Bursa

Cojocariu Flavius

Constantin Iasmin

Dieaconu Vlad

Stanciu Ovidiu

Cuprins

I. Specificații aplicație

Pentru a aprofunda problemele de concurență se folosește terminalul la simularea interacțiunilor dintre cumpărători și vânzători.

```
Command Options:
a: Start simulation
b: Show database
c: Show clients' requests
d: Show sellers' offers
e: Show tranzactions
q: Quit
```

La pornirea programului funcția *initDatabase()* va insera in baza de date toate datele de lucru prin randomizare. Inițial tabelele sunt create de noi, tipurile de acțiuni și vânzători fiind hardcodate, pentru a nu randomiza elemente de tip String. Odată încheiată executarea funcției, studentul va fi înștiințat și va avea posibilitatea de a da start simulării prin funcția *startSimulation().*

```java
public void initDatabase() throws SQLException {

    // setup connection
    con = new DatabaseConnection().connect();

    // clear all
    clearLists();
    // clear DB
    clearDBTables();

    // init lists
    initClients();
    initSellers();
    initStocks();
    initOffers();
    initRequests();

    LOG.info( msg: "Simulation is ready!");
}
```

Funcția *startSimulation()* se va ocupa de inserarea tranzactiilor in baza de date, precum și de crearea thread-urilor. Aceasta va modela și situațiile in care apare fenomenul de concurență și va actualiza baza de date la fiecare tranzacție efectuată. La oprirea funcției vom avea un meniu prin care se pot accesa informații despre tranzacționarile efectuate.

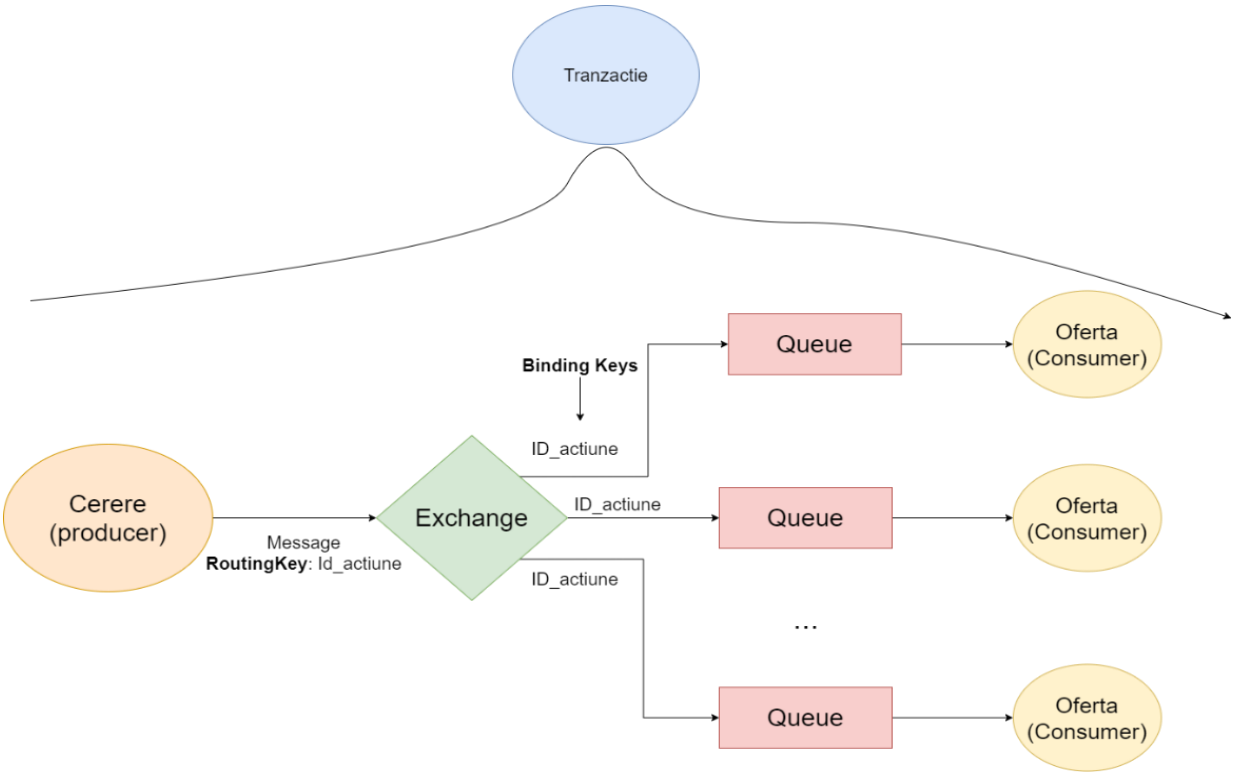
```
public void startSimulation() throws InterruptedException {
    // TODO functie startSimulation() - aici e concurenta

    LOG.info( msg: "Simulation started...");

    // make a thread for every client
    listClients.forEach(client -> {
        thd.add(new Thread(client));
    });
```

Pentru partea de evenimente, se va folosi framework-ul RabbitMQ. Procesul prin care se operează evenimentele cu RabbitMQ presupune că un Producer trimite un mesaj cu un RoutingKey in Exchange, apoi in funcție de tipul Exchange-ului se va verifica RoutingKey-ul mesajului cu BindingKey-urile fiecărui Queue. Când acestea se potrivesc mesajul este pus în Queue, acest proces se numește Publishing. Apoi Consumer-ul preia mesajul din Queue, proces denumit Consuming.

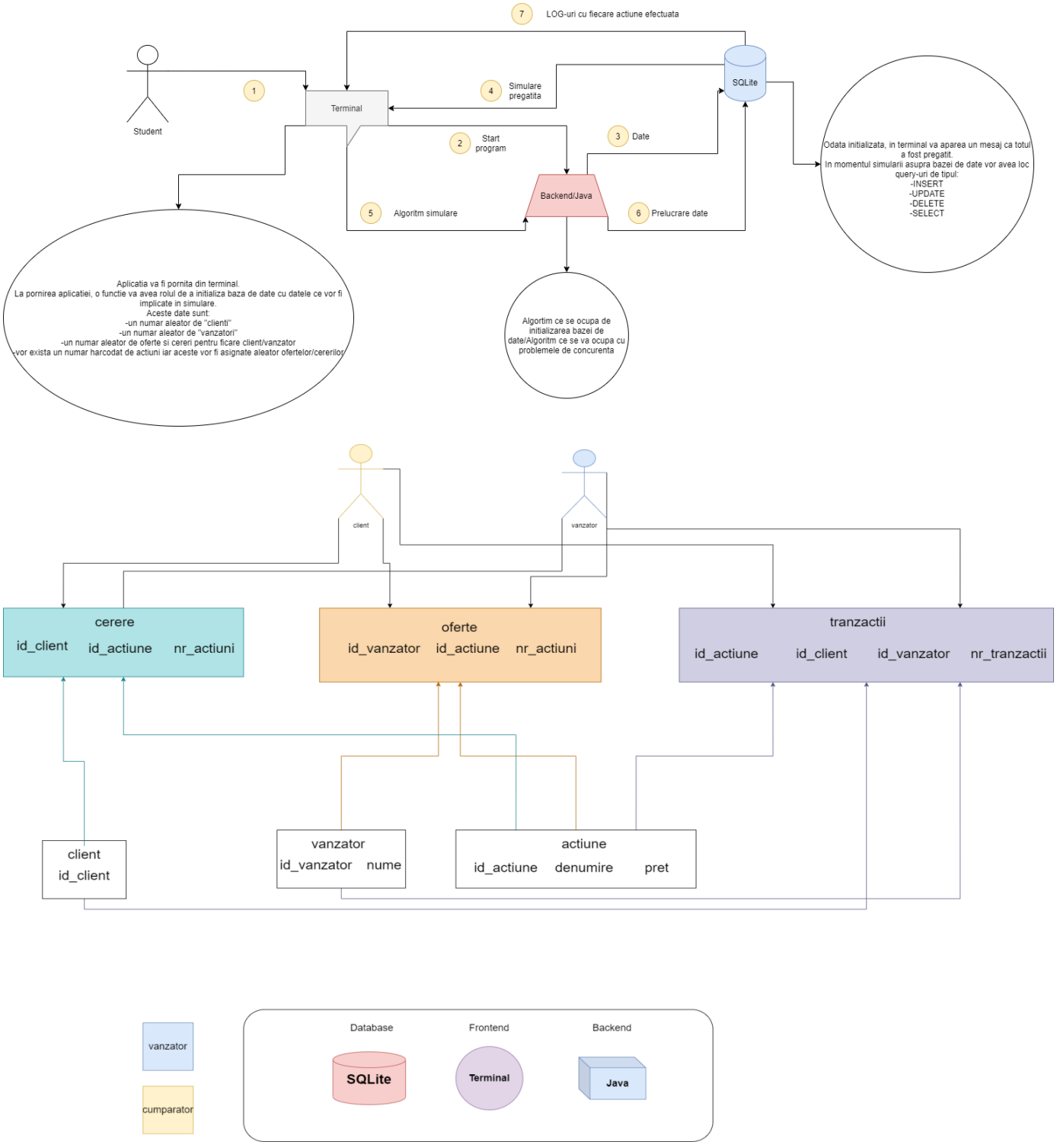
Pentru aplicația noastră interfața Producer a fost implementată prin clasa Cerere, RoutingKey-ul mesajului transmis către Exchange, este reprezentat de ID-ul acțiunii din cerere. Se va folosi Direct Exchange, adică RoutingKey trebuie sa fie egal cu BindingKey. BindingKey-ul este reprezentat de ID-ul acțiunii din ofertă. Când se trece de Exchange, mesajul merge in Queue-ul corespunzător. Fiecare Queue este specific pentru fiecare acțiune în parte. Acesta a fost procesul de Publishing.

Procesul de Consuming este efectuat de Consumer, în cazul nostru acesta fiind clasa Oferta. Oferta preia mesajul de la Cerere efectuând ultimul stagiu al procesului de tranzacționare.

Un eveniment este reprezentat de o tranzacție.



II. Design arhitectural



III. Probleme de concurență

Pentru a spori cazurile de concurentă aparute in timpul rularii functiei startSimulation(), functie care se ocupa de efectuarea tranzactiilor, am decis sa nu mai luam in considerare urmatoarea regula:

„atunci cind o oferta cu o cerere corespund ca pret, se tranzactioneaza un numar de actiuni n = min (oferta, cerere)" .

Prin asta ne referim la faptul ca toate actiunile existente se vand la acelasi pret, difera doar numarul de actiuni vandute de fiecare vanzator, respectiv de numarul de actiuni din cererile clientilor.
Modul in care se alege minimul cat si necesitatea lui au ramas la fel.

Algoritmul nostru de cautare si tranzactionare ruleaza pentru fiecare client/thread apeland *.start()* respectiv *.run()* in clasa *Client.*

In timpul rularii/procesului de debugg-ing am dat de cazul in care un client nu asteapta ca un alt client sa termine o tranzactie cu un vanzator si sa faca in acelasi timp si el o tranzactie cu acelasi vanzator dar cu un numar neactualizat de actiuni, ducand astfel la valori negative, exemplificate mai jos:

```
Nov 10, 2021 8:43:03 PM utils.Application startSimulation
INFO: Simulation started...
Nov 10, 2021 8:43:03 PM utils.Application startSimulation
INFO: Simulation ended...
Clientul: 7 a cumparat: 5, actiunea: 1 de la vanzatorul: 3
Clientul: 7 a cumparat: 3, actiunea: 1 de la vanzatorul: 4
Clientul: 2 a cumparat: 1, actiunea: 1 de la vanzatorul: 2
Clientul: 7 a cumparat: 3, actiunea: 2 de la vanzatorul: 1
Clientul: 2 a cumparat: 1, actiunea: 1 de la vanzatorul: 4
Clientul: 4 a cumparat: 4, actiunea: 1 de la vanzatorul: 3
Clientul: 7 a cumparat: 5, actiunea: 2 de la vanzatorul: 2
Clientul: 3 a cumparat: 9, actiunea: 1 de la vanzatorul: 1
Clientul: 6 a cumparat: 3, actiunea: 1 de la vanzatorul: 4
Clientul: 5 a cumparat: -9, actiunea: 1 de la vanzatorul: 1
Clientul: 1 a cumparat: 9, actiunea: 1 de la vanzatorul: 1
```

De asemenea, cu ajutorul functiilor care elimina cazurile de concurenta, vom *„semaforiza"* ordinea si prioritatea in care clientii tranzactioneaza actiunile.

Pana sa aprofundam acest aspect am folosit functia *Thread.sleep(x millis);* pentru ca fiecare client sa astepte un timp pana sa isi inceapa propriile tranzactii, astfel clientii anteriori sa isi incheie toate tranzactiile.

Pe langa aceste aspecte am intampinat si probleme de concurenta interne ale IDE-ului pe care il folosim, Intelij IDE, exemplificate mai jos:

```
Nov 08, 2021 9:50:44 PM Application startSimulation
INFO: Simulation started...
Exception in thread "main" java.util.ConcurrentModificationException Create breakpoint
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1513)
    at Application.lambda$startSimulation$3(Application.java:269)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at Application.lambda$startSimulation$4(Application.java:264)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at Application.startSimulation(Application.java:260)
    at Main.main(Main.java:18)
```

Problema a intervenit in momentul cand asupra unei liste pe care o parcurgeam, incercam sa facem modificari asupra ei, mai exact eliminare de elemente.

IV. Link repository : https://github.com/ELSGY/Bursa

Commit	Titlu	Descriere
26bace52db4431b481c4c77d4d938ace3e38796a	Created connection to database	Am inclus dependintele de care avem nevoie pentru sql Lite si am facut clasele specifice pentru lega programul nostru de baza de date.
6d50342f71dfe5a9ee90dd0a474c241b890d4a59	Added model package	Am creat pachetul unde ne vom creea modele pentru fiecare tabel din baza de date. Adica pachetul in care se face maparea intre tabelele din baza de date si programul nostru, folosindu-ne implicit de *Bridge Pattern.*

1ca01423ae28e0d1e2baa66b4bc052c50aa2b1a6	Added all model classes, init functions, TODOs	Am definit toate modele de care avem nevoie, functia de init, care are rolul de a ne popula tabelele in baza de date intr-un mod random.
4d2ce6af94690da7943fa20e3c087fd8e9be5dd6, 4d2ce6af94690da7943fa20e3c087fd8e9be5dd6	Resolved some TO DO.	Implementarea unor clase definite in comitul anterior.
196c36559a7b97578a8d3c4bbb67c5d03d61c894	Added menu, wrote complete initDatabase() function	Am creat un meniu interactiv prin care putem controla momentul in care se populeaza baza de date si momentul in care incep tranzitiile, adica momentul in care se creaza oferte si cereri intr-un mod aleatoriu, de asemnea avem la dispozitie si o optiune prin care putem vedea ce avem in baza de date la un moment dat.
4a68401eaa56bdf7df41421e6f545a9cc75b2d21	Solved problem with duplicated offers and requests	Am eliminat cazurile in care un vanzator poate sa aiba mai multe oferte deschise pe aceeasi actiune.
6910d66f22e6faa3e5cf2de837a24aa9504a40ae	Example for tranzaction	Am definit modul in care se accepta o trazactie. Daca numarul de actiuni este 0 pentru o anumita oferta, toate cererile pentru aceea actiune sunt ignorate, iar daca exista un numar mai mic de actiuni in oferta decat sunt cerere trazitia se face cu numarul de actiuni disponibile, iar cererea ramane deschisa cu diferenta de actiuni.

5d7d041dc051746c5751d02d274157c72fa8c8f5	Added threads, tranzaction function, competitive functions debugging	Functii si debugging in timpul tranzactionarii actiunilor.