Technical University of Cluj-Napoca

Programming Techniques

Laboratory – Assignment 2

Queue Simulator

Teacher: prof. Ioan Salomie

Teacher Assistant: Dr. Cristina Pop

Student: Rusu David

Group: E_30424

Contents

1.Assignment Objective

Design and implement a simulation application aiming to analyze queuing-based systems for determining and minimizing clients' waiting time. The application should provide configurable environments in which the number of queues can be modified as well as be able to generate a variety of costumers which are modeled based in two variables, what time they are done shopping and go to a queue and how much time they need to pay. An optional requirement is to have a user interface from which the user can configure the simulation as well as view it in real time.

2.Problem analysis, modeling, scenarios and use cases

I. Problem Analysis

In most real-world applications, it is not financially feasible to offer customers a zero-wait time for a service by having one employee for each customer, thus queues are used. A queue is a line or sequence of people awaiting their turn to be attended to, this way one employee can service multiple customers.

Now that we know we can use queues to become more efficient financially, the question of how many queues do we need arises. We cannot have just one queue because then the customer will have to wait a significant amount of time till he can be serviced, and we cannot have too many queues because then we become inefficient.

In order to solve the problem of how many queues are needed, we will create a program that simulates the queues environment. The dynamic environment is achieved by having and altering the following parameters: number of customers, number of queues, when the customer arrives to a queue ...etc. By using the mentioned parameters, the simulation can help us decide on the number of queues needed so that each customer does not have to wait too much until he is serviced.

II. Modeling

Now that we analyzed the problem, we must model a solution. In this case, it will have to main parts:

a) The user interface
   This will be the part that the user interacts with, so it must be intuitive, good looking and complete. The UI is complete only if the user can perform all the required operations through it. These are:
   - Configure the simulation options
   - Start the simulation
   - View the simulation in real time with all the changes that happen (A customer arrives at a queue)
   - Read all the required statistics gathered from the simulation

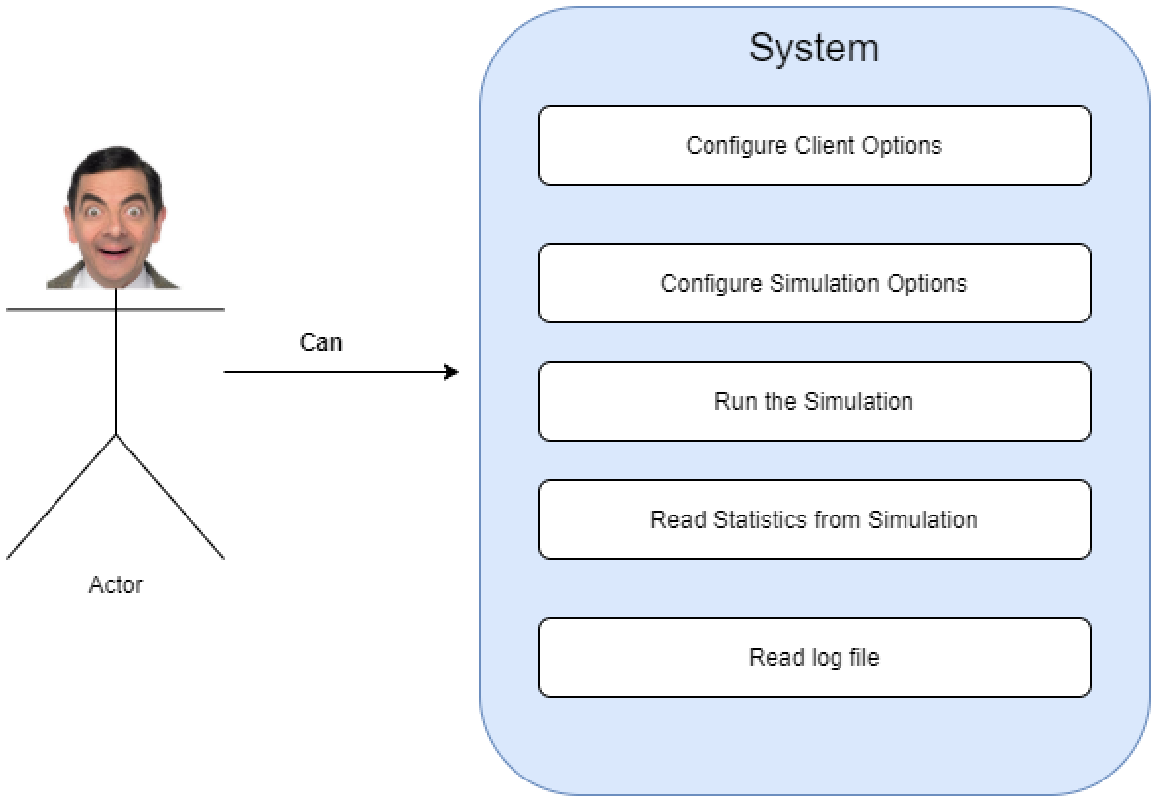
b) The back end
   This part will oversee managing and running the simulation. It will be composed of the following components:

- Simulation manager, this will generate our clients, send data to be displayed on the UI, as well as send clients who want to join a queue to a dispatcher.
- Dispatcher, this will be in charge of sending a client to a queue based on a criteria (For example the queue with the least amount of people waiting).
- Queue, this will hold all the waiting clients that are to be serviced.

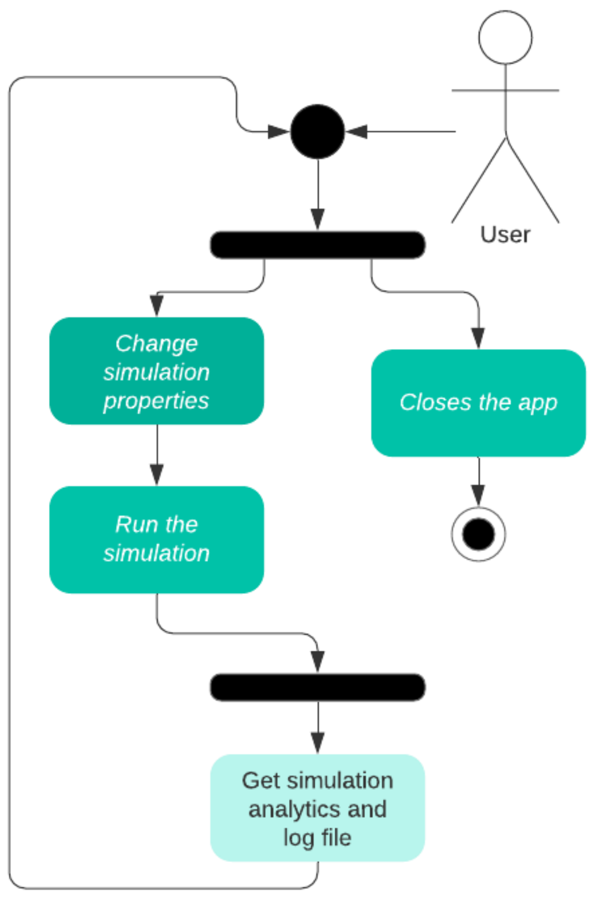
III. Use cases and scenarios

Below we can find the use case and scenarios diagrams.

Use case diagram:



Scenarios Diagram:


User
Change simulation properties
Closes the app
Run the simulation
Get simulation analytics and log file

Following the above diagram, we will describe each step.

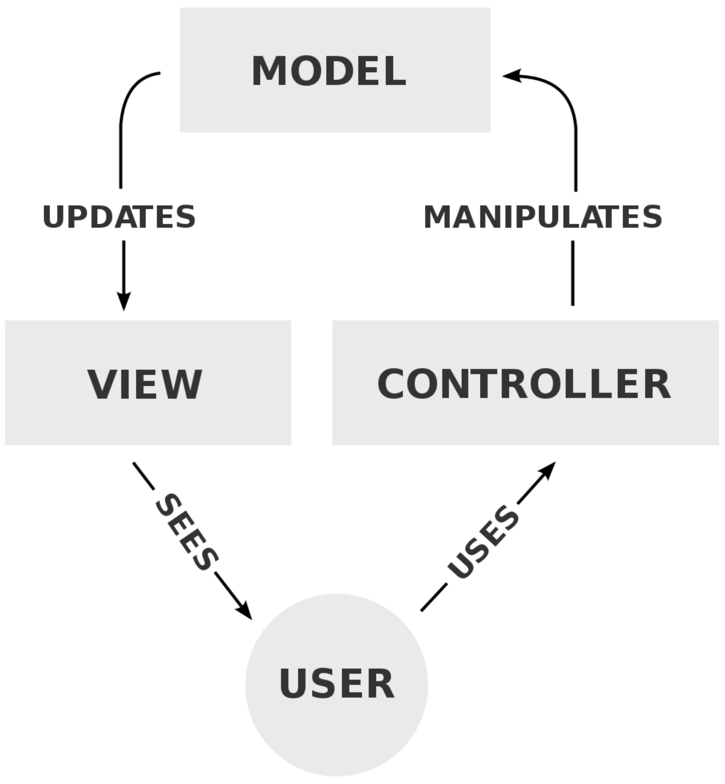
1. Choose whether to close the app or continue using it.
2. If the user does not close the app, he will then have to configure the simulation that will be run.
3. Run the simulation, here the user can see the changes in real time in the UI.
4. Based on the ran simulation, the user will get the calculated analytics as well as a log file in which each step of the simulation is found.

3.Design

I. Class design decisions

One of the most important parts in designing our app is to choose a structure. We have 2 options here, either a custom structure or an already existing one, better known as a design pattern. In this case we will go with a preexisting design pattern called MVC (Model View Controller). The aim of the MVC is to divide the program into 3 interconnected elements.

- **Model:** It is the application's dynamic data structure, independent of the user interface.
- **View:** Renders the user interface. Any representation of information such as a chart, diagram, or table.
- **Controller:** responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it, and then passes the input to the model.



In our case, the model will be made up of the following classes:

- Client

- Settings

The view will be the JavaFX app which is made up of multiple FXML files:

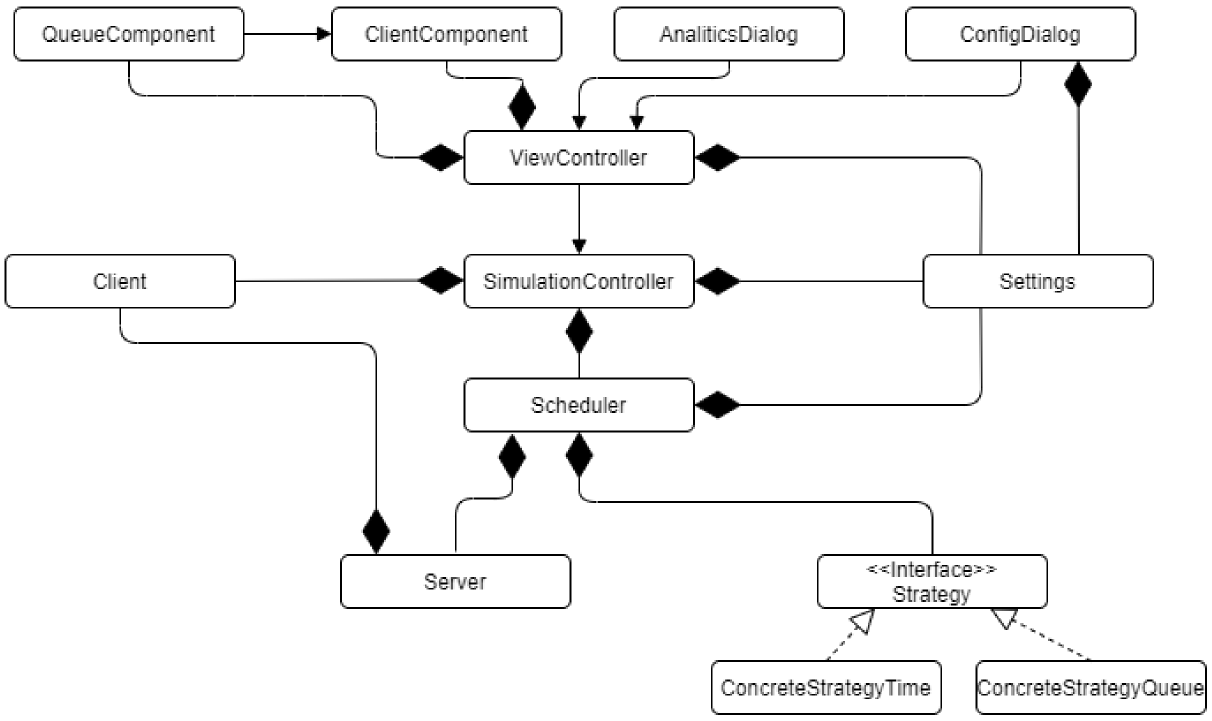
- AnalyticsDialog
- ClientComponent
- ConfigDialog
- QueueComponent
- View

The controller's are found in the controllers package, they manage the UI and its components, program flow and hold the algorithms.

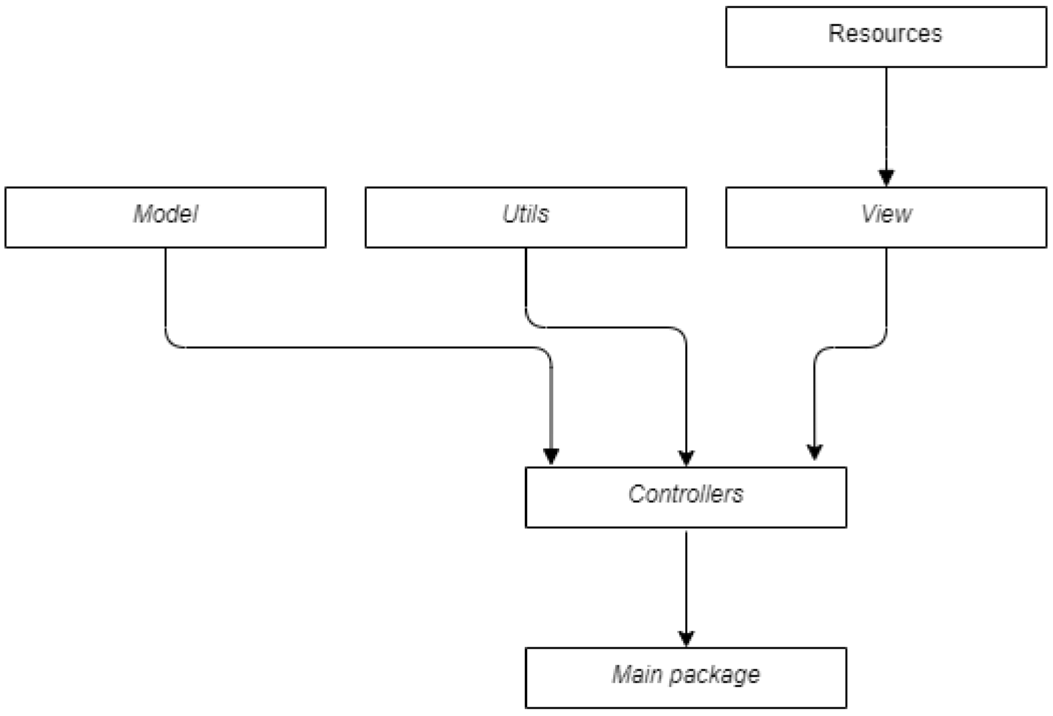
- AnaliticsDialog
- ClientsComponent
- QueueComponent
- ConfigDialog
- SimulationController
- ViewController

II. UML diagrams

Class wise dependency diagram:



Package wise UML diagram:



III. Data structures

All the application specific data structures can be found in the Model package. In that package we have 2 classes that represent a data type:

a) Client, this holds a client with all the necessary information for it to be processed. A client is formed from the following data:
   - clientId
   - arrivalTime
   - serviceTime
   - queue (Used to identify in which queue the client is in)
   - timeSpentInQueue

b) Settings, this is used to hold the simulation configuration. It is formed of:
   - nrClients
   - nrQueues
   - SimArrival
   - minArrivalTime
   - maxArrivalTime
   - minServiceTime
   - maxServiceTime

Besides the above data structures, we also used linked lists and BlockingQueue's were also used.

IV. Interfaces

In the application we have one interface, **Strategy**. Its role is to offer the Scheduler an easy way to switch what algorithm is used when a client is sent to a queue. The interface has one method **addTask**. In total two classes implement this interface:

- ConcreteStrategyQueue
- ConcreteStrategyTime

This interface is used in the Scheduler class.

V. Packages

The application is split into multiple packages in order to obey the MVC design pattern. The application has 6 main packages:

- controllers: Holds the Controller class

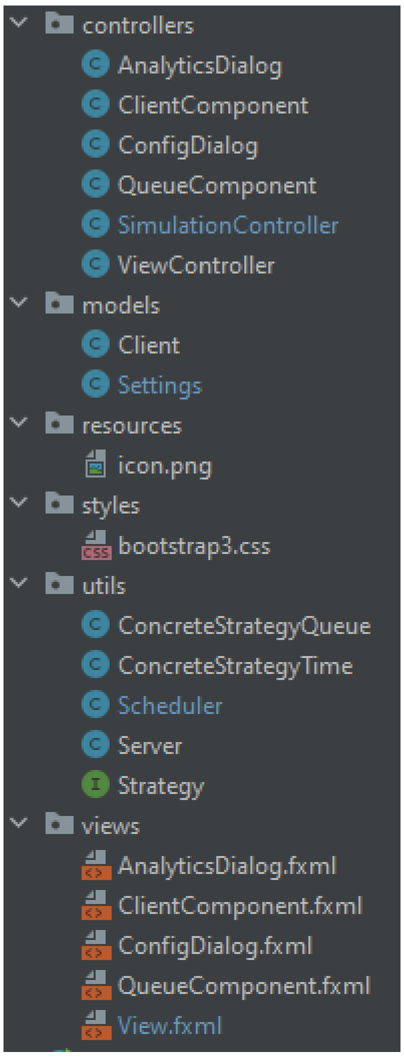
- model: Holds all our application's dynamic data structures

- resources: In this package we find images that are used in the view component

- view: In this package, the user interface is found

- styles: In this package, the css for the interface is found

- utils: Holds helper classes such as: Server, Scheduler, Strategy ...etc



```
controllers
    AnalyticsDialog
    ClientComponent
    ConfigDialog
    QueueComponent
    SimulationController
    ViewController
models
    Client
    Settings
resources
    icon.png
styles
    bootstrap3.css
utils
    ConcreteStrategyQueue
    ConcreteStrategyTime
    Scheduler
    Server
    Strategy
views
    AnalyticsDialog.fxml
    ClientComponent.fxml
    ConfigDialog.fxml
    QueueComponent.fxml
    View.fxml
```

VI. Algorithms

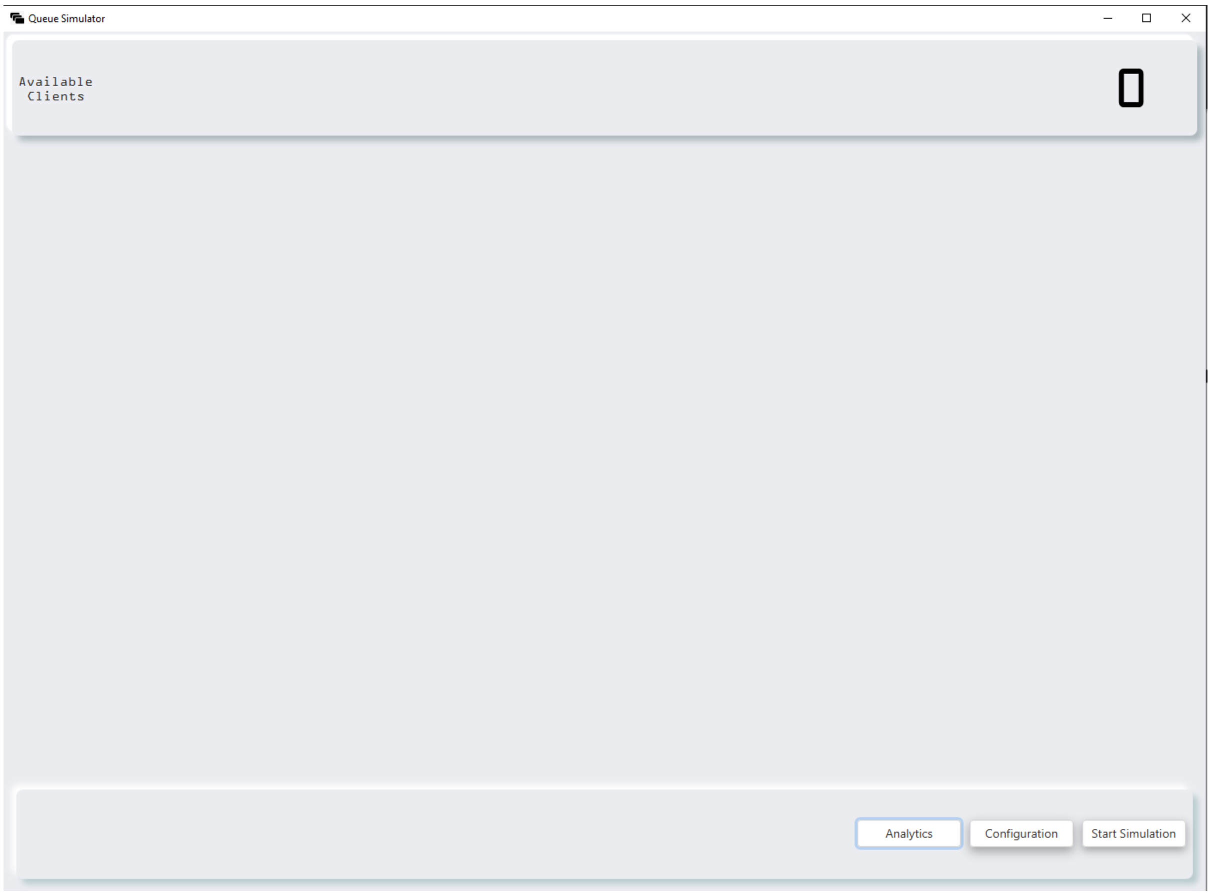
From an algorithm point of view, this app is rather weak. Most of the algorithms are found in the classes that implement the Strategy interface. Both of those classes use an algorithm to select the smallest element from a list.

VII. User Interface

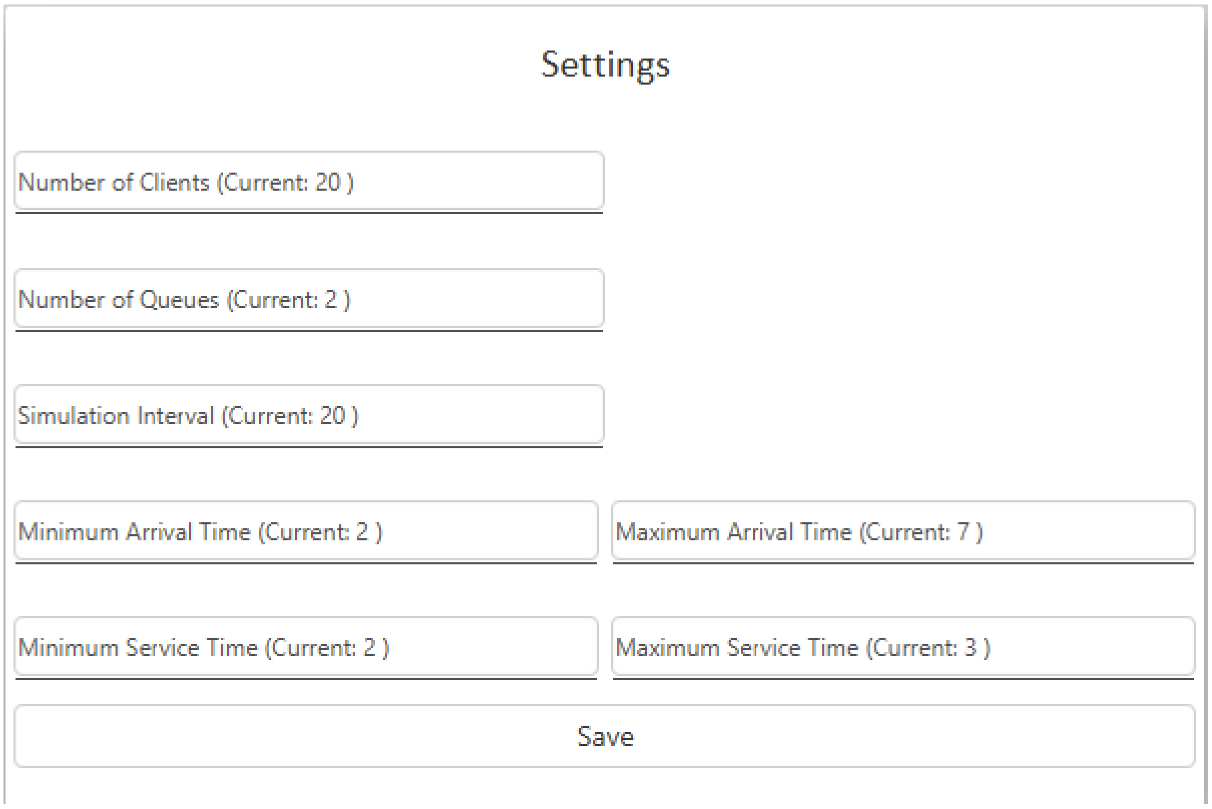
The UI represents a large portion of this app. Its main goal is to offer the user an easier method to interact with the backend. Through the UI the user can change the simulation parameters, run the simulation, and view all the changes that take place in real time.

The UI was implemented in JavaFX, and style with CSS as well as bootstrap for certain buttons. JavaFX was chosen over Swing because it offers a much more modern way of writing it, FXML compared to java objects.

I. When the app is first started, the user is presented with the following screen:

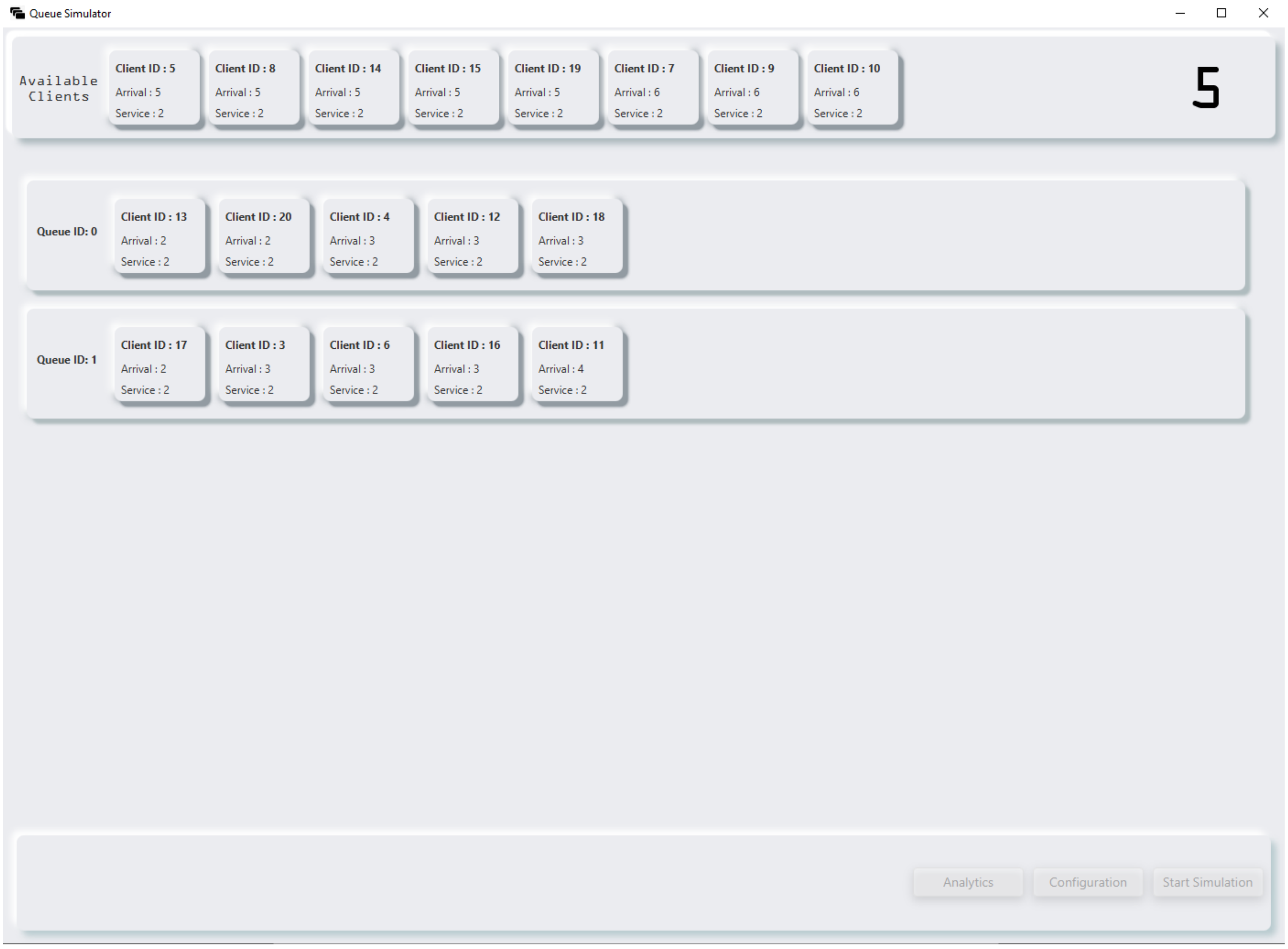


II. After the app is started the user can Configure the simulation by pressing the **Configuration** Button. A dialog will appear in which certain parameters can be altered:


Settings
Number of Clients (Current: 20)
Number of Queues (Current: 2)
Simulation Interval (Current: 20)
Minimum Arrival Time (Current: 2)
Maximum Arrival Time (Current: 7)
Minimum Service Time (Current: 2)
Maximum Service Time (Current: 3)
Save

After the users sets his desired config, he can then press save and the dialog will close. It is not necessary to alter all the settings, if just one option is altered and the save button is pressed, then only the altered option will change, the rest of the settings will still hold the set value.

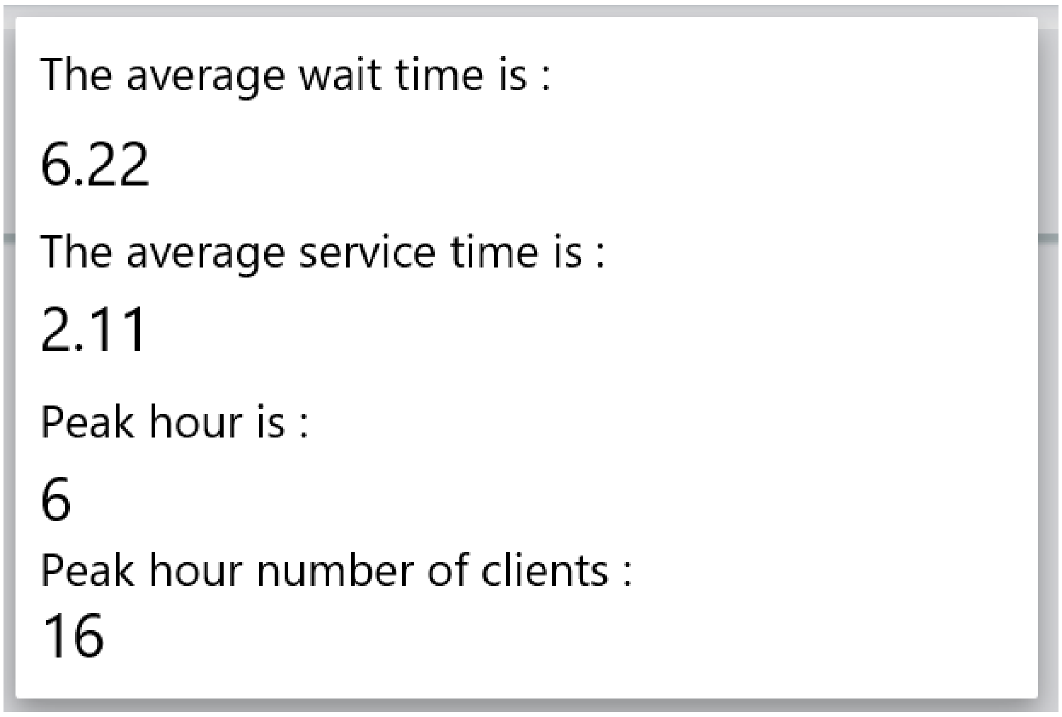
III. When the user is ready, he can then press the **Start Simulation Button**.



Available Clients	Client ID : 5 Arrival : 5 Service : 2	Client ID : 8 Arrival : 5 Service : 2	Client ID : 14 Arrival : 5 Service : 2	Client ID : 15 Arrival : 5 Service : 2	Client ID : 19 Arrival : 5 Service : 2	Client ID : 7 Arrival : 6 Service : 2	Client ID : 9 Arrival : 6 Service : 2	Client ID : 10 Arrival : 6 Service : 2
Queue ID: 0	Client ID : 13 Arrival : 2 Service : 2	Client ID : 20 Arrival : 2 Service : 2	Client ID : 4 Arrival : 3 Service : 2	Client ID : 12 Arrival : 3 Service : 2	Client ID : 18 Arrival : 3 Service : 2			
Queue ID: 1	Client ID : 17 Arrival : 2 Service : 2	Client ID : 3 Arrival : 3 Service : 2	Client ID : 6 Arrival : 3 Service : 2	Client ID : 16 Arrival : 3 Service : 2	Client ID : 11 Arrival : 4 Service : 2			

In the left upper corner, the Simulation time is displayed. To the left of the simulation time, we can find all the generated clients that are still shopping. Once a client is sent to a queue, he will then appear in the queue that he was assigned to. The queues can be seen below the **Available Clients** block. While the simulation is running the 3 buttons (Analytics, Configuration, Start Simulation) are grayed out and become inactive. Once the simulation is over the buttons return to their normal state.

IV. After the simulation ran, the user can access the analytics by pressing the **Analytics button.** The following dialog will appear:


The average wait time is :

6.22

The average service time is :

2.11

Peak hour is :

6

Peak hour number of clients :

16

To close the dialog, the user must left click outside the dialog area.

4. Implementation

The implementation of this project can be split into 2 categories, the frontend, and the backend. We will start with the back end.

a) The backend is composed of the following classes
   a. SimulationController (Runnable)

This class in charge of running the main simulation. This is done in the **run()** method. Once the class starts to run, a log file is created, all the clients are generated, and then the main loop starts. The main loop runs once every second for a period that is set in the simulation options. The loop checks if any clients want to go to a queue, if there are clients then they are dispatched using the schedular object, after that the thread sends to the UI a list which contains all the clients, after the thread writes in the log file all the changes that happened and then goes to sleep for 1000ms. Once the loop is over, the log file is closed and all the created queue threads are closed.

   b. Scheduler

This class generates all the queues and starts them on separate threads (Servers in code). Besides that, it also calls the appropriate strategy to use when a client is dispatched.

   c. Server (Runnable)

This class represents one queue, it is in charge of servicing one client at a time. The main loop is implemented in the **run()** method. The loop checks if there are any clients waiting, if so it takes the first one for servicing. After we increment a variable in all the other waiting clients that keeps track how long they have been waiting for, this is used for analytics. The thread then goes to sleep for

1000ms after which the loop is repeated. Clients can be added to a queue using the **addClient()** method.

d. ConcreteStrategyQueue

This class send a client to the queue that has the least amount of people waiting in line at the queue.

e. ConcreteStrategyTime

This class send a client to the queue that has the least amount of wait time till a client can be serviced.

f. Client

This class was covered in the following section: III. Data structures.

g. Settings

This class was covered in the following section: III. Data structures.

b) The front end is comprised of the following classes and FXML files:

a. ViewController and View.fxml

This class oversees the main view functionality. This view can be seen in the UI section I.

b. ConfigDialog and ConfigDialog.fxml

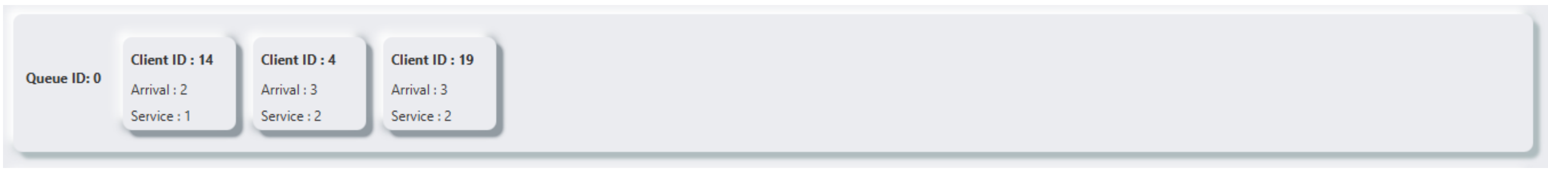
This class oversees the configuration dialog and all its functionality, such as saving any modified settings. This view can be seen in the UI section II.

c. AnalyticsDialog and AnalyticsDialog

This class oversees the analytics dialog and all its functionality, such as setting the calculated average wait time. This view can be seen in the UI section IV.

d. QueueComponent and QueueComponent.fxml

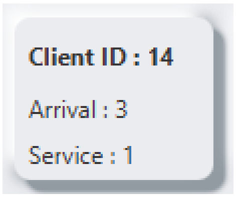
This class controls a single queue component Below we find an image with a single queue component:



As we can see, a queue can have multiple ClientComponents in it. The class is in charge of setting these components to its queue.

e. ClientComponent and ClientComponent.fxml

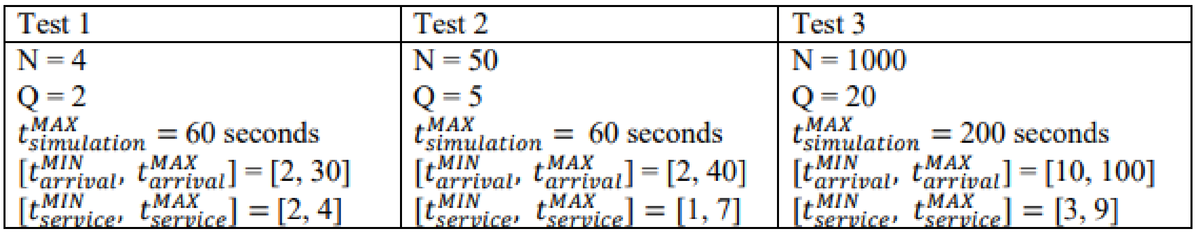
This class controls a single client component. Below we find an image with a single client component:



As we can see, a client component has an ID, Arrival, and service. The class is in charge to set those values.

5. Results

For testing the application, I used the following values:



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

Observation: For test 3, the UI cannot handle that many clients, so for details on the changes that took place, the log file must be consulted.

6. Conclusion

In conclusion this assignment helped me cover multithreading in java and all the safe practices that apply to it, not only in java but to multithreading in general. I was also able to improve on my JavaFX skills, this time by using dialogs and styling the application.

7. Bibliography

The support presentation (was very helpful)

https://stackoverflow.com/questions/13784333/platform-runlater-and-task-in-javafx

https://blog.zelinf.net/posts/java/2017-12-19-javafx-custom-component.html

https://www.youtube.com/watch?v=PLliIaBV2Sc

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html

https://www.baeldung.com/java-thread-stop