DOCUMENTAȚIE

TEMA 2

NUME STUDENT: **Neacșu Ioana-Alina**
GRUPA: 30221

Cuprins

1. Obiectivul temei

Obiectivul acestui proiect este de a dezvolta o aplicație de gestionare a cozilor care să atribuie clienților cozi într-un mod eficient, astfel încât timpul de așteptare să fie minimizat.

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

Cerințe Funcționale:

- Adăugare Client: Sistemul trebuie să permită adăugarea de clienți noi cu parametrii specificați: ID, timpul de sosire și durata servirii.
- Atribuire Cozi: Clienții trebuie să fie atribuiți în cozi astfel încât timpul de așteptare să fie minim.
- Servire Client: Clienții trebuie să fie serviți în ordinea în care ajung la capul cozii și să fie eliminați din coadă după ce sunt serviți.
- Simulare: Implementarea unei simulări care să reprezinte sosirea clienților, atribuirea lor, servirea și plecarea acestora.

Cerințe Non-Funcționale:

- Eficiență: Sistemul trebuie să gestioneze eficient sosirea și servirea clienților pentru a minimiza timpul de așteptare.
- Ușurința de Utilizare: Interfața aplicației trebuie să fie intuitivă și ușor de folosit.
- Performanță: Aplicația trebuie să poată gestiona un număr mare de clienți și cozi în timp real.

3. Proiectare

Diagrama Pachetelor:

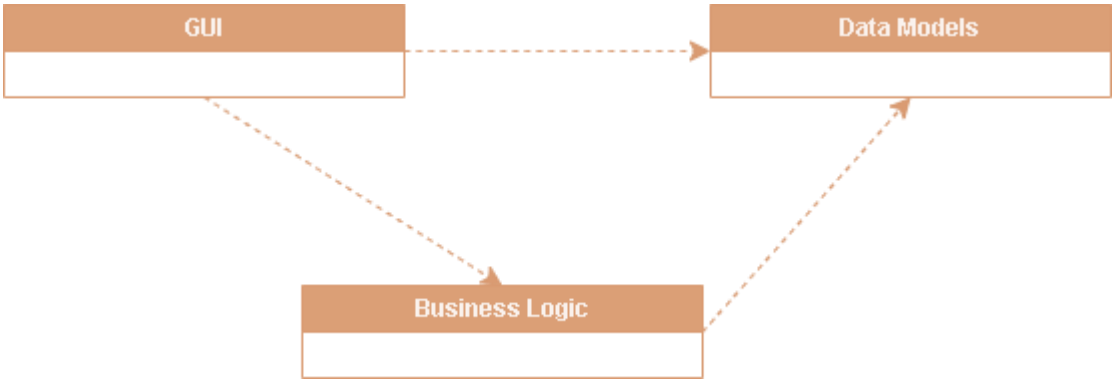
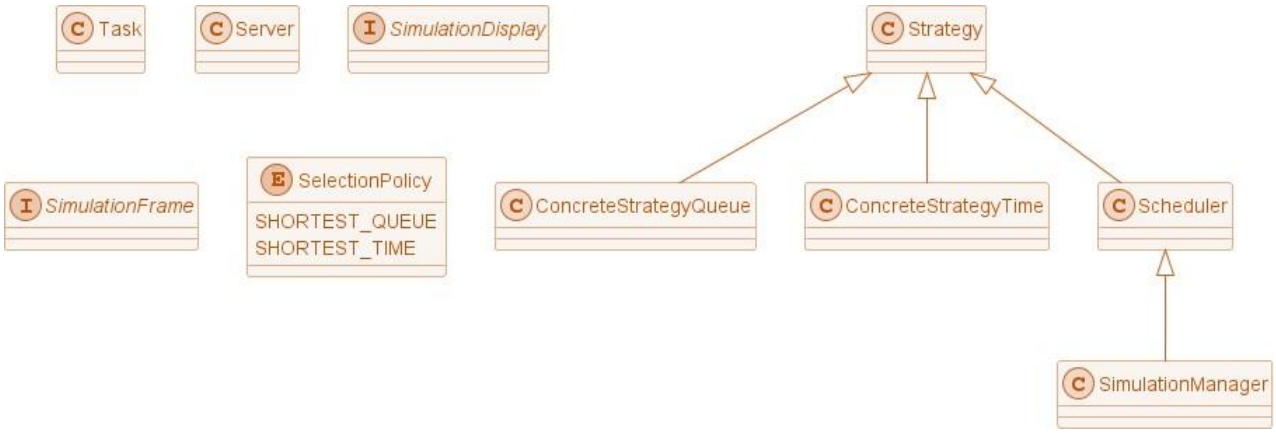


Diagrama Claselor:



4. Implementare

Clasa SimulationManager:

Este clasa principală care coordonează întreaga simulare care se ocupă de citirea datelor de intrare de la interfața utilizator, generarea sarcinilor, gestionarea serverelor și actualizarea stării simulării în timp real. Implementează interfața Runnable pentru a permite rularea simulării într-un thread separat. Metode importante:

- readUIData(): Citirea datelor de intrare din interfața utilizator.
- selectStrategy(): Selectarea strategiei de selecție a serverelor în funcție de opțiunile din interfața utilizator.
- generateNRandomTasks(): Generarea unui număr specificat de sarcini aleatorii pentru simulare.
- run(): Execuția simulării în timp real.
- updateServer(): Actualizarea stării serverelor în fiecare unitate de timp.
- writeLog(): Înregistrarea stării curente a simulării într-un fișier de jurnal.
- dispatchTask(): Transmiterea sarcinilor către servere în momentul sosirii acestora.

Clasa Scheduler:

Gestionează planificarea sarcinilor pe servere. Include logica pentru a distribui sarcinile către servere folosind o anumită strategie (de exemplu, cea mai scurtă coadă sau cel mai scurt timp de așteptare). Creează și lansează thread-uri pentru fiecare server pentru a permite execuția simultană.
Metode importante:

- Scheduler(strategy): Constructor pentru inițializarea unei instanțe a clasei cu o anumită strategie de selecție a serverelor.
- dispatchTask(Task t): Transmiterea unei sarcini către unul dintre servere folosind strategia specificată.

Interfața SimulationFrame:

Interfața grafică pentru configurarea simulării. Oferă câmpuri de introducere pentru parametrii simulării și butoane pentru a începe simularea. Permite utilizatorului să interacționeze cu aplicația și să controleze setările simulării.

Utilizatorul poate folosi interfața în următorii pași:

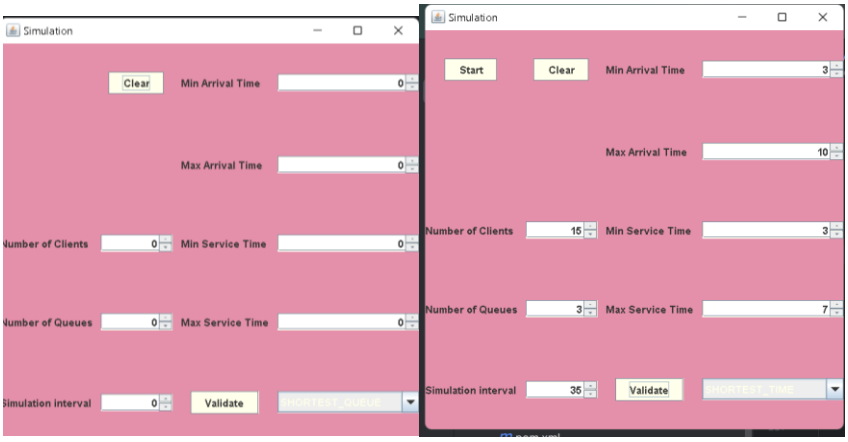
Pașii utilizatorului:

a) Utilizatorul deschide aplicația și își configurează simularea folosind câmpurile disponibile.

b) După ce a introdus parametrii, utilizatorul apasă butonul "Validate" pentru a valida datele introduse.

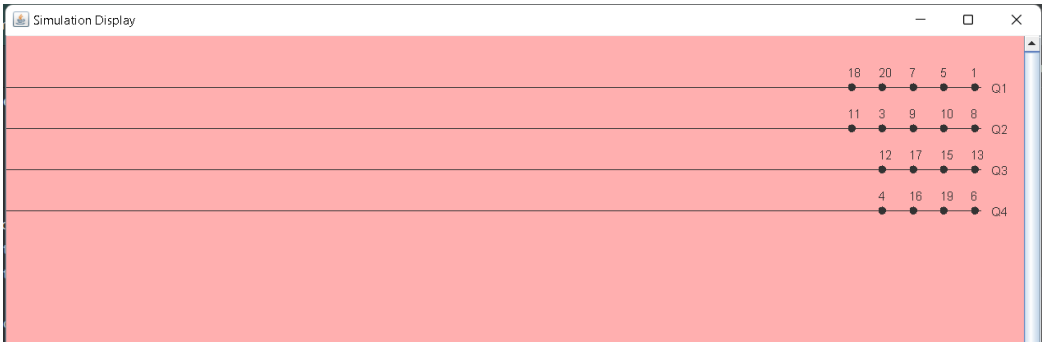
c) Dacă datele au fost introduse corect se va afișa butonul "Start". Acesta se apasă pentru a începe simularea

d) Utilizatorul poate analiza rezultatele prin deschiderea fișierului "log.txt" sau poate repeta simularea cu alte setări folosind din nou interfața grafică.



Interfața SimulationDisplay:

Interfața grafică pentru afișarea simulării în timp real. Desenează starea curentă a serverelor, sarcinile în așteptare și alte informații relevante despre simulare. Oferă o vizualizare clară și intuitivă a evoluției simulării pentru utilizator.



5. Concluzii

Thread-urile și interfețele grafice reprezintă aspecte esențiale în dezvoltarea aplicațiilor software moderne. Utilizarea lor corectă și eficientă poate îmbunătăți performanța, experiența utilizatorului și calitatea generală a aplicației. O înțelegere solidă a acestor concepte este crucială pentru dezvoltatorii de software care doresc să creeze aplicații robuste și ușor de utilizat. Pe parcursul acestui proiect, am învățat mai multe aspecte importante legate de dezvoltarea software:

1. Lucrul cu interfețe grafice (GUI)
2. Sincronizarea thread-urilor

Dezvoltări ulterioare posibile:

1. Optimizarea performanței: Se poate explora posibilitatea de a optimiza performanța aplicației prin îmbunătățirea algoritmilor existenți sau prin utilizarea mai eficientă a resurselor de sistem.
2. Optimizarea codului: Se poate explora posibilitatea de a optimiza codul pentru a îmbunătăți performanța și eficiența sistemului, cum ar fi optimizarea algoritmilor existenți.
3. Îmbunătățirea aspectului vizual: Se poate îmbunătăți aspectul vizual al interfeței grafice, utilizând stiluri și teme mai moderne sau personalizate pentru a oferi o experiență estetică mai plăcută utilizatorilor.

6. Bibliografie

1. https://www.tabnine.com/code/java/methods/javax.swing.JPanel/paintComponent
2. https://www.youtube.com/watch?v=9CLE33qPIvI&ab_channel=CSHero
3. https://www.digitalocean.com/community/tutorials/java-filewriter-example