

DOCUMENTAȚIE

TEMA 2

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

NUME STUDENT: Lupou Krisztián-Róbert
GRUPA: 30226

CUPRINS

1.	Obiectivul temei.....	3
1.1.	Obiectiv principal.....	3
1.2.	Obiective secundare.....	3
2.	Analiza problemei, modelare, cerințe funcționale și non-funcționale.....	3
2.1.	Analiza problemei.....	3
2.2.	Modelare.....	3
2.3.	Cerințe funcționale și non-funcționale.....	4
3.	Proiectare	4
3.1.	Diagrama UML a pachetelor.....	4
3.2.	Diagrama UML a claselor.....	5
3.3.	Structuri de date.....	5
3.4.	Interfața utilizator.....	6
4.	Implementare	6
5.	Rezultate	10
6.	Concluzii	10
7.	Bibliografie	11

1. Obiectivul temei

1.1. Obiectiv principal

Obiectivul principal al proiectului este de a proiecta și implementa o aplicație de gestionare a cozilor care alocă clienții la cozi astfel încât timpul de așteptare să fie minimizat. Sistemul trebuie să simuleze un mediu în care N clienți sosesc pentru a fi serviți, intră în Q cozi, așteaptă să fie serviți și părăsesc cozile după ce au fost serviți. Prin utilizarea unui algoritm eficient de distribuire a clienților, aplicația urmărește să minimizeze timpul total de așteptare și să optimizeze utilizarea resurselor (servere).

1.2. Obiective secundare

- Monitorizarea în timp real: Permite utilizatorilor să vadă în timp real starea cozilor și a clienților în așteptare.
- Analiza performanței: Calcularea și afișarea timpului mediu de așteptare, timpului mediu de serviciu și determinarea orei de vârf.
- Flexibilitate și configurabilitate: Permite configurarea variabilelor de simulare, cum ar fi numărul de cozi, numărul de clienți și intervalele de timp de sosire și serviciu.
- Raportare: Generarea de rapoarte detaliate despre desfășurarea simulării și performanța sistemului de cozi.

2. Analiza problemei, modelare, cerințe funcționale și non-funcționale

2.1. Analiza problemei

Gestionarea eficientă a cozilor este esențială pentru a minimiza timpul de așteptare al clienților într-un sistem de servicii. Această problemă apare frecvent în diverse domenii, de la gestionarea liniei de așteptare în supermarketuri până la administrarea cererilor într-un sistem informatic. Obiectivul principal este de a distribui clienții în mod optim între mai multe cozi (servere) astfel încât să se reducă timpul total de așteptare.

2.2. Modelare

Modelarea proiectului utilizează o structură orientată pe obiecte, cu clasele Task, Server, Scheduler, SimulationManager, ViewRealTime, View și Controller, fiecare având un rol specific. Clasa Task reprezintă sarcinile cu un id, timp de sosire și timp de procesare, ordonabile după timpul de sosire. Clasa Server procesează sarcinile folosind o coadă sincronizată și un contor atomic pentru perioada de așteptare, gestionând procesarea sarcinilor în fire de execuție separate. Clasa Scheduler distribuie sarcinile către servere, alegând serverul cu cea mai scurtă perioadă de așteptare și inițializând serverele. Clasa SimulationManager gestionează întreaga simulare: generează sarcini aleatorii, le distribuie folosind Scheduler, monitorizează starea sistemului, scrie raportul simulării și calculează parametrii de performanță. Clasa ViewRealTime

oferă o interfață grafică pentru vizualizarea în timp real a simulării, iar clasa View permite utilizatorului să introducă parametri de simulare și să inițieze simularea. Clasa Controller leagă interfața grafică de modelul de simulare, inițializând simularea pe baza parametrilor introduși de utilizator.

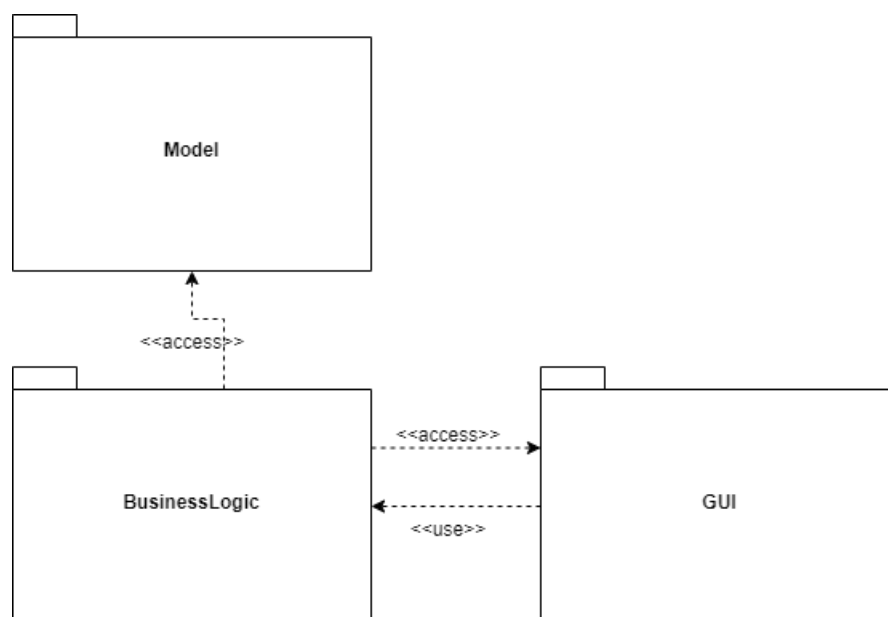
2.3. Cerințe funcționale și non-funcționale

Aplicația de management al cozilor trebuie să fie capabilă să genereze un număr N de clienți, fiecare caracterizat printr-un ID, un timp de sosire și un timp de serviciu, și să atribuie acești clienți cozii cu cel mai scurt timp de așteptare la momentul sosirii lor. Fiecare coadă va procesa clienții în ordinea sosirii, reducând timpul de serviciu până când fiecare client este servit complet. Aplicația trebuie să monitorizeze în timp real starea cozilor și a clienților în așteptare, afișând aceste informații într-o interfață grafică intuitivă și ușor de utilizat. În plus, aplicația trebuie să calculeze și să afișeze statisticile relevante, cum ar fi timpul mediu de așteptare, timpul mediu de serviciu și ora de vârf, și să genereze rapoarte detaliate ale simulării.

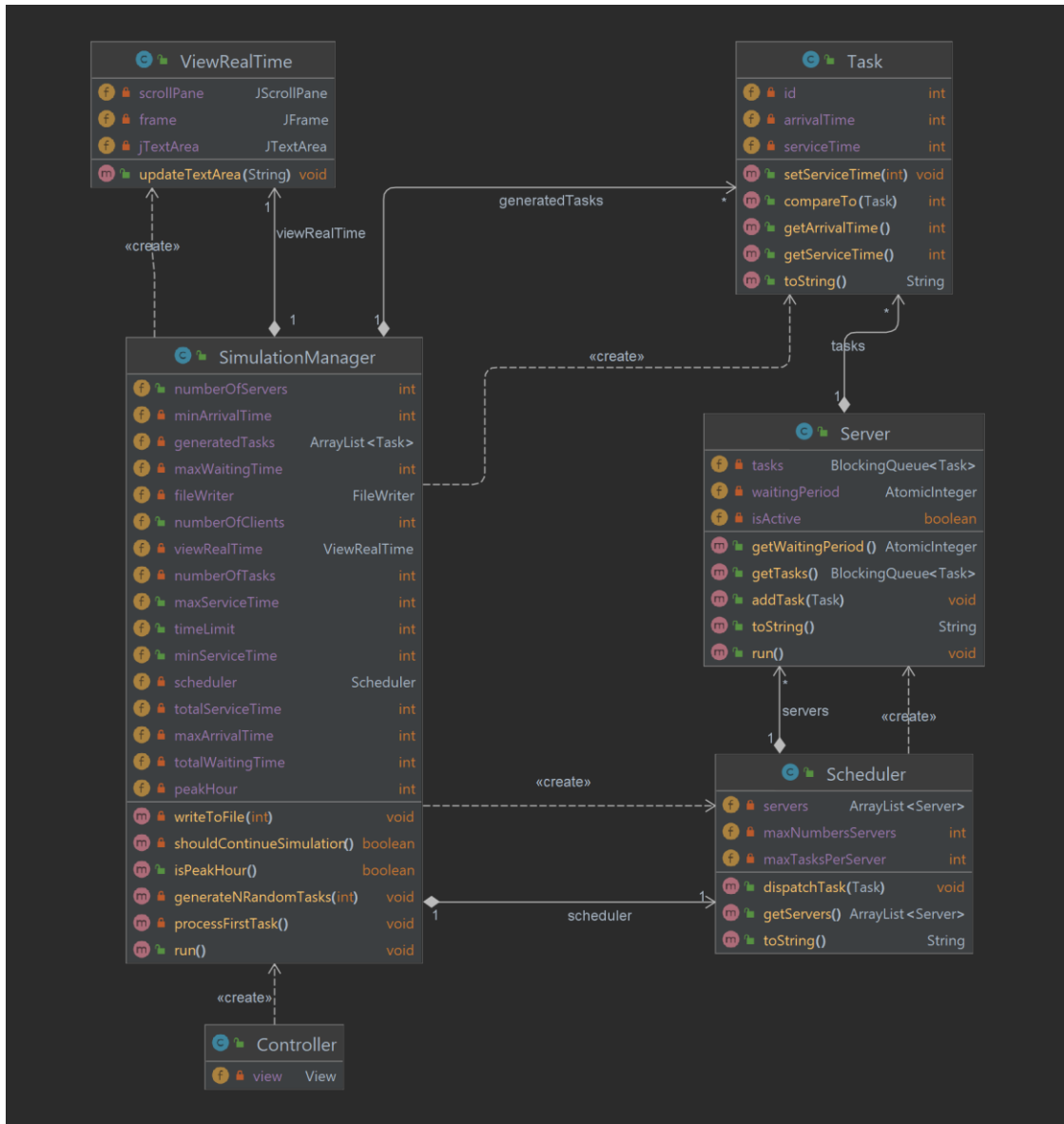
Din punct de vedere al cerințelor non-funcționale, aplicația trebuie să fie performantă, procesând și actualizând informațiile în timp real fără întârzieri semnificative. De asemenea, aplicația trebuie să fie scalabilă, fiind capabilă să gestioneze un număr variabil de cozi și clienți fără a compromite performanța. Flexibilitatea și configurabilitatea sunt și ele importante, permițând utilizatorilor să seteze variabilele de simulare în mod rapid și eficient. În cele din urmă, aplicația trebuie să includă facilități de raportare pentru a genera rapoarte detaliate despre desfășurarea simulării și performanța sistemului de cozi.

3. Proiectare

3.1. Diagrama UML a pachetelor



3.2. Diagrama UML a claselor



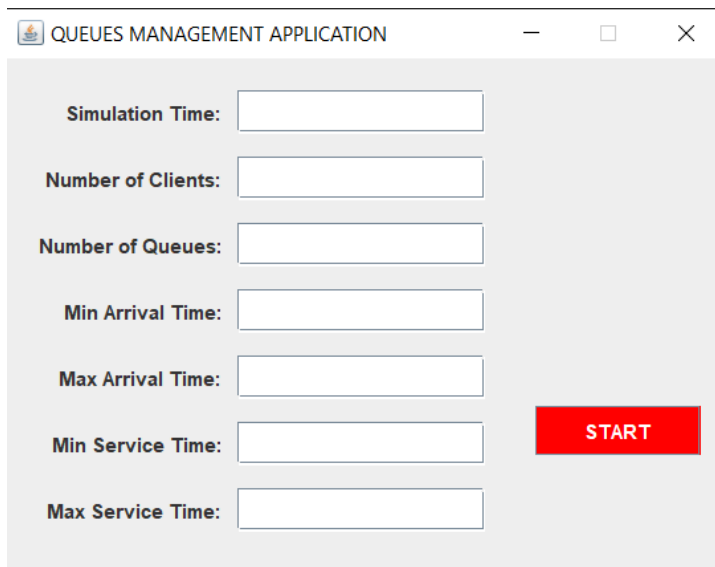
3.3. Structuri de date

În cadrul proiectului, ArrayList este utilizat pentru a gestiona lista de servere (servers) și lista de sarcini generate aleatoriu (generatedTasks). Această structură de date din Java oferă o flexibilitate crescută în adăugarea, accesarea și modificarea elementelor într-o colecție dinamică de dimensiuni cunoscute. În cazul clasei Scheduler, ArrayList permite să fie ținute în memorie mai multe instanțe de Server, fiecare dintre ele gestionând propriile sarcini într-o manieră independentă, dar coordonată. Pe de altă parte, BlockingQueue este utilizat în clasa Server pentru a gestiona coada de sarcini (tasks) care sunt procesate de fiecare server. Această implementare specifică din Java este ideală pentru a asigura operațiuni sigure și eficiente într-un mediu multi-

threaded, unde mai multe fire de execuție (thread-uri) pot adăuga și prelua sarcini în paralel. BlockingQueue garantează că operațiile de adăugare și preluare sunt sincronizate automat, protejând astfel integritatea datelor și prevenind condițiile de curse (race conditions).

3.4. Interfața utilizator

Pentru a realiza interacțiunea dintre problema noastră și utilizator, vom folosi o interfață grafică. Aceasta va conține 7 câmpuri de text unde utilizatorul va putea introduce datele necesare pentru rularea programului, respectiv: Timpul simulării, numărul de clienți, numărul de cozi, momentul minim în care un client poate ajunge în coadă, momentul maxim în care acesta ajunge în coadă, timpul minim și maxim în care clientul poate fi servit. Toate aceste date sunt de tip INT. În plus, avem un buton de START, care, după introducerea datelor, va porni programul și va afișa în timp real evoluția clienților în cozi.



4. Implementare

Clasele: Controller, View, ViewRealTime, Task, Server, Scheduler și SimulationManager.

Clasa Task

Clasa Task reprezintă sarcinile (clienții) care sosesc în sistem și sunt distribuite către servere pentru procesare. Aceasta implementează interfața Comparable pentru a permite sortarea bazată pe timpul de sosire. Atributele clasei sunt:

- id: identificator unic pentru fiecare sarcină.
- arrivalTime: timpul de sosire al sarcinii.
- serviceTime: timpul necesar pentru procesarea sarcinii.

Metodele principale ale clasei sunt constructorul Task pentru inițializarea obiectelor și metodele get pentru accesarea atributelor și setTime pentru modificarea timpului de serviciu.

Clasa Server

Clasa Server implementează logica de gestionare a sarcinilor într-o coadă folosind o implementare ArrayBlockingQueue pentru a asigura accesul sincronizat. Atributul waitingPeriod de tip AtomicInteger reprezintă perioada totală de așteptare pentru toate sarcinile din coadă. Metodele cheie includ:

- addTask(Task newTask): adaugă o nouă sarcină în coada serverului și actualizează perioada de așteptare.
- run(): gestionează execuția serverului într-un mediu simulat de gestionare a cozilor. Prin intermediul unei bucle while, serverul rămâne activ atâta timp cât variabila isActive este setată la true. În fiecare iterație, metodă încearcă să obțină prima sarcină din coadă folosind peek(). Dacă există o sarcină (currentTask != null), aceasta este procesată prin simularea timpului de servire utilizând Thread.sleep() și apoi eliminată din coadă cu poll(), actualizând în același timp perioada de așteptare. Dacă nu mai există sarcini în coadă, serverul se închide prin setarea lui isActive la false.
- Metoda toString() returnează o reprezentare textuală a stării curente a serverului.

Clasa Scheduler

Clasa Scheduler gestionează distribuirea sarcinilor către servere în funcție de cel mai scurt timp de așteptare. Atributele includ o listă de servere (servers) și parametrii pentru numărul maxim de servere și sarcini pe server (maxNumbersServers, maxTasksPerServer). Metodele principale sunt:

- dispatchTask(Task t): determină serverul cu cel mai scurt timp de așteptare și atribuie sarcina respectivă.
- toString(): oferă o reprezentare textuală a stării curente a fiecărui server.

Clasa SimulationManager

Clasa SimulationManager este responsabilă pentru simularea și gestionarea întregului proces de simulare a unui sistem de cozi în timp real. Aceasta coordonează interacțiunea între generatorul de sarcini, planificatorul (Scheduler) și monitorizarea stării sistemului în timpul execuției simulării. Vom explora fiecare metodă și funcționalitate a clasei pentru a înțelege cum este implementată simularea și cum sunt gestionate datele și starea sistemului.

Constructorul primește o serie de parametri care definesc parametrii simulării:

- timeLimit: Timpul maxim pentru care se va rula simularea.
- numberOfServers: Numărul de servere disponibile în sistem.
- numberOfClients: Numărul de clienți (sarcini) care vor fi simulate.

- minServiceTime, maxServiceTime: Timpul minim și maxim de service al sarcinilor.
- minArrivalTime, maxArrivalTime: Timpul minim și maxim de sosire al sarcinilor.

În constructor, se inițializează un obiect Scheduler care gestionează distribuirea sarcinilor către servere, se deschide un fișier pentru înregistrarea raportului de simulare și se generează sarcini aleatorii pentru simulare.

Metode:

- generateNRandomTasks(int numberOfClients): această metodă generează un număr specificat de sarcini aleatorii (Task) pe baza parametrilor specificați în constructor (minArrivalTime, maxArrivalTime, minServiceTime, maxServiceTime). Aceste sarcini sunt apoi sortate în funcție de timpul lor de sosire pentru a fi gestionate corect de către Scheduler.
- writeToFile(int currentTime): metoda înregistrează starea curentă a sistemului într-un fișier de jurnal (simulation_log.txt) și actualizează interfața vizuală în timp real (utilizând clasa ViewRealTime). Înregistrarea include timpul curent, lista sarcinilor așteptate și starea fiecărui server.
- processFirstTask(): această metodă gestionează procesarea primei sarcini din fiecare server în fiecare unitate de timp. Verifică dacă serverele au sarcini în așteptare și actualizează timpul de service al sarcinilor în funcție de durata de procesare. Dacă o sarcină este complet procesată (timpul de service devine 0), aceasta este eliminată din coadă, iar metricile cumulabile ale sistemului (timpul total de așteptare, numărul de sarcini finalizate) sunt actualizate.
- shouldContinueSimulation(): această metodă determină dacă simularea ar trebui să continue în funcție de starea actuală a sarcinilor și serverelor. Simularea continuă atâta timp cât există sarcini de procesat în servere sau sarcini de generat.
- isPeakHour(): detectează momentele de vârf în sistemul simulat de cozi. Aceasta funcționează în felul următor: inițializează o variabilă maxWaiting la începutul fiecărei verificări și parcurge fiecare server din lista de servere. Pentru fiecare server, adaugă timpul de service al fiecărei sarcini în așteptare la variabila maxWaiting. Apoi, compară valoarea maxWaiting cu pragul maxWaitingTime pentru a determina dacă durata totală de service a sarcinilor depășește limita precedentă. Metoda întoarce true dacă se identifică o oră de vârf și false în caz contrar.
- run(): această metodă coordonează întregul flux de execuție al simulării. În fiecare iterație a buclei, simularea verifică dacă există sarcini noi de procesat care ajung la timpul curent și le distribuie către serverele disponibile folosind Scheduler-ul. Apoi, înregistrează starea actuală a sistemului folosind metoda writeToFile cât și pe o interfață grafică în timp real. Metoda isPeakHour() este apelată pentru a detecta ora de vârf, în funcție de cantitatea și durata totală de service a sarcinilor în așteptare în servere. În fiecare secundă, procesul primei sarcini din fiecare server scade timpul de service, iar dacă acesta ajunge la zero, sarcina este eliminată folosind metoda processFirstTask. Simularea continuă până când timpul limită al simulării este atins sau nu mai există sarcini de procesat în servere. La finalizarea simulării, se calculează și se afișează parametrii de performanță cum ar fi media timpului de așteptare și de service, și ora de vârf identificată, oferind o evaluare completă a performanței sistemului.

Clasa Controller

Clasa Controller gestionează interacțiunea utilizatorului cu interfața grafică pentru simularea unui sistem de cozi. La inițializare, acesta ascultă evenimentul de start al interfeței grafice pentru a extrage și valida parametrii introduși de utilizator pentru simulare. Apoi, creează și lansează o instanță a clasei SimulationManager într-un fir de execuție separat pentru a gestiona logica de simulare.

Clasa ViewRealTime

Clasa ViewRealTime este utilizată pentru afișarea în timp real a informațiilor despre simularea sistemului de cozi. Ea gestionează actualizarea interfeței grafice în funcție de starea simulării, oferind o reprezentare vizuală a datelor relevante pentru utilizator.

În esență, ViewRealTime furnizează o metodă updateTextArea() pentru a actualiza textul afișat în interfața grafică în timp ce simularea rulează. Aceasta este utilizată de SimulationManager pentru a raporta starea curentă a sistemului de cozi, inclusiv informații precum timpul curent, sarcinile în așteptare în fiecare server, și alte detalii relevante pentru performanța simulării.

Scroll pane (JScrollPane) este utilizat pentru a gestiona textul din JTextArea, asigurând că utilizatorul poate naviga ușor prin conținutul afișat, indiferent de volumul de date prezentat în timp real în aplicație. Acest lucru contribuie la o experiență utilizator îmbunătățită și la o interfață grafică mai prietenoasă pentru utilizator.

Clasa View

Clasa View este o interfață grafică creată în Java Swing pentru a permite utilizatorului să introducă parametri necesari pentru simularea sistemului de cozi. Aceasta conține elemente precum etichete (JLabel) pentru descrierea câmpurilor de intrare și câmpuri de text (JTextField) pentru introducerea datelor. În plus, include un buton (JButton) "START" care inițiază simularea pe baza parametrilor introduși.

Interfața grafică are următoarele funcționalități:

- Etichetele (JLabel) sunt utilizate pentru a descrie fiecare câmp de intrare, cum ar fi "Simulation Time", "Number of Clients", "Number of Queues", etc.
- Câmpurile de text (JTextField) permit utilizatorului să introducă valorile pentru fiecare parametru al simulării, cum ar fi timpul de simulare, numărul de clienți, numărul de cozi, etc.
- Butonul "START" (JButton) inițiază simularea atunci când este apăsat de utilizator.
- Metoda start(ActionListener e) este folosită pentru a adăuga un ascultător de evenimente butonului "START", astfel încât să fie notificat când utilizatorul apasă butonul.
- Metodele getSimulationTime(), getNumberOfClients(), getNumberOfQueues(), getMinArrivalTime(), getMaxArrivalTime(), getMinServiceTime(), getMaxServiceTime() sunt utilizate pentru a obține valorile introduse de utilizator în câmpurile respective ale interfeței grafice.

5. Rezultate

Pentru verificarea rezultatelor ne-am folosit de fișiere generate pentru cele trei cazuri de testare din cerința proiectului. Datele și traseul clienților sunt înregistrate în fișiere pentru a ușura verificarea și pentru a permite accesul ușor la informații. În repository se pot găsi cele trei fișiere .txt: Test 1, Test 2 și Test 3 care reprezintă testele pentru valorile din tabelul de jos.

Test 1	Test 2	Test 3
N = 4 Q = 2 $t_{simulation}^{MAX} = 60 \text{ seconds}$ $[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 30]$ $[t_{service}^{MIN}, t_{service}^{MAX}] = [2, 4]$	N = 50 Q = 5 $t_{simulation}^{MAX} = 60 \text{ seconds}$ $[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 40]$ $[t_{service}^{MIN}, t_{service}^{MAX}] = [1, 7]$	N = 1000 Q = 20 $t_{simulation}^{MAX} = 200 \text{ seconds}$ $[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [10, 100]$ $[t_{service}^{MIN}, t_{service}^{MAX}] = [3, 9]$

6. Concluzii

O îmbunătățire posibilă ar fi adăugarea unei funcționalități pentru ordonarea serverelor în funcție de lungimea cozii lor, astfel încât să gestionăm mai eficient sarcinile. Aceasta ar permite distribuirea mai eficientă a sarcinilor și optimizarea resurselor.

În final, proiectul reprezintă o implementare eficientă a unui sistem de gestionare a cozilor, utilizând o interfață grafică pentru vizualizarea în timp real a simulării. Prin intermediul claselor Task, Server, Scheduler, SimulationManager, și interfeței ViewRealTime, am realizat o aplicație care permite simularea și monitorizarea performanței unui sistem de cozi.

7. Bibliografie

1. Interface BlockingQueue<E> -
<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html>
2. Java Threads - https://www.w3schools.com/java/java_threads.asp
3. Class JScrollPane -
<https://docs.oracle.com/javase%2F9%2Fdocs%2Fapi%2F%2F/javafx/swing/JScrollPane.html>
4. ASSIGNMENT 2 – SUPPORT PRESENTATION (PART 1) -
https://www.dsrl.eu/courses/pt/materials/PT_2024_A2_S1.pdf
5. ASSIGNMENT 2 – SUPPORT PRESENTATION (PART 2) -
https://www.dsrl.eu/courses/pt/materials/PT_2024_A2_S2.pdf