

QUEUES MANAGEMENT

DOCUMENTATION - ASSIGNMENT 2

Programming Techniques

STUDENT: DOMȘA-STÎNĂ ANDA

GROUP: 30423

# TABLE OF CONTENTS

[1. Objectives 4](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297885)

[2. Analizing the requirements 4](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297886)

[3. Design 7](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297887)

[4. Implementation 11](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297888)

[5. Results 14](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297889)

[6. Conclusions 14](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297890)

[7. Bibliography 15](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297891)

# Objectives

1. Main objective

*Project specification:*

Design and implement an application aiming to analyze queuing-based systems by (1) simulating a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues, and (2) computing the average waiting time, average service time and peak hour . Sub-objectives:

The purpose of the project is to create a program for simulating queues, so that people can make queuing more efficient. The problem of waiting in line is a real and common problem nowadays, and of course this can become very irritating for each person, and therefore, all sorts of solutions are being sought to reduce the time of standing in line more than it is only necessary, but this topic being just a simulation, this problem can be solved only virtually, in hope that in the future this simulation will be developed in a real solution for this problem, especially as technology seems to advance more and more. Some very good examples where time is allocated to the state in the queue would be: ATMs, where people want to perform various operations, or cash registers, but whatever example we would look at, the human nature is to look at the queue that goes the fastest, and move there to finish the process in a timely manner.

*Analyze the problem and identify requirements*

The first step in implementing the program is to understand the problem, to identify the tasks it should perform and to model its functionalities using an object-oriented approach. This will be forward detailed in chapter 2.

*Design the simulation application*

The next step in solving this project’s requirements is to design the queues management application by dividing it into smaller components using structural diagrams. This will be further discussed in chapter 3.

*Implement the simulation application*

Now that the requirements are understood and modeled, the next step is to implement the actual java program by writing code for the needed classes in Intellij. This step will be covered in chapter 4.

*Test the simulation application*

The last step of the project development is to test the application, a process which I have described in chapter 5 of the documentation.

# Analyzing the requirements

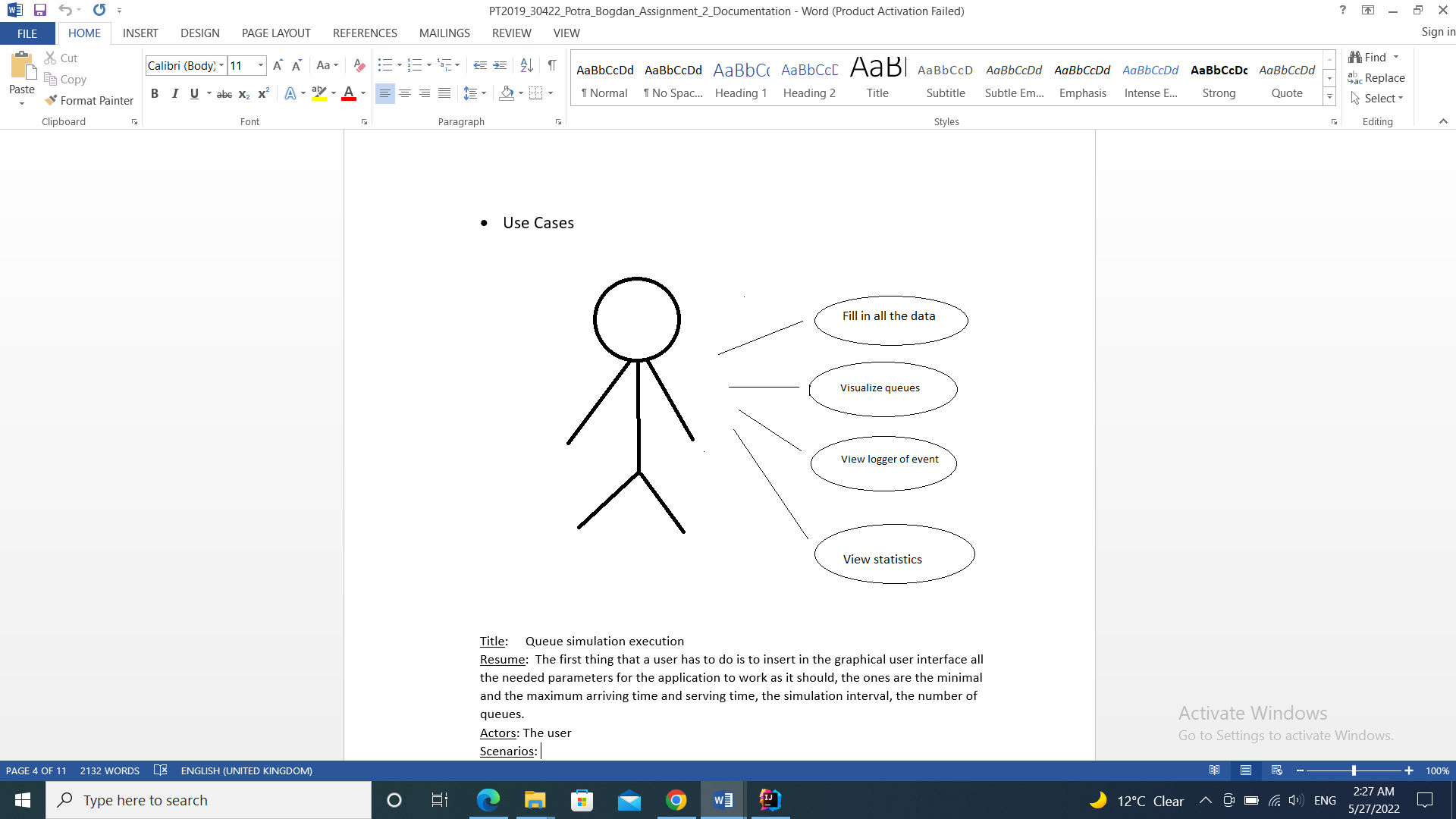
The first step in implementing the program is to understand the problem, to identify the tasks it should perform and to model its functionalities using an object-oriented approach.

The requirement of the project is to implement an application that systematizes some queues to which customers are added, so that the waiting time is minimal.

The queues management application should simulate by defining a simulation time (𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛), a series of N customers who arrive to be served, enter Q queues, wait, are served and finally leave. All clients are generated when the simulation is started and are characterized by three parameters: ID (a number between 1 and N), 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 (simulation time when the queue is ready to enter) and 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 (time interval or duration required to serve the client/ waiting time when the customer is in front of the queue). The application tracks the total time spent by each customer in order to queue it and calculates the average waiting time. Each customer is added to the queue with the minimum waiting time when his time 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 is greater than or equal to the simulation time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 ≥ 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛). The following data should be considered as input data for the application that should be entered by the user in the user interface of the application: - Number of customers (N); - Number of queues (Q); - Simulation interval (𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋); - Minimum and maximum arrival time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ≤ 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 ≤ 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋); - Minimum and maximum serving time (𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ≤ 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 ≤ 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋).

The user interacting with this application should be able to enter the input data mentioned above, see how the time increases during the simulation and stops when the simulation stops meaning that it is over and all customers have been served; to be able to see how the customers enter and leave the queue when their serving time has ended, and at the end, how the whole simulation was written in 3 files corresponding to the 3 cases to be tested.

**Use cases:**



The user can set:

* The maximum number of queues available to process customers.
* Minimum and maximum arrival interval: the delay between customers arriving to receive a service (in seconds)
* Minimum and maximum service time: the number of seconds needed for a client to be processed, a value is chosen randomly
* Simulation interval: the starting and finishing time of the simulation.

The user can read

* The total “dead time” for each queue (in seconds)
* The average waiting time for every queue (in seconds)
* The “peak hour”, when the most clients were served
* The average service time length (in seconds)
* The number of clients and the average waiting time for a user given interval

Use case example: Queue simulation execution - normal

Actor: The user

Scenario:

The user has to introduce all the data, I assume that all the introduced data is correct and it contains only digits. If the user has perfectly introduced all the required data inputs and it presses the “Start” button, the application is displaying the log and the real time evolution of the queues. After this he/she can look in the output file to see the statistics and the data which has been calculated during the serving process

Use case example: Queue simulation execution – bad input

Actor: The user

Scenario:

The user has to introduce all the data, I assume that all the introduced data is correct and it contains only digits. If the user doesn’t fill in all the input boxes the application will throw an error and the simulation will not start.

For a better understanding, the first given data set will be explained below:

|  |  |
| --- | --- |
| Changing time | Explenation |
| 1. Time: 0  Customers :  (2,2,3)  (1,7,2)  (4,15,4)  (3,30,4)  Queue1: -  Queue 2: - | As it can be seen, customers who each have an id, arrival time and serving time, are ordered in ascending order by arrival time. At the moment both queues are unoccupied, because the arrival time of the clients does not correspond to the current time, so that they can go to the queue. |
| ... |  |
| 2.Time: 2  Customers:  (1,7,2)  (4,15,4)  (3,30,4)  Queue1:(2,2,3)  Queue 2: - | You can see how the first customer waiting above went to the first queue, with the arrival time equal to the current time. |
| 3.Time: 3  Customers :  (1,7,2)  (4,15,4)  (3,30,4)  Queue 1:(2,2,2)  Queue 2: - | The customer's serving time decreases, and when he reaches 0, he can leave the queue. |
| ... |  |
| 4.Time: 30  Customers:  Queue 1:(3,30,4)  Queue 2: - | Here you can see how the last customer to be served went in line. |
| 5. Time: 34  Customers: -  Queue 1: -  Queue 2:- | The last customer waiting in line left. |

# Design

The next step in solving this project’s requirements is to design the queues simulator by dividing it into smaller components using structural diagrams. In this chapter, we will also present the OOP design of the application, the UML diagram, as well as the data structures and algorithms used.

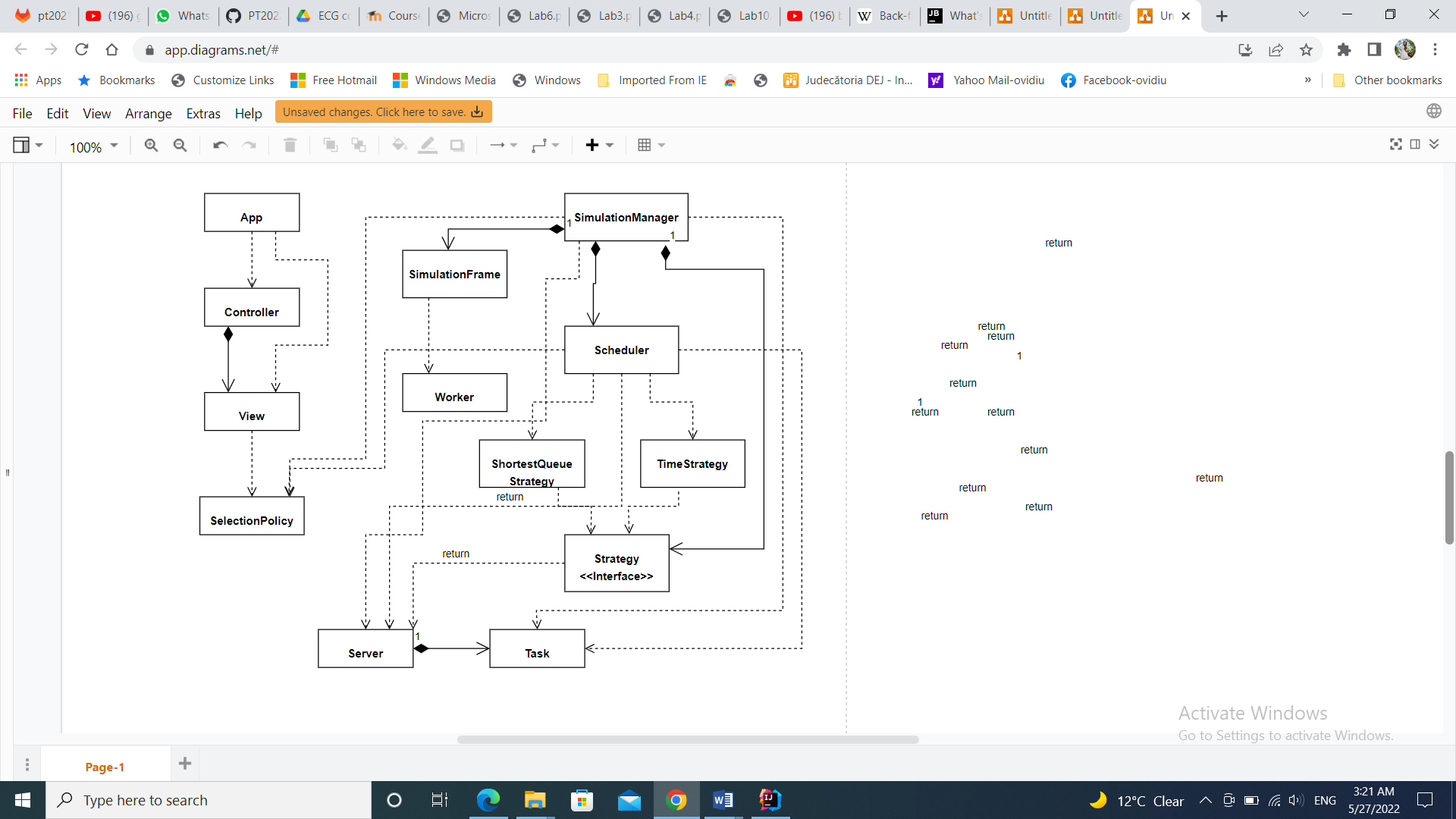
**Packages:**

The application is organized in three packages: Model, GUI and Logic, that group together related classes and interfaces.

The model package contains the classes “Server” (that represents the queues) and “Task” (that represents the clients).

The logic package contains the classes that are of importance for the simulation, like “SimulationManager” or “Scheduler”.

The GUI package contains the graphical user interfaces and the controller associated.



# Implementation

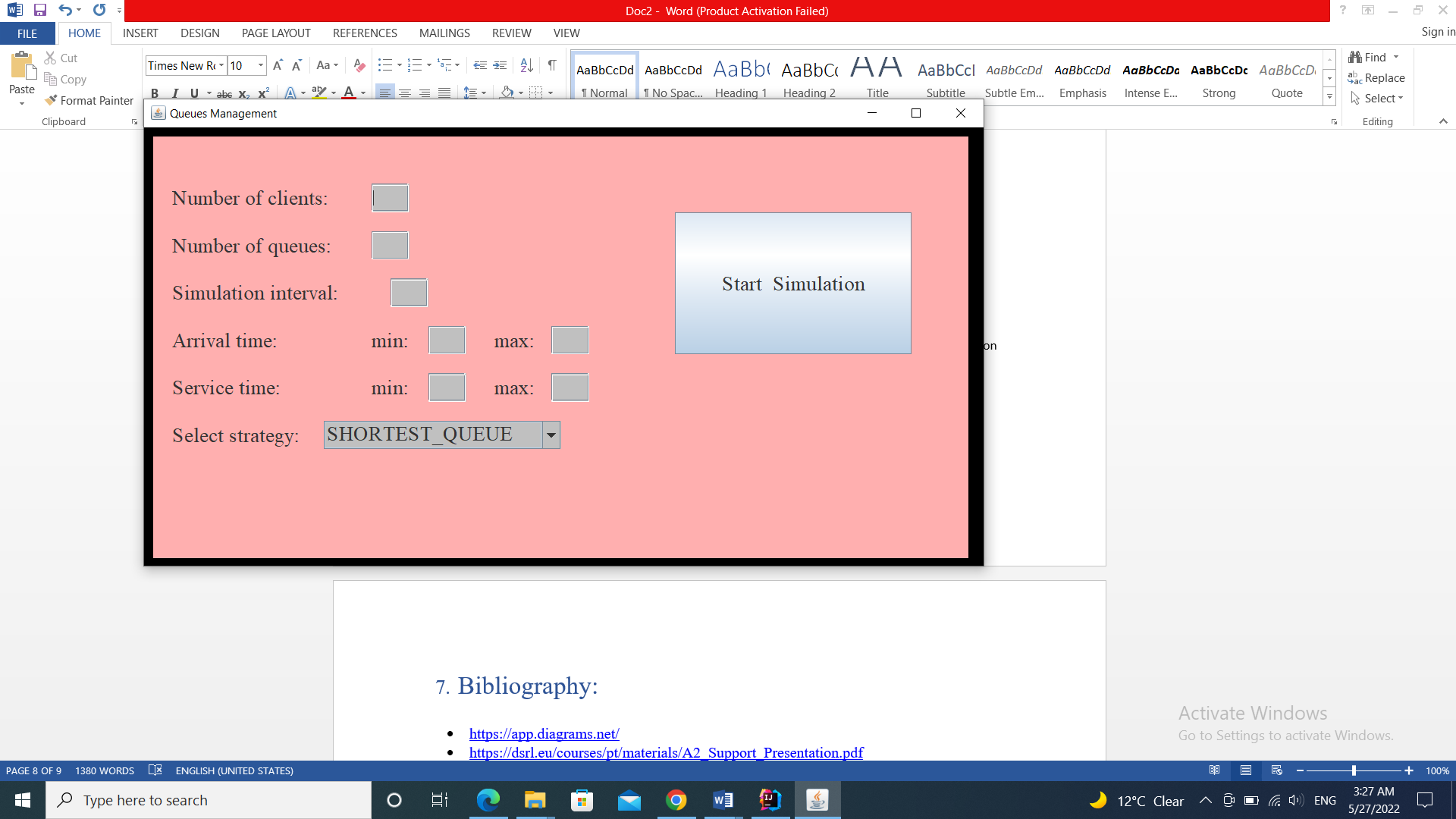
I started the implementation by working on the “Task” and “Server” classes, both of them being found in the “model” package.

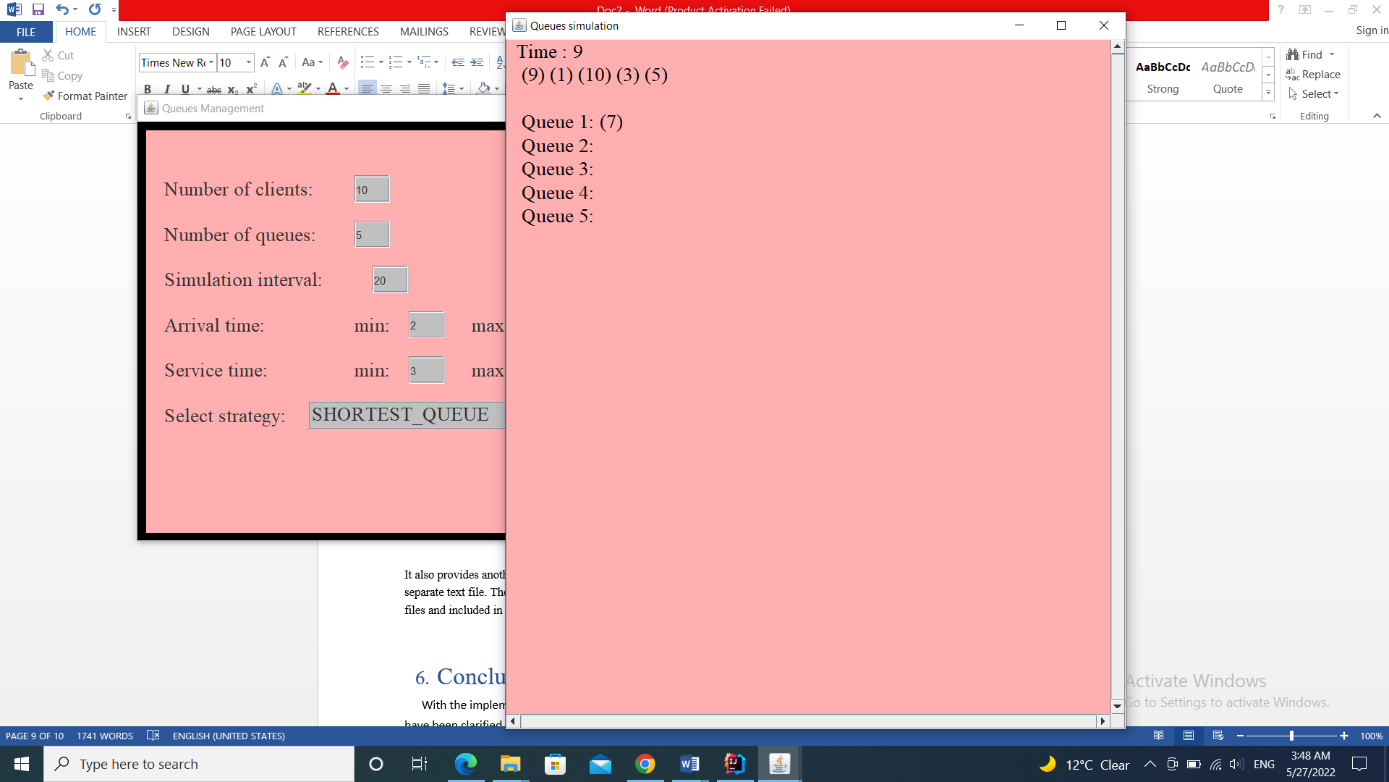
**Classes:**

* *Task* - This class stores the customer data such as ID, arrival time, serving time (how long it must be first in line until it is processed) etc. It has a constructor with parameters and it also contains getters and setters.
* *Server -* This class models the data using a queue that contains multiple clients (tasks). It has as fields a structure of type BlockingQueue <Task> and also a variable of type AtomicInteger that retains the waiting period corresponding to the respective queue. Each queue has a unique ID associated with it to make it easier to identify. At the same time, each queue corresponds to a thread that has the behavior described by the run () method, specific to the Runnable interface, implemented by this class.An important method is the addTask () method that deals with adding a client to the task list corresponding to the queue. Also, it implements the Runnable interface and is used to update the queues from the GUI.
* *SimulationManager –* This class is one of the more importantant classes of the project because it is the core that makes the simulation work and links all the other classes together. The generatedNRandomTasks () method generates N (from the user) number of random tasks that meet user-specified specifications. At the same time, this method deals with sorting the generated tasks according to their arrival time. The run () method is the most important method as it specifies the behavior of the thread running the simulation.
* *Scheduler* – This class sends the tasks to the Server, depending on the strategy chosen by the user. It contains application data, such as the list of existing servers, the number of servers, but also the number of tasks on a server.
* *SelectionPolicy* – Enumeration containing the two types of strategies that can be chosen in the simulation process.
* *ShortestQueueStrategy –* This class implements the method “addTask” for the case in which the strategy chosen is the shortest queue strategy.
* *TimeStrategy -* This class implements the method “addTask” for the case in which the strategy chosen is the shortest time strategy.
* *Strategy -* Interface containing the addTask method. It is implemented by both the ShortestQueueStrategy class and the TimeStrategy Class.
* *Controller –* Thisn is the class that links the bussniss logic classes to the GUI classes and sets the action listeners for the user interface.
* *Worker –* This class implements the SwingWorker class and it is used update the text area in which the simulation takes place.
* *SimulationFrame –* This class implements the frame in which the actual queues simulation takes place.
* *View –* It is the class that implements the main graphical user interface of the application. It implements the window in which the user can input data and start the simulation.
* *App –* This class contains the main() method which starts the controller and so, the application of simulating the queues service.

# Results

As presented, the application is a user-friendly one, as it provides the user an interface where he or she can input the data for the simulation and it signals if any errors occurred while introducing the details through a dialog text box that lets the user know where the error is.





It also provides another window that holds the simulation of the queues and writes the text logs and the statistics in a separate text file.

Here the user can see the real time simulation as the timer goes down and the random generated clients get in lines at the server queues and get dispatched when their turn comes.

The user can also choose between the shortest queue strategy or the shortest time strategy of the simulation.

At the end, he can see the statistics in the generated test file.

The three cases that were provided were tested and their results have been printed in three different text files and included in the project. For example, these are the first steps of the text log for the first given test details:

Given: N = 4, Q = 2, 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛: 𝑀𝐴𝑋 = 60 seconds

[𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁, 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙𝑀𝐴𝑋 ] = [2, 30]

[𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁, 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [2, 4]

Time 0  
Waiting clients:   
(2, 14, 4); (3, 15, 3); (1, 19, 3); (4, 30, 4);   
Queue: 1: Empty  
Queue: 2: Empty  
  
Time 1  
Waiting clients:   
(2, 14, 4); (3, 15, 3); (1, 19, 3); (4, 30, 4);   
Queue: 1: Empty  
Queue: 2: Empty  
  
Time 2  
Waiting clients:   
(2, 14, 4); (3, 15, 3); (1, 19, 3); (4, 30, 4);   
Queue: 1: Empty  
Queue: 2: Empty  
  
Time 3  
Waiting clients:   
(2, 14, 4); (3, 15, 3); (1, 19, 3); (4, 30, 4);   
Queue: 1: Empty  
Queue: 2: Empty  
  
Time 4  
Waiting clients:   
(2, 14, 4); (3, 15, 3); (1, 19, 3); (4, 30, 4);   
Queue: 1: Empty  
Queue: 2: Empty  
  
Time 5  
Waiting clients:   
(2, 14, 4); (3, 15, 3); (1, 19, 3); (4, 30, 4);   
Queue: 1: Empty  
Queue: 2: Empty  
  
Time 6  
Waiting clients:   
(2, 14, 4); (3, 15, 3); (1, 19, 3); (4, 30, 4);   
Queue: 1: Empty  
Queue: 2: Empty

# Conclusion

With the implementation of this application, the concepts related to Threads and their synchronization have been clarified much better. In terms of further development possibilities, the graphical interface can be improved so as to simulate in a more realistic way the arrival of customers, adding them to the queues, respectively removing them from the queue, as well as better synchronization of threads.

# Bibliography:

* <https://app.diagrams.net/>
* <https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>
* <https://google.github.io/styleguide/javaguide.html>
* <https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/packagediagram.html>
* [www.stackoverflow.com](http://www.stackoverflow.com)
* [www.wikipedia.org](http://www.wikipedia.org)
* <https://www.w3schools.com/java/java_threads.asp>
* <https://www.tutorialspoint.com/java/java_multithreading.htm>
* <https://www.draw.io/> (for diagrams)