de lucru, rulați următoarele comenzi în interpretorul vostru de comenzi (verificat în `bash` și `zsh`)

```
make create_venv
source venv/bin/activate
make install
```

Quickstart

O rută în cadrul unei aplicații web, cum ar fi în Flask, reprezintă un URL (Uniform Resource Locator) specific către care aplicația web va răspunde cu un anumit conținut sau funcționalitate. Atunci când un client (de obicei un browser web) face o cerere către serverul web care găzduiește aplicația Flask, ruta determină ce cod va fi executat și ce răspuns va fi returnat clientului. În Flask, rutele sunt definite folosind decoratori care leagă funcții Python de URL-uri specifice, permițând astfel aplicației să răspundă în mod dinamic la cereri (requesturi).

În Flask, puteți defini o rută care răspunde la un apel de tip **GET** folosind decoratorul **@app.route()** și specificând metoda *HTTP* (**methods=['GET']**). Pentru a răspunde la un apel de tipul **POST** (apel folosit pentru a trimite date de către un client către server) folosim același decorator și specificăm **methods=['POST']**. De exemplu:

```
from flask import request

@app.route('/', methods=['GET'])
def index():
    return 'Aceasta este o rută care răspunde la un apel de tip GET'

@app.route('/post', methods=['POST'])
def post_route():
    data = request.json  # Se obțin datele JSON trimise prin POST
    return 'Aceasta este o rută care răspunde la un apel de tip POST'
```

În cazul API-urilor este un best practice ca datele returnate să fie în format JSON, pentru a fi ușor de prelucrat de către alte servicii în mod programatic. Pentru a returna un obiect JSON în Flask, vom folosi helperul **jsonify()** ca în exemplul de mai jos:

```
from flask import request, jsonify

@webserver.route('/api/post_endpoint', methods=['POST'])
def post_endpoint():
    if request.method == 'POST':
        # Presupunem că metoda conține date JSON
        data = request.json
        print(f"got data in post {data}")

        # Procesăm datele primite
        # Pentru exemplu, vom returna datele primite
        response = {"message": "Received data successfully", "data": data}
        return jsonify(response)
    else:
        # Nu acceptăm o altă metodă
        return jsonify({"error": "Method not allowed"}), 405
```

Structura input-ului și a output-ului

Interacțiunea cu serverul se va face pe bază de mesaje JSON, după cum este descris mai jos. Vă recomandăm să vă uitați în suita de teste, în directoarele input și output pentru a vedea informațiile mult mai detaliat.

Input

Un input pentru un request care primește doar o întrebare în următorul format:

```
{
  "question": "Percent of adults aged 18 years and older who have an overweight classification"
}
```

Unul care așteaptă o întrebare și un stat are următorul format:

```
{
  "question": "Percent of adults who engage in no leisure-time physical activity",
  "state": "South Carolina"
}
```

Output

Un răspuns JSON va avea mereu structura:

```
{
  "status": "done",
  "data": <JSON_REZULTAT_PROCESARE>
}
```

JSON_REZULTAT_PROCESARE este un obiect JSON așa cum se regăsește în directorul output, pentru fiecare endpoint din directorul tests.

Testare

Testarea se va realiza folosind atât unitteste, cât și teste funcționale.

Rularea testelor

Pentru a rula testele, folosiți fișierul `Makefile`. Într-un shell 1) activați mediul virtual și 2) porniți serverul

```
source venv/bin/activate
make run_server
```

Într-un alt shell 1) activați mediul virtual și 2) porniți checkerul

```
source venv/bin/activate
make run_tests
```

Comenzile de mai jos sunt valabile pentru **Linux**. Dacă dezvoltați tema pe alt sistem de operare, adaptați comenzile pentru sistemul vostru (de regulă, diferă foarte puțin calea către scriptul de activare al mediului virtual). Trebuie să vă asigurați că ați activat mediul virtual înainte de a rula comenzile din make.

```
source venv/bin/activate
```

Dacă nu ați activat mediul virtual, `make` vă va arunca următoarea eroare (linia, ex 8, poate să difere).

```
Makefile:8: *** "You must activate your virtual environment. Exiting...".  Stop.
```

Unittesting

Pentru testarea funcțiilor din **server** veți folosi modulul de unittesting [https://docs.python.org/3/library/unittest.html] al limbajului Python.

Pentru a defini un set de unitteste trebuie să vă definiți o clasă care moștenește clasa `unittest.TestCase`

demo_unittest.py

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')
```

Pentru a defini un test, numele metodei trebuie să înceapă cu prefixul `test_`, așa cum puteți observa în exemplul de mai sus: `test_upper`. Verificările din corpul metodei se fac folosind metodele `assert*`, în exemplul de mai sus a fost folosită metoda `assertEqual`. O listă completă a metodelor de verificare disponibile este prezentată în documentație [https://docs.python.org/3/library/unittest.html#assert-methods].

Pentru a rula testele, folosim subcomanda unittest:

```
$ python3 -m unittest demo_unittest.py
$ # puteti folosi optiunea -v pentru mai multe detalii
$ python3 -m unittest -v demo_unittest.py
```

Pentru a testa comportamentul definiți în fișierul `unittests/TestWebserver.py` o clasă de testare numită `TestWebserver`. Clasa `TestWebserver` va testa funcționalitatea tuturor rutelor definite de voi. Dacă definiți alte metode, va trebui să adăugați teste și pentru acestea.

Vă recomandăm să folosiți metoda setUp [https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp] pentru a inițializa o instanță a clasei testate și orice altceva ce vă ajută în testarea codului. Un exemplu de utilizare a metodei `setUp` este disponibil în documentație [https://docs.python.org/3/library/unittest.html#organizing-test-code].

În arhivă aveți un exemplu, dar nu ceva fix, doar structura trebuie respectată. De exemplu, puteți adăuga fișiere de referință în directorul `unittests` sau ce vă este necesar pentru a vă rula testele.

Checkerul testează end-to-end: atât răspunsul, cât și faptul că serverul este în picioare și api-ul este cel așteptat.

Scopul unittestelor este să validați că implementarea calculelor este ok. De ex, pentru un csv de 2 linii, dat de voi ca input, funcția state_mean întoarce valoarea X (unde X l-ați calculat voi în alt mod și știți că este răspunsul corect).

Recomandarea noastră este să vă scrieți funcțiile care implemetează calculele în așa fel încât să puteți "injecta" input a.î. să puteți valida outputul. Vă recomandăm clipul acesta de pe YT despre dependency injection [https://www.youtube.com/watch?v=J1f5b4vcxCQ].

Logging

Vrem să utilizăm fișiere de logging în aplicațiile pe care le dezvoltăm pentru a putea urmări flowul acestora a.î. să ne ajute în procesul de debug.

Folosind modulul de logging [https://docs.python.org/3/library/logging.html], trebuie să implementați un fișier de log, numit “webserver.log”, în care veți urmări comportamentul serverului.

În fișierul de log veți nota, folosind nivelul `info()`, toate intrările și ieșirile în/din rutele implementate. În cazul metodelor care au parametrii de intrare, informația afișată la intrarea în funcție va afișa și valorile parametrilor. Fișierul va fi implementat folosind RotatingFileHandler [https://docs.python.org/3/library/logging.handlers.html#logging.handlers.RotatingFileHandler]: astfel se poate specifica o dimensiune maximă a fișierului de log și un număr maxim de copii istorice. RotatingFileHandler ne permite să ținem un istoric al logurilor, fișierele fiind stocate sub forma “file.log”, “file.log.1”, “file.log.2”, … “file.log.max”.

Vă încurajăm să folosiți fișierul de log și pentru a înregistra erori [https://docs.python.org/3/library/logging.html#logging.Logger.error] detectate.

În mod implicit, timestamp-ul logurilor folosește timpul mașinii pe care rulează aplicația (local time). Acest lucru nu este de dorit în practică deoarece nu putem compara loguri de pe mașini aflate în zone geografice diferite. Din acest motiv, timestampul este ținut în format UTC/GMT. Asigurați-vă că folosiți gmtime, și nu localtime. Pentru aceasta trebuie să folosiți metoda formatTime [https://docs.python.org/3/library/logging.html#logging.Formatter.formatTime].

O descriere completă a cum puteți utiliza modulul de logging este prezentă în categoria HOWTO [https://docs.python.org/3/howto/logging.html] a documentației.

Precizări biblioteci

În fișierul **requirements.txt** aveți specificate bibliotecile pe care le puteți folosi, pe lângă cele standard. Nu se vor putea folosi alte biblioteci.

Precizări încărcare

Arhiva temei va fi încărcată pe moodle [https://curs.upb.ro/2024/mod/assign/view.php?id=115677]

Arhiva (fișier .zip) trebuie să conțină:

- fișierele temei și alte fișiere `.py` folosite în dezvoltare
- `README`
- fișierul `git-log` (îl obțineți rulând comanda `git log > git-log`)
- un exemplu de conținut al arhivei este mai jos

```
api_server.py
app/
app/routes.py
app/task_runner.py
app/data_ingestor.py
app/__init__.py
README
unittests/
unittests/mytests.py
git-log
```

Repository-ul pe care îl folosiți în procesul de implementare este necesar să fie privat.

Pentru a documenta realizarea temei, vă recomandăm să folosiți template-ul de aici [https://gitlab.cs.pub.ro/asc/asc-public/-/blob/master/assignments/README.example.md]

Punctare

Tema va fi verificată automat, folosind infrastructura de testare, pe baza unor teste definite în directorul `tests`.

Tema se va implementa **Python>=3.7**.

Notarea va consta în 80 pct acordate egale între testele funcționale, 10 pct acordate pentru unitteste și 10 pct acordate pentru fișierul de logging. Depunctări posibile sunt:

- folosirea incorectă a variabilelor de sincronizare (ex: lock care nu protejează toate accesele la o variabilă partajată, notificări care se pot pierde) (-2 pct)
- prezența print-urilor de debug (maxim -10 pct în funcție de gravitate)
- folosirea lock-urilor globale (-10 pct)
- folosirea variabilelor globale/statice (-5 pct)
  - Variabilele statice pot fi folosite doar pentru constante
- folosirea inutilă a variabilelor de sincronizare (ex: se protejează operații care sunt deja thread-safe) (-5 pct)
- alte ineficiențe (ex: creare obiecte inutile, alocare obiecte mai mari decât e necesar, etc.) (-5 pct)
- lipsa organizării codului, implementare încâlcită și nemodulară, cod duplicat, funcții foarte lungi (între -1pct și -5 pct în funcție de gravitate)
- cod înghesuit/ilizibil, inconsistența stilului - vedeți secțiunea Pylint
  - pentru code-style recomandăm ghidul oficial PEP-8 [https://www.python.org/dev/peps/pep-0008/]
- cod comentat/nefolosit (-1 pct)
- lipsa comentariilor utile din cod (-5 pct)
- fișier README sumar (până la -5 pct)
- nerespectarea formatului .zip al arhivei (-2 pct)
- lipsa fișierului `git-log` (-10 pct)
- alte situații nespecificate, dar considerate inadecvate având în vedere obiectivele temei; în special situațiile de modificare a interfeței oferite

Temele vor fi testate împotriva plagiatului. Orice tentativă de copiere va fi depunctată conform regulamentului. Rezultatele notării automate este orientativă și poate fi afectată de corectarea manuală.

Pylint

Vom testa sursele voastre cu pylint [https://www.pylint.org/] configurat conform fișierului **`pylintrc`** din cadrul repo-ului dedicat temei. Atenție, rulăm pylint doar pe modulele completate și adăugate de voi, nu și pe cele ale testerului.

Deoarece apar diferențe de scor între versiuni diferite de pylint, vom testa temele doar cu ultima versiune [https://www.pylint.org/#install]. Vă recomandăm să o folosiți și voi tot pe aceasta.

Vom face depunctări de până la 10pct dacă verificarea făcută cu pylint vă dă un scor mai mic de 8.

Observații

- Pot exista depunctări mai mari decât este specificat în secțiunea Notare pentru implementări care nu respectă obiectivele temei și pentru situatii care nu sunt acoperite în mod automat de către sistemul de testare
- Implementarea și folosirea metodelor oferite în schelet este obligatorie
- Puteți adăuga variabile/metode/clase etc.
- Bug-urile de sincronizare, prin natura lor sunt nedeterministe; o temă care conține astfel de bug-uri poate obține punctaje diferite la rulări succesive; în acest caz punctajul temei va fi cel dat de tester în momentul corectării
- Recomandăm testarea temei în cât mai multe situații de load al sistemului și pe cât mai multe sisteme pentru a descoperi bug-urile de sincronizare

Resurse necesare realizării temei

Pentru a clona repo-ul [https://gitlab.cs.pub.ro/asc/asc-public] și a accesa resursele temei 1:

```
student@asc:~$ git clone https://gitlab.cs.pub.ro/asc/asc-public.git
student@asc:~$ cd asc/assignments
student@asc:~/assignments$ cd 1-le_stats_sportif
```

Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți forumul temei [https://curs.upb.ro/2024/mod/forum/view.php?id=115669].

Orice intrebare e recomandat să conțină o descriere cât mai clară a eventualei probleme. Întrebări de forma: “Nu merge X. De ce?” fără o descriere mai amănunțită vor primi un răspuns mai greu.

ATENȚIE să nu postați imagini cu părți din soluția voastră pe forumul pus la dispoziție sau orice alt canal public de comunicație. Dacă veți face acest lucru, vă asumați răspunderea dacă veți primi copiat pe temă.

asc/teme/tema1.txt · Last modified: 2025/04/06 09:17 by giorgiana.vlasceanu