*Lior Shtaimberg ,313418469, ליאור שטיימברג, liorst@mta.ac.il*

*Dor Wolfgor ,205892177, דור וולפגור,* *dorwo@mta.ac.il*

Overview

```

The Program, as expected, creates and handles target graphs that a task can run on,   
created by an Admin desktop app and run by Worker desktop app. While having a server  
handling all the targets sending and holding info.

The Engine runs the tasks using multithreading.

```

*UI*

```

GraphFx Module:

A lot of the base FX features are thanks to the library "JavaFXSmartGraph" by brunobrunomnsilva which i "forked" and changed to fit the task in hand

a lot of disecting and adding features later we got a root to leaf top to bottom graph with pressing and changing states capabalities you can use to see graph info

and task info even while running on it.  
added to it is the GraphApp module handling my added features and needs for this project.

```

Admin:

In the admin module the task can be created by going to the graphs tab then choosing the graph in the comboBox after that in the sideController

You can press the run task button and fill up the relevant task details. After choosing the targets by clicking or the targets comboBox,

The task will be uploaded to the server for workers to sign.

Dashboard - responsible for the screen that shows all the graphs uploaded to the page as well as the tasks to be performed.

TaskControllerAdmin - This class is responsible for The second tab in the dashboard, in this screen the admin will be able to work and receive information as part of one task to perform.

From the moment you enter the screen, the admin will be able to run the task.

All information is as in Exercise 2 and in addition While clicking on the target you can see the logs below.

TaskSettings – responsible for getting the task settings from the admin, so that after choosing the targets the task created will be sent to the server

```

```

Worker

The application will open in a login screen where the user will be required to choose a name for himself and provide the number of parallel resources.

After the login screen the user will move to the next screen - the dashboard screen.

file engine:

TaskProcessor - Each worker will connect to the server and request target targets to perform according to his or her registration request for tasks to perform.

The server will reply to the required targets.

In addition, the department makes a request for more tasks and the execution of the targets waiting in queue.

file screens:

Dashboard - This class is responsible for the screen that will display a list of all users in the system: for each user, his name and role will be displayed.

In addition to that, list of to-do tasks defined in the system.

WorkerApp - This class is responsible for the login screen where the user will be required to choose a name for himself and provide the number of parallel resources.

file utils:

Different classes represent different buttons in the system which each have a different role in running the program.

In the worker module the worker can sign to tasks by clicking on the checkbox for signing in or out to the task, and can pause and stop his tasks from the task tab as well.

He will process the targets he ask for by TaskProcessor class that holds all the task processing logic handling everything with concurrent maps holding all the info needed

By taskName and targetName. The runningTasks Map is holding all the tasks the worker is actively asking for targets from, and the signedTasksTargets Map is holding all the

Tasks he is currently signed to and the targets being processed from each one.

```

Both the admin and the worker module depend on the sharedEngine holding all the data types that fill the tables and being sent from the client to the server.

*Engine*

```

Like in ex2 only now it holds the queue and when being asked for is sending the targets to the asker in accordance to his request

1. the graph methods and logic where the directed graph is represented by the `TargetGraph.AdjacentMap` Class that is two Dictioneries of targets names to their childeren in the graph and to their parents

2. The task runner is the task running dedicated class that controls the thread pool using threadExcecutor and a queue that holds the waiting targets and sends to execute when the terms are met.

3. the Tasks are Simulation and Compilation both with giving update on all the task running and gives a status and a result in the end

```

Task running on the graph implementation

In the TargetGraph class we have two TargetGraph.AdjacentMap graphs

\* the Original that was loaded from the xml file

\* the Current graph that we would be running the task on

We start from the RunTask button in the admin module opening a task settings window, and waiting for the corrct task information for the client.

from there in case the task already ran on the graph and the user can run on it from the last point or from scratch like was requested for the project.

Then the user can choose the targets to run on implemented by ChoosingController,

the listener for the chooser is added in the graph panel with onClicked consumer we send it with its initializing

the bottom controller helps you choose all or WhatIf depends or required quickly

after that the task is sent to the server for publish and starting.

In the server engine we create a Queue with the leafs and Independent targets of the relevant graph then when targets are asked by a worker we go through the queue poping targets and checking their status if its waiting or frozen then we check if all their childeren finished successfully and send it to the worker for execution.

While all of this the worker is posting updates and the admin is asking for updates so you can  
see in the admin UI all the targets being processed and the worker going through targets and then asking for more

After a target is finished the worker sends an update that its finished

if it failed we update all of its ancecstors to skipped.

we keep a counter that counts the finished and skipped and update it when neccecery.

when all the targets done or skipped we are done.

Bonuses

1. Postman - Information about the postman will be in the appropriate readme file.

Postman Collection -

1.) login -

this is the path to the login with postman - You need to put in 2 parametrs -

username - name of the user

threads - number of threads between 1 to 5.

2.) Get all users -

this is the path to get all the users - If you send this request you will get the list of all users in the system and their roles.

3.) Get all tasks -

Allow the user to get back the list of all "to-do" tasks.

4.) Sign to task -

Allow the user to register as a worker for a task to be performed, getting back general details about the task.

You need to put taskName - task name.

5.) Get targets -

Allow the user to request from the server specific tasks to perform as part of the task to which he is subscribed

You need to put targets- number of targets you want to get from the server that ready to work on.

6.) Unsign to task -

Allow the user to un-register as a worker for a task to be performed.

You need to put taskName - task name.

7.) Update target when finished -

Allow the user to update the server on a task processing result.

You need to put in 2 parametrs -

targetName - target name.

taskName - task name.