Project Stage II

În ceea ce privește progresul până în momentul de față, am implementat o parte din logica de business in spring boot, care ne permite să interogăm măsurătorile stocate în baza de date, precum și filtrele asociate fiecărei măsurători. De asemenea, avem posibilitatea de a obține măsurătorile efectuate într-o perioadă de timp determinată, cu posibilitatea de a selecta una sau mai multe filtre, iar în viitor urmează să adaug și opțiunea de filtrare pe intervale de timp, filtrarea după tag-uri sau crearea unei eșantionări pentru perioade de timp extinse cu opțiunea de alegere a granularității.

Baza de date, care rulează într-un container Docker, este monitorizată de un serviciu Telegraf, ce preia metricile legate de performanta bazei de date și le trimite către un microserviciu Grafana. În paralel, microserviciul de logică de business trimite metrici prin Prometheus către Grafana, pentru a putea monitoriza și funcționalitatea API-ului.

În ceea ce privește serviciul de autentificare și limitare a accesului la API, după o analiză asupra unor tehnologii deja existente am identificat următoarele variante:

1. Bucket4j: Acesta permite implementarea unui sistem de limitare a ratei pe baza unui "bucket" (reservor) de tokenuri, unde fiecare cerere consumă un token, iar dacă tokenurile sunt epuizate, cererile vor fi respinse. Bucket4j poate fi utilizat pentru a implementa rate limiting la nivel de utilizator sau IP.
   Ce aduce Bucket4j: Oferă o scalabilitate mai bună, în special pentru aplicațiile care au o utilizare mai intensă a resurselor. Poate fi implementat cu atât într-un mediu in-memory sau distribuit.

2. Resilience4j: Acesta este un toolkit de librării care permite implementarea de pattern-uri de reziliență, inclusiv limitarea ratei (rate limiting). Resilience4j poate fi folosit pentru a crea politici de limitare a cererilor în funcție de numărul de cereri permise pe o unitate de timp. De asemenea, poate fi integrat ușor cu Spring Boot și poate oferi un control detaliat asupra comportamentului API-ului în condiții de trafic crescut.
   Ce aduce Resilience4j: În plus față de rate limiting, oferă și funcționalități pentru gestionarea fallback-urilor, retry și circuit breaker.

3. Spring Rate Limiter: Această metodă permite limitarea accesului API-ului folosind un sistem simplu de rate limiting bazat pe cereri per utilizator, per IP, sau pe alte criterii configurabile. Este o metodă mai ușor de implementat și integrat cu Spring

Security, ideală pentru aplicațiile care nu necesită o complexitate ridicată în ceea ce privește configurarea rate limiting-ului.
Ce aduce Spring Rate Limiter: O abordare simplă, rapidă direct în Spring, ideală pentru aplicații mici.

Autentificarea se va face pe baza unui token. În viitor, se va putea conecta la sistemul de autentificare al facultății, în cazul unui nou deploy. Acest microserviciu va fi utilizat pentru a efectua verificări de permisiuni, având rolul de a limita accesul utilizatorilor la anumite operații, în funcție de poziția lor: de exemplu, studenții vor avea un număr redus de operații disponibile, iar profesorii vor avea acces la un set mai larg de operațiuni.

O problemă cu care ma confrunt este generarea și stocarea seturilor de date în mediul local pentru testare. În prezent, datele sunt generate folosind scripturi Python, iar acestea sunt introduse în InfluxDB pentru testare. Un dezavantaj al acestui mod de generare este comportamentul predictibil al datelor, care nu reflectă complet scenariile de utilizare reală si totodată mărimea setului de date.

În ciuda acestui lucru, InfluxDB oferă instrumente care pot fi utilizate pentru a agrega date provenite din API-uri externe dedicate. Un exemplu relevant este API-ul OpenWeather, care furnizează informații meteo din diverse orașe. Aceste date ne vor permite să testăm filtrarea pe baza tagurilor de orașe la o scară mai largă și să evaluăm performanța aplicației pe seturi de date reale, cu un comportament mai natural și imprevizibil.

Totuși, pentru a asigura o diversitate mai mare a datelor și pentru a îmbunătăți procesul de import, trebuie să identificăm și alte surse și metode de integrare a informațiilor în aplicație. De exemplu, putem explora API-uri din alte domenii (economie, trafic, sănătate etc).

Un alt element esențial în dezvoltarea acestui API este designul său, care trebuie să fie atât intuitiv, cât și flexibil, pentru a răspunde unui spectru larg de cerințe, folosind totodată un număr restrâns de endpointuri. Este crucial ca API-ul să fie ușor de înțeles și de utilizat de către dezvoltatori, astfel încât să nu necesite o documentație complexă pentru utilizare.

În concluzie, obiectivul este de a acoperi restul funcționalităților de bază propuse în prezentarea anterioară până la următoarea sesiune de prezentare. Acum că mare parte din set-up-ul inițial a fost finalizat, următorul pas este implementarea funcționalităților rămase și crearea unui sistemul de gestionare a erorilor bine definit. Este important ca utilizatorii să poată înțelege rapid și clar ce probleme au apărut în cadrul apelurilor API și să poată lua măsurile necesare pentru a le rezolva.