Andreea-Lavinia Stingu

Documentatie aplicatie

Federated Search Engine

Cuprins

1. Continut arhiva

Arhiva contine fisierele sursa separate in 5 pachete

- Pachetul dao: MovieDAO(interfata), MovieDAOImplemen
- Pachetul model: Content, MoviesDetails, PoolMessageNotReady, SyncServer
- Pachetul service: MovieService(interfata), MovieServiceImplemen
- Pachetul mainserver: MainService
- Pachetul mvc: HelloController (controller-ul principal), PoolController (controller-ul pentru serverul Pooling), SyncController (controller-ul pentru serverul SyncWebServer)

Fisierele jsp se gasesc in webapp/WEB-INF/pages: hello.jsp(interfata cu utilizatorul contine field-ul in care utilizatorul introduce cuvantul dupa care se realizeaza cautarea), displayInformation.jsp (se afiseaza informatiile gasite).

Fisierul db_schema.sql contine schema bazei de date.

2. Tehnologii folosite

Am lucrat in IntelliJ, am folosit Spring MVC, Maven, JPA. Baza de date folosita este MySQL.

3. Modul de rulare al aplicatiei

In primul rand se pornesc cele doua servere SyncWebServer si Pooling server. Se deschide baza de date (am utilizat MySQL WorkBench), se ruleaza in MySQL WorkBench scriptul db_schema.sql. Se deschide proiectul cu IntelliJ, se ruleaza aplicatia; in browser se va deschide o pagina in care utilizatorul poate introduce intr-un text field un cuvant cheie dupa care se va face cautarea. Dupa apasarea butonului Submit controller-ul va incepe prelucrarea datelor dupa care va trimite informatiile gasite utilizatorului, afisandu-le in pagina.

4. Implementarea aplicatiei

Am realizat interogarea pe doua servere SyncWebServer si PoolServer. Fiecare server are un controller separat SyncController si PoolController.

Din serverul SyncWebServer salvez in baza de date doar anumite informatii: title, poster, url si year (clasa Content contine aceste 4 campuri). Clasa SyncServer contine un camp noEntries si o lista de Content. Din serverul PoolingServer salvez doar informatiile: Genre, Actors, Plot, Country.

Clasa PoolMessageNotReady contine doua campuri message si detail. Serverul PoolServer fiind asincron, in momentul in care este accesat va returna prima data un mesaj si detalii despre acel mesaj. Din aceasta cauza se va crea un obiect de tipul PoolMessageNotReady care va retine aceste informatii.

Clasa MoviesDetails contine campuri ce se regasesc atat pe primul server cat si pe al doilea server.

Atat clasa SyncController cat si clasa PoolController implementeaza interfata Runnable. In metoda run se realizeaza prelucrare json-ului. Fiecare clasa are o lista de obiecte de tipul MoviesDetails in care sunt salvate informatiile dupa ce este

parsat json-ul. Inainte de a fi salvate in lista, json-ul este transformat intr-un array de bytes pentru a i se aplica UTF-8.

In clasa HelloController se afla controller-ul principal. Se creeaza doua obiecte de tipul SyncController respective PoolController, dupa care se creeaza doua thread-uri ce iau ca paramteri obiectele create:

```
SyncController syncController = new SyncController(server_urlSync);

PoolController poolController = new PoolController(server_urlPool);

Thread thread1 = new Thread(syncController);

Thread thread2 = new Thread(poolController);
```

Dupa pornirea si join-ul thread-urilor (thread.start(), thread.join()) se face merge intre rezultatele gasite de fiecare thread in parte, rezultatele fiind adaugate in lista moviesDetails. Aceasta lista finala este preluata in displayInformation.jsp unde vor fi afisate rezultatele finale.

Proiectul creat este de tipul SpringMVC, este configurat cu Maven, in pom.xml sunt adaugate toate dependintele de care am avut nevoie. Atat conectarea la baza de date cat si metodele ce insereaza/cauta/sterge in/din baza de date au fost realizate pas cu pas urmarind proiectul de la prezentarea de JPA. In fisierul config.properties se face conexiunea cu baza de date .

La partea de cod am adaugat mai multe comentarii in surse. In baza de date inregistrarile sunt salvate pentru 15 minute (900000 milisecunde);

Initial am realizat partea de cache in felul urmator: daca in intervalul de 15 minute utilizatorul cauta un film care este deja in baza de date, informatiile despre acesta vor fi preluate direct din baza de date, in cazul in care au fost depasite cele 15 minute, informatiile despre filmul respectiv se sterg din baza de date si se face iar cautare pe servere. Dar nu am stiut cat de corecta este aceasta solutie, asa ca am regandit problema altfel: in momentul in care se face o cautare, se verifica mai intai pentru toate inregistrarile din baza de date acum cat timp au fost adaugate. Daca timpul de cand au fost adaugate depaseste 15 minute, inregistrarile respective vor fi sterse din baza de date.

Nu am reusit sa integrez si serverul de Callback deoarece nu am inteles cum functioneaza acel server.

Desi nu am terminat aplicatia am invatat foarte multe lucruri si am mai inteles cate putin din notiunile teoretice din prezentari.