Graduation Project Report

for

A GUI-based Software Package for the Simulation of Queuing Networks

Prepared by
Fatma Erem Aksoy – 2315075
Ece Erseven – 2385383
Ceazar Hatokah – 2456119

Middle East Technical University Northern Cyprus Campus
Computer Engineering

Supervised by Enver Ever

06/17/2023

Contents

1 Introduction

1.1 Purpose

This project aims to develop a GUI-based tool (web application) to be used in queue simulations, so the users of the application can evaluate the queue simulations by making the software easier to use with the help of GUI. Rather than having command-line-based simulation platforms, a GUI-based software platform is handier for users. They can see the process and make the necessary changes easily when a problem occurs, so they can save so much time during the simulation process. There are some existing GUI-based simulation software in the current industry, but we want to create a more efficient one in terms of the interface. For instance, one of the existing toolbox qnetworks (Marzolla, n.d), which is a free software package for QN (queuing networks) analysis for GNU Octave (Marzolla, 2012), is compatible with MATLAB, not with C/C++. Although it is available on many operating systems (Windows, MacOSX, and most Unix variants), execution time could have been better if it was also compatible with other languages. So, for this reason, we used Python as an implementation language for the back-end operations, such as queue simulation and statistical calculations, and HTML, JavaScript, and CSS for the front-end to implement the GUI part. The difference in our project is that it is planned to be a web application that is different from the existing packages in the current industry. Some of the advantages of having a web application for queuing simulations can be listed as the ease of use for users as they will not have to download any extra package or tool to run the simulations, ease of availability as this application only requires a stable internet connection (other than the basic knowledge needed) and the fact that it does not consume any memory space as no extra installations are required for the execution process. We decided to use Python for the back-end because Python has many libraries and tools, and it supports a variety of features, making integrating the back-end and front-end very easy by using Flask. It provides a solution to service traffic by using drag-and-drop GUI simulation. The system can be used to create a simulation for banks to handle money transaction traffic or emergency services, such as hospital queues to be seen by the doctor or for any other medical operation. It can be used to create a simulation for websites, especially e-commerce-related ones; in case of an exceeding number of requests to reach the website, the system will handle the queuing efficiently (Software Solutions Studio, 2021). In addition, the system can create a simulation for delivery queues to investigate delayed email delivery and deal with the problems that occur.

1.2 Scope

The scope of our project is to provide and help the users use a queueing simulation tool for different types of systems by creating a GUI for the execution of these simulations using a web-based platform. The GUI eases the interface between the user and the simulation so that the users will not face any difficulties running the queuing simulations; in addition to that, our project is designed in a way to attract the users and make it simple for them to use by including:

- Drag and drop feature allows users to easily select the elements they want (queue, server(s) and a connection) from the Toolbox.

- A selection page for the user to select which simulation type they would like to run (M/M/1, M/M/1/L, M/M/C, M/M/C/L).
- The clickable feature allows the user to click the items and enter the inputs directly. For queue: arrival rate, number of entities, the queue size limit for M/M/1/L and M/M/C/, and for server: service rate.
- Calculating the MQL value for the selected simulation type using the simulation approach and verifying the result by comparing it with the analytical approach by coding.
- Plotting the MQL values in a line graph and stopping the simulation when the MQL value calculated by simulation is verified as close enough to the one calculated using the analytical formulas.

1.3 Identification of constraints

Our project has some constraints which are the following:

- Time for simulation as the simulation should stop exactly when MQL results for analytical and simulation approaches are close enough to each other.
- Number of entities (number of iterations) should be at least 1024 since in order to get the accurate results for MQL, we need to iterate through at least 1024 times or else results will not be accurate.
- Users with disabilities such as blind users will not be able to use the simulation. No safety will be considered for these types of users.
- Accuracy value between the simulation and analytical results should not exceed a 0.25 margin, meaning that the results are close enough to be considered as accurate.
- System will not accept negative values for any of the inputs as they are considered invalid. Also lambda (arrival rate) value entered should be less than mue (service rate) value since if the arrival rate is greater than the service rate, the queue will not end and it will keep having customers forever. Validation checks regarding these will be done before running the simulation.

1.4 Related Work

The application areas of queuing networks can be briefly mentioned as modeling service centers, evaluating the performance of computer systems, production, and flexible manufacturing systems, as well as communication networks (Filipowicz & Kwiecien, 2008). Some of the applications and platforms that are used in these areas for the queuing simulations are discussed in this section.

JSIM (J-Sim, 2023) is a Java simulation and animation platform that allows web-based simulation. Web-based simulation is a rapidly growing subject of simulation research and development. JSIM makes use of MML (Miller, 2000), a declarative framework for creating algebraic and differential equations. Procedure calls are used to access imperative components defined in other programming languages such as MATLAB, FORTRAN, and C++. MML syntax is simple, requiring only the establishment of parameters and variables before presenting the

equations in a clear and intelligible mathematical form. Most of the difficult decisions that must be taken for simulations to run well, such as identifying transient components in samples that must be ignored, have been automated. The tool also provides instructions for the graphical structure of the network, as well as for the analysis and results visualization.

Some non-GUI simulation applications/tools, such as SimPy, Arena, and AnyLogic, are also available in the industry. SimPy is a Discrete Event Simulation (DES) package for Python allowing users to create simulation models. However, only a few sources are available online for advanced queuing systems for this package. Arena is a general-purpose simulation software that allows users to build and analyze simulation models using a process-oriented modeling approach (Salsabela, 2021). It provides a command-line interface for executing simulations and analyzing results. AnyLogic, on the other hand, is a simulation software with a GUI but also provides command-line execution and batch-mode simulations (AnyLogic, n.d.). In this project, we will utilize JSIM as a starting point to create a GUI simulator that supports queuing networks. In our example, we simplify the interface by managing complicated equations and variables while limiting inputs as much as feasible to manage the queue and servers' outputs.

Furthermore, Python is utilized as a programming language to assist us in overcoming the back-end. The web pages created and graphs produced for visual representation are written in HTML and JavaScript. Then, the front-end and back-end are connected using Flask to transfer data and do the necessary calculations and statistics before sending it back to the front-end to display the resulting graph.

A sample comparison with one of the platforms mentioned, JSIM, can be found in the table below, Table 1.

Aspect	JSIM	Our project
Programming language	Java	Python
Object oriented paradigm	The component processes the data in an execution context (a Java thread). Asynchronous components are those that can handle diverse data at the same time without synchronizing with one another. (Android Developers, n.d.)	Python Flask will be used to interact between the front and back ends, including variables and basic data structures. Also included are HTML and JavaScript scripts for managing various functionalities in the front-end.
Performance scalability	Use of parallel simulation kernels makes it highly scalable. (J-Sim Official, 2003)	Because the jobs are split between the back end and the front end, the simulation will operate smoothly because the software does not consume heavy traffic or lengthy time computations.

Table 1: Related Work Comparison Table with JSIM

1.5 Product Overview

1.5.1 Product perspective

As there is no any external interface/tool that our GUI-based Simulation Software needs in order to function as it is designed, there is only the actual system that we worked with. Figure 1 also shows that our application is not interacting with other external systems.

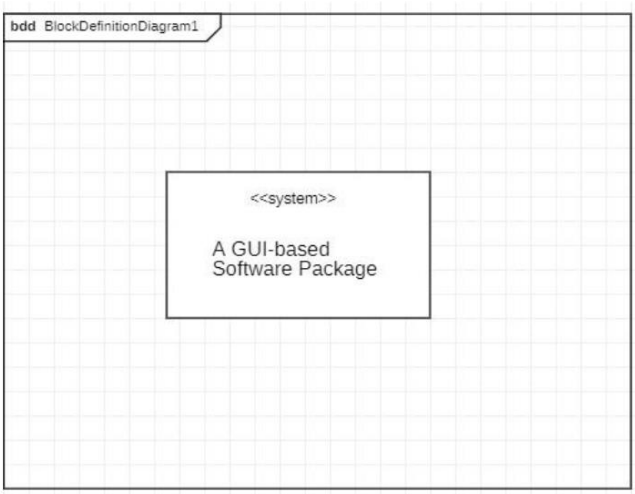

Figure 1: Block Diagram of the System

1.5.1.1 System Interfaces

N.A. As we do not have any data exchange with other systems, like having a database, the system has no interfaces in this application other than user interface. The system is not dependent on other systems.

1.5.1.2 User interfaces

Users will have an interface where they can select the simulation type, drag and drop elements (server(s) and queue) and connect them using the toolbox; in addition to that, the user can run the simulation with the desired inputs for the selected items, and an output graph will be displayed accordingly. An illustration of a sample simulation process can be seen below.

Figure 2 illustrates the home page of the GUI. At the top of the page, available pages can be seen, and the user can move to any of those pages if needed. To create a simulation model, they are required to continue with the 'Create Simulation' page.

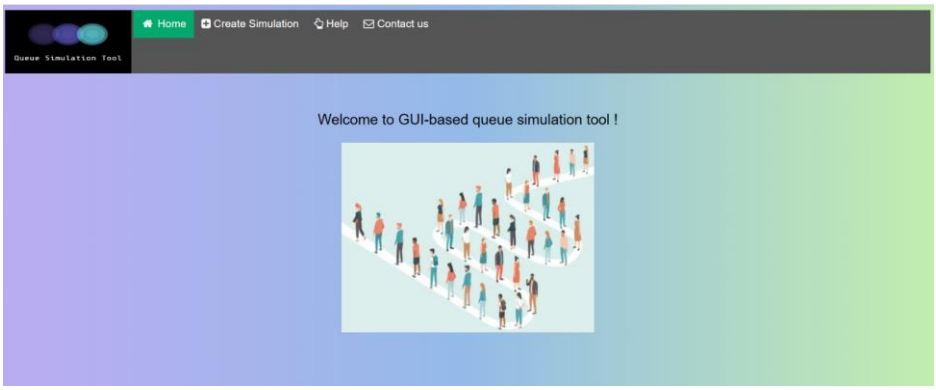

Figure 2: Home Page

Figure 3 shows the simulation creation page. The user is expected to select the simulation type they want to execute to proceed. Short explanations are provided for each type of simulation as well. (M/M/1/L simulation is selected for this step, so the figures after are based on this type)

Figure 3: Create Simulation Page

Figure 4 illustrates the execution page, which is the same for all types of simulations. The left side of the page has a toolbox, which includes server, queue, and connection items. The canvas is for dropping items and creating a connection between them. When an item is dropped on the canvas, the user will click on the item and enter the required parameters. (A server and a queue are connected to each other, and their inputs are entered. For ease of illustration purposes, only one sample for the queue to enter input is provided here. More detailed figures are provided in the following sections)

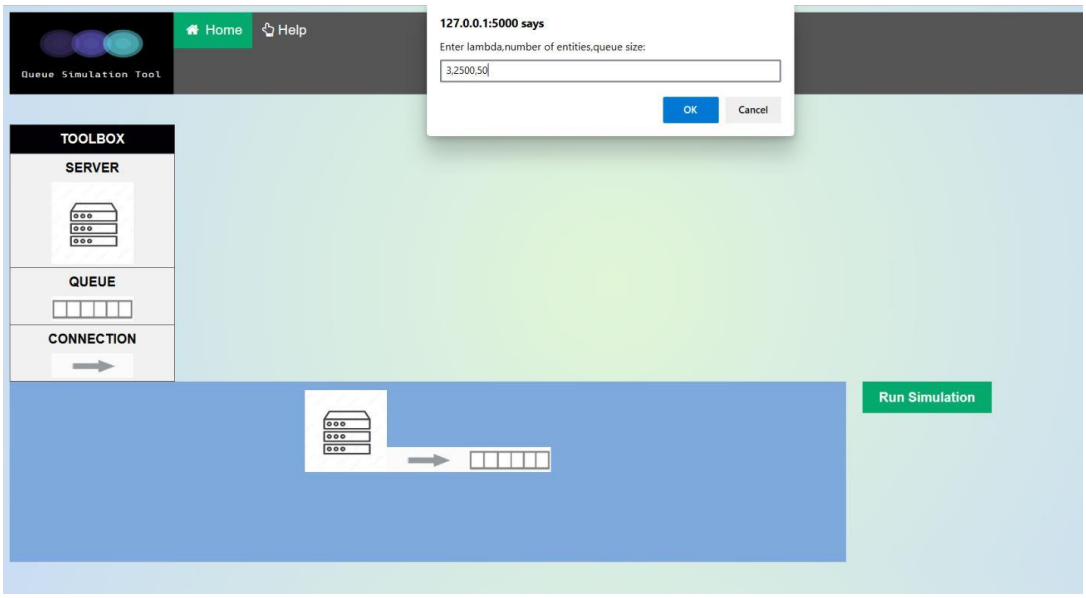

Figure 4: Selecting Items and Entering Desired Inputs

Figure 5 illustrates where the user enters valid inputs, clicks the 'Run Simulation' button, and executes the simulation without any error. A corresponding graph for the entered inputs is displayed. The user can also download the graph plotted at this stage of the simulation (see the 'Download' button at the very left bottom of the page).

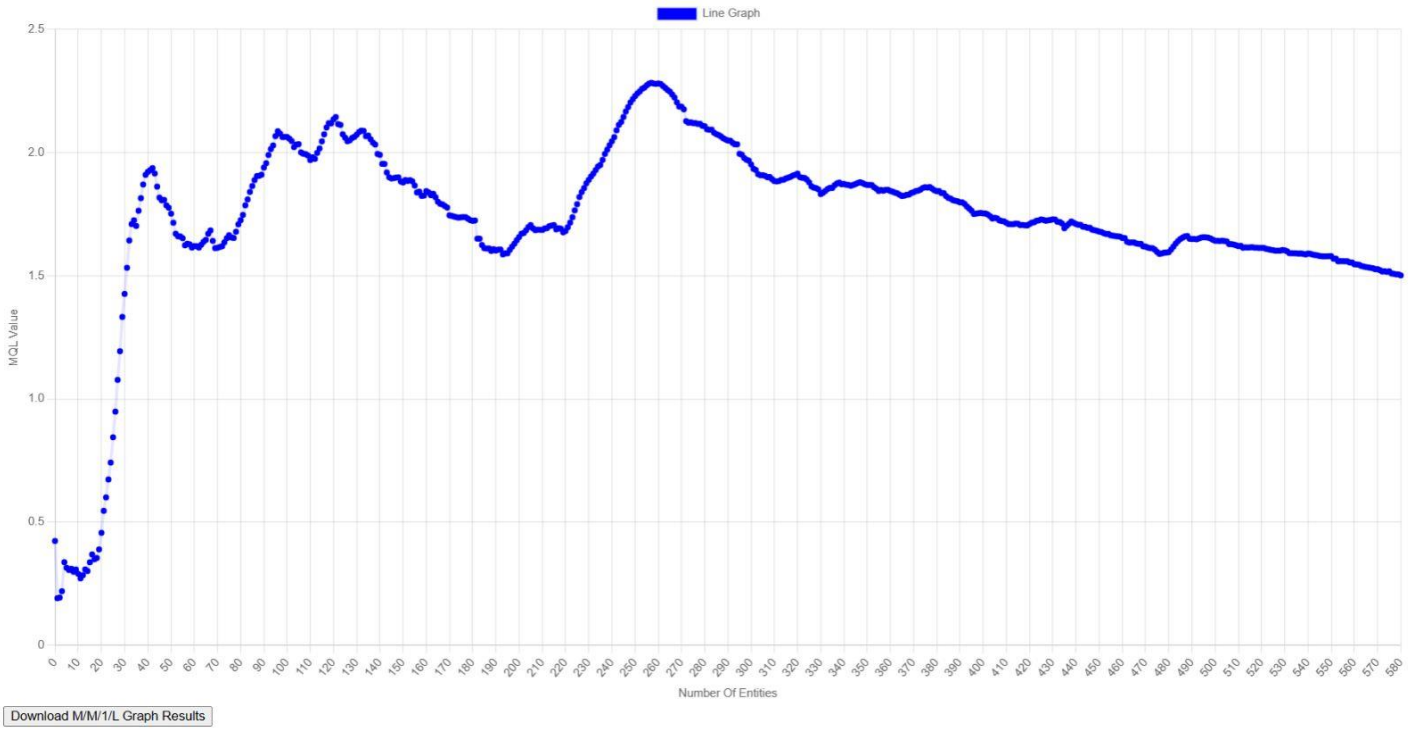

Figure 5: Graph Plotted

1.5.2 Product functions

• The system allows users to drag and drop servers, queue and connection from the toolbox to connect all these together.

• The system allows users to select a simulation type between four options: M/M/1, M/M/1/L, M/M/C and M/M/C/L, which are discussed further with their functionalities in the next sections.

• The system allows users to enter inputs such as number of entities, lambda (arrival rate), mue (service rate) or a queue limit (depending on the simulation type selected) to calculate the average number of entities in the system per time unit (patient, person, jobs), whole simulation time, discrepancy, arrival time, etc.

• The system is able to generate random numbers for the interarrival time and service time for the simulation approach.

• The system is able to visualize results as a graph at the end of the simulation.

• The system provides a download option of the graph plotted to users.

The below figure, Figure 6, is a simple demonstration of the connection between a queue and a server (single server queuing system, M/M/1) where lambda corresponds to the arrival rate and mue (inside the service node) represents the service time. Waiting area is the queue of the system and it doesn't have a limit here (as it is an M/M/1 system), so it is considered to take infinite number of entities but for an M/M/1/L system, the queue will have a limit and the system serves accordingly. For an M/M/C system, the difference is to have multiple servers instead of just one like in the figure, and an M/M/C/L system is basically an M/M/C system with a limited queue size.

Figure 6: A Queuing System (Wikipedia, n.d)

1.5.3 Identified stakeholders and design concerns

Developer: They are responsible from the code going on at the back. The code is written for the calculations of some necessary statistics (execution time, response time, start time of the service, MQL (Mean Queue Length), etc.). The implementation of the GUI part of the software package is also done by them. A web application is created as the simulation environment.

User: User is the person who will use the GUI-based simulation software for queueing networks. They will be using the web application created for the simulation and drag-and-drop feature to make the connection between the queue and the server(s). They need to enter all the necessary inputs for a successful simulation. They will be provided a technical help (done with a help page) in any case but they are expected to have stable internet connection for the simulation.

1.5.4 User characteristics

The following are the characteristics required for the users of this system, other than that users will have a difficult time using this system.

• The users of the system must have a good understanding of queues because the queues will be used for the data structures of this software.

• The users must be able to understand the usage of GUI-based tools because the user needs to know the meaning of the output based on the entered inputs and how to use the tool to benefit it the most.

• The users of the system need to have a basic knowledge of networking, especially computer networks as this project is focused on Queuing Networks.

1.5.5 Limitations

• In case of a power failure during the simulation, the tasks that the system were performing will be disappear.

• This web application can only be used by the users that has an enough background knowledge for the required aspects.

• If the users do not have an internet connection during the simulation, the program will not be executable as it is a web application.

• As the queue size is fixed by user (for M/M/1/L and M/M/C/L options), the number of tasks that the system can perform will be based on the time of each task so it will be limited, and that's a limitation for the execution time of the simulation as a whole for these simulation types.

• Blind users will not be able to use this application as there is no support implemented for this case.

1.5.6 Assumptions and dependencies

• The users of the software are assumed to have a pre-knowledge of queues, and how to use GUI-based software.

• The users of the software are assumed to have a stable (or at least enough for the program to run) internet connection during the simulation process as it will be a web application.

• Internet connection can be counted as a dependency for the system as it will be needed for the whole process. In case of an internet interrupt, the user will need to execute the program again to get the correct output.

2 Specific requirements

2.1 External interfaces

N.A. There is no external interfaces with our project, no other external system is involved or used.

2.2 Functions

Figure 7 shows the Use-Case Diagram of the GUI-based Simulation Software. It shows the inner interactions in the system to perform the tasks as specified (where lambda represents the arrival rate and mue represents the service rate):

1. System should allow users to create a GUI-based queueing simulation.

2. System should allow users to select a simulation type from the four existing options (M/M/1, M/M/1/L, M/M/C and M/M/C/L).

3. System should allow users to drag items (queue and server(s)) from the toolbox and drop them into the canvas.

5. System should allow users to run the simulation by entering input values based on the selected simulation type.

6. System may allow users to download the graph plotted after the simulation is executed.

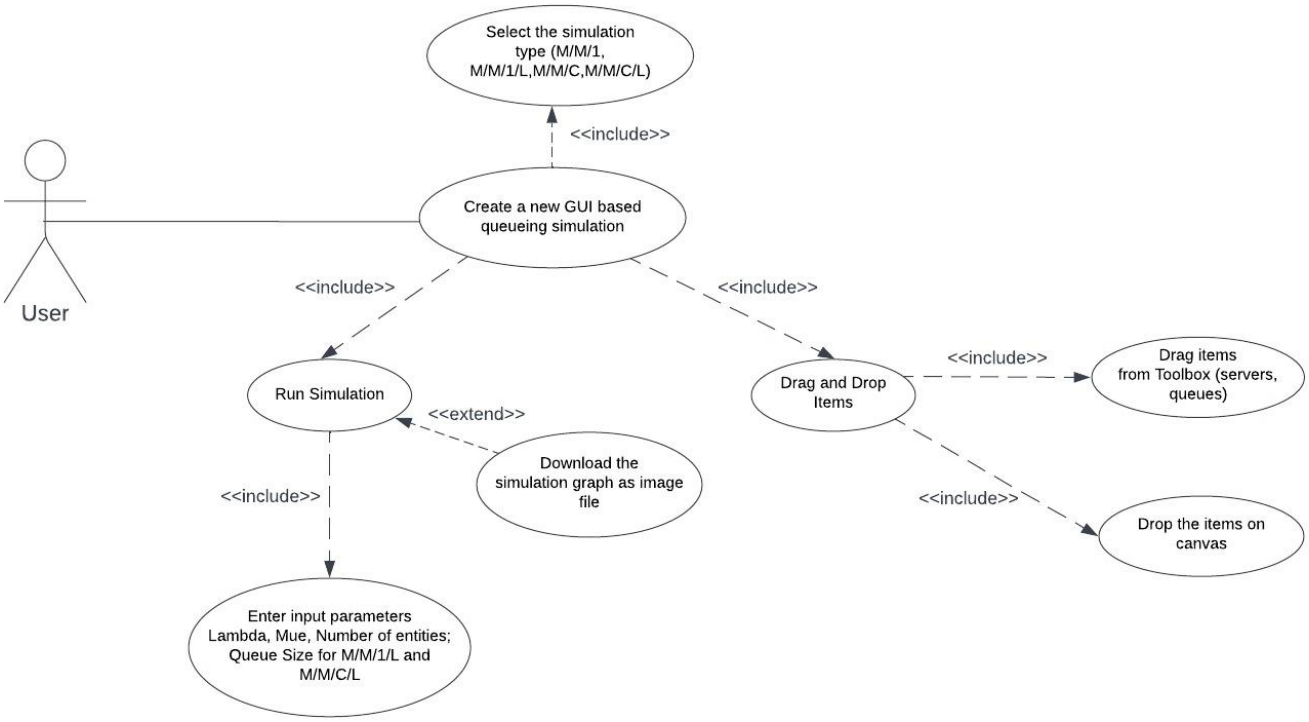

Figure 7: Use Case Diagram

Use case	Create GUI based simulation
Actors	User
Cross references	
Typical Course of Events	
Actor Intentions	**System Responsibility**
1.User clicks on create simulation button.	
	2.System redirects user to the simulation type selection page.
3.User selects the simulation type.	
	4.System redirects user to the simulation page related to the selected simulation type.
Alternative Courses	
Step 2&3 : Internet connection is not available, web application gives missing connection error, process terminates.	

Table 2: Create GUI-based Simulation Use-Case Description

Use case	Drag and drop items
Actors	User
Cross references	
Typical Course of Events	
Actor Intentions	**System Responsibility**
1. User drag items(queue,server,connection) from the toolbox and drop them into canvas.	
	2.System receives the drop items server(s), queue and connection
Alternative Courses	
N/A	

Table 3: Drag and Drop Use-Case Description

Use case	Run Simulation
Actors	User
Cross references	
Typical Course of Events	
Actor Intentions	**System Responsibility**
1.User clicks on queue component.	
	2.System asks user to enter lambda, number of entities (and queue size for M/M/1/L, M/M/C/L simulation types).
3.User enters lambda, number of entities and queue size.	
	4.System check if all parameters are provided and in appropriate range.
5.User clicks on server component.	
	6.System asks user to enter mue value.
7.User enters mue value.	
	8.System checks if the mue value is provided and not a negative value.
9.User clicks the run simulation button.	
	10.System checks if the lambda value is less than mue.
	11.System runs the simulation and redirects user to the simulation graph page.
Alternative Courses	
Step 3&7 : System detects there is a missing input parameter or parameter is out of range and asks user to enter the parameter.	
Step 10 : System detects that lambda is greater than mu, and asks user to enter lambda again.	
Step 11 : Internet connection is not available, web application gives missing connection error, process terminates.	

Table 4: Run Simulation Use-Case Description

2.3 Usability Requirements

1. The GUI screen is easy to understand and use for users.

2. GUI contains informative labels for the components and operations (such as server, queue, connection, run button), so that the users understand the representation easily.

3. System is designed to be traversed easily for its different sections (Toolbox section, canvas section, inputs section).

4. The help page is presented with possible operations to perform for first-time users or users who face any problem related to the interface or running the simulation to demonstrate how the application works.

5. The clickable feature to enter inputs for the selected items is easy to use, especially for the multiple server systems (M/M/C and M/M/C/L), as the user can see and choose which server to enter input for. Doing it this way instead of having a separate page to enter inputs avoids confusion for the application users.

2.4 Performance requirements

1. A huge number of users are tolerated for the system usage to avoid traffic usage.

2. Any command or instruction (running the simulation for example) should take milliseconds to be executed and not long enough for users so they get the best experience.

3. The clickable feature to enter inputs for the selected items (by using drag-and-drop feature) should take less time comparing to the option where we have a separate page to enter inputs, so it is a time efficient option for users.

4. Depending on the input values such as: arrival rate, service rate, number of entities and queue size limit (limit is only for M/M/1/L and M/M/C/L simulations), the execution time of the simulation will vary since the simulation should stop as soon as the MQL value calculated from the simulation approach is close enough to the result of the MQL value calculated from the analytical method.

5. Desired amount of accuracy should be provided for results by comparing, using the discrepancy value, simulation and analytical results for MQL and verifying that they are as close as expected to each other.

2.5 Logical database requirements

N.A. We do not have a database implementation as this is a web application. The data used is not saved for the simulations (there is only a download option for the graph plotted at the end), so we do not need a database.

2.6 Software system attributes

• Reliability:

- The system should not give an inappropriate output when the requirements are met because the simulation process should be reliable in terms of accuracy. For example, the output calculated from the simulation approach should be close enough (depending on the discrepancy value set as a limit, it is 0.25 in this project, showing the closeness of the two values mentioned) to the analytical result we get to make sure that the graph plotted is reliable.

• Availability:

- The time for the system to perform the requested tasks should not take longer than expected (unnecessarily long) so that the users can access to it whenever they need it without suffering from the time consumption of the task performed.

- The web page should be available for its audience as long as there is a stable internet connection.

• Security:

- As the system does not use a database and keep the simulation outputs in it, it does not require any security-based implementations. The security will be needed depending on the area that this system is used. For example, if it is used in a bank, then extra security precautions need to be taken by the bank itself.

• Maintainability:

- When a new feature is wanted to add to the application, the system will make sure that it is done through the GUI as it is responsible for the whole simulation process and the changes made will affect the interface as well, such as adding a new type of simulation or changing the structure of clickable feature or changing the graph representations.

• Portability:

- System will be implemented on python, which is an interpreted language, in addition to that system can run on devices that use windows OS and Macintosh, except Linux because some features for instance drag and drop, is not adjustable for Linux.

2.7 Supporting information

A queue is a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end (Technopedia, 2011).

Queuing theory is a mathematical method that aims to comprehend and analyze the dynamics of waiting lines or queues. It involves examining the arrival patterns of customers to a service system, their waiting duration, and the process of being served or processed. The objective of this theory is to identify the most effective setup and structure for queuing systems, with the ultimate goal of enhancing efficiency, reducing waiting times, and optimizing the utilization of resources. In our application, the structure will be a single queueing network system with different types of simulations available (Investopedia, 2011).

3 Software Estimation

In this part we will calculate the software estimation of our simulation project. For the first thing we will calculate the effort, we will consider the three methods we learned (Organic, Semi-detached, Embedded), but before that, complexity of our project and the Compute Adjusted Total Function Points (ATFP) are going to be calculated. Regarding the programming languages for our project, Python, Html, JavaScript were all used to complete the GUI simulation. (Note that python will be considered as a 4^{th} generation language)

Program Characteristic	Complexity	UFP (Unadjusted Function Points)
Number of inputs = 5	Low	5 x 3 = 15
Number of outputs = 11	Low, Medium	(9 x 4) + (2 x 5) = 46
Inquiries = 16	Low, Medium	(14 x 3) + (2 x 4) = 42 + 8 = 50
Logical internal files = 0	-	
External interface files = 0	-	

Table 5: Unadjusted Function Points

Inputs	Complexity	UFP (Unadjusted Function Points)
Number of entities	Low	3
Average arrival time (Lambda)	Low	3
Average service rate (Mue)	Low	3
Number of Servers	Low	3
Queue Size Limit	Low	3
		Total = 15

Table 6: Inputs Table

Outputs	Complexity	UFP (Unadjusted Function Points)
Interval arrival time	Medium	5
Service Time	Medium	5
Arrival Time	Low	4
Start time of service	Low	4
Finish Time of Service	Low	4

Waiting Time	Low	4
Simulation Time	Low	4
Traffic Intensity	Low	4
Analytical MQL	Low	4
Simulation MQL	Low	4
Discrepancy	Low	4
		Total = (4 x 9) + (5 x 2) = 46

Table 7: Outputs Table

Inquiries	Complexity	UFP (Unadjusted Function Points)
Number of entities	Low	3
Average arrival time (Lambda)	Low	3
Average service rate (Mue)	Low	3
Number of Servers	Low	3
Queue Size Limit	Low	3
Interval arrival time	Medium	4
Service Time	Medium	4
Arrival Time	Low	3
Start time of service	Low	3
Finish Time of Service	Low	3
Waiting Time	Low	3
Simulation Time	Low	3
Traffic Intensity	Low	3
Analytical MQL	Low	3
Simulation MQL	Low	3
Discrepancy	Low	3
		Total = (3 x 14) + (4 x 2) = 42 + 8 = 50

Table 8: Inquiries Table

General System Characteristics	Degree of Influence (0-5)	Explanation
Data Communications	0	There will be no communication facilities with the system
Distributed Processing	0	There will be no data processing handled in the system
Heavily Used Configuration	0	The current hardware platform won't be heavily used at all
Transaction Rates	0	No transactions will be held
Online Date Entry	0	No percentage of the data will be on-line
Design For End User Efficiency	4	System should be efficient to a high degree for the users.

Online Updates	0	There are no logical files to be updated
Complex Processing	5	System will use a lot of processing to calculate formulas from Table
Usable in other applications	0	The system simulation is not developed for other applications
Installation Ease	5	Installation of the software packages of the back end will be easy
Operational Ease	5	Any operations related with simulation will be done effectively
Multiple Sites	0	System will not be available in multiple sites
Facilitate Change	0	System won't be supported to facilitate changes
Total Of Degree Of Influence	TDI = (4+5+5+5) = 19	
Value Adjustment Factor	VAF = TDI x 0.01 + 0.65 = (0.19) + 0.65 = 0.84	

Table 9: General System Characteristics Table

Program Characteristics	Low Complexity	Medium Complexity	High Complexity	Total
Inputs	5	0	0	15
Outputs	9	2	0	46
Inquiries	14	2	0	50
Logical Internal Files	0	0	0	0
External Interface Files	0	0	0	0
Unadjusted Total Of Function Points				15+46+50 = 111
Adjusted Total Of Function Points				AFTP = UTFP X VAF = 111 x 0.84 = 93

Table 10: Adjusted Total of Function Points

Programming Language	Percentage in Project	Language Unit Size	Language Unit Size in Project
Python	22%	20	0.22 X 20 = 4.4
HTML	50%	15	0.5 X 15 = 7.5
JavaScript	28%	80	0.28 x 80 = 22.4

Language Unit Size	34
Lines Of Code	Lines Of Code = AFTP x Language Unit Size = 93 x 34 = 3,162
Kilo Delivered Source Instruction	KDSI is ATFP x language unit size/1000 = 93 x (34/1000) = 3.16

Table 11: KDSI Table

1. **Organic Method**
   Effort = a x $(KDSI)^b$ (a = 2.4 and b = 1.05) = 2.4 x $(3.16)^{1.05}$ = 8.03
   Development time = c x $(Effort)^d$, (c=2.5 and d=0.38) = 2.5 x $(8.03)^{0.38}$ = 5.51
   Staff size = Effort/Development time = 8.03/5.51 = 1.45
   Productivity = KDSI/Effort = 3.16/8.03 = 0.39

2. **Semi-detached**
   Effort = a x $(KDSI)^b$ (a = 3.0 and b = 1.12) = 3.0 x $(3.16)^{1.12}$ = 10.88
   Development time = c x $(Effort)^d$, (c=2.5 and d=0.35) = 2.5 x $(10.88)^{0.35}$ = 5.76
   Staff size = Effort/Development time = 10.88/5.76 = 1.88
   Productivity = KDSI/Effort = 3.16/10.88 = 0.29

3. **Embedded**
   Effort = a x $(KDSI)^b$ (a = 3.6 and b = 1.20) = 3.6 x $(3.16)^{1.20}$ = 14.31
   Development time = c x $(Effort)^d$, (c=2.5 and d=0.32) = 2.5 x $(14.31)^{0.32}$ = 5.85
   Staff size = Effort/Development time = 14.31/5.85 = 2.44
   Productivity = KDSI/Effort = 3.16/14.31 = 0.22

Development Type	**Organic**
Estimated Effort in Man-Months	8.03
Estimated Development Time in Month	5.51
Estimated Team Size	1.45

Table 12: COCOMO I/Basic/'81 for software effort and schedule estimation

Development Type of our project is organic, since it is not very complicated and simple to develop, in addition to that, project group size for our project is considered small and not huge, we can actually see that our project is also organic due to the estimated team size we calculated using COCOMO software estimation method which is not accurate in our case but is an indicator that our project shouldn't have so many members for the project to be developed.

Adjusted Total of Function Points (ATFP)	93
Kind Of Software	Systems
Software Organization's Skills/Abilities	Best in Class

Estimated Effort in Man-Months	Effort in man-months = $(AFTP)^{3xClass\ Exponent}/27$ = $(93)^{3x0.43}/27$ = 12.8
Estimated Development Time in Month	Estimated Development Time in Month = 3.0 x $(\text{Estimated Effort in Man} - \text{Months})^{1/3}$ = 3.0 x $(12.8)^{1/3}$ = 7.0
Estimated Team Size	1.45

Table 13: Jones's First-Order Estimation for software effort and schedule estimation

Our project is not related with any businesses or licenses, terms, and conditions, so it a systems type of software. Organization's skills for our project are best in class with it being designed in a way for maximum performance such as accuracy, speed, etc.

	Systems Products		Business 1'roducts		Shrink-WrapProduct	
System Size (lines of code)	Schedule (months)	Effort (man-months)	Schedule (months)	Effort (man-months)	Schedule (months)	Effort (man-months)
10,000	10	48	6	9	7	15
15,000	12	76	7	15	8	24
20,000	14	110	8	21	9	34
25,000	15	140	9	27	10	44
30,000	16	185	9	37	11	59
35,000	17	220	10	44	12	71
40,000	18	270	10	54	13	88
45,000	19	310	11	61	13	100
50,000	20	360	11	71	14	115
60,000	21	440	12	88	15	145
70,000	23	540	13	105	16	175
80,000	24	630	14	125	17	210
90,000	25	730	15	140	17	240
100,000	26	820	15	160	18	270
120,000	28	1,000	16	200	20	335
140,000	30	1,200	17	240	21	400
160,000	32	1,400	18	280	22	470
180,000	34	1,600	19	330	23	540
200,000	35	1,900	20	370	24	610
250,000	38	2,400	22	480	26	800
300,000	41	3,000	24	600	29	1,000
400,000	47	4,200	27	840	32	1,400
500,000	51	5,500	29	1,100	35	1,800

Table 14: Nominal Schedule Sample

In our project, nominal schedules were used for our software estimations, assuming team members are familiar a little bit and have some experience with programming languages, and in addition to that, considering there will be conflicts occurring between the team members as well. Overall nominal schedules are considered as average schedules that are used in average projects.

4 Architectural Views

In this report, the architecture of software systems will be described based on the Logical View, Process View, Development View, and Deployment View, as suggested by Kruchten (1995)1.

4.1 Logical View

The logical view shows the key abstractions in the system as object classes.

4.1.1 Class Diagram

Figure 8 below shows the classes that this application has, their interaction with each other and their methods. It has four classes and these are: Simulation (GUI), Queue (two types are available as limited and unlimited), Server and Web Application. The web application is basically the interface used for the simulation, a connection between the user and the simulation. The user will use this platform to run the simulations. Almost all the calculations and running process will be handled by the Simulation functions.

When a user wants to run a simulation, the simulation screen will be shown by calling the items that are available, which are server, queue and connection, from the GUI with the help of the interface (web application class) and the toolbox will be displayed for the user. The user then will be able to drag and drop the items, and enter the corresponding inputs for the selected item by clicking the item. When the user clicks the item, a function is called for that item and the inputs are taken by GUI. After all the require inputs are entered, the user clicks the run button and validation checks are performed at this stage by the GUI. If all the conditions for the inputs are satisfied, the simulation will be executed. The graph will be plotted with the outputs generated according to those inputs on the web page.

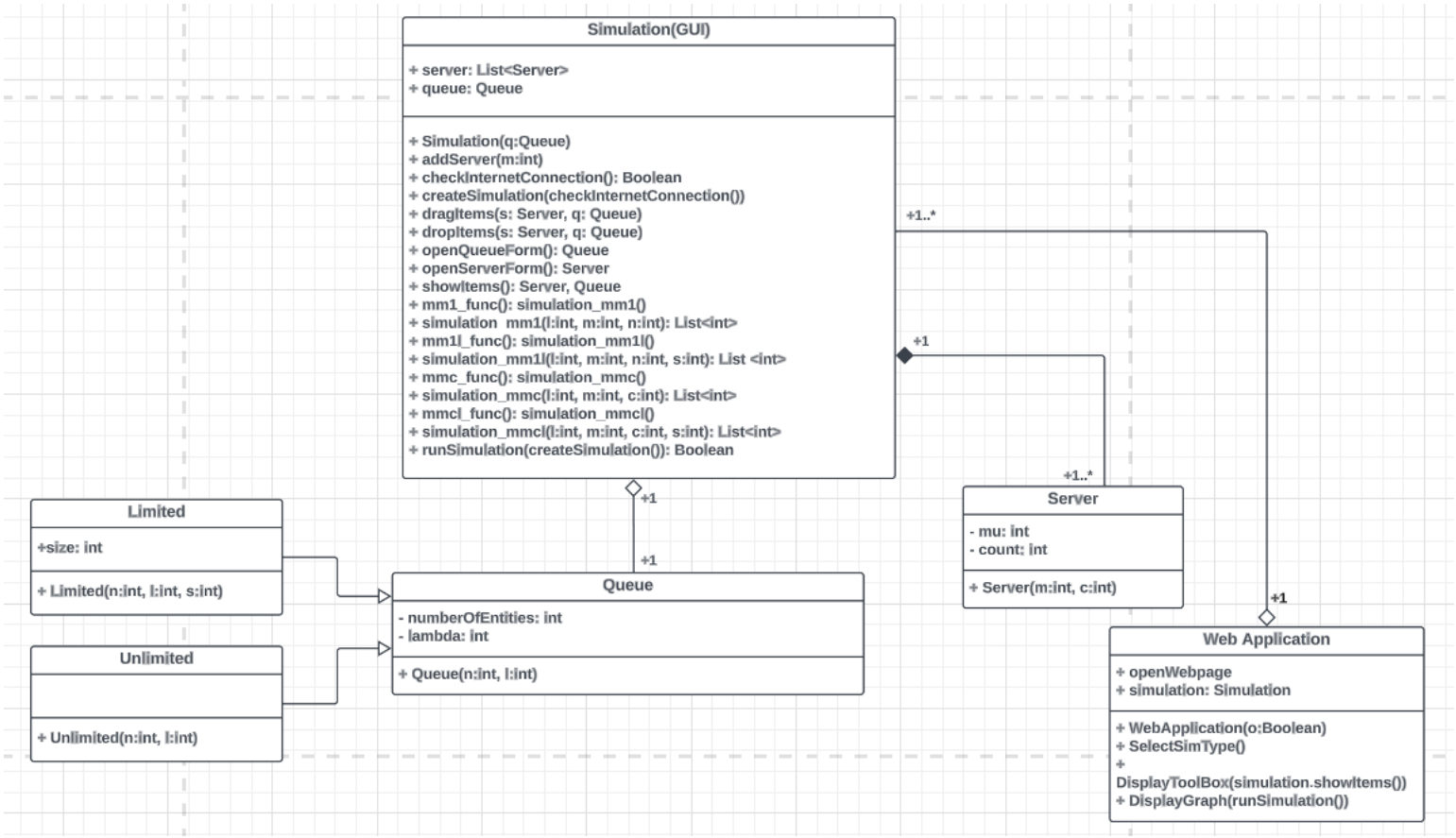

Figure 8: Class Diagram

4.2 Process View

The process view shows how the system is composed of interacting processes.

4.2.1 Activity Diagram

First, the system receives simulation type(M/M/1, M/M/1/L, M/M/C and M/M/C/L). Then, it receives dropped items: server, queue, and connection. The system checks if the selected simulation type has a limited queue size(M/M/1/L or M/M/C/L) or an unlimited queue size (M/M/1, M/M/C). If the selected simulation type has a limited queue size system receives lambda, mu, the number of entities, and queue size as inputs; if the selected simulation type has an unlimited queue size system receives lambda, mu, and the number of entities as inputs. The system checks for a missing parameter or a parameter out of range. If there is, the system receives inputs again. At the same time, the system checks the availability of an internet connection. In case of an internet connection cut, the procedure terminates and returns to the initial stage. The system calculates the related statistics based on the selected simulation type if both the internet connection and the required parameters are available. The system displays the simulation, and if the user requests to download the simulation graph system performs the download operation. All are shown below in Figure 9.

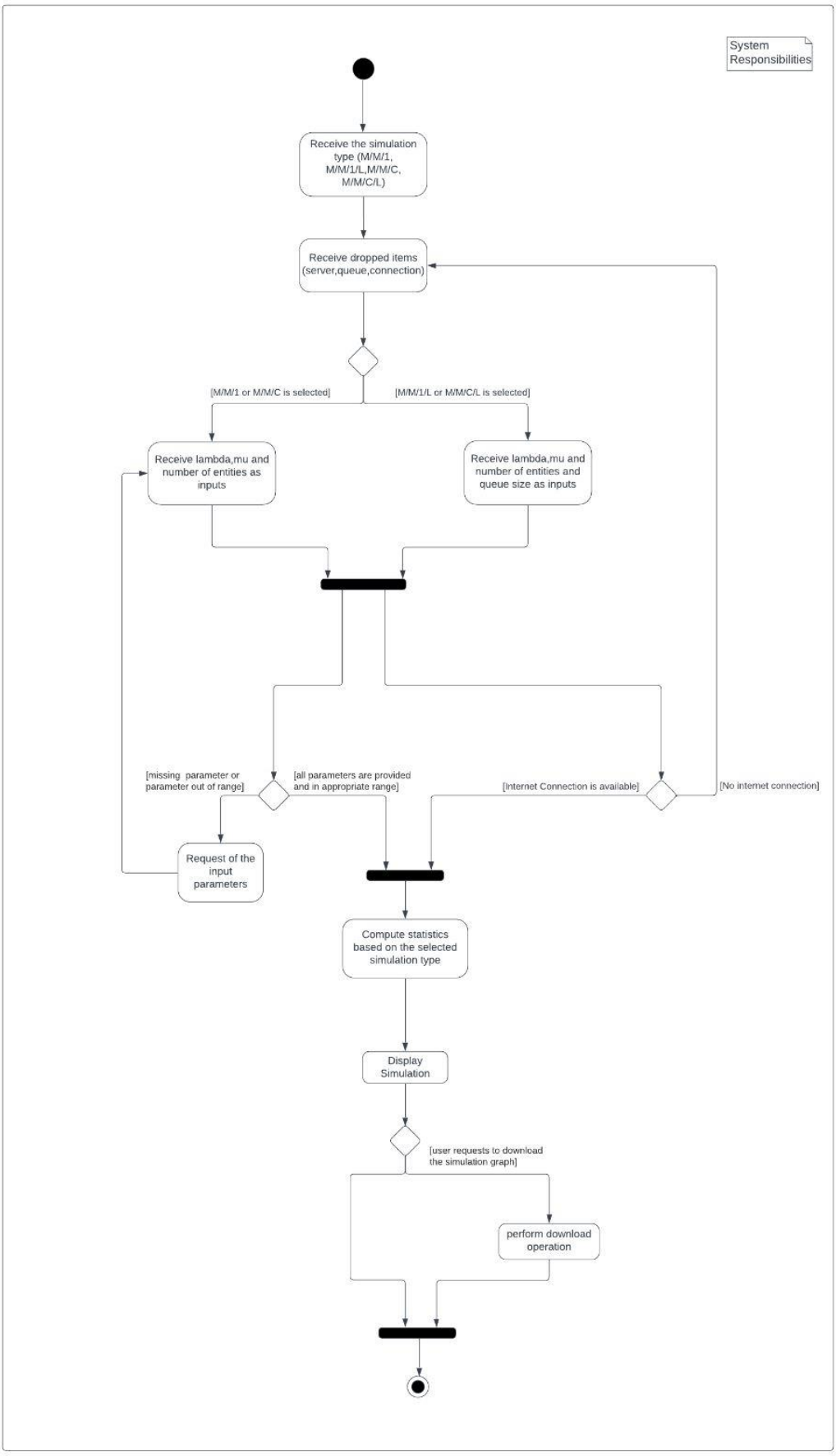

Figure 9: Activity Diagram

This activity diagram, Figure 10, is an example of how the system computes statistics as outputs. In this example, we would like to represent only how the traffic intensity and MQL analytic outputs are calculated. The system receives queues, servers, and connections; then, it receives parameters: lambda, mu, and the number of entities. The system checks if the user provides all connections and parameters. It also checks whether the Internet connection is available or not. If there is an Internet connection, all parameters and connections are appropriate lambda/mu will be calculated, which is equal to the traffic intensity value. After this step, the system traffic intensity/(1- traffic intensity) will be calculated; the result equals the MQL analytic (average number of entities in the system per time unit). When calculations are done, the system will display the statistic to the user.

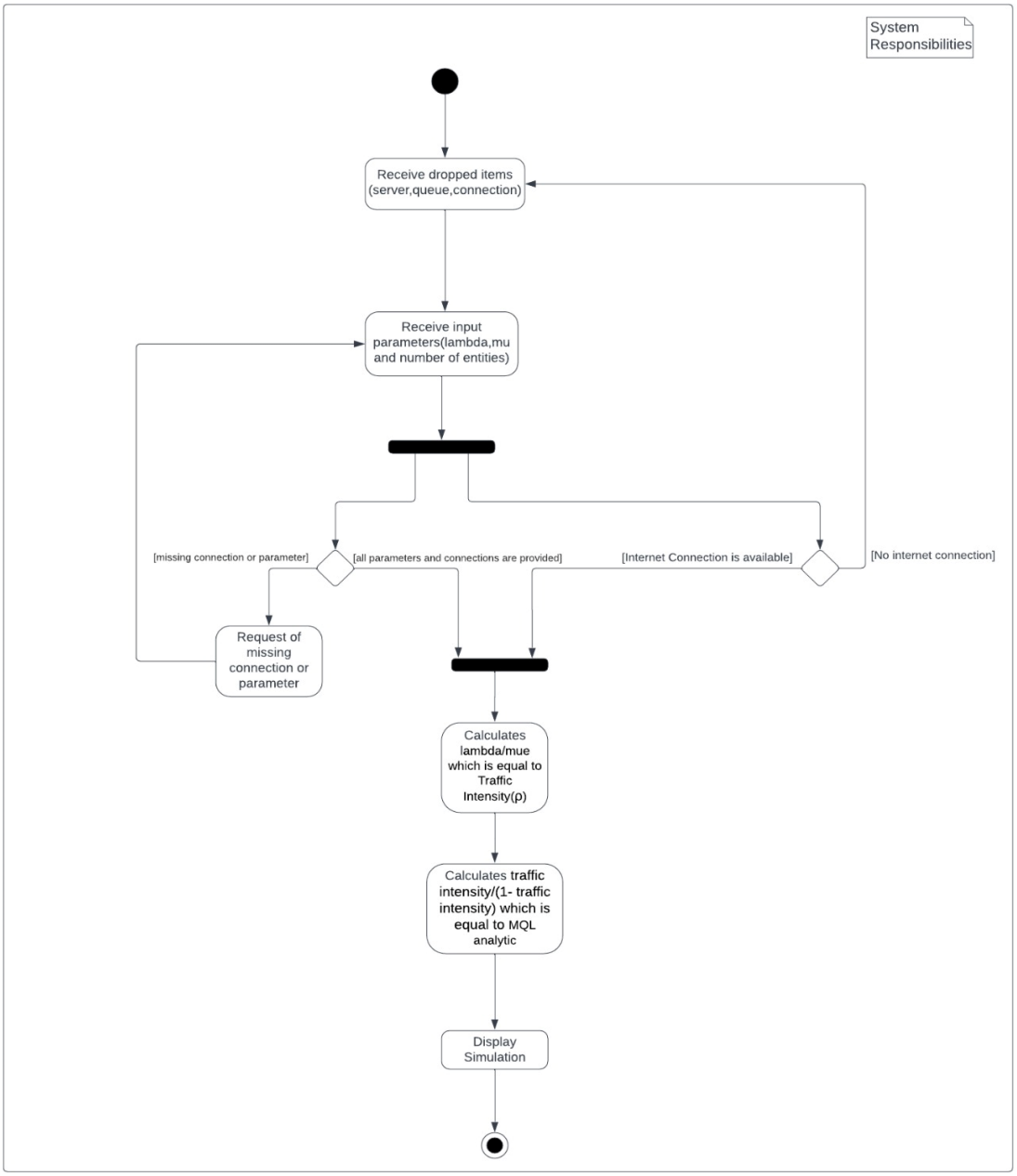

Figure 10: Sample Statistics Computation Activity Diagram

4.2.2 Sequence Diagrams

1) Sequence Diagram for Create a new GUI-based Simulation Use Case:

This sequence diagram, Figure 11, represents the steps of the "Create a new GUI-based queueing simulation" use case. It explains the sequence of the functions when the user wants to create a new simulation. The interactions occur between the user (actor), Web application, and Simulation classes. To determine the lifelines, we consider the calling order of the functions and intervals they will execute. For example, the selectSimType function is called first so that the user selects the simulation type. Then createSimulation is called, and this function calls the checkInternetConnection function. Internet connection is checked in the alternative fragment. If available, boolean = 1 is sent to the Web application class so that the DisplayToolBox function is called, and finally, the toolbox is displayed by calling the showItems function.

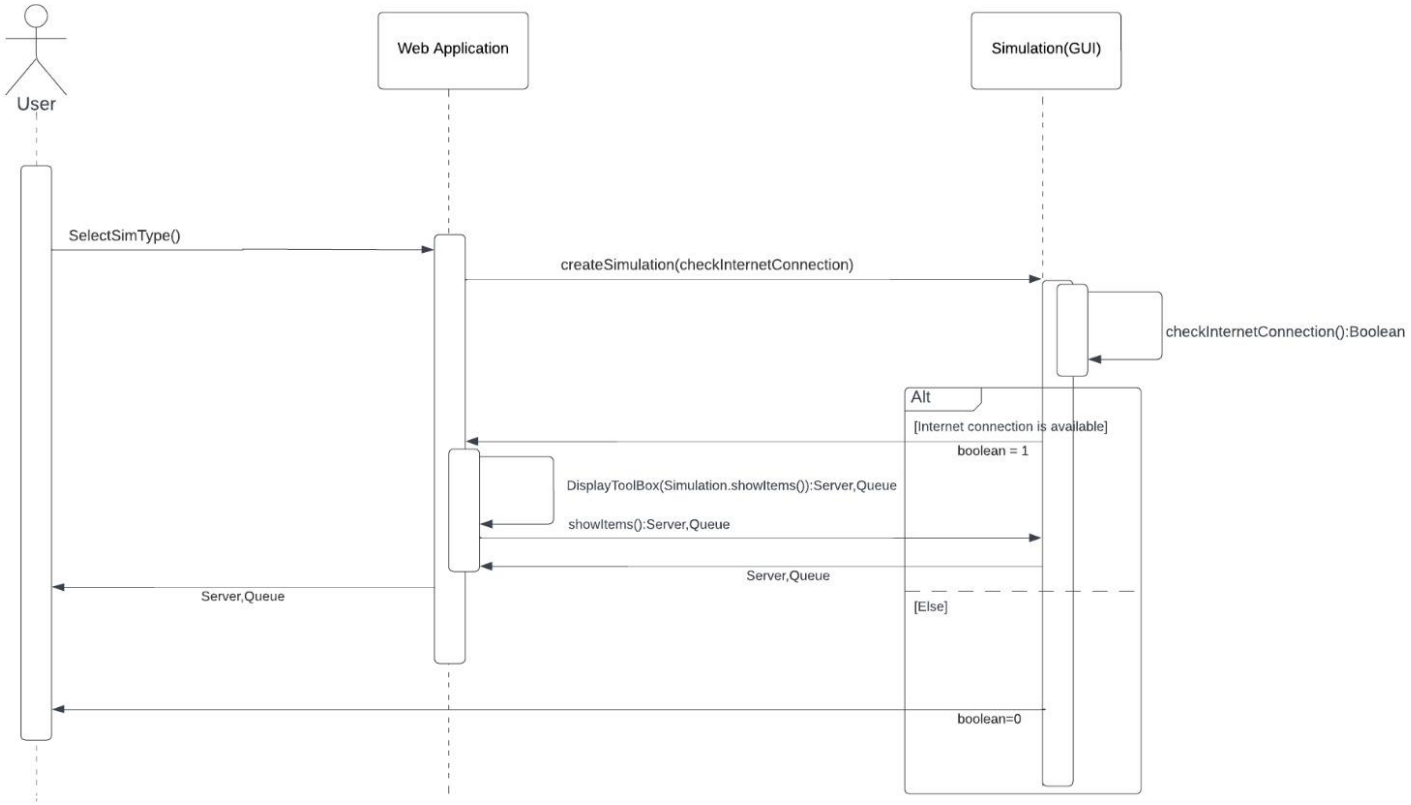

Figure 11: Create Simulation Case Sequence Diagram

2) Sequence Diagram for Drag and Drop Items Use Case:

This sequence diagram, Figure 12, represents the steps of the "drag and drop items" use case. It explains the sequence of the functions when the user wants to drag items from the toolbox and drop them into the canvas. The interactions occur between the user (actor), Simulation, Queue, and Server classes. To determine the lifelines, we consider the calling order of the functions and intervals they will execute. For example, the dragItems function will be called first; then, the function interacts with the Server and Queue classes and returns the server and queue for the user. Another significant interaction is that in the Server class, the simulation type is checked in the alternative fragment; if the simulation type is M/M/C or M/M/C/L, the addServer function is called so that the user can add multiple servers. dropItems is called to drop items on the canvas.

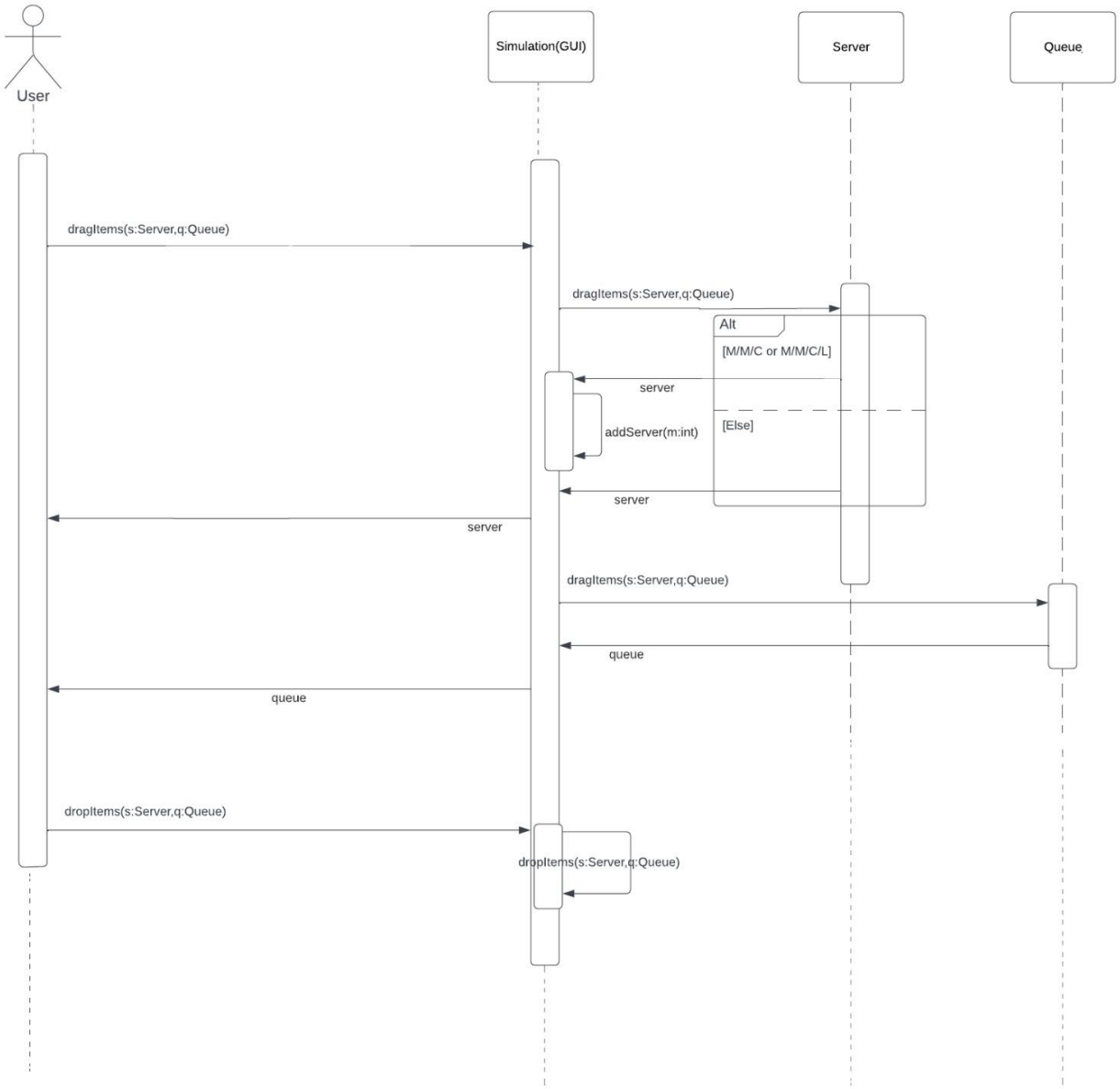

Figure 12: Drag and Drop Items Sequence Diagram

3) Sequence Diagram for Run Simulation Use Case:

This sequence diagram, Figure 13, represents the steps of the "run simulation" use case. It explains the sequence of the functions needed when the user wants to run the queue simulation. The interactions occur between the user(actor), Simulation(GUI), Web Application, Server, Queue, Limited, and Unlimited classes. The results will not be saved into the database, so the database class is not implemented in this sequence diagram. To determine the lifelines in the diagram, we consider calling orders of the functions and the intervals they will execute. For example, the user must click on the server and queue to enter inputs; to enter input for a queue item openQueueForm is called first, then this function calls the Queue function. At this step, queue type(limited or unlimited) is checked by an alternative fragment; for example, if it is a limited queue, it will visit the Limited class and call the Limited function to take the necessary parameters from the user. To run the simulation RunSimulation function is called, and in the Simulation class, the simulation type is checked, and for each of them, the related functions are called. For example, for M/M/1 simulation, mm1_func is called to perform necessary calculations, and it calls simulation_mm1 to perform graph-generating related operations.

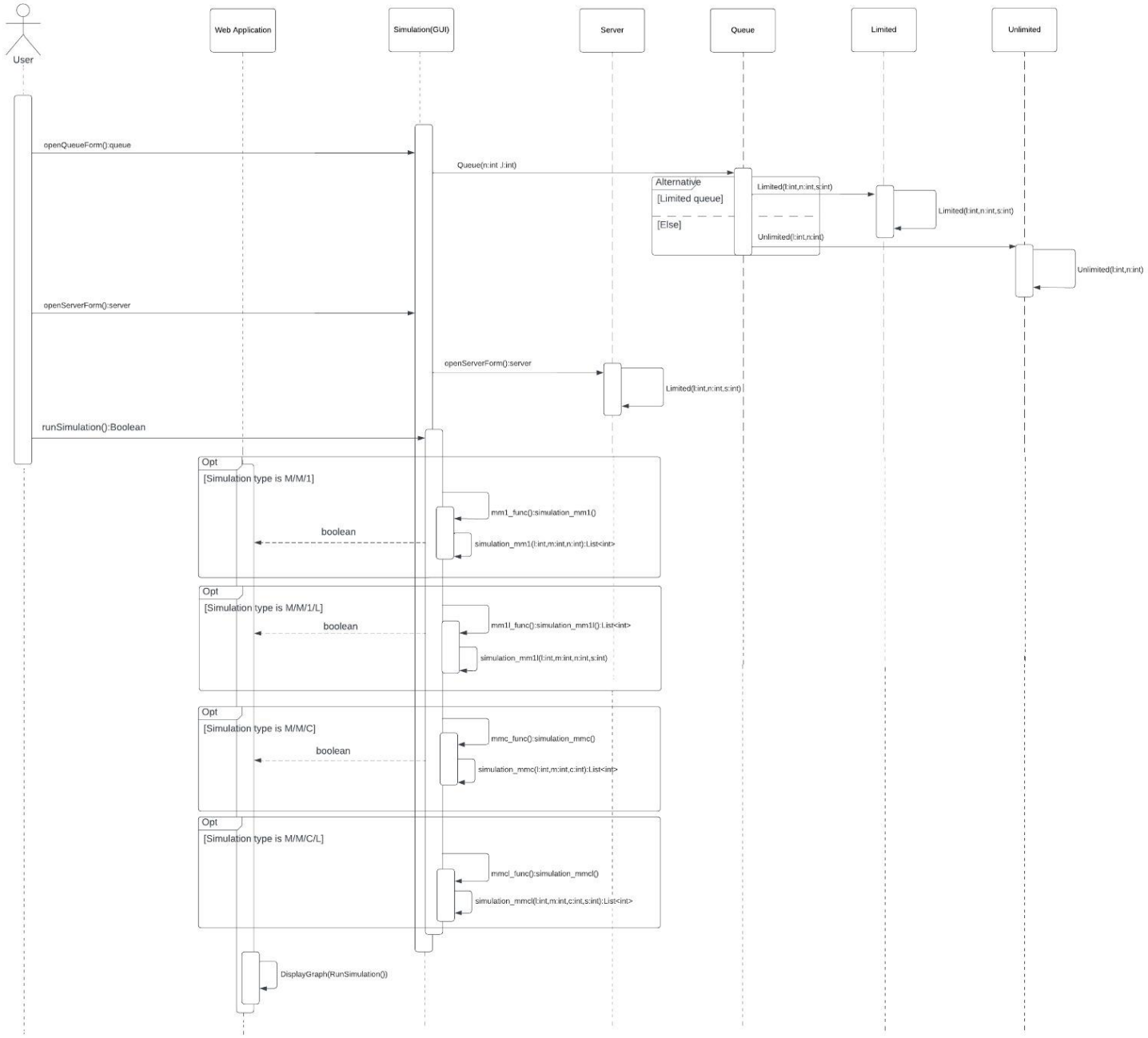

Figure 13: Run Simulation Sequence Diagram

4.2.3 Data Flow Diagrams

The data flow diagrams of the system map out the flow of information for all the processes in the system. Context level DFD shows the fundamental flows between the user and the GUI through system itself. Those flows are further divided into their detailed processes in the Level-0 DFD, and divided into more detailed ones in Level-1 DFD. We use data flow diagrams in our design to show the relationship between all the processes in our system and map the flow of the information of all these processes.

Context level DFD:

This Context level DFD, Figure 14, represents the essential data flows between the user and GUI-based Simulation Software System. The user sends the simulation (create) to the GUI, and GUI delivers Toolbox to the user. Then the user sends parameters and simulation (run) to the GUI. Finally, the GUI provides the corresponding graph to the user.

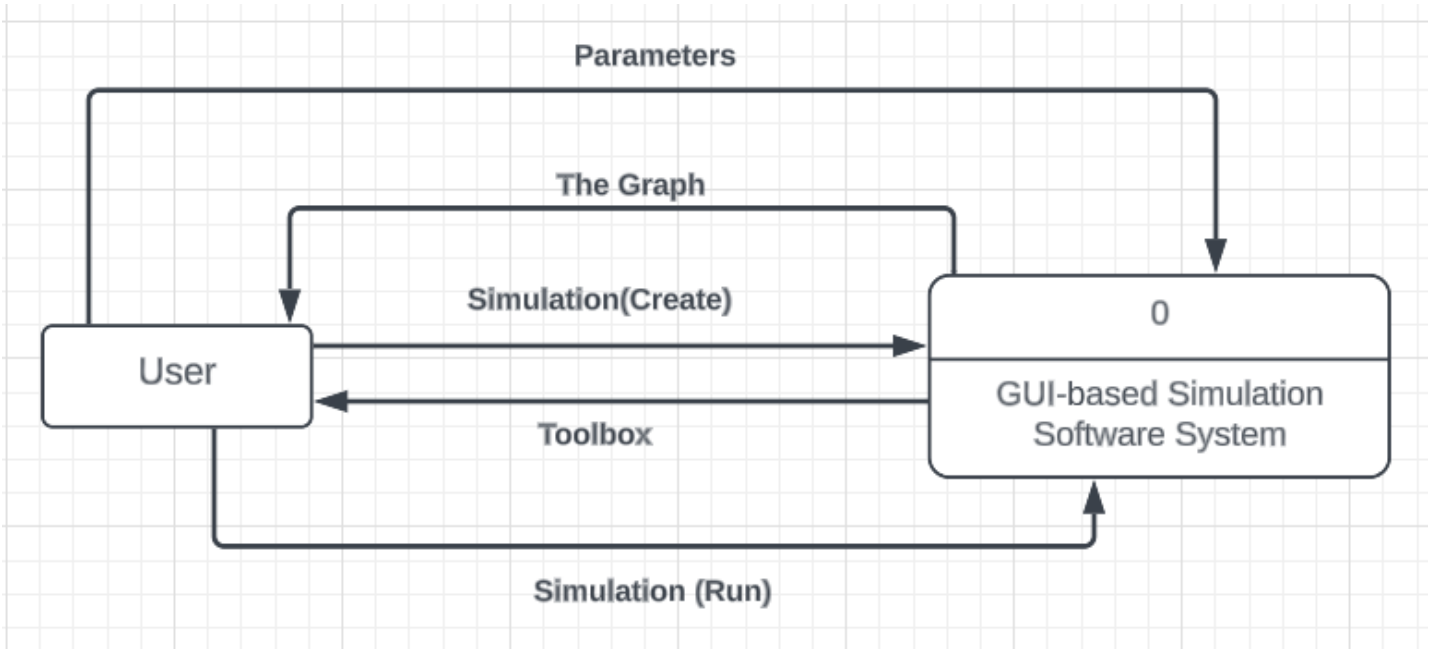

Figure 14: Context Level DFD

LEVEL-0 DFD:

This Level-0 DFD, Figure 15, shows the data flow between the user and divided processes of Context level DFD. When user creates a simulation, one of the simulation types will be selected and the Toolbox items will be displayed accordingly (the Toolbox for all the simulation types are same but the inputs saved in the items selected will be different). After the user drags and drops the desired items, the system receives input parameters from the user. The user runs the simulation, and if there is a stable internet connection, a confirmation will be sent to the run simulation. Calculation results then will be sent to plot the corresponding graph and it will be displayed on the screen.

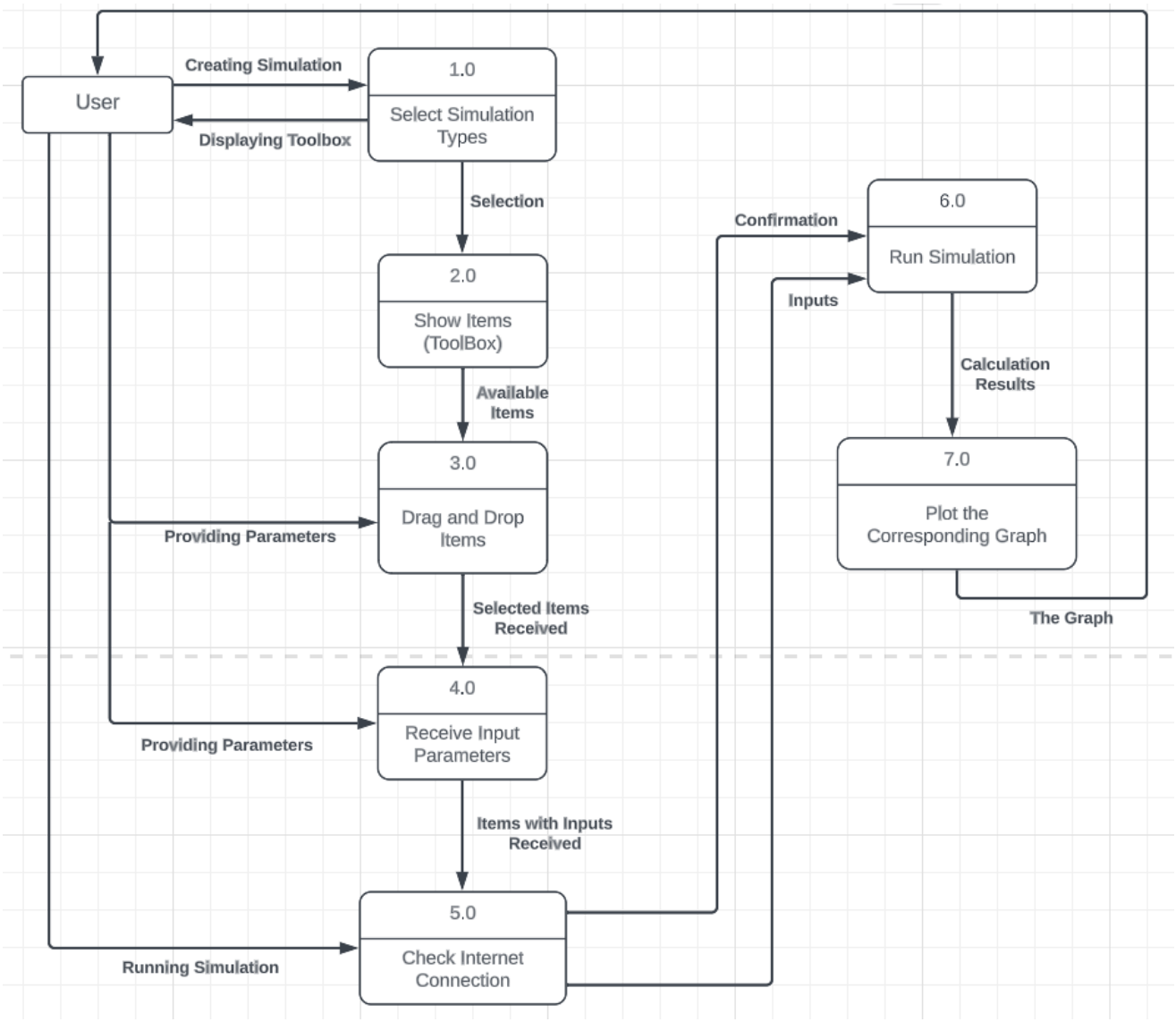

Figure 15: Level-0 DFD

LEVEL-1 DFD:

This Level-1 DFD, Figure 16, represents the data flow between the user and the divided and detailed processes of Level-0 DFD. The user creates a simulation and selects the type of simulation they want to run. The system then will generate the items: queue, server and connection to be selected and the Toolbox will be shown to the user. The user drag and drops items, and makes connections. The system receives lambda, mue, the number of entities and queue size (based on the simulation type selected by user) from the user. The user runs the simulation, and the system checks for input validations and the internet connection. A confirmation will be sent to the run simulation after all the checks are done. Inputs then will be sent to the back-end for the calculations. A corresponding graph will be plotted and returned to the user.

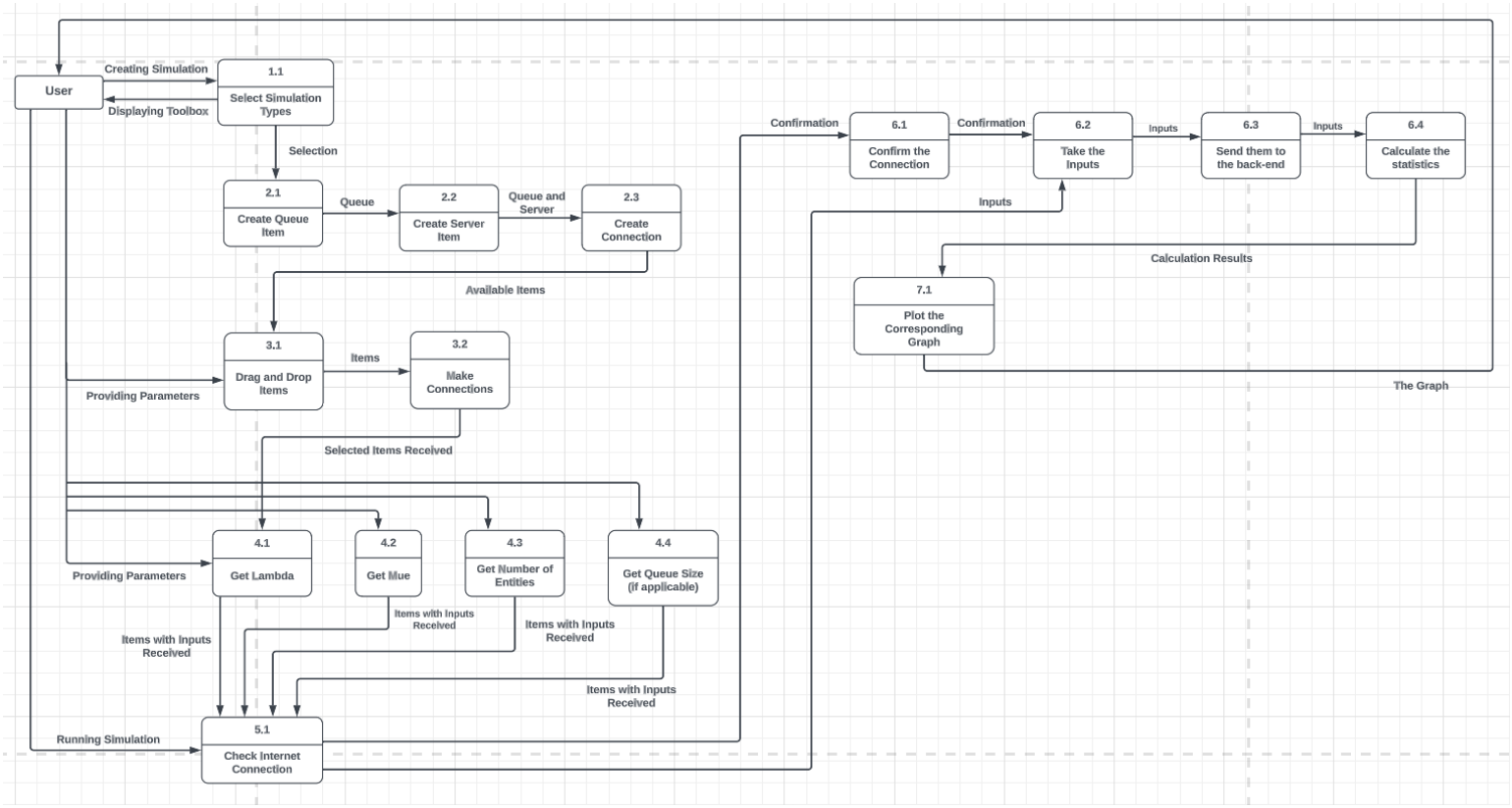

Figure 16: Level-1 DFD

4.3 Development View

The development view shows how the software is decomposed for development.

4.3.1 Component Diagram

Figure 17 below is a Component Diagram of this application and it shows the structural relationships between the components of the GUI-based simulation software. There are only software objects shown with this diagram.

According to the diagram, there are four components of this system: Queue, Server, Simulation (GUI) and Web Application. Also there is a component box to represent the back-end where all the statistical calculations are performed. A Toolbox is provided to the Simulation by the Web Application to display the available items and the simulation type is also sent to the Simulation from the Web Application. Both Queue and Server have their corresponding inputs and the Simulation requires both of these items with their parameters to perform statistics. These values are required by the Back-end, and the outputs generated from the calculations are provided to the Simulation. Simulation requires the output values to plot a graph and then it provides the corresponding graph to the Web Application.

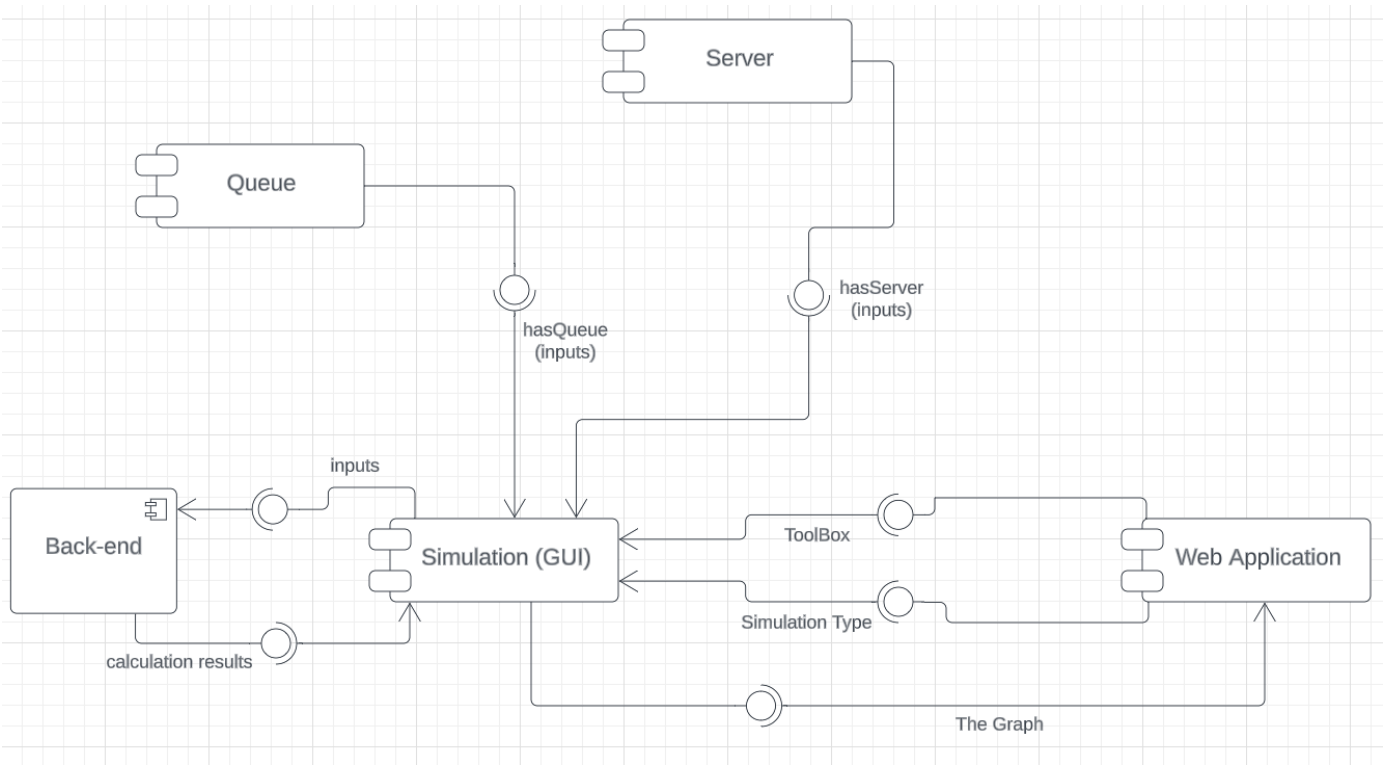

Figure 17: Component Diagram

4.4 Physical View

The physical view shows the mapping of software onto hardware.

4.4.1 Deployment Diagram

Figure 18, the Deployment Diagram of the system, shows the software configuration of the GUI-based simulation software application. Web Server and PC are nodes, run-time physical objects, of this system. They both are connected to each other to communicate properly. The UserInterface is the artefact of this system. As we do not have a database implementation, we do not have it as another node in this diagram as well. So basically this diagram shows the static aspect of the physical nodes and their relationships in the GUI-based simulation software system.

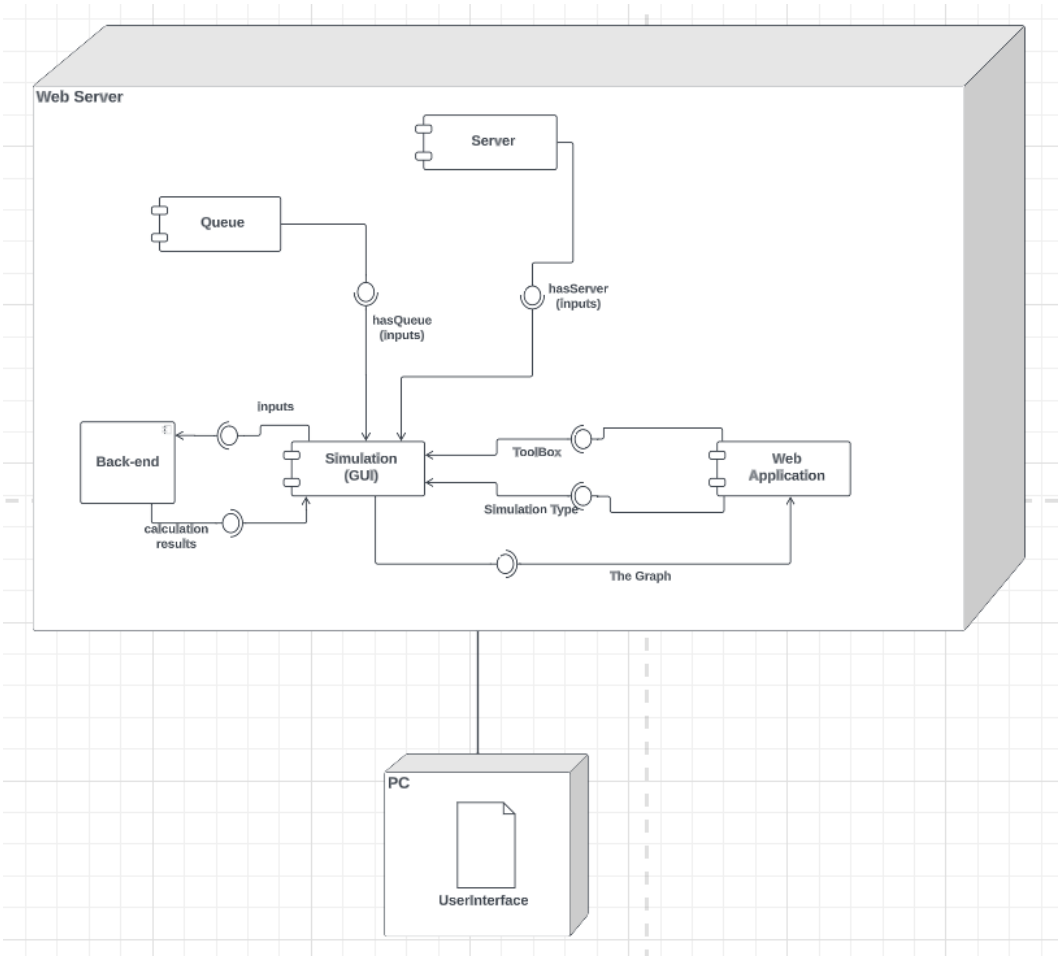

Figure 18: Deployment Diagram

5 Software Implementation

Python is used as a programming language for the back-end part of the project. The functions for the statistical calculations are written in Python. **Math** and **Random** modules are used to make the calculations easier and faster.

HTML, JavaScript and **CSS** languages are used for the front-end part of the project. The user interface (web pages) are written using HTML and CSS. The code to display the graphs for the outputs of the simulations is written in JavaScript. Some external scripts are also added as a source to the related HTML files to provide some functionalities such as using a canvas, creating a graph (chart) and plotting the graph. All these scripts can be found in the whole project file (the package submitted) for more details.

Flask framework is used for the integration between back-end and front-end of the application.

6 Software Testing

This section explains how the testing period for the final product is conducted and the results. During the testing period, we conducted Unit, Integration, System, Performance, and User Testing as planned in the test plan. Regarding Unit testing, we individually verified the correctness and functionality of simulation-related code elements. Regarding Integration testing, we confirmed the connection between different pages in the web application. We also tested the integration of the front-end and back-end components to ensure the inputs provided by the user successfully can be accessed and used by the simulation code. We performed the System testing to ensure that the entire software functions as intended and meets the specified requirements by checking all aspects of the software to ensure it satisfies the desired expectations. Considering Performance testing, we verified the performance of generating simulations under different workloads. In terms of user testing, we performed it by creating a questionnaire and delivering it to participants who are from the computer engineering department having the required amount of knowledge to use the application. Regarding the questionnaire answers, we verified that all the components, including the drag-and-drop feature, home page, and help page, are easy to use and understand. To sum up, we have completed all the tests specified in the test plan to confirm the quality and reliability of the GUI-based Simulation of Queuing software.

6.1 Unit Testing

6.1.1 Test Procedure

For Unit Testing, plotting of the graphs for M/M/1, M/M/1/L, M/M/C and M/M/C/L simulations were tested by running the HTML files for them in a browser and manually checking if the plotting function works and plots a line graph with the calculated results from the code without any problems; in addition to that, validation checking for the inputs entered for each function for M/M/1, M/M/1/L, M/M/C and M/M/C/L was done on PyUnit. Erem created the test cases and performed the validation checks, and Ceazar did the other parts of the testing listed below in the next section, 2.1.2. A sample code we used for M/M/1/L Unit Testing in PyUnit is provided below, Figure 19, and this code calculates the MQL value and prints it. When the user enters the parameter inputs, the inputs are checked; if there is a negative input it will print an error message, otherwise it will calculate the MQL value and then send it to the front end. This code is used for all test cases regarding M/M/1/L. Other sample codes used regarding M/M/1/L calculations are provided in the Test Cases section.

```
1  import unittest
2  import random
3  import math
4  from mm1Ldiscrepancy import calculate_discrepancy_mm1l
5
6  class test_mm1(unittest.TestCase):
7
8   def test_discrepancy(self):
9     discrepancy = calculate_discrepancy_mm1l(2,4,10000,500)
10    self.assertNotEqual(discrepancy,-1)
11
12 unittest.main()
```

Figure 19: Sample Code Used for M/M/1/L Simulation in PyUnit

6.1.2 Test cases

ID	Description	Steps	Test Data	Pre-condition	Expected Output	Passed/ Failed
1	*Testing M/M/1 with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue, Number of entities) for M/M/1 validation function.* *2. Run the code*	*Lamda (Arrival rate) = 2* *Mu (Service rate) = 5* *Number Of Entities = 10,000*	-	*Print the MQL value, MQL is expected to be close to 0.67*	*Passed*
2	*Testing M/M/1 with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue, Number of entities) for M/M/1 validation function.* *2. Run the code*	*Lamda (Arrival rate) = 2* *Mu (Service rate) = 5* *Number Of Entities = 1,000,000*	-	*Print the MQL value, MQL is expected to be close to 0.67*	*Passed*
3	*Testing M/M/1 with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue, Number of entities) for M/M/1 validation function.* *2. Run the code*	*Lamda (Arrival rate) = 2* *Mu (Service rate) = 1* *Number Of Entities = 10,000*	-	Print an error message stating that the Lambda value should be less than Mue	*Passed*
4	*Testing M/M/1/L with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue, Number of entities, Queue size limit) for M/M/1/L validation function.* *2. Run the code*	*Lamda (Arrival rate) = 2* *Mu (Service rate) = 4* *Number Of Entities = 10,000* *Queue size = 500*	-	*Print the MQL value, MQL is expected to be close to 1.0*	*Passed*
5	*Testing M/M/1/L with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue, Number of entities, Queue size limit) for M/M/1/L validation function.* *2. Run the code*	*Lamda (Arrival rate) = -3* *Mu (Service rate) = 5* *Number Of Entities = 10,000* *Queue size = 1000*	-	*P*rint an error message stating that the Lambda value cannot be a negative number	*Passed*
6	*Testing M/M/C with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue for each server, Number of entities, Number of servers) for M/M/C validation function.* *2. Run the code*	*Lamda (Arrival rate) = 2* *Mu (Service rate) = 5 for all servers* *Number Of Entities = 10,000* *Number of servers = 2*	-	*Print the MQL value, MQL is expected to be close to 0.416 with 2 servers*	*Passed*
7	*Testing M/M/C with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue for each server, Number of entities, Number of servers) for M/M/C validation function.* *2. Run the code*	*Lamda (Arrival rate) = 2* *Mu (Service rate) = 5,8,6* *Number Of Entities = 50,000* *Number of servers = 3*	-	*Print the MQL value, MQL is expected to be close to 0.40 with 3 servers*	*Passed*
8	*Testing M/M/C/L with validation checking in Python using PyUnit library*	*1. Enter values for the input parameters (Lambda, Mue for each server, Number of entities, Number of servers, Queue size limit) for M/M/C/L validation function.*	*Lamda (Arrival rate) = 2* *Mu (Service rate) = 5 for all servers* *Number Of Entities = 10,000*	-	*Print the MQL value, MQL is expected to be close to 0.416 with 2 servers*	*Passed*

		2. Run the code	Number of servers = 2 Queue size = 50			
9	Testing M/M/C/L with validation checking in Python using PyUnit library.	1. Enter values for the input parameters (Lambda, Mue for each server, Number of entities, Number of servers, Queue size limit) for M/M/C/L validation function. 2. Run the code	Lamda (Arrival rate) = 2 Mu (Service rate) = 5,8,6 Number Of Entities = 50,000 Number of servers = 3 Queue size =50	-	Print the MQL value, MQL is expected to be close to 0.40 with 3 servers	Passed

Table 15: Sample Cases for Unit Testing

```
4
5   def calculate_discrepancy_mm1l(lamda, mue, Number_of_Entities,L_size):
6
7     if(lamda>=mue) or (lamda<0) or (mue<0) or (Number_of_Entities<0) or (L_size<0):
8       mql = 0
9       return -1
10    time_Arrival = 0
11
12    Iat = -(1 / lamda) * math.log(random.random())
13    Start_Serving = 0
14    entity_list = []
15    finish_Time = Start_Serving + (-(1 / mue) * math.log(random.random()))
16    total_waiting_time = 0
17    finish_list = []
18    MQL_analytic_list = []
19    mql_list = []
20    mql = 0
21    l=0
22    j=0
23    int()
24    for i in range(int(Number_of_Entities)):
25
26        if(l == L_size):
27          continue
28        else:
29          time_Arrival += Iat
30          Iat = -(1 / lamda) * math.log(random.random())
31          Start_Serving = max(time_Arrival, finish_Time)
32          if(Start_Serving == time_Arrival):
33              l = l+1
34          else:
35              if(l>0):
36                l = l-1
37              else:
38                l = 0
39          St = -(1 / mue) * math.log(random.random())
40          finish_Time = Start_Serving + St
41        entity_list.append(i)
42        finish_list.append(finish_Time)
43        waiting = finish_Time - time_Arrival
44        mql += waiting
45        if(abs((((mql/finish_Time)-(lamda/(mue-lamda)))*100)/(lamda/(mue-lamda)))<=0.25):
46            mql_list.append(mql / finish_Time)
47            j = i
48            break
49        mql_list.append(mql / finish_Time)
50        total_waiting_time += waiting
51        j=i
52
53
54    mql = mql/finish_list[j]
55    print("Analytical Mql: ",(lamda/(mue-lamda)))
56    print("MQL: ",mql)
57    Discrepancy = abs((((mql)-(lamda/(mue-lamda)))*100)/(lamda/(mue-lamda)))
58    return Discrepancy
```

Figure 20: Discrepancy Calculation Code Used for M/M/1/L Simulation in PyUnit

6.1.3 Test Results

Referring to the table above, Table 15, all of our test cases for the unit testing that has been done passed and no failures occurred.

6.1.4 Test Log

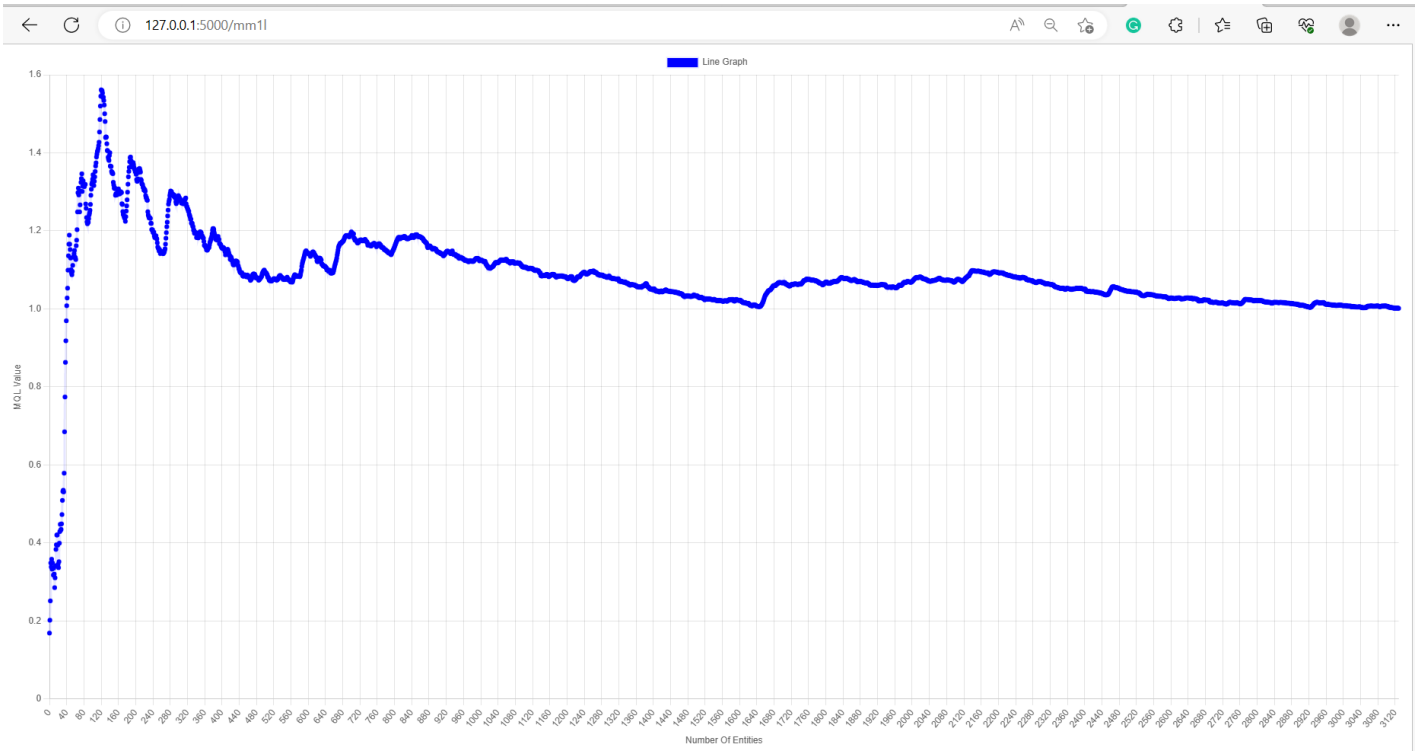

Figure 21: Case 4 (successful validation and plotting the line graph)

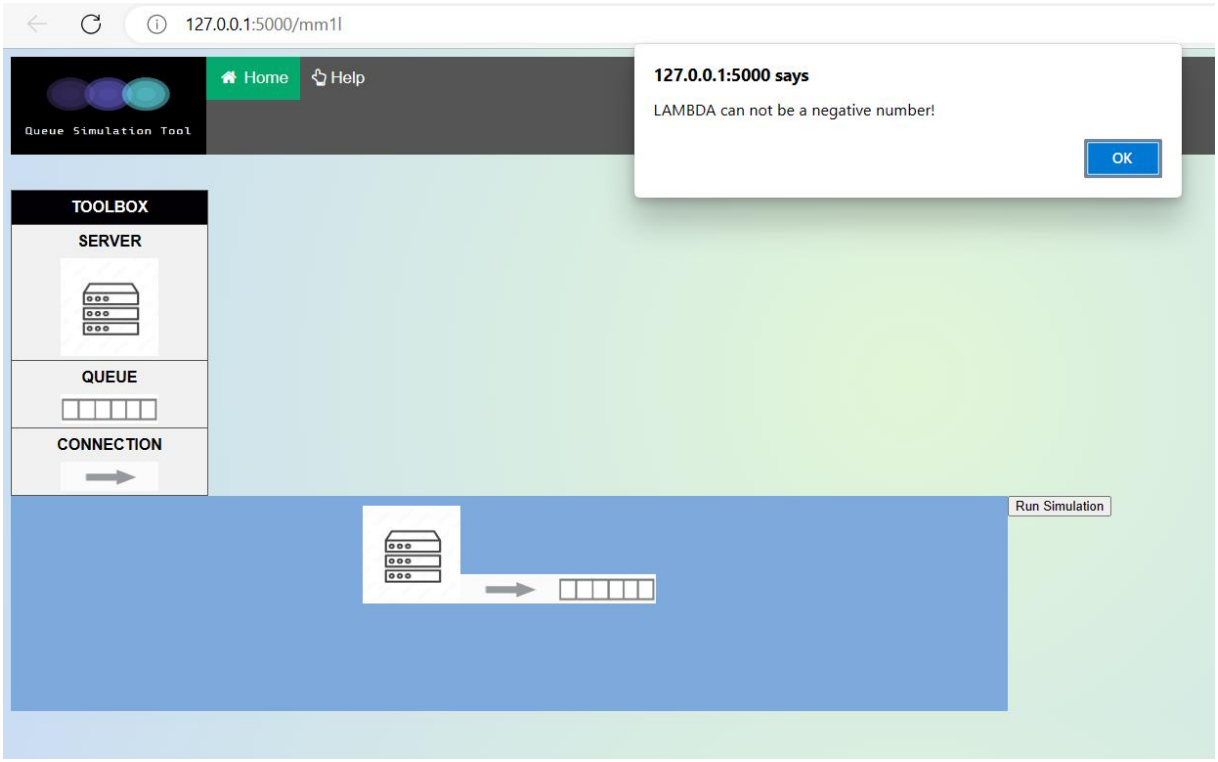

Figure 22: Case 8 (one of the inputs, Lambda, is negative)

6.2 Integration Testing

6.2.1 Test Procedure

Integration testing has been performed by using the bottom-up technique. We first tested each file/module separately and then did the testing for the whole program. We used Python Flask for the integration of the GUI part and the simulation part. We have .html and .js files for the GUI part and .py and .js files for the simulation part of the project. We performed all the test cases manually as we needed to check the different values and errors separately for each specified case. For the GUI part, we tested if all the web pages were connected to each other successfully and if we could switch between pages without any problem. For the simulation part, we tested if the values entered for each test case were saved correctly and if the corresponding graphs were plotted correctly with these inputs. We have tested nine graph-related files (five .js and four .html files) for this part. In addition, some of these files are related to both sides, so their testing was performed after completing separate tests for each part. For instance, the files containing drag-and-drop-related features where the values needed to be taken from the user in the GUI part and sent to the main .py file to be checked for validations and then to the corresponding .js file to plot the graph.

Ece performed the GUI part and drag-and-drop-related cases (1&2), and Erem performed the rest of the testing which are related to the simulation part (saving values and plotting graphs-cases 3&4&5).

6.2.2 Test cases

The test cases (after case 2) given in the table below, Table 16, are related to each other so they are tested together to show if their integration is successful or not.

ID	***Description***	***Steps***	***Test Data***	***Pre-condition***	***Expected Output***	***Passed /Failed***
1	*Connection of each web page to each other*	*1*	*-*	*-*	*Being able to switch between web pages*	*Passed*
2	*Entering values for Server(s) and Queue*	*1*	*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*	*Selection of M/M/1 simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*
		2	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Selection of M/M/1/L simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Selection of M/M/C simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*
		4	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Selection of M/M/C/L simulation in the Home page*	*Being able to send the entered values to the corresponding parameters*	*Passed*

3	*Accessing values for Server(s) and Queue in the main.py file*	*1*	*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 3-step 1*	*Printing the entered values correctly in the main.py console*	*Passed*
		2	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Verifying test case 3-step 2*	*Printing the entered values correctly in the main.py console*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 3-step 3*	*Printing the entered values correctly in the main.py console*	*Passed*
		4	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Verifying test case 3-step 4*	*Printing the entered values correctly in the main.py console*	*Passed*
4	*Accessing values for Server(s) and Queue in the corresponding .js files for each graph*	*1*	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Verifying test case 4-step 2*	*Printing the entered values correctly inside the mm1L_graph.js*	*Passed*
		2	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 4-step 3*	*Printing the entered values correctly inside the mmc_graph.js*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Verifying test case 4-step 4*	*Printing the entered values correctly inside the mmcL_graph.js*	*Passed*
5	*Plotting the correct graphs with the values entered for each case*	*1*	*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 5-step 1*	*Plotting an MM1 graph as expected*	*Passed*
		2	*Server(mue)=5, Queue(lambda,# of entities,queue size)=3, 2500, 20*	*Verifying test case 5-step 2*	*Plotting an MM1L graph as expected*	*Passed*
		3	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*	*Verifying test case 5-step 3*	*Plotting an MMC graph as expected*	*Passed*
		4	*Server0(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities,limit)=3, 2500,10*	*Verifying test case 5-step 4*	*Plotting an MMCL graph as expected*	*Passed*

Table 16: Sample Cases for Integration Testing

6.2.3 Test Results

All the test cases provided in the table above are passed the integration testing and how they are tested are provided with more details on the next section, 2.2.4, with their corresponding screenshots as well.

6.2.4 Test Log

One of the cases tested is provided with detailed steps below:

The sample test case for an M/M/C simulation with the correct inputs and its graph plotted:

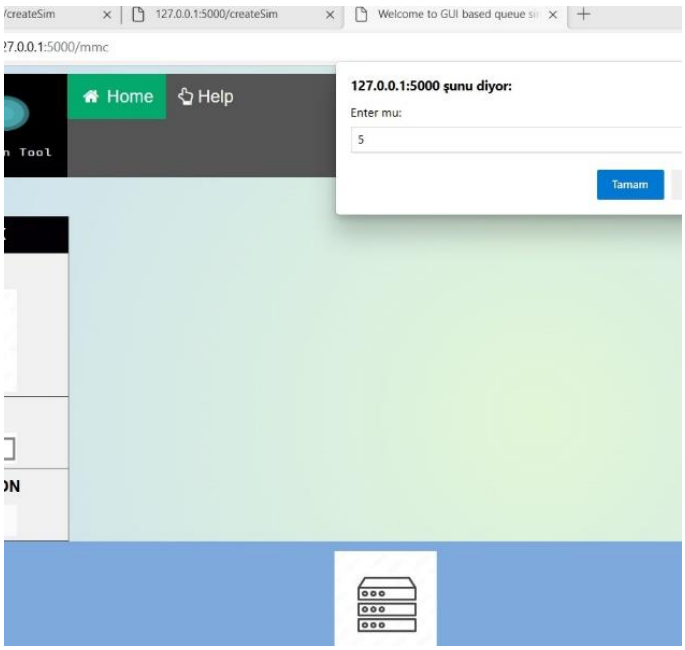

Figure 23: Entering the values, Mue value for Server 0

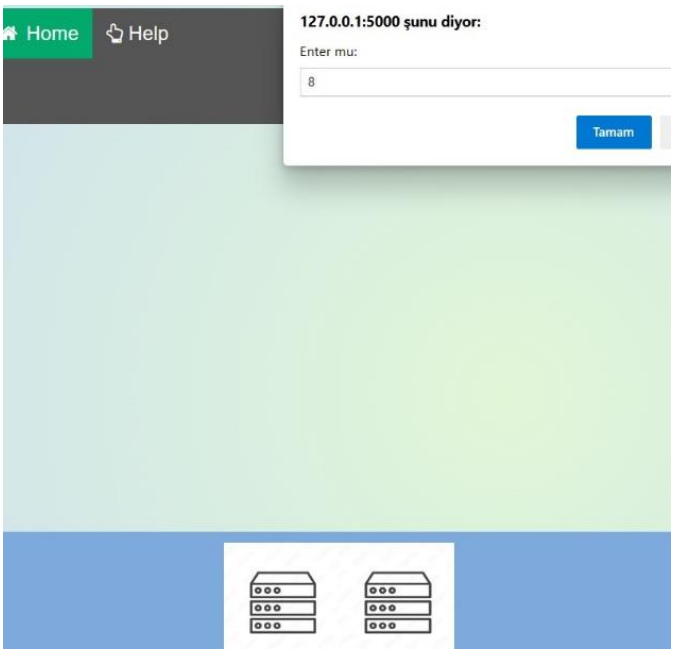

Figure 24: Entering the values, Mue value for Server 1

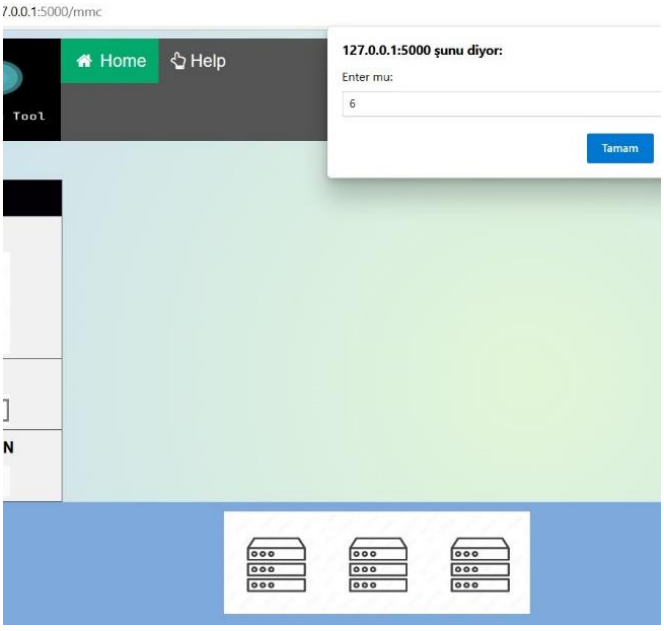

Figure 25: Entering the values, Mue value for Server 2

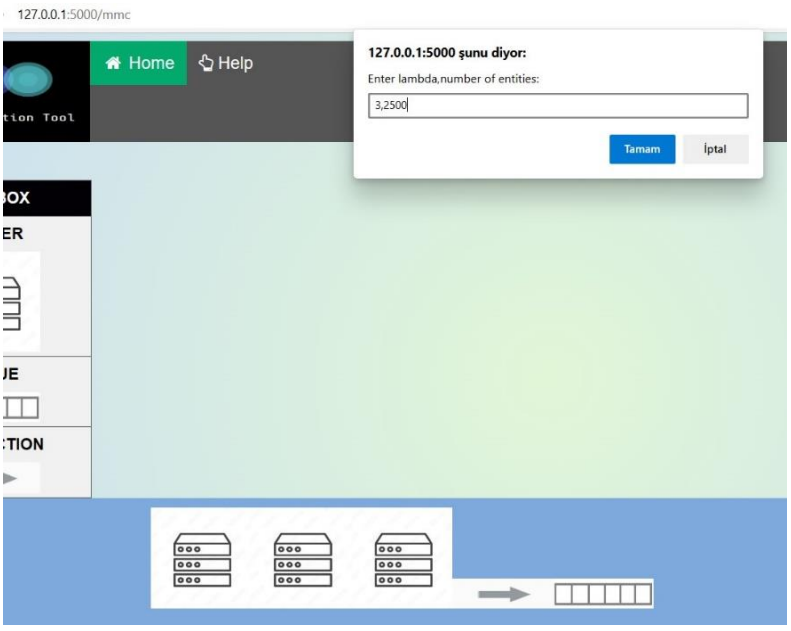

Figure 26: Entering the values, Lambda and Number of Entities

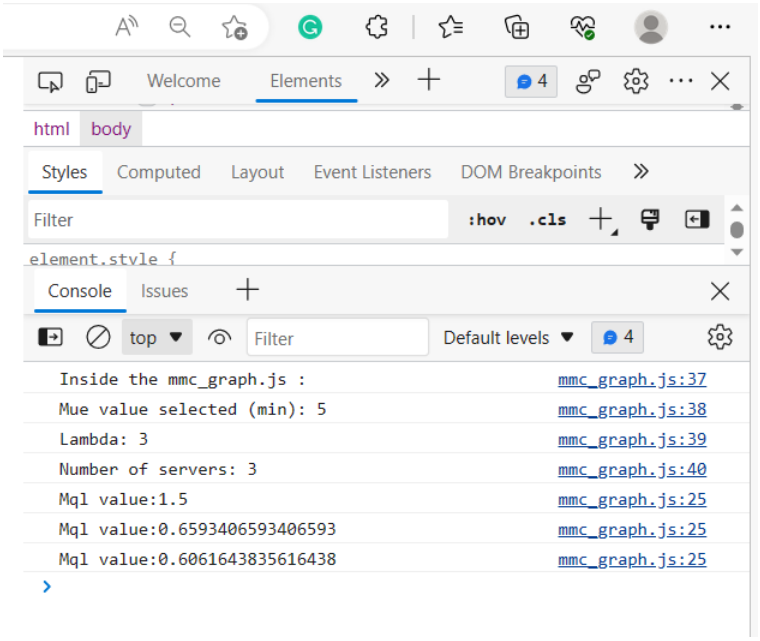

Figure 27: Values saved inside the mmc_graph.js file

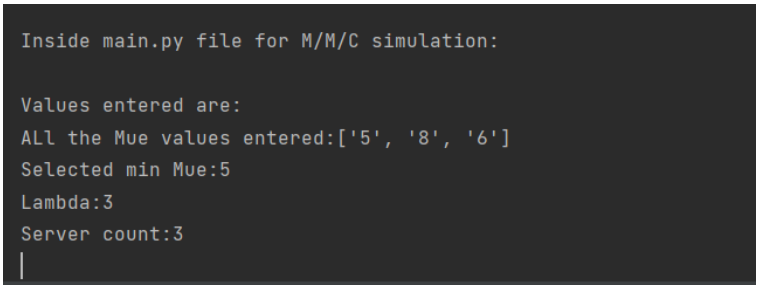

Figure 28: Values saved inside the main.py file

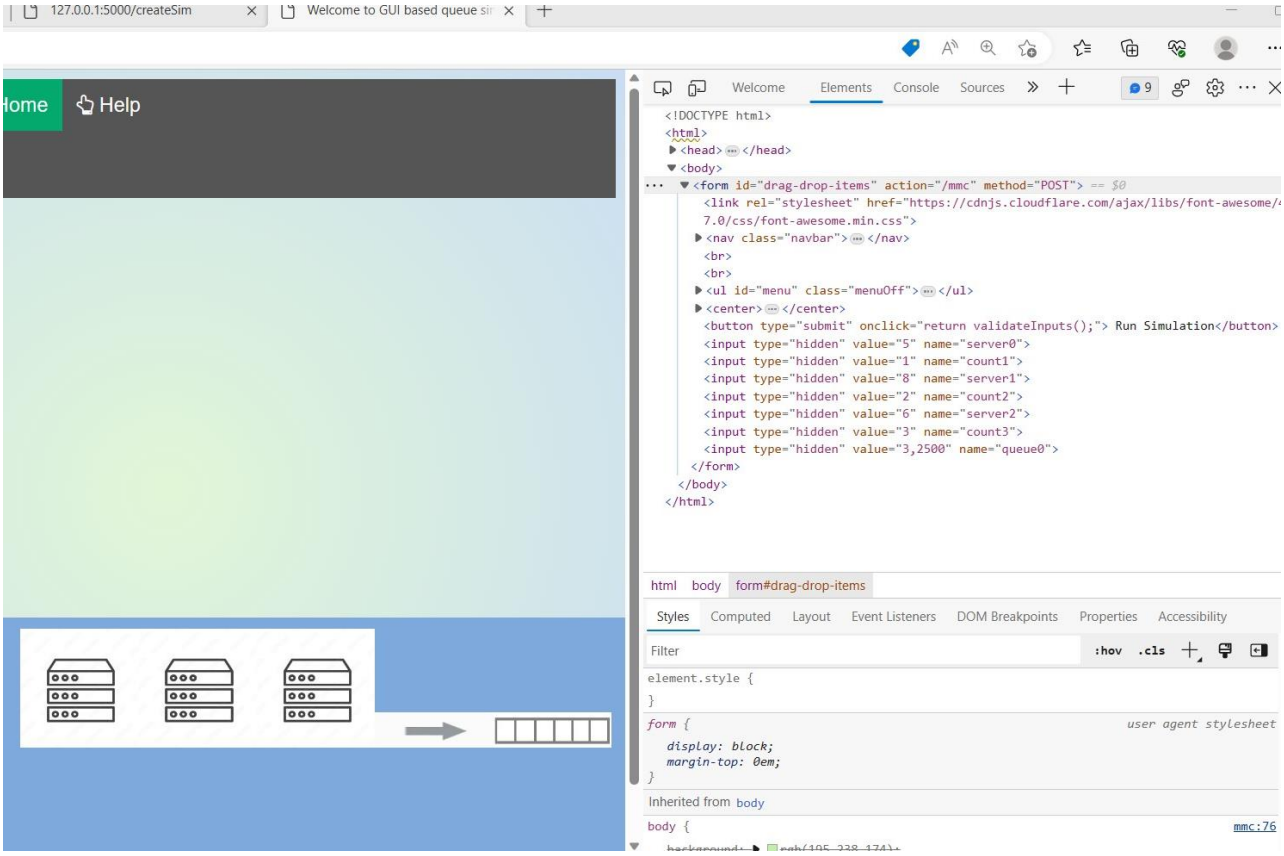

Figure 29: Values entered saved in the web page

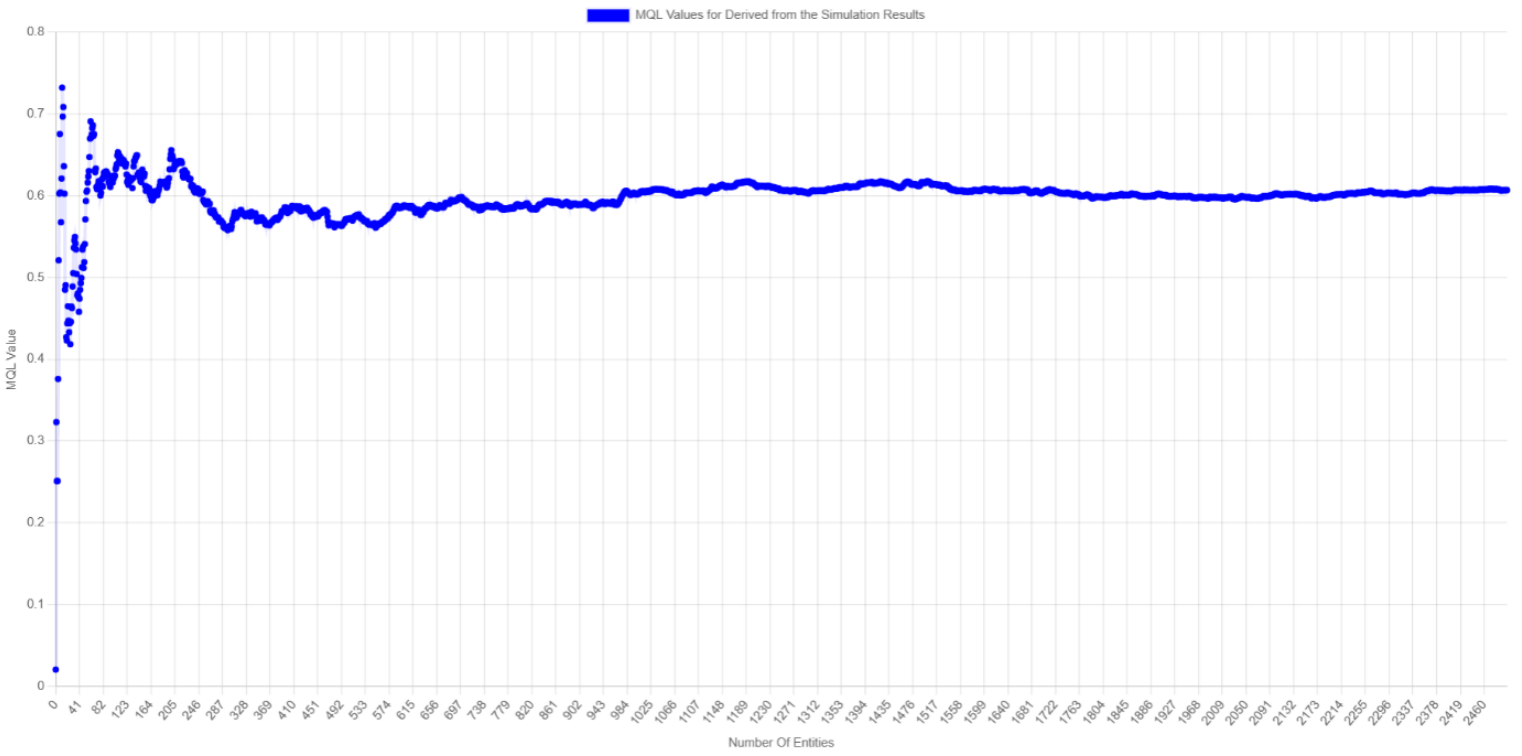

Figure 30: Graph plotted for the MQL value calulated according to the entered values (the MQL values can be verified with Figure 7 outputs as well)

6.3 System Testing

6.3.1 Test Procedure

We have used scenario-based system testing to perform system testing for this project. Several scenarios were created to split between the group members, which were tested to see if the whole system worked flawlessly in harmony. We manually tested these scenarios by following the steps provided below. Ece tested cases 1&2&3, Erem tested cases 4&5& 6.

6.3.2 Test Scenarios

The cases tested are as listed below:

1. Home -> Create Simulation -> M/M/1 -> Entering values (*Server(mue)=5, Queue(lambda, # of entities)=3, 2500*)->Run Simulation

2. Home -> Create Simulation -> M/M/1/L -> Entering values (*Server(mue)=5, Queue(lambda,# of entities, queue size)=3, 2500, 20*)->Run Simulation

3. Home -> Help -> Creating a new simulation -> Create Simulation -> M/M/1 -> Entering values (*Server(mue)=6, Queue(lambda, # of entities)=2, 3700*)->Run Simulation

4. Home -> Contact Us -> Home -> Create Simulation -> M/M/C -> Entering values (*Servero(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities)=3, 2500*)->Run Simulation

5. Home -> Help -> Create Simulation -> M/M/C/L -> Entering values (*Servero(mue)=5, Server1(mue)=8, Server2(mue)=6, Queue(lambda, # of entities, limit)=3, 2500,10*)->Run Simulation

6. Home -> Create Simulation -> M/M/C/L -> Entering values (*Servero(mue)=5, Server1(mue)=3, Server2(mue)=11, Server3(mue)=4, Queue(lambda, # of entities, limit)=2, 10000,8*)->Run Simulation

6.3.3 Test Results

All the test scenarios given in section 2.3.2 are passed the system testing without any minor/major errors and the screenshots are provided for one of those cases in the section 2.3.4.

6.3.4 Test Log

Case 3:

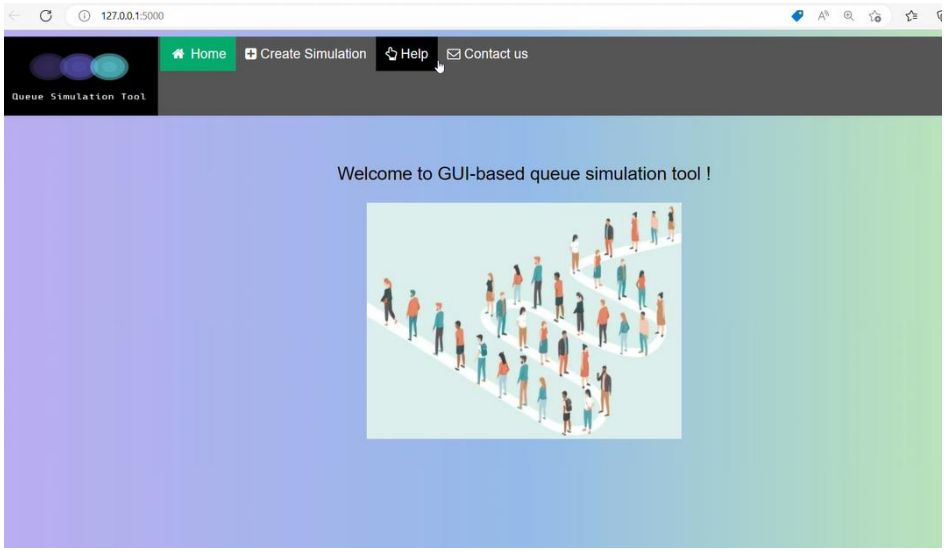

Figure 31: Home Page->Help Page

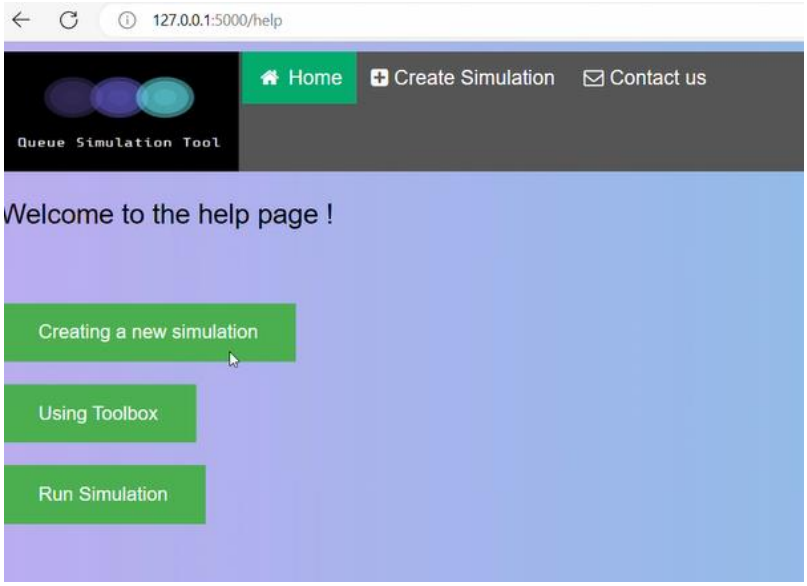

Figure 32: Help Page -> Creating a new simulation

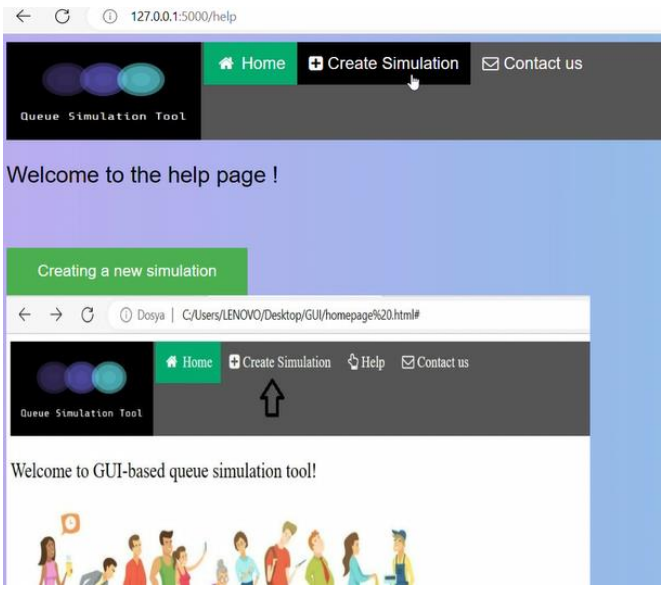

Figure 33: Creating a new simulation ->Create Simulation

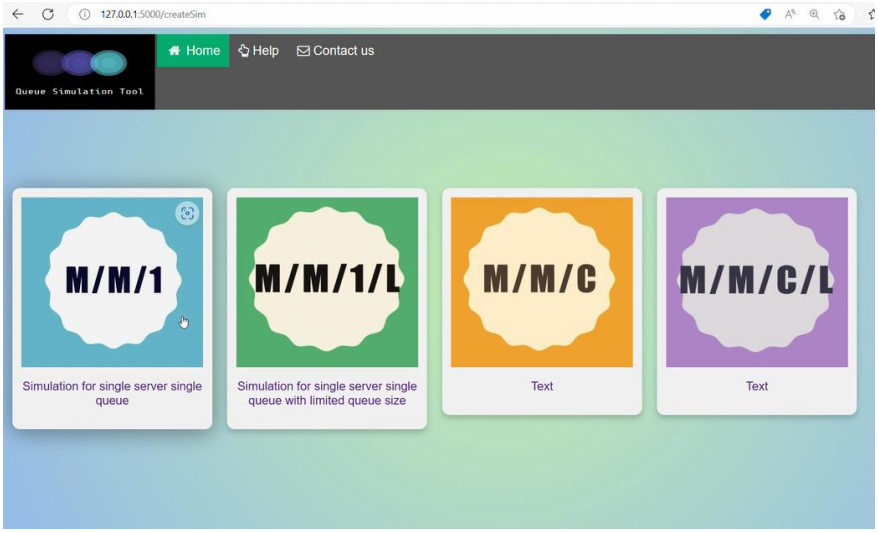

Figure 34: Create Simulation -> M/M/1

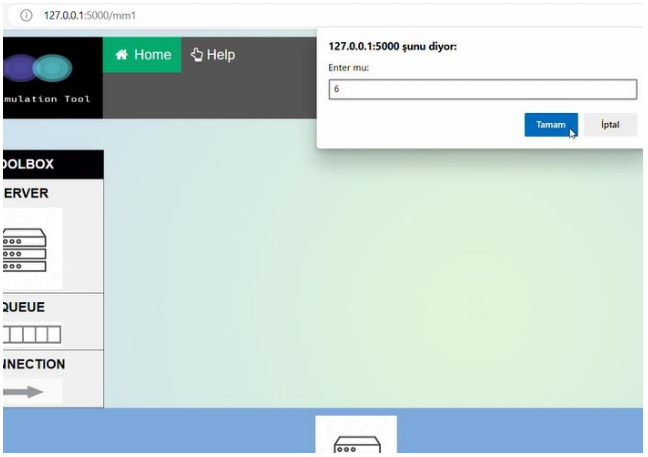

Figure 35: Entering values, Mu for Server

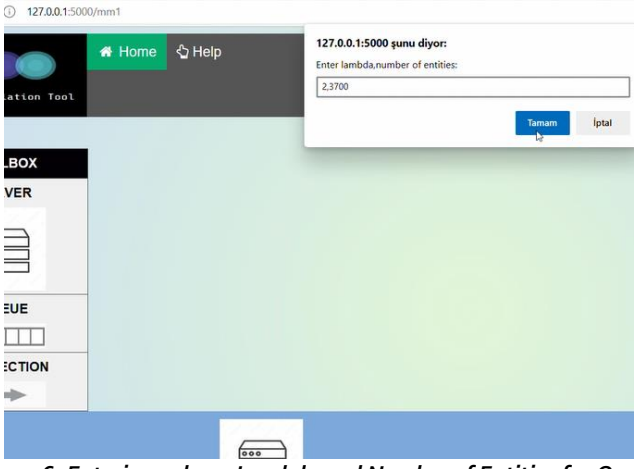

Figure 36: Entering values, Lambda and Number of Entities for Queue

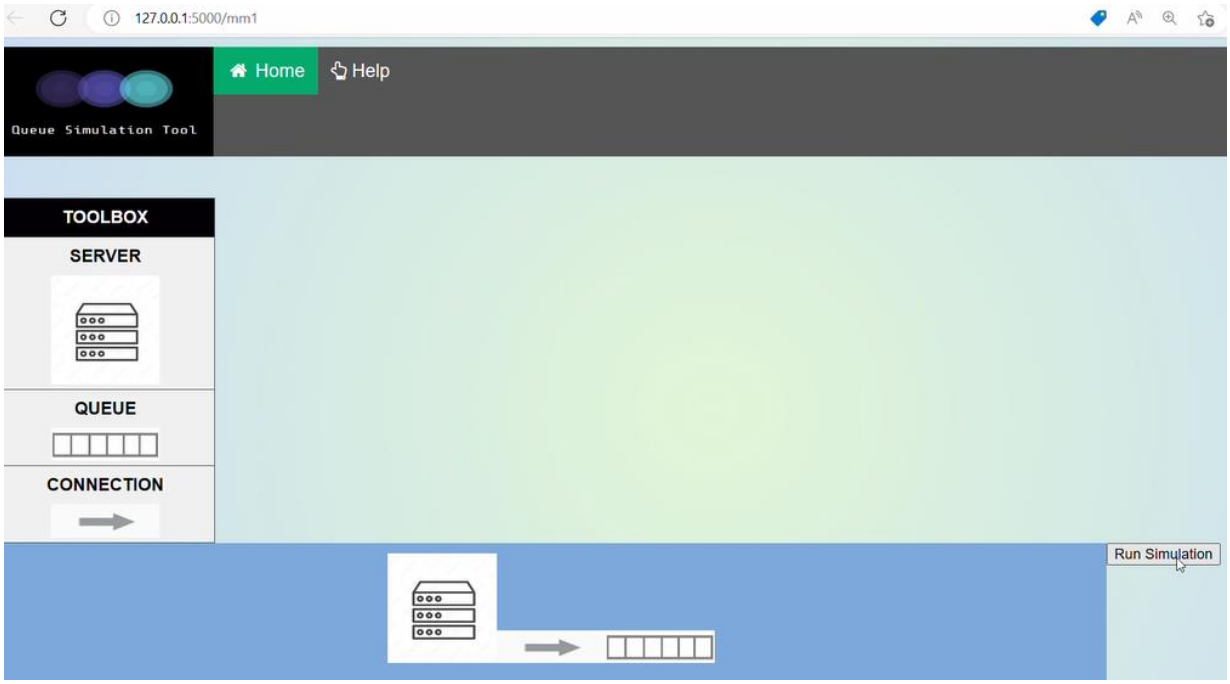

Figure 37: Run Simulation

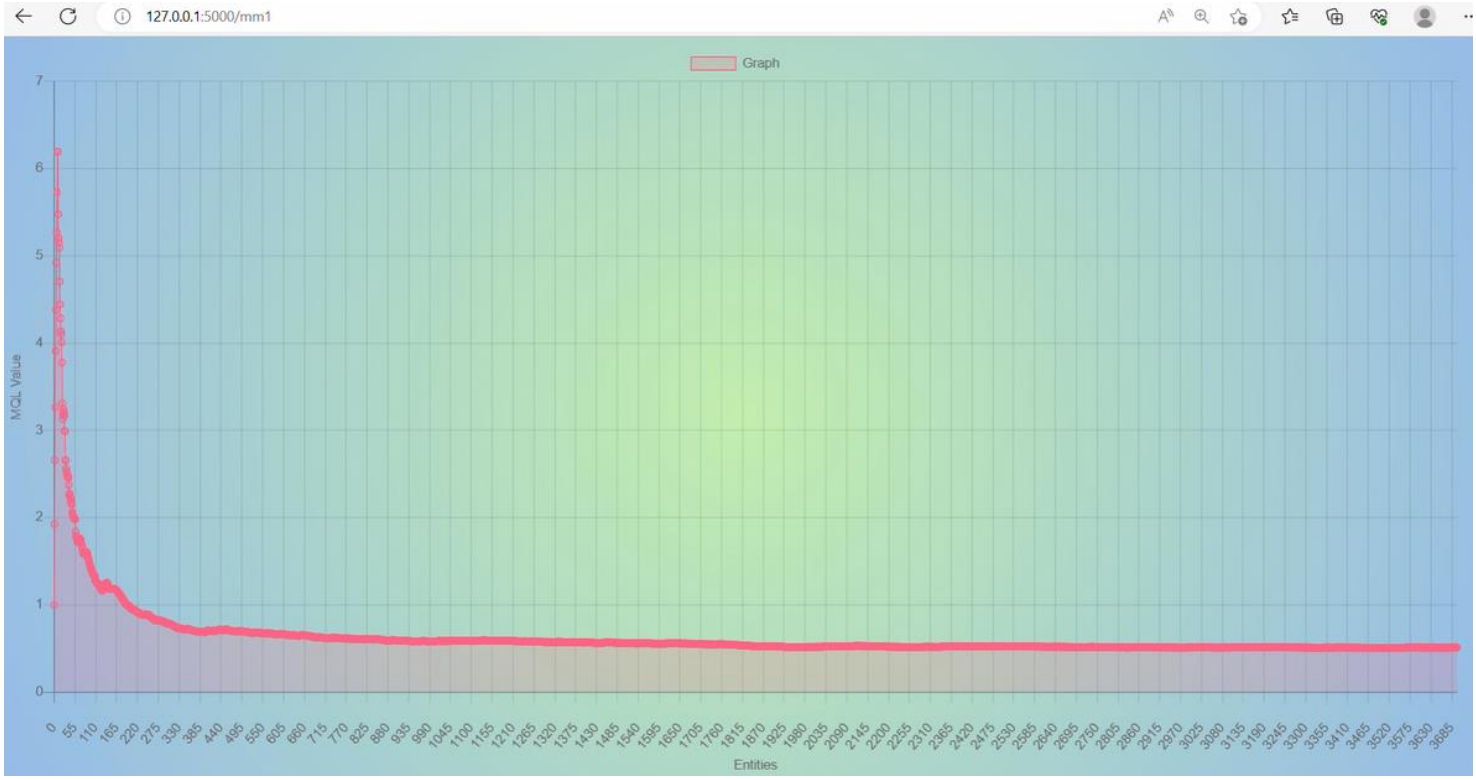

Figure 38: Plotted Graph of the Entered Values

6.4 Performance Testing

6.4.1 Test Procedure

For performance testing, we used stress testing technique to see how the system behaves with a huge amount of data, in addition to that, we want the system to be very accurate in terms of results, the expected result is calculated from the analytical approach (for M/M/1=(lamda/mue)/(1 - (lamda/mue)) and for M/M/1/L = lamda/(mue – lamda)), and then discrepancy of each entity is calculated by absolute of [(MQL - lamda/(mue – lamda) x 100) / lamda/(mue – lamda)]. Discrepancy is used to check how close are the results for MQL for both simulation and analytical methods, in performance we are testing the accuracy of the simulation by calculating the discrepancy formula which is mentioned before, in addition to that, it should be maximum equal to 0.25 so we can get very accurate results. PyUnit has been used to test the performance of the simulation with very large number of entities, and to check the discrepancy to see if passes the criteria or fails. Ceazar and Erem were responsible for performance testing.

6.4.2 Test cases

ID	Description	Steps	Test Data	Pre-condition	Expected Output	Passed/ Failed
1	Testing M/M/1 by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.	1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities. 2. Run the code. 3. Get result for the test, either Pass or Fail.	Lamda = 2 Mue = 5 Number of entities = 10,000	-	Discrepancy <= 0.25	Passed
2	Testing M/M/1 by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.	1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities. 2. Run the code. 3. Get result for the test, either Pass or Fail.	Lamda = 2 Mue = 5 Number of entities = 100,000	-	Discrepancy <= 0.25	Failed
3	Testing M/M/1 by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.	1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities. 2. Run the code. 3. Get result for the test, either Pass or Fail.	Lamda = 2 Mue = 5 Number of entities = 1,000,000	-	Discrepancy <= 0.25	Passed
4	Testing M/M/1 by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.	1. Enter data for the input parameters for M/M/1: Lamda, Mue, Number of entities. 2. Run the code. 3. Get result for the test, either Pass or Fail.	Lamda = 2 Mue = 5 Number of entities = 2,000,000	-	Discrepancy <= 0.25	Passed
5	Testing M/M/1/L by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.	1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size. 2. Run the code. 3. Get result for the test, either Pass or Fail.	Lamda = 2 Mue = 5 Number of entities = 10,000 Queue limit = 500	-	Discrepancy <= 0.25	Passed
6	Testing M/M/1/L by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.	1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size. 2. Run the code. 3. Get result for the test, either Pass or Fail.	Lamda = 2 Mue = 5 Number of entities = 100,000 Queue limit = 500	-	Discrepancy <= 0.25	Passed
7	Testing M/M/1/L by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.	1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size. 2. Run the code. 3. Get result for the test, either Pass or Fail.	Lamda = 2 Mue = 5 Number of entities = 1,000,000 Queue limit = 500	-	Discrepancy <= 0.25	Passed

8	*Testing M/M/1/L by checking accuracy of the simulation with discrepancy formula in Python using PyUnit library.*	*1. Enter data for the input parameters for M/M/1/L: Lamda, Mue, Number of entities, Queue limit size.* *2. Run the code.* *3. Get result for the test, either Pass or Fail.*	*Lamda = 2* *Mue = 5* *Number of entities = 2,000,000* *Queue limit = 500*	-	*Discrepancy <= 0.25*	*Passed*

Table 17: Sample Cases for Performance Testing

6.4.3 Test Results

Referring to the table above, Table 17, all of our test cases except **case 2** for the performance testing that has been done passed and no failures occurred.

Case 2: Fail error message, failed since discrepancy was greater than 0.25, it must be less than 0.25, otherwise our simulation fails the performance testing due to not achieving high accuracy for the results.

In Figure 39, the speed results for M/M/1 simulation, stress testing, are shown in a graph generated by using the values in Table 4. How the execution time changes under huge amount of data, number of entities when other parameters are fixed, can be seen from the graph.

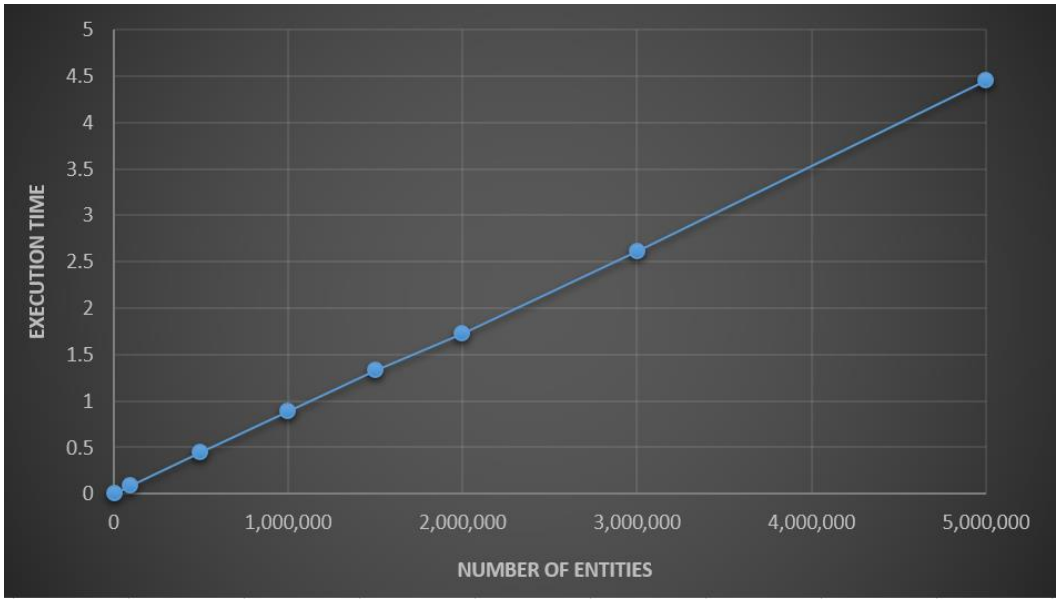

Figure 39: Performance Testing Speed Results

Number of Entities	***Execution Time (in sec)***
10,000	0.009
100,000	0.084
500,000	0.442
1,000,000	0.889
1,500,000	1.334
2,000,000	1.728
3,000,000	2.605
5,000,000	4.452

Table 18: **M/M/1 Simulation Sample Speed Results**

6.4.4 *Test Log*

During the execution of the system performance testing, we do a validation check for the input data as well similar to the unit testing, in addition to that, we check the value of the discrepancy and depending on that, we decide if the test fails or passes.

```
D: > 492 > testresults > testmm1.py > test_mm1 > test_discrepancy
  1  import unittest
  2  import random
  3  import math
  4  from mm1discrepancy import calculate_discrepancy
  5
  6  class test_mm1(unittest.TestCase):
  7
  8   def test_discrepancy(self):
  9     discrepancy = calculate_discrepancy(2,5,100000)
 10     self.assertNotEqual(discrepancy,-1)
 11     self.assertTrue(abs(discrepancy)<=0.25)
 12
 13  unittest.main()

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

:\492\testresults\testmm1.py'
Mql Analytical is:  0.6666666666666667
MQL is:  0.6730827562393688
F
======================================================================
FAIL: test_discrepancy (__main__.test_mm1)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "d:\492\testresults\testmm1.py", line 11, in test_discrepancy
    self.assertTrue(abs(discrepancy)<=0.25)
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.104s
```

Figure 40: Sample Fail Case for Discrepancy Value

```
D: > 492 > testresults > mm1ltest.py > ...
  1  import unittest
  2  import random
  3  import math
  4  from mm1Ldiscrepancy import calculate_discrepancy_mm1l
  5
  6  class test_mm1(unittest.TestCase):
  7
  8   def test_discrepancy(self):
  9     discrepancy = calculate_discrepancy_mm1l(2,5,10000,500)
 10     self.assertNotEqual(discrepancy,-1)
 11     self.assertTrue(abs(discrepancy)<=0.25)
 12
 13  unittest.main()

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\492\testresults>  d:; cd 'd:\492\testresults'; & 'C:\Users\THINK\AppData\Local\Programs\Python\Python310\python.exe' 'c:\User
s\THINK\.vscode\extensions\ms-python.python-2023.8.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '49687' '--' 'd
:\492\testresults\mm1Ldiscrepancy.py'
'; & 'C:\Users\THINK\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\THINK\.vscode\extensions\ms-python.python-2023.8
.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '49794' '--' 'd:\492\testresults\mm1ltest.py'
Analytical Mql:  0.6666666666666666
MQL:  0.6676610605526728
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS D:\492\testresults>
```

Figure 41: Sample Successful Case for the Discrepancy Value

6.5 User Testing

6.5.1 Test Procedure

To conduct user testing, we selected the usability testing method. To perform the user testing, we delivered the System Usability Scale (SUS) questionnaire to a selected group of participants with pre-existing Computer Engineering backgrounds. By selecting participants from the Computer Engineering department, we desired to receive feedback from individuals familiar with the concepts and requirements related to our software tool. First of all, we provided a brief introduction to the simulation tool. Then we asked participants to create a new simulation using the home page, select the simulation type from the selection page, drag and drop server and queues from toolbox to canvas, enter parameters by clicking on the dropped items, and run the simulation. Participants are asked to perform all these operations for all the simulation types. Moreover, participants are also requested to download the simulation graph and visit the help page and contact page. After participants completed all the steps, we shared the questionnaire link and asked them to fill it out objectively. The questionnaire covered the general user experience and various aspects of the software, including complexity, consistency, and integration of the system's functions. The participants are asked to rate the statements on a 5-point Likert scale, ranging from "Strongly Disagree" to "Strongly Agree." Their objective answers supported us in measuring their perceptions of the software's usability and identifying the parts of the system that need improvement. Using the SUS questionnaire, we aimed to get standardized feedback on our software's usability from the participants familiar with Computer Engineering. This enabled us to recognize our software's strengths and weaknesses and complete the necessary changes to improve usability and satisfy better user experiences.

6.5.2 Expectations

It is expected that the user testing will help to evaluate the usability of the web application using the System Usability Scale (SUS). This test's main focuses are the overall usability and user-friendliness of the software. By delivering the SUS scale to users, we can evaluate users' interaction with the graphical user interface (GUI) and determine whether the design elements, layout, and navigation are easy to use and meet the user's expectations. SUS results will also allow us to validate whether the software functions are well-integrated and consistent. 70-75 is the target range for the SUS score since it means a good usability rank. We expect that the user testing process and the calculated SUS score will direct us to improve the software's usability and fix any problems.

6.5.3 Test Results

To calculate the SUS score for each participant, 1 is subtracted from odd-numbered questions' scores, and the even-numbered questions' scores are subtracted from 5 (to prevent negative scores). Then, the sum of the scores per question is calculated. To convert the scale out of 100, the sum is multiplied by 2.5. These calculations are performed for each participant who answered the questionnaire, in order to explain in detail, the calculations for two participants are provided as follows:

User 1:

Question 1: 3 – 1 = 2

Question 2: 5 – 1 = 4

Question 3: 5 – 1 =4

Question 4: 5 – 2 = 3

Question 5: 5 – 1 =4

Question 6: 5 – 1 = 4

Question 7: 5 – 1 =4

Question 8: 5 – 2 = 3

Question 9: 5 – 1 =4

Question 10: 5 – 1 = 4

Sum of the scores = 36
Sum is multiplied by 2.5:
36 * 2.5 = 90

User 2:

Question 1: 5 – 1 = 4

Question 2: 5 – 1 = 4

Question 3: 5 – 1 =4

Question 4: 5 – 2 = 3

Question 5: 5 – 1 =4

Question 6: 5 – 2 = 3

Question 7: 4 – 1 =3

Question 8: 5 – 1 = 4

Question 9: 5 – 1 =4

Question 10: 5 – 3 = 2

Sum of the scores = 35
Sum is multiplied by 2.5:
35 * 2.5 = 87.5

The table in the following figure displays each participant score individually:

Participants	Scores
1	90
2	87.5
3	90
4	82.5
5	82.5
6	80
7	100
8	82.5
9	77.5
10	57.5
11	90
12	67.5
13	100
14	77.5
15	95
16	87.5
17	42.5
18	77.5
19	72.5
20	65
21	97.5
22	70
23	90

Table 19: Participant Scores

To find the final score, sum is computed and divided by the number of users that answered the questionnaire to find the average:

(90 + 87.5 + 90 + 82.5 + 82.5 + 80 + 100 + 82.5 + 77.5 + 57.5 + 90 + 67.5 + 100 + 77.5 + 95 + 87.5 + 42.5 + 77.5 + 72.5 + 65 + 97.5 + 70 + 90) / 23 = 80

The resulting score is calculated as 80. According to an article published in usability.gov (n.d.), a SUS score greater than 68 is considered above the average, so it can be concluded that the SUS result of the simulation software tool is above the average. As a result, the simulation software meets the expectations of the users in many aspects, and the user testing is successful.

6.5.4 *Test Log*

Total of 23 participant answered the SUS questionnaire. The results are transferred into the Google tables as follows:

L2 | fx =sus(B2,C2,D2,E2,F2,G2,H2,I2,J2,K2)

	B	C	D	E	F	G	H	I	J	K
1	1. I think that I woul	2. I found the systen	3. I thought the syst	4. I think that I woul	5. I found the variou	6. I thought there wa	7. I would imagine t	8. I found the syster	9. I felt very confider	10. I needed to learı
2	3	1	5	2	5	1	5	2	5	1
3	5	1	5	2	5	2	4	1	5	3
4	4	1	5	1	5	1	4	1	4	2
5	4	1	5	2	5	1	3	3	5	2
6	4	1	4	2	4	2	4	2	5	2
7	5	1	5	1	5	1	5	1	5	1
8	5	1	5	2	4	2	4	2	4	2
9	4	2	4	4	5	1	4	2	4	1
10	3	2	3	3	4	2	2	2	3	3
11	4	1	5	2	4	1	5	1	5	2
12	5	1	3	2	4	4	4	3	4	2
13	5	1	5	1	5	1	5	1	5	1
14	4	2	5	2	4	3	4	2	4	1
15	5	1	5	2	5	1	5	1	5	2
16	5	2	5	2	5	2	5	2	5	2
17	4	4	4	4	3	5	2	4	4	3
18	4	1	4	2	4	3	4	1	4	2
19	4	2	4	3	4	2	4	2	4	2
20	4	3	4	3	4	3	4	3	5	3
21	5	1	5	1	5	1	5	1	4	1
22	4	2	4	3	5	2	4	3	5	4
23	5	2	5	2	5	1	5	1	4	2

Table 20: SUS Questionnaire Results

The questionnaire link is: https://forms.gle/yWEtntULhtSVBgdv8

7 Project Scheduling

7.1 Milestones and Tasks

ID	Milestone	Date
M1	Implementing clickable feature and updating drag and drop to add and count multiple servers	Week 5
M2	Implementing M/M/1/L queueing simulation and verifying results	Week 6
M3	Improving the design of our project such as adding a new page for selecting simulation type and fixing the contact page	Week 9
M4	Implementing M/M/C queueing simulation and verifying results	Week 10
M5	Implementing M/M/C/L queueing simulation and verifying results	Week 10
M6	Applying unit, integration, user testing to the project	Week 12
M7	Finalizing last details of our project's whole package	Week 13

Table 21: Milestones

ID	Task	Duration	Task Dependency	Responsible Member
T1	Updating the drag and drop by adding multiple servers and counting them	1 Week	-	Ece, Erem
T2	Adding a new feature onto the drag and drop that is a clickable feature to enter inputs directly when you click onto an element (queue, server)	1 Week	(M1)	Ece, Erem
T3	Implementing the M/M/1/L simulation code on JavaScript and validating results	1 Week	(M2)	Ceazar
T4	Adding a new page into our project which the user can select the simulation type they want to run, updating and improving contact page by adding real pictures	1 Week	(M3)	Ece
T5	Implementing the M/M/C simulation code on JavaScript and validating results	1 Week	(M4)	Ceazar, Erem
T6	Implementing the M/M/C/L simulation code on JavaScript and validating results	1 Week	T5 (M5)	Ceazar
T7	Applying unit testing to each function for M/M/1, M/M/1/L, M/M/C, M/M/C/L	1 Week	T3, T5, T6	Ceazar
T8	Applying integration testing by connecting multiple web pages into the main code file	1 Week	T1,T2,T3,T4,T5,T6,T7	Erem
T9	Applying user testing by creating a questionnaire form and asking users about their opinions of the system	1 Week	T1,T2,T3,T4,T5,T6,T7,T8	Ece
T10	Finalizing the end product and preparing it for demo presentations	1 Week	T1,T2,T3,T4,T5,T6,T7,T8,T9	All members

Table 22: Tasks

7.2 Gantt Chart

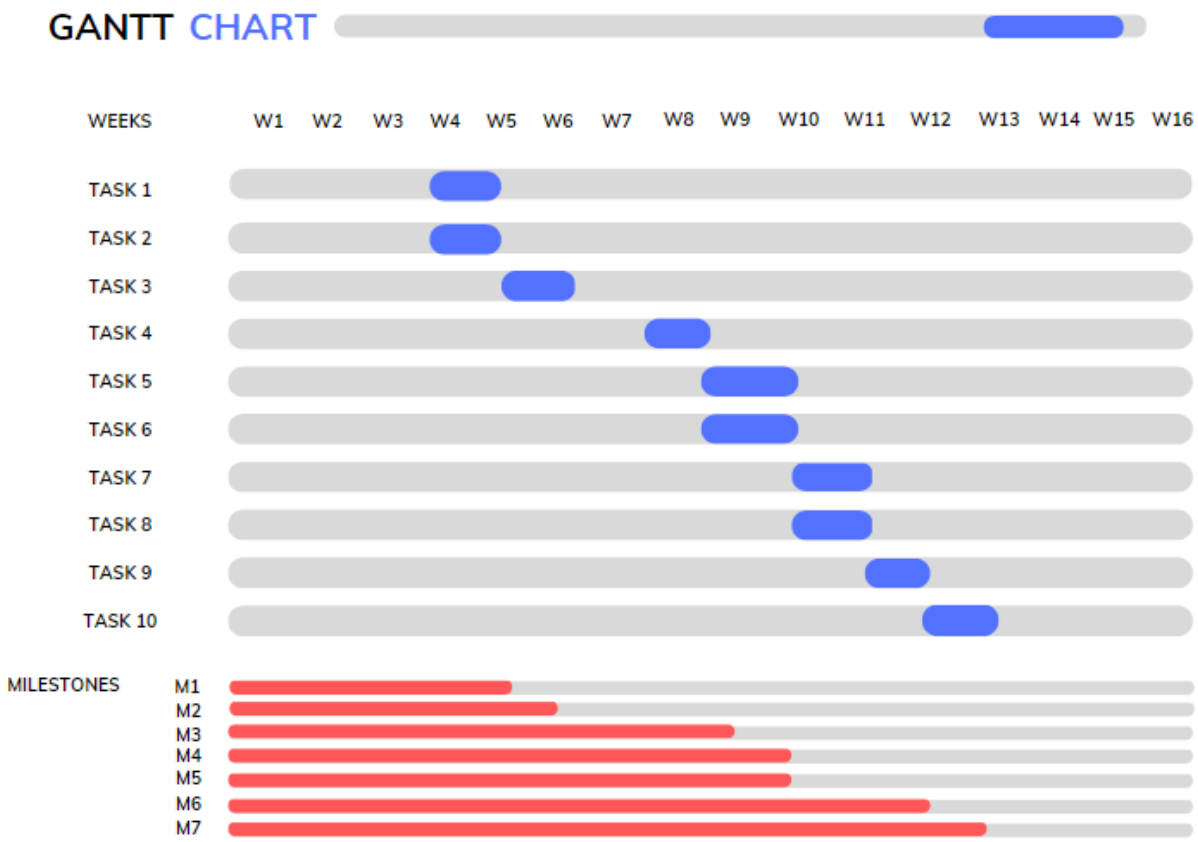

Figure 42: The Gantt Chart

8 Conclusion

A GUI-based Software Package for the Simulation of Queueing Networks is designed and implemented in this project. The software product is available as a web application and builds queueing simulations by supplying a GUI interface to the user. M/M/1, M/M/1/L and M/M/C, and M/M/C/L are four available queueing simulation types of the software. GUI allows users to create simulations, drag the server, queue, and connection from the toolbox, drop on the canvas, enter inputs by clicking on the dropped items, run the simulation, and download the simulation graph as an image file. The software package aims to simulate different queueing models and calculate and visualize the Mean Queue Length (MQL) based on the simulation type by evaluating the input parameters provided by the user. This goal is achieved by implementing the simulation queuing models in backend code using Python, visualizing in fronted by JavaScript (Chart.js), HTML, and CSS, and integrating backend and frontend code with Python Flask. The calculations of MQL provided a helpful understanding of the performance of queuing systems under various configurations, enabling us to explore their behavior accurately. Considering the retrospective of the project, about the successes, as a project team, we successfully completed the

implementation of a GUI-based web application for queueing network simulation for different simulation types and implemented a user-friendly interface. We validated the simulations' accuracy and correctness and achieved the integration of the back and front end. Regarding the challenges, dealing with complex calculations for simulations and implementing drag-and-drop features and clickable components was difficult for us. About lessons learned, the significance of clear communication within the team and task division within the project team, and the importance of having feedback about the implementations can be counted. We also have ideas to implement as a future aspect of our project. Priority queues can be implemented as future work, or we might introduce server failures for M/M/C and M/M/C/L simulation types. To conclude, the project "A GUI-based Software Package for the Simulation of Queueing Networks" is implemented successfully, and thanks to this project, we gained excellent knowledge of queuing theory and its applications.

9 References

7 essential usability metrics and how to use them. (2022, November 4). SaaS UI/UX Design Agency – Eleken. https://www.eleken.co/blog-posts/usability-metrics

Andrzej Chydzinski. (2020). *Response time of the queue with the dropping function, Applied Mathematics and Computation, Volume 377*. https://www.sciencedirect.com/science/article/pii/S0096300320301338

AnyLogic. (n.d.). AnyLogic: Simulation Modeling Software Tools & Solutions for Business. https://www.anylogic.com/

Assistant Secretary for Public Affairs. (2013, September 6). *System usability scale (SUS)*. Usability.gov. https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Average network delay and queuing theory basics. (2018, May 5). Packet Pushers - Where Too Much Technology Would Be Barely Enough. https://packetpushers.net/average-network-delay

B. FILIPOWICZ and J. KWIECIEŃ. (2008). *Queueing systems and networks. Models and applications*. Bulletin of the Polish Academy of Sciences - Technical Sciences. https://bulletin.pan.pl/(56-4)379.pdf

Bitran, Gabriel & Dasu, Sriram. (1992). *A review of open queueing network models of manufacturing systems. Queueing Systems. 12. 95-133*. ResearchGate | Find and share research. https://www.researchgate.net/publication/226090753_A_review_of_open_queueing_network_models_of_manufacturing_systems

Constructive cost model (COCOMO) - TutorialAndExample. (n.d.). Tutorial And Example - A Tutorial Website with Real Time Examples. https://www.tutorialandexample.com/constructive-cost-model-cocomo

John A Miller, Andrew F Seila, Xuewei Xiang. (2000). *The JSIM web-based simulation environment, Future Generation Computer Systems*. https://www.moreno.marzolla.name/publications/papers/The-qnetworks-Toolbox.pdf

Moreno Marzolla. (2012, December 4). *The queueing Package Queueing Networks analysis with GNU Octave*. https://www.cs.unibo.it/~donat/octave-queueing-package.pdf

Moreno Marzolla. (n.d.). *The qnetworks Toolbox: a Software Package for Queueing Networks Analysis*. https://www.moreno.marzolla.name/publications/papers/The-qnetworks-Toolbox.pdf

Queuing theory definition, elements, and example. (2011, January 4). Investopedia. https://www.investopedia.com/terms/q/queuing-theory.asp

Queueing theory for simulation. (2021, September 11). Software Solutions Studio | Software Solutions Studio. https://softwaresim.com/blog/queueing-theory-for-simulation/#:~:text=Queueing%20theory%20can%20utilize%20queueing,Telecommunications%20and%20computer%20systems

Running PyUnit tests and selenium tests created with PyUnit. (2023). Product Support Portal | SmartBear Software. https://support.smartbear.com/testcomplete/docs/working-with/integration/unit-test-frameworks/pyunit.html#:~:text=PyUnit%20is%20a%20standard%20unit,the%20tests%20to%20be%20run

Salsabela, A. A. (2021, February 8). *Make a queuing simulation with ARENA software*. Medium. https://aurumuyunaas.medium.com/make-a-queuing-simulation-with-arena-software-5c2e7a511e70

Vasileios KaryotisVasileios Karyotis, M.H.R. KhouzaniM.H.R. Khouzani. (2016). *Queuing-based malware diffusion modelling*. https://www.sciencedirect.com/topics/computerscience/queuing-network

What is a queue? - Definition from Techopedia. (2011, December 16). Techopedia.com. https://www.techopedia.com/definition/1155/queue-networking

10 Appendices

10.1 Acronyms and abbreviations

GUI: Graphical User Interface

MQL: Mean Queue Length

10.2 Glossary

GUI: It is a digital interface in which a user interacts with graphical components such as items in the Toolbox, buttons, and menus.

Queue: A queue is defined as a linear data structure that is open at both ends and the operations are performed in first in first out order. It might be used for represent a line waiting to purchase tickets, where the first person in line is the first person served (Technopedia, 2011).

Lambda: Arrival rate of the system.

Mue: Service rate of the system.

Queuing Networks: Queuing Networks (QN) are models where customers (service requests) arrive at service stations (servers) to be served. When customers arrive at a busy service station, they are queued for a waiting time until the service station is free (Investopedia, 2011).

Queueing Time: Queueing time is the time spent in the interval from the arrival instant to the instant at which the customer first enters service (Chydzinski & Adamczyk, 2020).

Service Time: The difference between completion time and starting time of the simulation process is the service time (Chydzinski & Adamczyk, 2020).

Response Time (Delay): The response time of a queueing system is defined as the total time a packet spends in the system, including the service time of this packet. It is random in general, so for a full description of the response time, its probability distribution has to be given. Or it can also be calculated by the addition of queuing time and service time of the system (Chydzinski & Adamczyk, 2020).

Traffic Intensity: It is indicated by the ratio of arrival rate and service rate in a queuing system. It is a measure of how close a system is to capacity (Balaji, 2023).