DOCUMENTATIE

TEMA 2

Nume student:VELICEA ANDREEA – IOANA

Grupa: 30228

Cuprins

Table of Contents

1. Obiectivul temei

i) Obiectivul principal

- Proiectarea si implementarea unei aplicatii menite sa analizeze sisteme bazate pe cozi prin (1) simularea a o serie de N clienti care ajung pentru a fi serviti, care intra in cozile Q, asteapta, sunt serviti si in cele din urma parasesc cozile si (2) calcularea timpului mediu de asteptare, a timpului mediu de serviciu si a orelor de varf.

ii) Obiectivele secundare

- Analizarea problemei si indentificarea cerintelor – capitolul 2
- Proiectarea aplicatiei de simulare – capitolul 3
- Implementarea aplicatiei de simulare – capitolul 4
- Testarea aplicatiei de simulare – capitolul 5

2. Analiza problemei si identiifcarea cerintelor

i) Cerintele functionale

- Aplicatia de simulare ar trebui sa permita utilizatorului sa seteze datele pentru simulare
- Aplicatia de simulare ar trebui sa permita utilizatorului sa porneasca simularea
- Aplicatia de simulare ar trebui sa afiseze in timp real evolutia cozilor
- Aplicatia de simulare ar trebui sa salveze intr un fisier evolutia cozilor

ii) Cerinte non functionale

- Aplicatuia de simulare ar trebui sa fie intuitiva si usor de folosit de catre utilizator
- Ar trebui sa afiseze intr un mod placut placut rezultatul pentru a putea fi inteles de catre utilizator

iii) Cazurile de functionare

Caz de functionare : configurarea simularii

Actor principal: utilizatorul

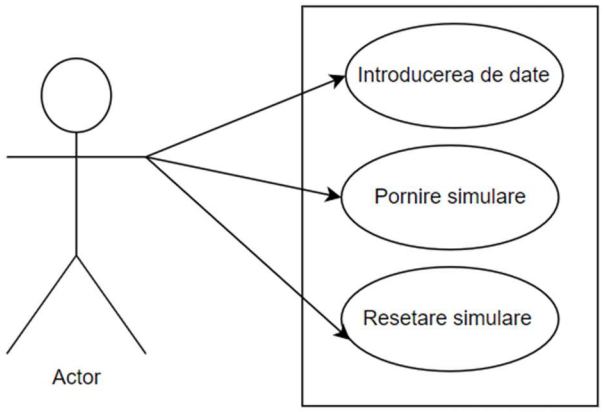
Scenariul principal de success:

- Utilizatorul insereaza valori pentru: numarul de clienti, numarul de cozi, timpul de simulare, timpul minim si maxim de sosire si timpul minim si maxim de servire
- Utilizatorul apasa pe butonul de RUN pentru validarea datelor
- Aplicatia valideaza datele si simularea incepe.

Secventa alternativa: Valori invalide pentru parametrii de simulare

- Utilizatorul introduce valori invalide pentru parametrii de configurarea a aplicatiei
- Aplicatia afiseaza un mesaj de eroare si cere utilizatorului inserarea de valori valide
- Scenariul se intoarce la primul pas.

iv) Diagrama



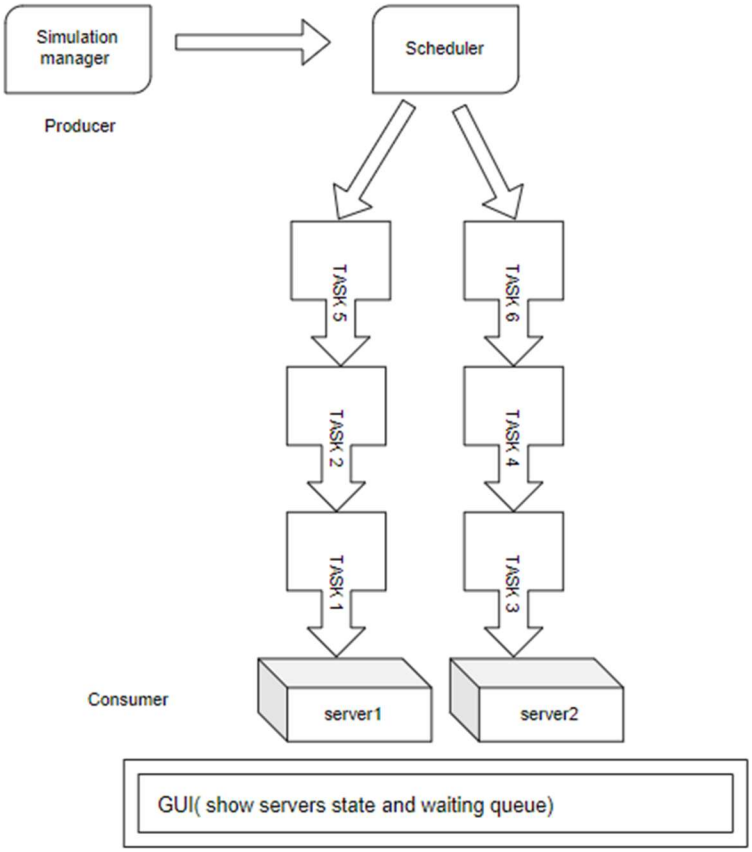
3. Proiectarea aplicatiei de simulare

a) Structuri de date folosite

- LinkedBlochingQueue<Task> pentru stocarea task-urilor unui anumit server
- ArrayList<Server> - pentru stocarea datelor despre server
- AtomicInteger – pentru stocarea timpului de asteptare la coada

b) Arhitectura conceptuala

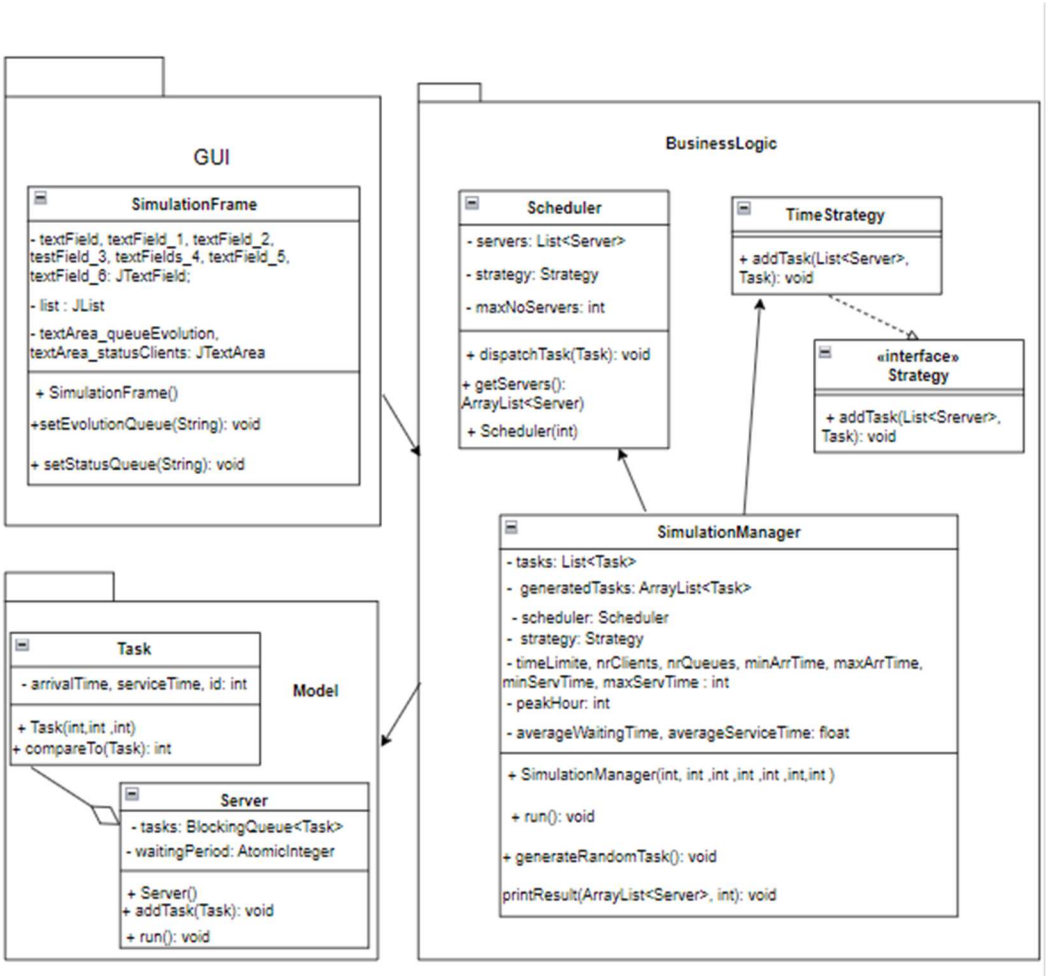
Am realizat diagrama conceptuala pentru a intelege mai bine ceea ce avem de implementat.



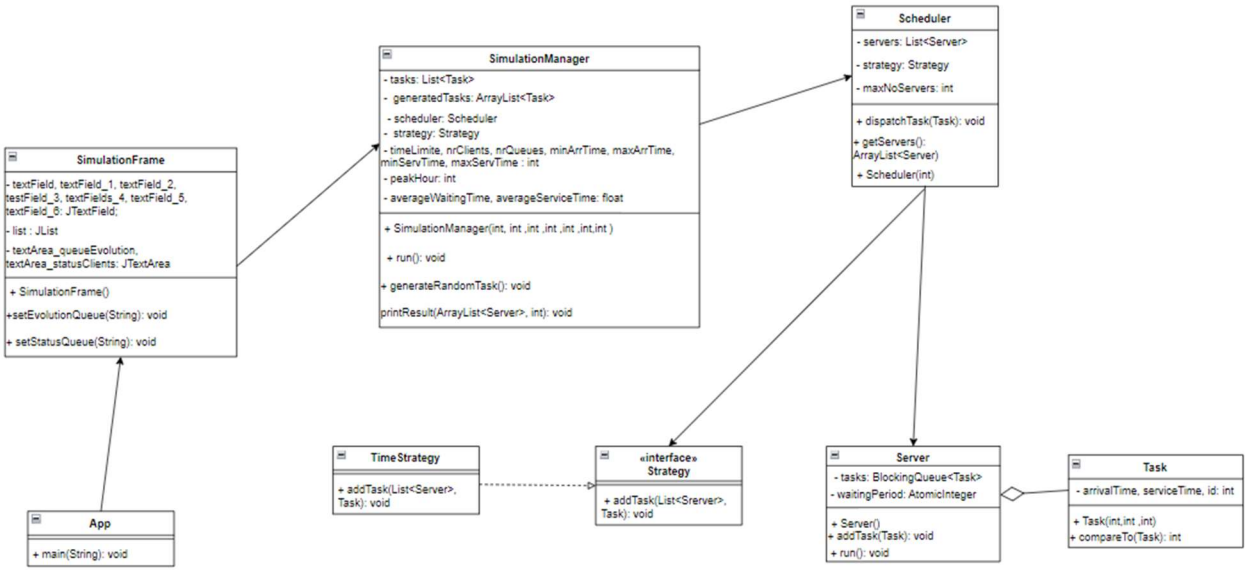
c) Divizarea in pachete

Am impartit clasele in 3 pachete:

- BussinesLogic
  - Scheduler
  - SimulationManager
  - Strategy (interface)
  - TimeStrategy
- Model
  - Server
  - Task
- GUI
  - SimulationFrame



d) Divizarea in clase



4. Implementarea

a) Clasa App

- Implementeaza metoda main(String) : void , unde este apelat constructorul interfetei

```
-   public static void main( String[] args ){
          EventQueue.invokeLater(new Runnable() {
       public void run() {
          try {
             SimulationFrame frame = new SimulationFrame();
             frame.setVisible(true);
          } catch (Exception e) {
             e.printStackTrace();
          }
       }
    });
      }
```

b) TimeStrategy

- Implementeaza metoda addTask(List<sServer>, Task) care adauga task-ul current la coada cu cel mai mic timp de asteptare

```
-   public void addTask(List<Server> servers, Task t) {
        int minTime = Integer.MAX_VALUE;
        Server serverMin = servers.get(0);
        for(Server s: servers){
            if(s.getWaitingPeriod().get() < minTime){
                serverMin = s;
                minTime = s.getWaitingPeriod().get();
            }
        }
        serverMin.addTask(t);}
```

b) Scheduler

- Are ca si atribute :
  - List<Server> servers
  - maxNoServers: int
  - Strategy: strategy
- Ca si metode are constructorul, metoda dispatchTask care apeleaza metoda addTask implementata in TimeStrategy

```
-   public void dispatchTask(Task t){
        this.strategy.addTask(this.servers,t);
    }
```

c) Task

- Are ca si atribute:
  - arrivalTime, serviceTime, id: int
- Are ca si metode:
  - Constructor
  - Setter si Getter pentru fiecare atribut
  - ToString
  - compareTo(Task) – ne ajuta la sortarea task urilor in functie de timpul de sosire
- toString (String) :

```
-   public String toString()
    {
        return "Task (" + "id= "+ id + ", arrivalTime = " +
```

```
    arrivalTime + ", serviceTime= " + serviceTime + ")";
    }
```

- compareTo(Task):

```
-   public int compareTo(Task o) {
        return this.getArrivalTime()-o.getArrivalTime();
    }
```

d) Server

- Are ca si atribute:
  - Tasks: BlockingQueue<Task>
  - waitingPeriod: AtomicInteger
- Are ca si metode:
  - Constructor
  - Setter si Getter pentru fiecare atribut
  - toString()
  - addTask(Task)
  - run()
- metoda addTask(Task):

```
-   public void addTask(Task newTask){
        tasks.add(newTask);
        waitingPeriod.addAndGet(newTask.getServiceTime());
    }
```

- metoda run():

```
-   public void run() {
        Task item = new Task(0,0,0);
        while(true){
            if(tasks.size()>0) {
                item = tasks.peek();
                if(item!=null) {
                    try {
                        Thread.sleep(1000 *
    item.getServiceTime());
                        tasks.remove(item);

    this.waitingPeriod.set(this.getWaitingPeriod().get() - 1);
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }

                }
            }
        }
    }
```

- metoda toString()

```
-   public String toString(){
        StringBuilder allClients = new StringBuilder();
        if(tasks.isEmpty()){
            return "closed";
        }
        for(Task task: tasks){
            allClients.append(task.toString()+"\n");
        }
        return allClients.toString();
    }
```

e) SimulationManager

- Are ca si atribute:
  - Int : timeLimit, nrClients, nrQueues, minArrTime, maxArrTime, minServTime, maxServTime
  - Int : peakHour
  - Float : averageWaitingTime, averageServiceTime
  - List<Task> tasks
  - Scheduler : scheduler
  - ArrayList<Task> generatedTasks
  - Strategy : strategy
- Iar ca metode are :
  - Constructorul
  - Setter si Getter pt fiecare atribut
  - generareRandomTask(): void – genereaza random un numar de nrClients clienti in functie de minArrivalTime si maxArrivalTime si minServTime si maxServTime
  - run() : void
  - printResult(ArrayList<Server>, int) : void – pentru afisarea solutiei in consola
- metoda generateRandomTask

```
-   private void generateRandomTask(){
        this.generatedTasks = new ArrayList<>();
        for(int i=0;i<nrClients;i++) {
            Task task = new
    Task((int)Math.floor(Math.random()*(maxArrTime-
    minArrTime+1)+minArrTime),
    (int)Math.floor(Math.random()*(maxServTime-
    minServTime+1)+minServTime),i);
            generatedTasks.add(task);
        }
        Collections.sort(generatedTasks);
```

```
        averageServiceTime = 0;
        for(Task t: generatedTasks){
            averageServiceTime += t.getServiceTime();
        }
        averageServiceTime /=(double) generatedTasks.size();
    }
```

- metoda run()

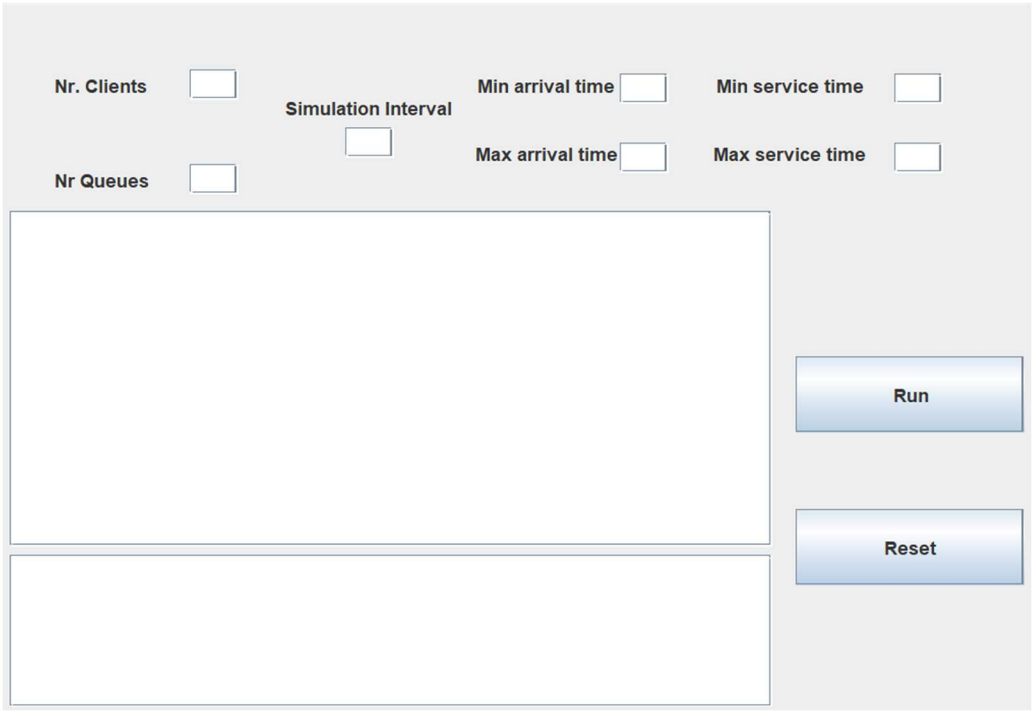
```
-   public void run() {
            int currentTime = 0;
            int maxNrClient = Integer.MIN_VALUE;
            try {
                FileWriter writer = new FileWriter("log.txt");
                StringBuilder evolutionQueue = new
    StringBuilder();
                StringBuilder statusQueue = new StringBuilder();
                peakHour = Integer.MIN_VALUE;
                while(currentTime < timeLimit ) {
                    evolutionQueue = new StringBuilder();
    //                waitingQueue = new StringBuilder();
                    writer.write("Time "+currentTime+"\n");
                    //evolutionQueue.append("Time
    "+currentTime+"\n");
                    int i=0;

                    while(i<generatedTasks.size()) {
                            if
    (generatedTasks.get(i).getArrivalTime() == currentTime) {

    scheduler.dispatchTask(generatedTasks.get(i));
                                generatedTasks.remove(i);
                                i=0;
                            }else{
                                i++;
                    }
                    writer.write("Waiting clients: \n");
                    evolutionQueue.append("Time "+currentTime+":
    \n");
                    evolutionQueue.append("Waiting clients: \n");
                    for(Task t: generatedTasks){
                        writer.write(t + ";");
                        writer.write("\n");
                        evolutionQueue.append(t+";");
                        evolutionQueue.append("\n");
                    }
                    writer.write("\n");
                    evolutionQueue.append("\n");
                    int id =0;
                    int nrClienti =0;
                    for(Server s: scheduler.getServers()){
                        writer.write("Queue "+(id+1 )+ ": \n");
                        evolutionQueue.append("Queue "+(id+1)+":
    \n");
                        id++;
                        writer.write(s.toString());
                        evolutionQueue.append(s.toString());
                        writer.write("\n");
```

```
                    evolutionQueue.append("\n");
                    nrClienti += s.getTasks().size();
                    if(s.getTasks().size()!=0)
                        averageWaitingTime +=
s.getWaitingPeriod().intValue();
                }
                if(nrClienti > maxNrClient){
                    maxNrClient = nrClienti;
                    peakHour = currentTime;
                }

                writer.write("\n");
                evolutionQueue.append("\n");
                printResult(scheduler.getServers(),
currentTime);
                writer.write("-------------------------------
-------------------------\n");
                evolutionQueue.append("----------------------
----------------------\n");
                Thread.sleep(1000);
                currentTime++;
 SimulationFrame.setEvolutionQueue(evolutionQueue.toString());
            }
            averageWaitingTime /= nrQueues*timeLimit;
            writer.write("Average service time :"
+averageServiceTime+"\n");
            writer.write("Average waiting time: "+
averageWaitingTime+"\n");
            writer.write("PeakHour: "+peakHour);
            statusQueue.append("Average service time :"
+averageServiceTime+"\n");
            statusQueue.append("Average waiting time: "+
averageWaitingTime+"\n");
            statusQueue.append("PeakHour: "+peakHour);
 SimulationFrame.setStatusQueue(statusQueue.toString());
         writer.close();
      }catch(Exception e){
          e.printStackTrace();
      }

  }
```

f) SimulationFrame

- Are ca si atribute:
  - textField, textField_1, textField_2, textField_3, textField_4, textField_5, textField_6 : JTextField
  - textArea_queueEvolution, textArea_statusClients: JTextArea
- Are ca si metode:
  - Constructorul

- setStatusQueue – seteaza valorile finale averageServiceTime, averageWaitingTime, peakHour
- setEvolutionQueue – seteaza evolutia cozilor in timp real, afisand clientii care asteapta si fiecare coada si clientii ei



-
- Contine de asemenea
  - 2 butoane:
    - Run – care porneste simularea
    - Reset – permite resetarea aplicatiei si reintroducerea altor date
  - 7 label-uri reprezentative fiecarei valori de intrare

5. Rezultate

Testarea a fost facute pe urmatoarele teste, iar stocarea rezultatelor s-a facut in fisierele test1.txt, test2.txt si test3.txt

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

Rezultatele fiecarui test sunt:

- Test1:
  - Average service time :3.5
  - Average waiting time: 0.55833334
  - PeakHour: 6
- Test2:
  - Average service time :3.54
  - Average waiting time: 9.996667
  - PeakHour: 29
- Test3:
  - Average service time :5.966
  - Average waiting time: 171.7055
  - PeakHour: 100

6. Concluzii

Din aceasta tema am invatat cum sa folosesc si sa sincronizez thread-urile, de asemenea am invatat sa folosesc structure de date ca si BlockingQueue si sa folosesc Atomic Integer.

O actualizare ulterioara a acestui proiect este, adaugarea unei alte strategii pentru adaugarea clientilor in coada, cum ar fi in functie de cea mai scurta coada.

7. Bibliografie

- https://dsrl.eu/courses/pt/
- https://app.diagrams.net/
- https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html
- https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html