Optimizarea Api-urilor pentru date IoT folosind Spring Boot si InfluxDB

Autor:

Sofia Dragoș

Profesor coordonator:

Prof. Ciprian Mihai Dobre

Cuprins

1.Introducere

Un Open API (Application Programming Interface) reprezintă un set de reguli și protocoale care permit diferitelor aplicații software să comunice între ele într-un mod standardizat, fiind accesibil în mod public pentru dezvoltatori și organizații terțe [1]. Aceste API-uri sunt esențiale pentru ecosistemele moderne de software, deoarece permit integrarea rapidă între servicii, dezvoltarea mai eficientă a aplicațiilor și inovarea prin reutilizarea componentelor existente [2]. Ele permit dezvoltarea de aplicații cu o cuplare slabă, ceea ce înseamnă că serviciile software pot evolua independent unele de altele, fără a afecta funcționalitatea întregului sistem [3]. Aceasta este o caracteristică esențială în arhitecturile moderne bazate pe microservicii, unde fiecare componentă poate fi dezvoltată, testată și scalată separat [4]. Open API-urile sunt preferate în dezvoltarea aplicațiilor care nu au aceiași contribuitori, oferind un mod documentat de utilizare care standardizează interacțiunile dintre servicii. Astfel, acestea facilitează colaborarea între echipe și eficientizează reutilizarea codului în diverse interfețe, reducând redundanța și accelerând procesul de dezvoltare software.

La momentul actual, în cadrul campusului Facultății Naționale de Știință și Tehnologie Politehnică București, există o multitudine de senzori care furnizează date pentru monitorizarea și optimizarea mediului urban. Acești senzori includ people counters, senzori de temperatură și umiditate, dispozitive care colectează informații despre condițiile meteo, traficul și chiar gestionarea deșeurilor. Datele colectate au un potențial mare de utilizare în diverse aplicații, de la optimizarea resurselor energetice și reducerea amprentei de carbon până la cercetare științifică și dezvoltare urbană inteligentă.

Un mod uniform și standardizat de acces la aceste date, cum ar fi printr-un Open API, ar putea facilita integrarea acestora în diverse platforme și aplicații. Acest lucru ar permite dezvoltatorilor să creeze soluții inteligente pentru orașe sustenabile, să îmbunătățească managementul infrastructurii și să sprijine inițiativele academice bazate pe analiză de date. Un astfel de sistem ar oferi interoperabilitate între diferite servicii și ar încuraja colaborarea între cercetători, instituții publice și companii private, contribuind la eficientizarea proceselor și dezvoltarea unor soluții bazate pe date în timp real.

O astfel de aplicație ne poate oferi o bază pentru a putea testa modalități de optimizare a aplicațiilor folosind metode diverse, precum optimizarea interogărilor în InfluxDB, îmbunătățirea performanței codului scris în Java și observarea unui impact real, măsurabil, al acestor optimizări care, teoretic, aduc beneficii semnificative. Totodată, putem testa diverse metode de documentare automată pentru a analiza modul în care aceste tooluri influențează experiența utilizatorilor și contribuie la înțelegerea și utilizarea eficientă a API-urilor.

Mai mult decât atât, această aplicație poate reprezenta punctul de plecare pentru dezvoltarea unei librării Java care să faciliteze accesul la baze de date de tip InfluxDB fără a fi necesară utilizarea directă a interogărilor sub formă de șiruri de caractere (string queries). O astfel de librărie ar putea funcționa similar unui ORM (Object-Relational Mapping).

2.State of the art

2.1Optimizări InfluxDB

Există numeroase studii și metode care pot ajuta la optimizarea utilizării unei baze de date InfluxDB, însă aplicabilitatea lor într-un sistem real, bazat pe microservicii, merită analizată în detaliu. Deși multe dintre aceste tehnici sunt deja folosite, obiectivul nostru este să evaluăm impactul lor într-un scenariu concret.

Una dintre strategiile importante este utilizarea politicilor de retenție a datelor și a interogărilor continue pentru o gestionare mai eficientă a stocării și o performanță mai bună a analizelor. Politicile de retenție ajută la eliminarea automată a datelor mai vechi sau mai puțin relevante, reducând astfel consumul de spațiu și accelerând execuția interogărilor [7]. În plus, interogările continue permit pre-agregarea datelor la niveluri mai mici de granularitate, cum ar fi medii orare sau zilnice, ceea ce scade considerabil costurile de procesare pentru interogările care vizează volume mari de date [7].

O altă tehnică esențială este scrierea în loturi (batch writing), care contribuie la reducerea solicitării asupra discului și optimizează fluxul de date. Pe lângă aceasta, există și diverse tactici pentru îmbunătățirea interogărilor: evitarea scanărilor complete, utilizarea câmpurilor indexate și scrierea unor interogări mai eficiente cu InfluxQL [8]. De asemenea, ajustarea modului în care sunt alocate resursele și memoria poate preveni degradarea performanței în cazul unor sarcini intense, asigurând astfel un sistem mai stabil și mai eficient [8].

2.2Optimizări Spring Boot

La nivelul optimizărilor Spring Boot, una dintre abordări este optimizarea stream-urilor Java, care poate reduce overhead-ul de performanță în procesarea datelor. Operațiunile de stream pot deveni ineficiente atunci când nu sunt gestionate corect, iar instrumentele de profilare, precum StreamProf, permit identificarea punctelor de blocaj în fluxurile de date [9]. Astfel, prin analiza și optimizarea acestora, se poate îmbunătăți semnificativ performanța aplicațiilor care depind de procesarea intensivă a datelor.O altă metodă importantă de optimizare în Spring Boot este utilizarea procesării asincrone, care permite executarea unor operațiuni în paralel, fără a bloca firul principal al aplicației. Prin utilizarea adnotării @Async, anumite metode pot rula pe thread-uri separate, permițând aplicației să gestioneze mai eficient cereri multiple și să

îmbunătățească timpul de răspuns. Această abordare este utilă în special pentru apeluri externe, scrierea de fișiere sau alte operațiuni consumatoare de timp.

Beneficiile procesării asincrone includ reducerea latenței și creșterea capacității de scalare a aplicației, însă poate introduce și provocări, cum ar fi dificultăți în propagarea contextului sau gestionarea erorilor. De asemenea, utilizarea necorespunzătoare a thread-urilor poate duce la supraîncărcarea serverului și la consum excesiv de resurse.

În ceea ce privește gestionarea thread-urilor, există mai multe modalități de implementare, fiecare având avantaje și dezavantaje.

2.3.Optimizarea documentației automate

Un aspect important în dezvoltarea aplicațiilor este documentarea automată a codului, în special pentru API-urile REST. Redactarea manuală a acestor descrieri poate fi un proces consumator de timp și predispus la erori. Pentru a eficientiza acest proces, au fost dezvoltate diverse soluții care permit generarea automată a specificațiilor OpenAPI direct din codul sursă Java, precum: Respector, Prophet, AutoOAS și springdoc-openapi. Printre acestea, AutoOAS se remarcă printr-o precizie îmbunătățită în identificarea căilor API [10]. Cu toate acestea, pe lângă acuratețea tehnică, este important să se evalueze și claritatea documentației generate, asigurându-se că aceste instrumente asigură înțelegerea funcționalităților sistemului. În acest sens, testarea acestor soluții pe proiectul final este necesară pentru a determina impactul lor real asupra dezvoltării și mentenanței aplicației dezvoltate.

3.Tehnologii și arhitectura

InfluxDB este o bază de date time-series (TSDB) pe care o folosim în cadrul aplicației pentru gestionarea și interogarea eficientă a datelor cu marcaj temporal, fiind ideală pentru colectarea datelor provenite de la senzori IoT, metrici de performanță și log-uri. Aceasta utilizează un model schemaless, permițând adăugarea flexibilă de noi serii de date fără necesitatea modificării structurii existente. Arhitectura sa bazată pe Time-Structured Merge Tree (TSM) optimizează performanța la scriere și reduce semnificativ timpul de răspuns pentru interogările recente.

Pentru backend, utilizăm Spring Boot, datorită integrării excelente cu InfluxDB și a ecosistemului său robust pentru dezvoltarea API-urilor REST. Aplicația integrează InfluxDB prin Spring Data Influx, permițând interogări reactive și o stocare eficientă a datelor. Totodată, folosim Spring Actuator pentru monitorizarea stării aplicației și Spring WebFlux pentru programare reactivă, îmbunătățind scalabilitatea și reducând latența în interacțiunile backend-ului.

În completare, folosim o bază de date Oracle pentru gestionarea informațiilor legate de utilizatori, cum ar fi emailurile, parolele (hash-uite și securizate), token-urile de

autentificare și metadatele aferente accesului. Această separare între datele de infrastructură (time-series) și cele de autentificare/utilizator asigură o mai bună organizare, securitate și scalabilitate a sistemului.

Pentru deployment, sistemul rulează într-un mediu orchestrat cu Kubernetes, care ne permite scalare atât pe verticală, cât și pe orizontală. Acesta asigură echilibrarea automată a sarcinii, gestionarea fail-urilor și o disponibilitate crescută în contextul unei cereri variabile. Monitorizarea performanței aplicației este realizată cu Prometheus, pentru colectarea de metrici, și Grafana, pentru vizualizarea acestora, oferind o imagine clară asupra timpilor de răspuns, utilizării resurselor și comportamentului aplicației în timp real.

Pentru testarea rezilienței backend-ului în condiții de rețea instabilă sau degradată, utilizăm Toxiproxy, un tool care ne permite să simulăm întârzieri, pierderi de pachete și alte scenarii de instabilitate a conexiunii cu InfluxDB. Acest lucru este esențial pentru identificarea timpurie a problemelor de performanță și pentru validarea comportamentului aplicației în condiții adverse.

Pentru testarea scalabilității și a capacității de a susține trafic ridicat, folosim Gatling, un instrument performant de testare a API-urilor. Cu ajutorul acestuia putem genera un volum mare de cereri simultane și analiza în detaliu metrici precum timpul mediu de răspuns, rata erorilor și distribuția latenței. Fiind integrat într-un mediu Kubernetes, Gatling ne ajută să identificăm bottleneck-urile aplicației și să ajustăm alocarea resurselor în funcție de cerințele reale de trafic și performanță.

Aplicația este structurată în două microservicii principale. Primul este microserviciul de business logic, responsabil pentru preluarea datelor din InfluxDB și pentru gestionarea principalelor funcționalități, precum sortarea, filtrarea și selecția datelor în funcție de cerințele utilizatorului. Al doilea este un microserviciu de autentificare și logare, care nu este expus public, dar are rolul de a valida token-urile primite de la utilizatori, de a genera token-uri noi pe baza credențialelor, de a crea conturi noi și de a aplica politici de acces prin rate limiting. Acest microserviciu este conectat la o bază de date Oracle, unde stochează informațiile legate de utilizatori (emailuri, parole, token-uri) și datele necesare pentru aplicarea regulilor de rate limiting.

Așa cum este ilustrat în Figura 1 de mai jos, utilizatorul interacționează exclusiv cu microserviciul de business logic, care are rolul de a prelua datele din baza de date InfluxDB. Acesta comunică la rândul său cu microserviciul de autentificare, care validează fiecare cerere în funcție de token-ul primit, precum și din perspectiva politicilor de rate limiting. În cazul în care cererea nu este validată, utilizatorul va primi un mesaj de eroare corespunzător. Activitatea celor două microservicii este monitorizată prin Prometheus, care colectează metrici de performanță și le transmite către Grafana pentru vizualizare. În plus, Grafana integrează și informații despre starea celor două baze de

date (InfluxDB și Oracle), permițând analiza detaliată a comportamentului fiecărei componente a sistemului.

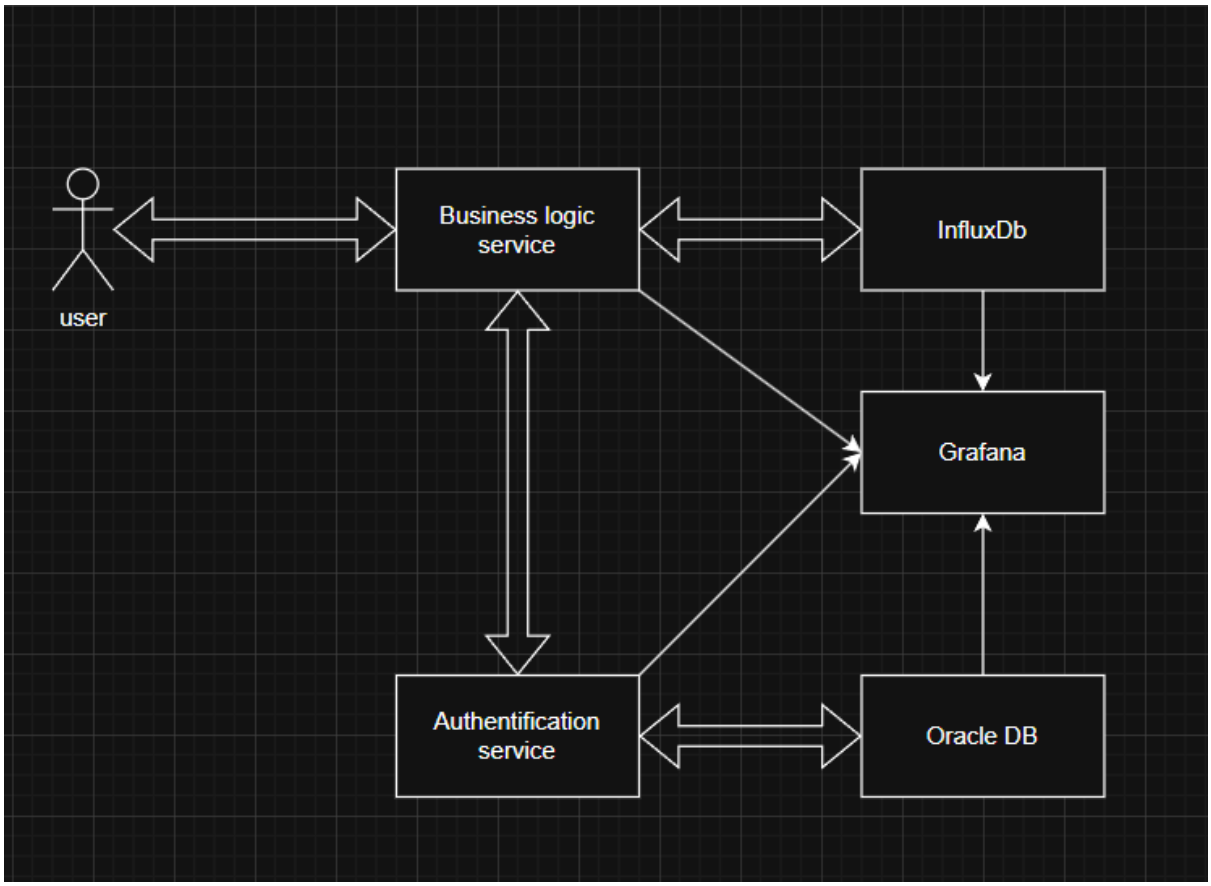

Figura 1. Arhitectura aplicației

4.Metodologie

În ceea ce privește metodologia proiectului, putem împărți procesul de dezvoltare în mai multe etape distincte:

4.1. Dezvoltarea aplicației de bază

Această etapă constă în construirea celor două microservicii, fiecare cu roluri bine definite. Microserviciul de business logic va implementa funcționalitățile principale legate de procesarea și interogarea datelor, iar microserviciul de autentificare va gestiona accesul pe bază de token și credențiale.

Funcționalitățile de bază urmărite în cadrul aplicației includ:

I. Interogarea datelor colectate de senzori pe intervale de timp personalizate

- Utilizatorii vor putea efectua interogări personalizate pentru a accesa date colectate de senzori într-un anumit interval de timp

- Interogările vor suporta atât date custom, cât și date preprocesate.

II. Paginarea rezultatelor

- Pentru a optimiza timpul de răspuns și a evita supraîncărcarea infrastructurii, datele vor fi paginare.
- Paginarea va permite utilizatorilor să navigheze prin volume mari de date fără a solicita resurse excesive din backend.

III. Agregarea datelor pe intervale mari de timp

- Pentru interogări extinse, datele vor putea fi agregate folosind metode precum:

  1. Media (average)

  2. Minimul și maximul valorilor dintr-un interval

  3. Sumă totală și alte statistici relevante

- Granularitatea va fi configurabilă, permițând utilizatorilor să selecteze niveluri precum minute, ore sau zile pentru reducerea numărului de rezultate și optimizarea performanței.

IV. Autentificarea pe baza credențialelor UPB

- Sistemul va permite autentificarea utilizatorilor folosind identitatea instituțională (e.g., email UPB).

V. Mecanisme de siguranță prin rate limiting și monitorizare a accesului

- Fiecare utilizator va avea un număr limitat de interogări permise într-un anumit interval de timp, prevenind supraîncărcarea sistemului.
- Accesul la date va fi monitorizat și logat, iar în caz de activitate suspectă, sistemul va putea bloca automat utilizatorii sau aplicațiile care depășesc limitele impuse.

4.2. Testarea impactului optimizărilor

După implementarea aplicației de bază, va urma o etapă de testare și evaluare a impactului pe care diverse optimizări tehnice îl au asupra performanței sistemului.

Această etapă presupune:

- Implementarea aceleiași funcționalități în mai multe variante optimizate diferit (la nivel de query-uri InfluxDB și la nivel de cod Java/Spring Boot).
- Măsurarea metodelor prin:
  1. timpul de execuție,
  2. timpul petrecut în stare de blocare (lock),
  3. utilizarea threadurilor și a resurselor sistemului.

- Încărcarea bazei de date cu un volum considerabil de date pentru a simula condiții reale de funcționare.

4.3. Evaluarea metodelor de documentare automată

În paralel cu testele de performanță, se va realiza și o evaluare comparativă a diferitelor metode de documentare automată prezentate în capitolul *„Optimizarea documentației automate”*.

Scopul acestei comparații este să identifice soluțiile care oferă cea mai bună experiență în înțelegerea și utilizarea API-ului.

4.4. Concluzii experimentale

Pe baza datelor colectate din testele de performanță și din analiza modului de documentare, se vor trage concluzii privind:

- cele mai eficiente metode de optimizare aplicabile aplicațiilor bazate pe InfluxDB și Spring Boot,
- validitatea și acuratețea metodologiilor de testare folosite,
- modul în care aceste optimizări pot fi transpuse în practică pentru aplicații reale de tip microserviciu.

5Contribuție practică

Datele colectate în cadrul aplicației sunt stocate în baza de date InfluxDB, care organizează informațiile sub formă de measurements (măsurători). Fiecare measurement reprezintă o categorie distinctă de date, cum ar fi vreme, număr de persoane sau nivel de zgomot. În cadrul fiecărei măsurători pot exista mai multe fields (câmpuri) care descriu valorile măsurate, de exemplu: temperatură, umiditate, presiune sau număr de persoane. Aceste date pot fi filtrate și analizate în funcție de tag-uri specifice, precum locația senzorului sau tipul dispozitivului.

În ceea ce privește contribuția practică la proiect, am dezvoltat componenta de business logic, care oferă următoarele funcționalități esențiale:

- Afișarea tuturor măsurătorilor disponibile și a câmpurilor (fields) aferente;
- Selectarea datelor pe baza valorilor asociate tag-urilor;
- Filtrarea datelor în intervale de timp specificate de utilizator;
- Returnarea informațiilor doar din câmpurile dorite de utilizator;
- Paginarea rezultatelor pentru optimizarea performanței și a răspunsului API-ului;
- Apelarea serviciului de autentificare pentru:

    - Validarea tokenului utilizatorului înaintea executării cererii;
    - Crearea de conturi;
    - Obținerea de tokenuri de autentificare;
- Validarea datelor primite în request-uri, pentru a preveni erorile și abuzurile.

Componenta de autentificare implementează:

- Crearea și gestionarea conturilor de utilizator;
- Generarea de tokenuri JWT folosind Spring Security;
- Mecanisme de rate limiting, implementate prin biblioteca *Resilience4j*, pentru a preveni supraîncărcarea aplicației și a limita abuzurile.

Ambele aplicații Spring sunt configurate pentru monitorizare prin Prometheus, care colectează metrice legate de starea și performanța serviciilor. Aceste metrice sunt transmise către Grafana, unde sunt vizualizate sub formă de grafice și dashboards.

În plus, informațiile referitoare la starea bazei de date InfluxDB sunt colectate prin intermediul unui container Telegraf, care preia automat date precum:

- Numărul de scrieri pe secundă;
- Timpul mediu de răspuns al interogărilor;
- Utilizarea memoriei și procesorului;
- Numărul de serii active în baza de date.

Întregul proiect rulează într-un mediu containerizat, fiecare componentă (business logic, autentificare, InfluxDB, OracleDB, Telegraf, Prometheus, Grafana) fiind izolată în propriul container pentru o mai bună scalabilitate și mentenanță.

Pentru a realiza testarea eficientă a funcționalităților implementate, datele utilizate în baza de date InfluxDB au fost generate sintetic cu ajutorul unor scripturi Python dedicate. Aceste scripturi simulează comportamentul senzorilor reali prin crearea de seturi mari de date, distribuite pe multiple measurements și structurate pe baza unor timestamp-uri realiste. Valorile generate acoperă câmpuri precum temperatură, umiditate, număr de persoane, etc., și permit testarea interogărilor pe diferite intervale de timp, niveluri de granularitate și scenarii de încărcare.

Funcționalitățile implementate acoperă majoritatea cerințelor definite inițial și oferă o bază solidă pentru desfășurarea testelor de performanță, stabilitate și evaluarea impactului optimizărilor propuse în cadrul proiectului.

6.Metrici de performanță

În cadrul proiectului, o etapă esențială va fi reprezentată de analiza performanței aplicației, cu scopul de a evalua comportamentul sistemului în condiții reale de utilizare și de a determina impactul diferitelor optimizări aplicabile atât bazei de date InfluxDB, cât și arhitecturii Spring Boot.

Monitorizarea performanței va fi realizată folosind o suită de unelte dedicate:

- **Prometheus** va colecta datele de monitorizare din microserviciile Spring Boot, incluzând metrici precum timpi de răspuns, throughput, rate de eroare sau starea componentelor interne.
- **Grafana** va fi folosită pentru vizualizarea acestor metrici, atât pentru partea de aplicație, cât și pentru baza de date InfluxDB. Informațiile referitoare la starea InfluxDB vor fi colectate printr-un container **Telegraf**, care va urmări indicatori precum: rata de scriere, latența medie a interogărilor, utilizarea resurselor și statusul conexiunilor.
- **Spring Boot Actuator** va furniza date legate de starea internă a aplicației: consumul de memorie, threaduri active, cache-uri, conexiuni către baze de date etc.
- **Gatling** va fi utilizat pentru testarea performanței API-ului la sarcină crescută, prin simularea unui număr mare de utilizatori concurenți.
- **Toxiproxy** va permite simularea unor probleme de rețea (latență, pierderi de pachete, timeouturi).

Analiza performanței se va concentra pe următorii indicatori:

1. **Timpul de răspuns** – măsurarea timpului necesar pentru procesarea interogărilor simple și complexe, în diferite condiții de sarcină.
2. **Throughput-ul** – evaluarea numărului de cereri procesate într-o anumită perioadă.
3. **Utilizarea resurselor** – CPU, memorie, număr de threaduri active, în funcție de tipul interogărilor sau numărul de utilizatori.
4. **Starea de blocare a microserviciilor** – identificarea eventualelor blocaje în timpul execuției interogărilor extinse sau în scenarii de utilizare intensă.

7.Analiza rezultatelor

Aceeași funcționalitate va fi implementată în mai multe moduri pentru a permite compararea performanței. Vor fi urmărite următoarele direcții:

- Compararea timpilor de execuție pentru variante optimizate și neoptimizate ale endpoint-urilor.
- Evaluarea impactului granularității interogărilor (minute, ore, zile) asupra timpului de răspuns.
- Măsurarea impactului asupra resurselor și latenței atunci când sunt aplicate diferite tehnici de agregare.

Un alt aspect important este compararea metodelor de generare automată a documentației (ex. Swagger), cu accent pe eficiența în interacțiunea dezvoltatorilor cu API-ul.

Scopul acestor analize va fi acela de a furniza un set de date clare și reproductibile care să demonstreze eficiența optimizărilor propuse. Totodată, se vor putea extrage concluzii privind scalabilitatea aplicației, comportamentul în scenarii reale și bune practici pentru dezvoltarea sistemelor distribuite bazate pe microservicii, InfluxDB și Spring Boot.

8.Concluzie si viitori pași

Proiectul propus a pus bazele unei aplicații containerizate, alcătuită din două microservicii principale – unul de business logic și unul de autentificare – capabile să răspundă în mod eficient interogărilor utilizatorilor asupra datelor stocate în InfluxDB. Structura modulară, integrarea mecanismelor de autentificare, paginare, validare, precum și infrastructura de monitorizare prin Prometheus și Grafana, oferă o baza solidă pentru dezvoltarea și testarea ulterioară a aplicației.

În ceea ce privește analiza performanței, aceasta urmează a fi realizată cu ajutorul unor unelte precum Prometheus, Grafana, Gatling și Toxiproxy, vizând metrice precum timpul de răspuns, încărcarea resurselor, rata de erori, comportamentul la limitări de acces și reziliența în fața eșecurilor.

Datele utilizate în testare vor fi generate cu ajutorul unor scripturi Python, care incearcă modelarea unui set de date real, fiind organizate în multiple *measurements* corespunzătoare unor categorii precum vreme, trafic sau mediu, fiecare având fielduri multiple (temperatură, umiditate, număr de persoane etc.).

Pentru viitor, proiectul va fi extins în următoarele direcții:

- Adăugarea unei funcționalități de selecție a datelor pe baza granularității temporale, permițând utilizatorilor să aleagă niveluri precum minute, ore sau secunde, în funcție de nevoile analitice. Aceste selecții vor putea fi însoțite de metode statistice diverse precum media, mediana, minim, maxim sau sumă totală.

- Aplicarea concretă a optimizărilor identificate în etapele anterioare ale documentației, atât asupra interacțiunii cu InfluxDB, cât și la nivel de Spring Boot (configurarea thread pool-urilor, gestionarea eficientă a conexiunilor, optimizarea endpoint-urilor etc.).
- Testarea comparativă a performanței înainte și după optimizări, în diferite scenarii de utilizare, pentru a putea determina impactul real al acestora asupra funcționării aplicației în condiții de producție.

9.Referințe

[1] Fielding, R. T., & Taylor, R. N. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.

[2] Medjahed, B., Bouguettaya, A., & Elmagarmid, A. K. (2003). Composing Web Services on the Semantic Web. VLDB Journal.

[3] Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

[4] Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J., & Josuttis, N. (2017). Microservices in Practice, Part 1: Reality Check and Service Design. IEEE Software

[5] InfluxData. (2021). InfluxDB Documentation. Retrieved from https://docs.influxdata.com [6] Kaplan, J. (2020). Time Series Databases: New Ways to Store and Analyze Data. O'Reilly Media.

[7] Admantium. (2025, January). InfluxDB: Optimize your Data with Data Retention Policies and Continuous Queries

[8] Toxigon. (2025, February). Best Practices for InfluxDB Performance: Enhance Your Time-Series Data

[9] Smith, J., & Doe, A. (2023). Profiling and optimizing Java streams

[10] Doe, J., & Smith, A. (2024). Automated OpenAPI Documentation Generation