Proiectarea și implementarea unui API RESTful

Sumar:

Creează un API (RESTful) care simulează o bază de date in-memory, capabil să proceseze și să stocheze eficient datele primite, să răspundă rapid la interogări, și să fie ușor de scalat și gestionat.

Specificații generale:

Serviciul va trebui să fie capabil să primească și să proceseze cereri HTTP care includ texte de la utilizatori. Acesta va contoriza apariția fiecărui cuvânt din textele primite, cuvintele trebuie tratate case-insensitive. Ulterior, serviciul va permite interogarea acestei baze de date simulate pentru a afla de câte ori a apărut un anumit cuvânt.

Pe lângă manipularea datelor în memorie, pentru a asigura răspunsuri rapide, serviciul va trebui să aibă grijă de persistența acestor date, astfel încât să poată restaura starea exactă în care a fost oprit, la o repornire ulterioară.
Pentru a răspunde unui număr ridicat de requesturi, serviciul trebuie să fie proiectat astfel încât să permită rularea simultană a mai multor instanțe.

Codul trebuie scris de la zero, doar cu biblioteci native ale limbajului.

Detalii tehnice:

Procesarea și Stocarea Datelor: La primirea unui text printr-un request HTTP POST, serviciul va analiza textul și va stoca numărul de apariții pentru fiecare cuvânt in-memory. Totul trebuie tratat case-insensitive, astfel încât textul să fie normalizat la lowercase înainte de procesare.

Interogarea Datelor: Serviciul va expune un endpoint HTTP GET care va permite utilizatorilor să solicite numărul de apariții pentru unul sau mai multe cuvinte. Răspunsul va fi în format JSON și va conține perechi de tipul cuvânt -> număr apariții.

Persistența și Recuperarea Datelor: Pentru a preveni pierderea datelor la oprirea serviciului, acesta trebuie să scrie periodic starea actuală într-un fișier. La repornire, serviciul trebuie să fie capabil să își refacă starea din acest fișier.

Disponibilitate ridicată: Având în vedere că dorim o soluție cu scalabilă cu accent pe high availability, serviciul va trebui să fie proiectat astfel încât să permită rularea simultană a mai multor instanțe. Arhitectura trebuie sa asigure redundanță între instanțe și consistență eventuală, astfel încât Availability și Partition Tolerance din teorema CAP sunt obligatorii (https://en.wikipedia.org/wiki/CAP_theorem).

Demonizare și Logging: Serviciul trebuie să fie ușor de demonizat, astfel încât să poată fi integrat cu un manager de servicii precum systemd. În același timp, trebuie să fie implementat un sistem de logging care să înregistreze atât accesul, cât și orice alte informații relevante, pentru a putea monitoriza și depana serviciul.

Provocări suplimentare:

1. Odată ce ai implementat API-ul REST, un pas suplimentar ar fi migrarea acestuia de la HTTP la gRPC, utilizând doar bibliotecile native. Acest lucru va implica refactorizarea codului pentru a folosi protocoale gRPC în loc de HTTP, păstrând în același timp toată funcționalitatea existentă.
2. Identifică potențialele bottleneck-uri în aplicație și propune soluții pentru a îmbunătăți performanța.
3. Ce măsuri de securitate ai implementa pentru a proteja serviciul împotriva atacurilor comune precum injecții SQL, XSS, etc.?

Livrabile:

Te rugăm să livrezi codul sursă complet, împreună cu un fișier README care să explice arhitectura aleasă, modul în care ai implementat persistența și redundanța, și orice alte detalii tehnice relevante.

De asemenea, include teste unitare și de performanță pentru a demonstra că serviciul răspunde cerințelor specificate.

Scopul acestui exercițiu este de a evalua nu doar abilitățile tale tehnice, ci și capacitatea ta de a proiecta soluții scalabile și reziliente, utilizând principii moderne de arhitectură software. Succes!