DOCUMENTATIE

TEMA *NUMARUL 2*

NUME STUDENT:Chiorean Bogdan
GRUPA:30223

CUPRINS

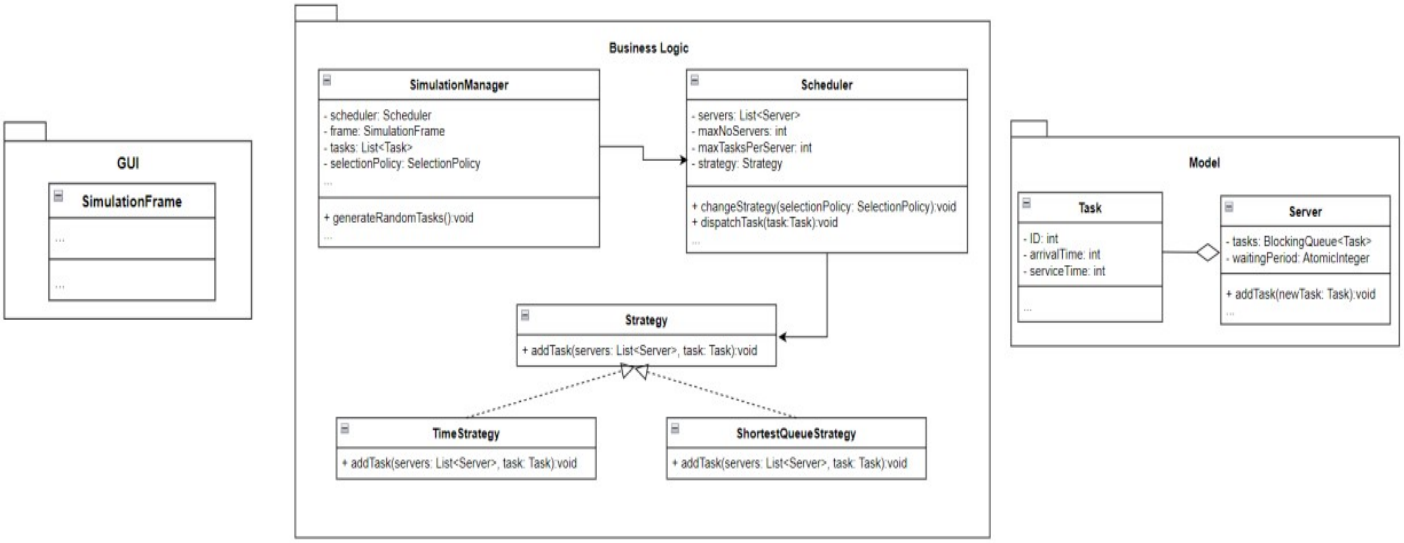
1. Obiectivul temei

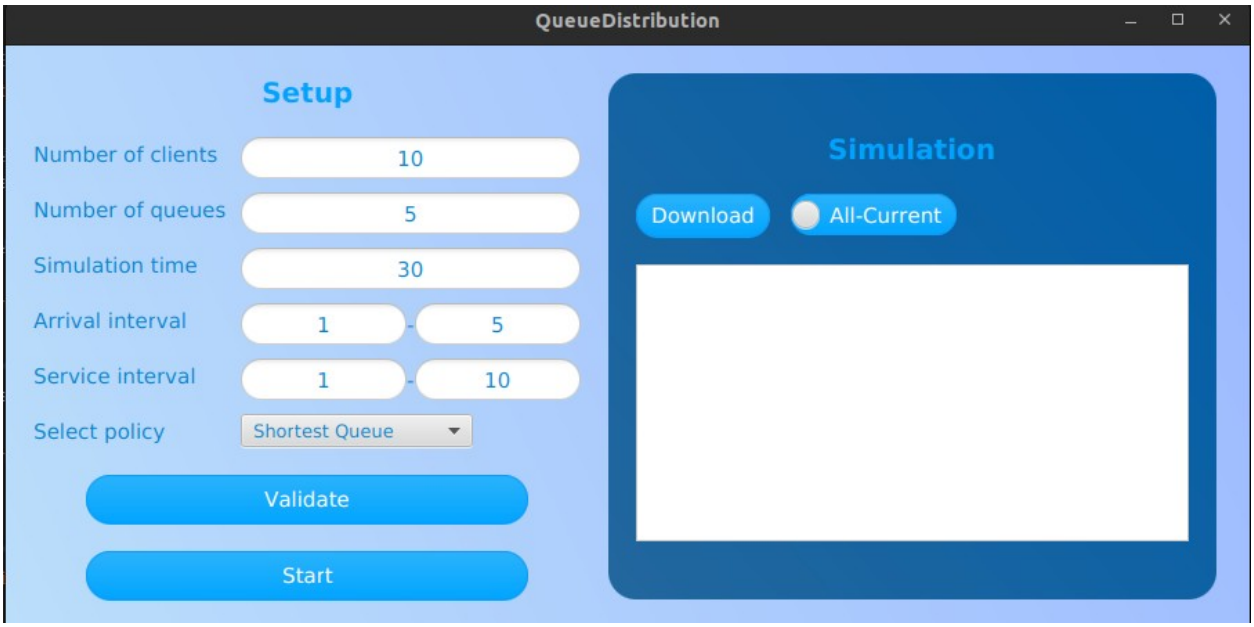
Obiectivul temei este de a realiza o simulare a impartirii mai multor task-uri in mai multe servere pentru a crea paralelism si eficienta realizarii acestora. Tema modeleaza problema prin a crea impartirea unor clienti la niste cozi/ghisee.

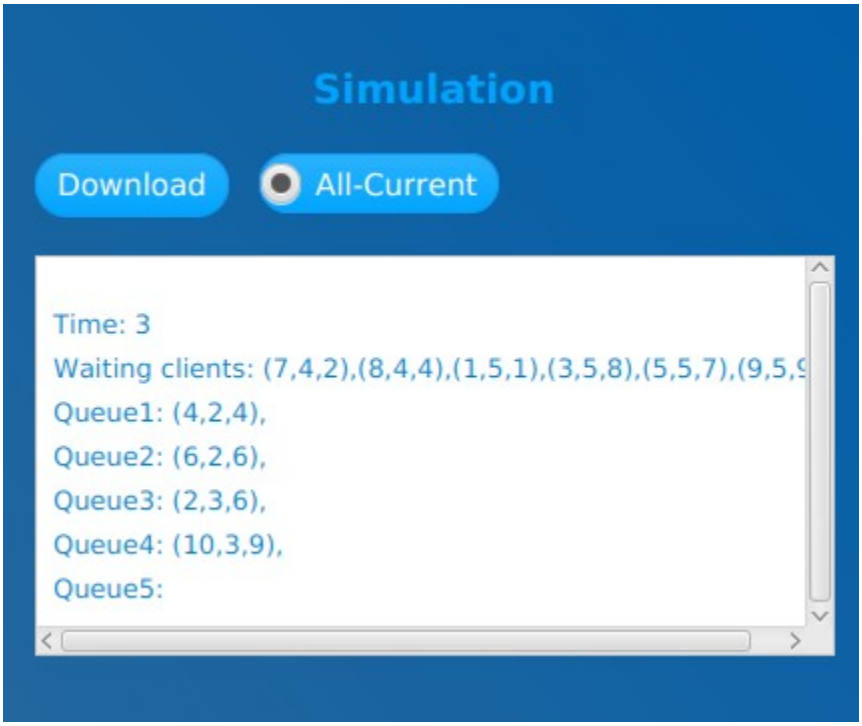
2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Rezolvarea problemei consta in lansarea a mai multor fire de executie care sa lucreze in paralel, si sa se sincronizeze pentru a putea realiza impartirea clientilor in cozi. Fiecare coada o sa aiba cate un fir de executie, iar pe langa acestea, vom avea unul pentru a le controla pe acestea, si unul pentru a actualiza interfata greafica. Aplicatia are un singur caz de utilizare: utilizatorul completeaza datele/scenariul simularii, iar apoi da start, restul facandu-se automat.

3. Proiectare







Am folosit modelarea prezentata in resursele temei. Suplimentar am creat un fisier css pentru stilizarea interfetei. Modelul de proiectare este un Model - Bussiness Logic – GUI.

4. Implementare

Clasele care realizeaza simularea sunt Task,Server,Scheduler si SimulationManager. Clasele legate de strategie, si enumeratia asociata au rolul de

a defini politica de atribuire la cozi:cea mai scurta coada sau coada la care se asteapta cel mai putin. Clasa de SimulationFrame creeaza interfata folosind tehnologia JavaFX si realizeaza interactiunea programului cu utilizatorul.

Clasa Task reprezinta un client care doreste sa isi rezolve problemele (sa fie procesat) la gisee asteptand la cozi. Aceasta clasa are ca atribute un id unic, ora la care ajunge si timpul in care isi rezolva problemele. Metodele acestei clase constau doar in accesarea si actualizarea datelor (constructor,settere si gettere).

Serverul reprezinta o coada, acesta are ca parametru o lista de task-uri (clienti) care vor urma sa fie procesate. Acesta implementea interfata Runnable definind firele de executie care se ocupa de gestiunea clientilor.

Scheduler este clasa care contine toate cozile valabile si selecteaza politica de repartizare. Acesta adauga un client in cozi, creaza firele de executie si le inchide.

SimulationManager contine toate datele cu care se creaza scenariul simularii, acesta creeaza clientii, si porneste firul de executie care se ocupa cu gestionarea cozilor (serverelor).

Sincronizarea firelor de executie:

-pentru a asigura corectitudinea si securitatea datelor, atributele care se acceseaza de mai multe fire de executie trebuie sa fie accesate in ordine, pentru a nu se generale rezultate nedorite in urma unor scrieri sau citiri nedorite. Pentru asta, folosim variblile care au lacate asociate: Atomic Integer, BlockingQueue.

-pentru a sincroniza firele de executie in timp real, acestea trebuie sa fie organizate astfel incat toate sa isi realizeze intructiunile in acelas timp, iar dupa terminarea actualizarilor, sa fie afistate in interfata. Pentru rezolvarea problemei, punem ca fiecare fir de executie,dupa ce isi face "treaba" pentru timpul respectiv, sa adoarma pentru 1 secunda.

5. Rezultat

Testele facute sunt atasate pe GitHub sub forma .txt. Textele arata corectitudinea repartizarii, iar simncronizarea se poate observa in testarea real-time prin afisarea aflata in interfata.

6. Concluzii

Problema este una clasica, care prezinta un concept important de inteles, si anume sincronizarea real-time al firelor de executie. Am invatat sincronizarea firelor de executie, si actualizarea interfetelor folosindu-le.

7. Bibliografie

https://stackoverflow.com/