netEval

**Software Design Document**

**Document**

Yehonatan(#204507743)\* and Carmel(#315858340)\*\*

\*[Cyoni10@gmail.com](mailto:Cyoni10@gmail.com) \*\*[caron1211@gmail.com](mailto:caron1211@gmail.com)

**Version: 1 Date: 3.11.2020**

**Contents**

[1. INTRODUCTION 3](#_Toc40953854)

[1.1 Purpose 3](#_Toc40953855)

[1.2 Scope 3](#_Toc40953856)

[1.3 Overview 3](#_Toc40953857)

[2. System Overview 4](#_Toc40953858)

[3. System Architecture 4](#_Toc40953859)

[3.1 Architectural Design 4](#_Toc40953860)

[3.2 Decomposition Description 5](#_Toc40953861)

[4. Data Design 8](#_Toc40953862)

[4.1 Data Description 8](#_Toc40953863)

[4.2 Data Dictionary 10](#_Toc40953864)

[5. Component Design 20](#_Toc40953865)

[6. Human Interface Design 24](#_Toc40953866)

[6.1 Overview of User Interface 24](#_Toc40953867)

[6.2 Screen Images 24](#_Toc40953868)

[6.3 Screen Objects and Actions 28](#_Toc40953869)

# INTRODUCTION

## Purpose

The purpose of the Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to be built.

## Scope

This SDD is for a base level system which will work as a proof of concept for the use of building a system the provides a base level of functionality to show feasibility for large scale production use. This SDD is focused on the base level system and critical parts of the system.

## Overview

This ISDD is divided into nine sections in order to provide a complete and understandable perception about the system to the target readers. First section is mostly about the scope and purpose of the document.

In the second part, system overview, a general description of the software system including its functionality and matters related to the overall system and its design is provided.

The third section states the design considerations and consists of two parts. In the first part, design assumptions, dependencies and constraints of the system are defined. In the second part, design goals and guidelines are given in terms of reliability, usability, portability and extensibility of the system.

The organization of the data structures of the system is explained in section four. Subsequently, a data dictionary is provided in order to provide a detailed description of the system major data, including data objects, their attributes and methods.

Sixth section is all about the user interface design. In this section the functionality and expected features of the user interface is given. In addition, some possible screenshots showing the interface from the user’s perspective are provided and purpose of the screen objects are explained.

# System Overview

This project targets primarily the owner of the network and advertisers who want to invest in the network.

This project (netEval) lets the user to construct a visualization of a network and it lets the user to load a network of various kinds with the option to modify it. The program will evaluate how much the network is worth and will present it to the user.

The program can compute all the basic network properties, such as graph diameter, radius, distance between two vertices, shortest path, eccentricity of a vertex, the total number of members, groups, traffic in the network, and how long each user spends in the network every day.

Furthermore, the program lets the user to present the network graphically. The user can filter data as he would like

# System Architecture

## Architectural Design

System will consist of two major components. The first part will deal with the processing of data on the network and the calculation of this data.

The second part will deal with turning all this data into a visual graph with arcs and vertices. Each of these two sections will have classes, objects, and methods that will be detailed later in this document.

In addition to these two parts, the system will also contain a convenient and user-friendly interface that will include logging in and registering for the software as well as displaying the user's network data on the timeline.

All these possibilities are shown in the following diagram (Figure 1)

**A close up of text on a white background

Description automatically generated**

**Figure 1. Use case:** All options the user has in the system

## Decomposition Description

As mentioned before, the first part of the system handles the absorption of data and processing.

This part of the system is divided into a subsystem that captures and processes the data and another subsystem which is responsible for calculating and analyzing the data.

At the user level the system will allow it to load the data file, after the system has processed the data they will be shown on the screen: the number of members on the network, the average number of responses per day, the average time on the network, the number of advertisers, etc.

This part of the system consists of many departments that create the network department, and departments that use this class to perform the calculations on the network.

Details of these classes will be found later in the document and its structure can be seen in the following diagrams ( Figure 2 and Figure 3 )

**A screenshot of a cell phone

Description automatically generated**

**Figure 2. Class diagram for Network**

**A picture containing screenshot

Description automatically generated**

**Figure 3. Class diagram for Person**

The second system deals with the presentation of the network data in the form of a graph with vertices and edges.

This system is also subdivided into subsystems:

One subsystem deal with presenting the graph to the user in a friendly way

And another subsystem is responsible for calculating the graph such as the diameter of the graph, finding the shortest track, etc.

There are also different departments that make up the system as seen in the following diagram ( Figure 4)

A screenshot of a cell phone

Description automatically generated

**Figure 4. Class diagram for Graph**

# Data Design

## Data Description

The main data that the program will deal with is a database structure that will be stored in a Personal Storage Table file (.pst). The user will be able to import or to retain such a database. The program will process such a file by reading it, collecting relevant data and finally present the data to the user.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **field** | **type** | **null** | **default** |
| **person** |  |  |  |  |
|  | person\_id | int(9) | yes | null |
|  | person\_name | varchar(50) | yes | null |
|  | age |  | yes | null |
|  | city | varchar(30) | yes | null |
|  | role | enum ('member', 'owners', 'advertisers ', ) | yes | null |
| **member** |  |  |  |  |
|  | person\_id | int(9) | yes | null |
|  | register\_date | timestamp | yes | null |
|  | time\_spent | int(9) | yes | null |
| **advertisers** |  |  | yes | null |
|  | person\_id | int(9) | yes | null |
|  | company\_name | varchar(50) | yes | null |
| **owner** |  |  | yes | null |
|  | person\_id | int(9) | yes | null |
|  | assets | double | yes | null |
| **Employees** |  |  |  |  |
|  | person\_id | int(9) | yes | null |
|  | salary | int | yes | null |
| **advertisement** |  |  |  | null |
|  | advertisement\_id | int(9) | yes | null |
|  | price | int(9) | yes | null |
|  | start\_date | timestamp | yes | null |
|  | end\_date | timestamp | yes | null |
|  | views | int(9) | yes | null |
| **Product** |  |  |  |  |
|  | product\_id | int(9) | yes | null |
|  | product\_name | varchar(50) | yes | null |
|  | advertisers\_id | int(9) | yes | null |
|  | value | int(9) | yes | null |
| **post** |  |  |  |  |
|  | post\_id | int(9) | yes | null |
|  | member\_id | int(9) | yes | null |
|  | creation\_date | timestamp | yes | null |
|  | data | varchar(800) | yes | null |
|  | shares | int(9) | yes | null |
|  | likes | int(9) | yes | null |
|  | views | int(9) | yes | null |
| **like** |  |  |  |  |
|  | like\_id | int(9) | yes | null |
|  | member\_id | int(9) | yes | null |
|  | postable\_id | int(9) | yes | null |
| **share** |  |  |  |  |
|  | share\_id | int(9) | yes | null |
|  | member\_id | int(9) | yes | null |
|  | postable\_id | int(9) | yes | null |
| **view** |  |  |  |  |
|  | view\_id | int(9) | yes | null |
|  | member\_id | int(9) | yes | null |
|  | postable\_id | int(9) | yes | null |

## Data Dictionary

In this section we specified all the functions we intend to impelement.

|  |  |
| --- | --- |
| Class name: Gui\_graph\_chart | |
| Description: This gui allows the user to see the value of the network in terms of time | |
| **Description(Optional)** | **Attributes (fields)** |
|  | Database connection |
|  | **Methods (operations)** |
| Get data from the database | fetchData () |
| Draw the chart | drawChart() |

|  |  |
| --- | --- |
| Class name: Gui\_graph\_visualization | |
| Description: This gui lets you enter to your account. | |
| **Description(Optional)** | **Attributes (fields)** |
|  |  |
|  |  |
|  | **Methods (operations)** |
| Draw the graph | drawGraph() |
| Filter graph by subjects | filterGraph() |
|  | calculateShortestDistance() |
| Mark a path between two vertices | markPath(Vertex vertex1, Vertex vertex2) |

|  |  |
| --- | --- |
| Class name: Gui\_network | |
| Description: This gui allows the user to see general info about the network. | |
| **Description(Optional)** | **Attributes (fields)** |
|  | Database connection |
|  | **Methods (operations)** |
| Load a database file | loadFile() |
|  | saveResults() |
|  | openGraphVisualization() |
|  | openNetworkChart() |

|  |  |
| --- | --- |
| Class name: Gui\_login | |
| Description: This gui lets you enter to your account. | |
| **Description(Optional)** | **Attributes (fields)** |
|  | Database connection |
|  | **Methods (operations)** |
| Connect to the network | connect(int networkId, String email, String password) |
| Open registration window | register() |

|  |  |
| --- | --- |
| Class name: Gui\_signup | |
| Description: This gui lets you register to the software | |
| **Description(Optional)** | **Attributes (fields)** |
|  | Database connection |
|  | **Methods (operations)** |
| Register a new owner | register() |
| Check validation | validMail() |

|  |  |
| --- | --- |
| Class name: gui\_manageUsers | |
| Description: This class allows the owner to add/remove users | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int id |
| Object that represents a person | Person person |
|  | **Methods (operations)** |
| Add a permission to someone to use someone’s network | addAccount(String email, String password) |
| Revoke permission | RemoveAccount(string email) |

|  |  |
| --- | --- |
| Class name: owner | |
| Description: This class represents the owner. | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int id |
| Object that represents a person | Person person |
|  | Int assets |
|  | **Methods (operations)** |
|  | getAssert() |

|  |  |
| --- | --- |
| Class name: advertiser | |
| Description: This class represents an advertiser | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int id |
| Object that represents a person | Person person |
|  | List < advertisement> advertisements |
|  | **Methods (operations)** |
|  | Get advertisements () |

|  |  |
| --- | --- |
| Class name: member | |
| Description: This class represents a member | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int id |
| Object that represents a person | Person person |
| timespan | register\_date |
|  | time\_spent |
|  | **Methods (operations)** |
|  | getTime\_spent() |

|  |  |
| --- | --- |
| Class name: post | |
| Description: This class represents a post. | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int post\_id |
| Object that represents a person | Int member\_id |
| When the post was created | creation\_Date |
| Data of post | String data |
| List of shares | List<Share> shares |
| List of likes | List <like> likes |
| List of views | List <view> views |
|  | **Methods (operations)** |

|  |  |
| --- | --- |
| Class name: page | |
| Description: This class represents a page in the network. | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int id |
| Object that represents a person who is the owner of the page. | Person person |
| The members who are affiliated to the page. | List<Person> members |
|  | **Methods (operations)** |
|  |  |

|  |  |
| --- | --- |
| Class name: group | |
| Description: This class represents a group in the network. | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int id |
| Object that represents a person who is the owner of the group. | Person person |
| The members who are affiliated to the group. | List<Person> members |
|  | **Methods (operations)** |
|  |  |

|  |  |
| --- | --- |
| Class name: Advertisement | |
| Description: This class represents a single ad | |
| **Description(Optional)** | **Attributes (fields)** |
| #ID | Int id |
| Object that represents a person who is the owner of the ad. | Person person |
|  | **Methods (operations)** |
|  |  |

|  |  |
| --- | --- |
| Class name: ReadDBFile | |
| Description: read the database and create a network | |
| **Description(Optional)** | **Attributes (fields)** |
| This object represents the entire network | Network network |
|  |  |
|  | **Methods (operations)** |
| Goes through the table that has all the posts and adds them to the network | readPostTable |
| Goes through the table that has all the members and adds them to the network | readMemberTable |
|  |  |

|  |  |
| --- | --- |
| Class name: network | |
| Description: This class represents the entire network. | |
| **Description(Optional)** | **Attributes (fields)** |
| Name of the network | String name |
| List of the members | List<member> members |
| List of the pages | List <page> pages |
| List of the groups | List <group> groups |
| List of the posts | List <post> posts |
| List of the advertisements | List < Advertisement > Advertisements |
|  | **Methods (operations)** |
| Insert a list of Page to the network | insertPages (List <page> pages ) |
| Insert a list of Member to the network | insertMember (List <member> members) |

|  |  |
| --- | --- |
| Class name: NetworkCalculations | |
| Description: This class evaluates the network with some calcualtions | |
| **Description(Optional)** | **Attributes (fields)** |
| Object that represents the network | Network network |
|  |  |
|  | **Methods (operations)** |
| How many members in the network | calculateNumMembers() |
| How long each user spends per day in the network | calculateAvgTime() |
| How many likes | calculateNumLikes() |
| How many shares | calculateNumShares() |
| Calculates the amount of expenses that the owner must spend | calculateExpenses() |
| Calculates the profit of the network | calculate Revenue () |
| Evaluates the value of the network | evaluateNetwork() |

|  |  |
| --- | --- |
| Class name: networkToGraph | |
| Description: Convert the network to a graph | |
| **Description(Optional)** | **Attributes (fields)** |
|  | Network network |
|  |  |
|  | **Methods (operations)** |
|  | convert |

|  |  |
| --- | --- |
| Class name: Graph | |
| Description: This class represents a graph. | |
| **Description(Optional)** | **Attributes (fields)** |
|  | List < Vertex> vertices |
|  | List <edge> edges |
|  |  |
|  | **Methods (operations)** |
|  | calculateRadius() |
|  | calculateDiameter() |
|  | calculateShortestDistanceBetween(Vertex vertex1, Vertex vertex2) |
|  | setVertical(double x, double y, double p, double q) |
|  | setEdge(Vertex vertex1, Vertex vertex2) |
| Degree between two vertices | numDegree(Vertex vertex) |
| Is there a path between two vertices | isConnected(Vertex vertex1, Vertex vertex2) |

|  |  |
| --- | --- |
| Class name: Vertex | |
| Description: This class represents a vertex | |
| **Description(Optional)** | **Attributes (fields)** |
|  | Double x |
|  | Double y |
|  | **Methods (operations)** |
|  | setVertex(double x, double y) |
|  | getVertex |

|  |  |
| --- | --- |
| Class name: Edge | |
| Description: This class represents an edge | |
| **Description(Optional)** | **Attributes (fields)** |
|  | Vertex v1 |
|  | Vertex v2 |
|  |  |
|  | **Methods (operations)** |
|  | setEdge() |
|  | getEdge() |

# Component Design

1.Find the shortest route in the graph

|  |  |
| --- | --- |
| 1: | **function** Dijkstra(Graph, source): |
| 2: | **for each** vertex v in Graph: | // Initialization |
| 3: | dist[v] := infinity | // initial distance from source to vertex v is set to infinite |
| 4: | previous[v] := undefined | // Previous node in optimal path from source |
| 5: | dist[source] := 0 | // Distance from source to source |
| 6: | Q := the set of all nodes in Graph | // all nodes in the graph are unoptimized - thus are in Q |
| 7: | **while** Q **is not** empty: | // main loop |
| 8: | u := node in Q with smallest dist[ ] |  |
| 9: | remove u from Q |  |
| 10: | **for each** neighbor v of u: | // where v has not yet been removed from Q. |
| 11: | alt := dist[u] + dist\_between(u, v) |  |
| 12: | **if** alt < dist[v] | // Relax (u,v) |
| 13: | dist[v] := alt |  |
| 14: | previous[v] := u |  |
| 15: | **return** previous[ ] |  |

2.find the diameter of a graph

After run Floyd-Warshall's all pair shortest path algorithm then traversing the whole distance matrix, and find diameter of graph

**let** dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)

**for each** edge (*u*, *v*) **do**

dist[*u*][*v*] ← w(*u*, *v*) *// The weight of the edge (*u*,* v*)*

**for each** vertex *v* **do**

dist[*v*][*v*] ← 0

**for** *k* **from** 1 **to** |V|

**for** *i* **from** 1 **to** |V|

**for** *j* **from** 1 **to** |V|

**if** dist[*i*][*j*] > dist[*i*][*k*] + dist[*k*][*j*]

dist[*i*][*j*] ← dist[*i*][*k*] + dist[*k*][*j*]

**end if**

3.Chart Line Graph

DefaultCategoryDataset dataset = new DefaultCategoryDataset( );

Connect to the database of your network.

List data\_list = Fetch your network data

For (item : data\_list)

dataset.addValue( item.value , "value" , data\_list.month );

return dataset;

4.Evaluate Network:

Class cat\_item{

double p ;  
int counter // how many ads under the same cat there are

Public x(int p){this.p = p; }

Public getCounter(){return counter}  
public getP(){return p}

Public increaseCounter(){counter++}

}

double result = $0.0;

double worth\_like = $0.2   
double worth\_share = $0.5  
double worth\_post = $0.2  
double worth\_advertiser = $1  
double worth\_time\_spent = $2 *// the amount of money per 1h of use*

list members = getMembers()

list advertisers = getAdvetisers()

list ads = getAds()

list category = getCat()// *category of Advetisers*

*// Each advertiser is worth $1 (Not necessarily active advertisers)*

result = advertisers.size() \* worth\_advertiser

*// An active ad adds 50% of the expense to the evaluation otherwise it adds 20%*

Size\_ads = ads.size // size of ads in the net

cat\_item Have\_interest\_in\_product[cat\_size]

for (ad : ads)

if (ad.active()) result+= ad.adveriser.getExpense() \* 0.5

else result+= ad.adveriser.getExpense() \* 0.2 *// not an active ad*

cat\_item current = Have\_interest\_in\_product[ad.getCat()];

current.setP(cat\_item.getP + ( ad.getInterest() / members.size()))

*// Updating the probability like this: Some people were interested in the specific advertisement divided by the number of members of the network plus the old probability that was there*

current.increaseCounter () *// Raise the meter by 1. The meter says how many advertisements from the same category there are*

}

Double size\_members\_to\_see[Have\_interest\_in\_product.size] *// Some people will be interested in the advertisement*

For (I to Have\_interest\_in\_product.size()){

cat\_item Current Have\_interest\_in\_product[i]

size\_members\_to\_see [i] = current.getP()/current.getCounter \* members.size()

*// The likelihood that a user will be interested in an ad of that category is the probability that we now calculated the average: divided by the number of people who viewed past ads twice the number of network members*

}

Double profit\_from\_ad[cat.size()] *// Profit from every advertisement*

For (i=0 to cat.size){

Double random = rand [0,1) *// Grate probability*

Int expected = size\_members\_to\_see[i*] // The number of people who will see the advertisement*

double told\_to\_others = expected\* rando *// The chance of something telling a friend about the advertisement*

Expected+= told\_to\_others

random = math.random [0,1)

*// how many people are expected to buy the product*

Double size\_members\_to\_buy = expected \* random *// how many will buy*

profit\_from\_ad[i] = size\_members\_to\_buy

result+= profit\_from\_ad[i]

}

for (member : members) {

list member\_likes = getLikes(member)   
list member\_shares = getShares(member)  
list member\_posts = getPosts(member)

result+= member.timeSpent() \* worth\_time\_spent // an average time of use every month

}

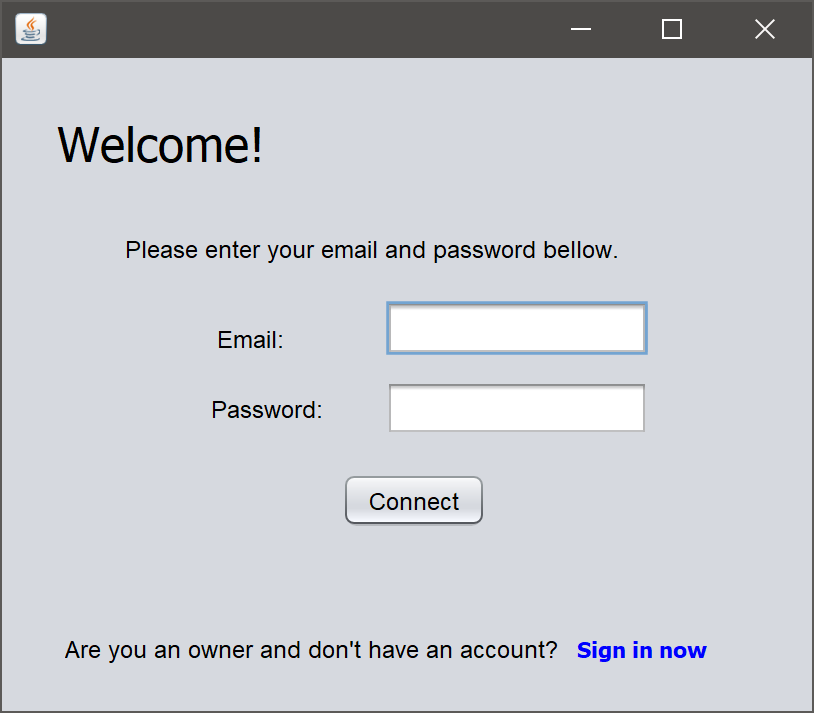
**return result;**

# Human Interface Design

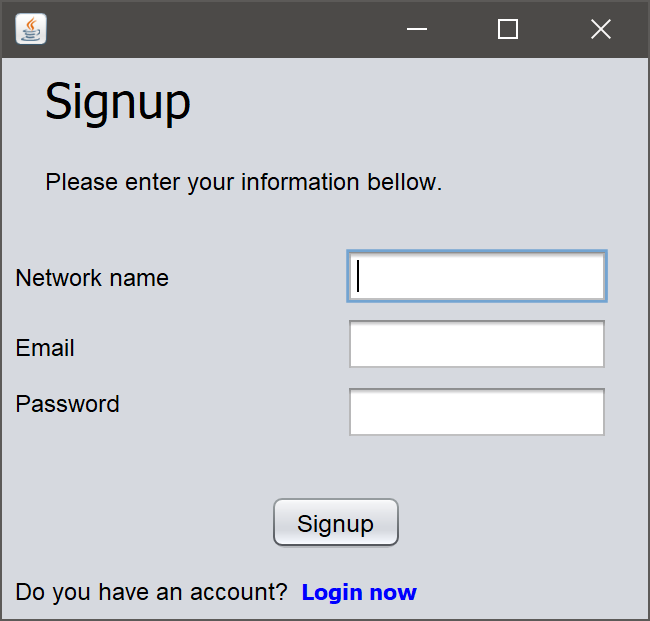
* 1. Overview of User Interface   
       
     First of all the user enters the software, enters their email and password and verifies that a verifier has permission to use the software (Figure 5). Once the user has logged in, it loads a .pst file into the software. After the software has read the file, the user will see the network data on the screen (Figure 8).

From this point the user will be able to see the graph of the network visually and also be able to filter results (Figure 9). The user will be able to run graph algorithms on the graph.  
In addition, the user will be able to see statistic data on the network using various diagrams (Figure 10).

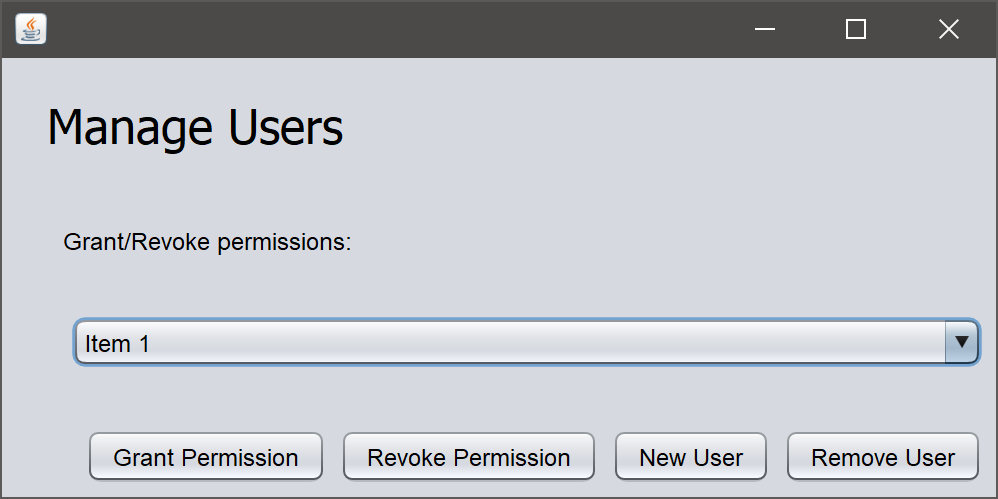
## Screen Images



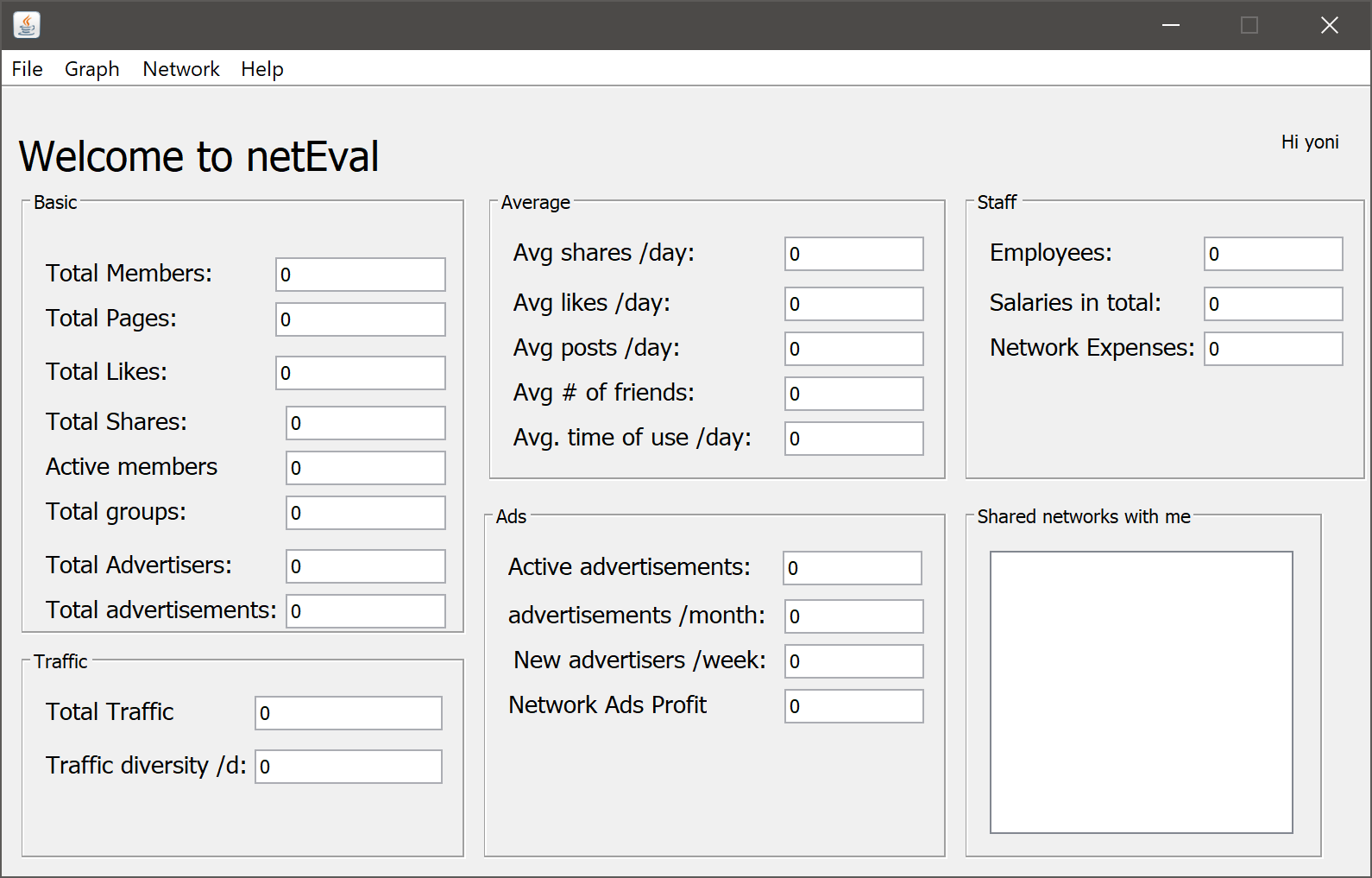
**Figure 5. Gui\_login**: The user is login by email and password



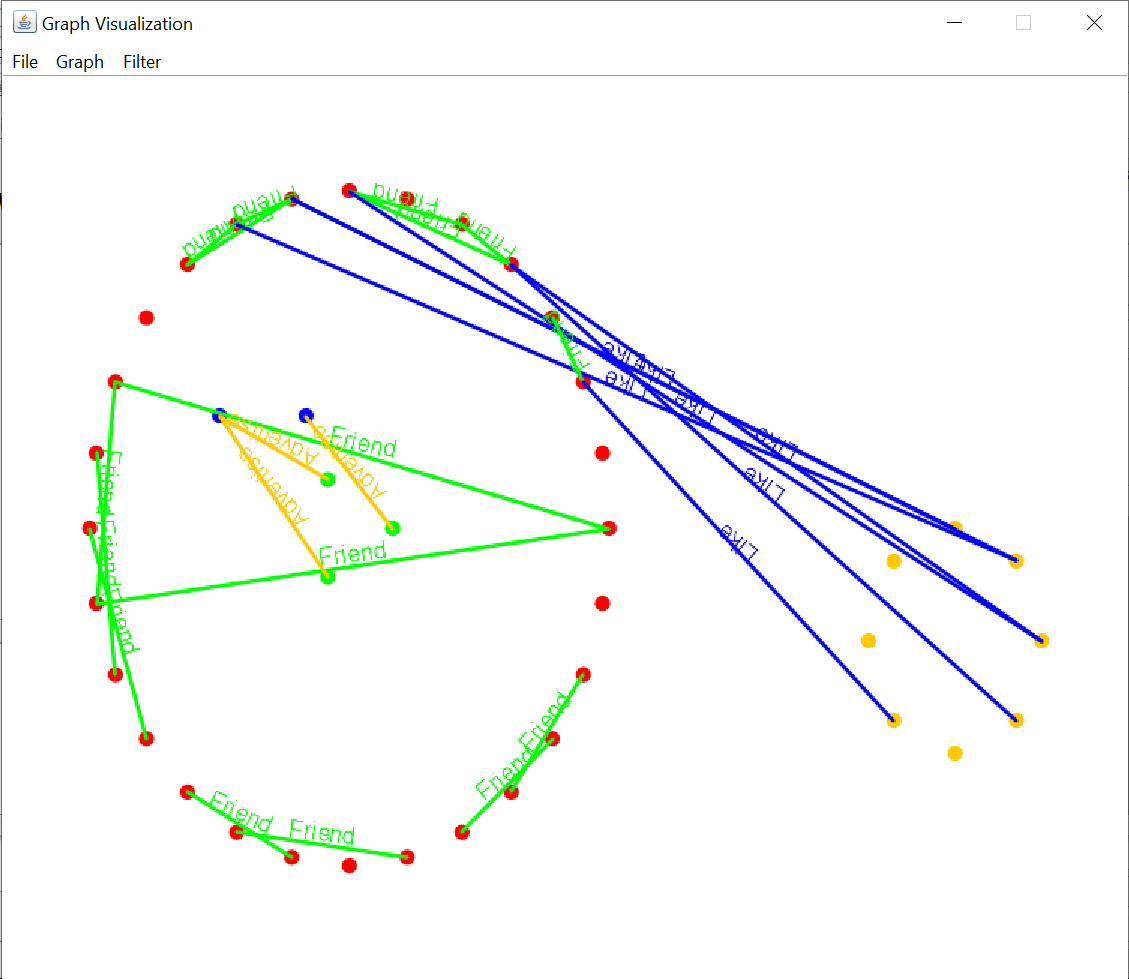
**Figure 6. Gui\_signup**: The user is registered by email and password



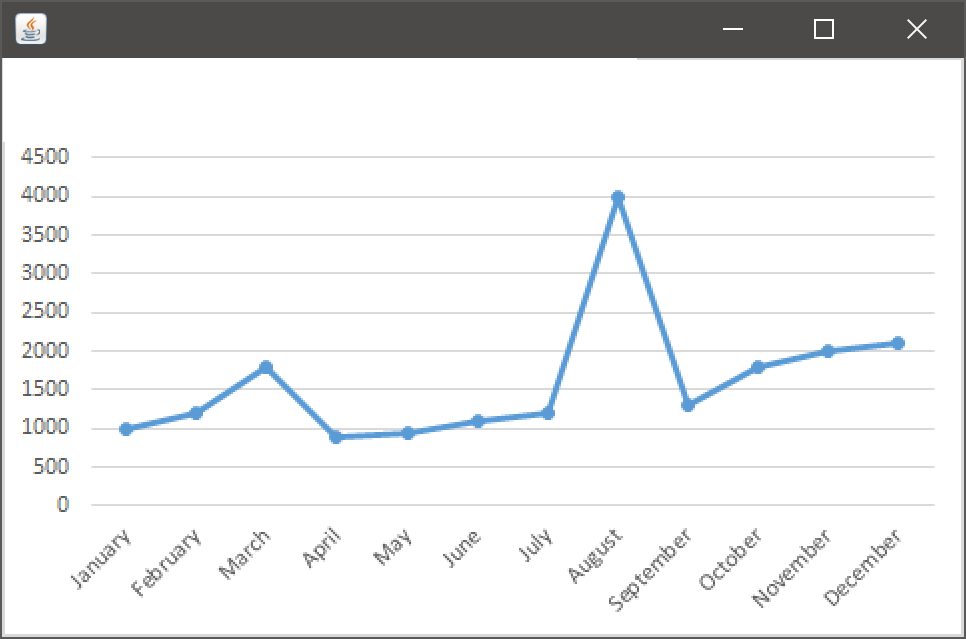
**Figure 7.Gui\_manage\_Users:** add and remove pernission



**Figure 8. Gui\_graph\_visualization:** All fields of the network and advertisers after the user uploads a file



**Figure 9. Gui\_graph\_visualization**: The graph is divided into colors by the different groups: users, posts, etc. The vertices are connected by edges for example: friend, like



**Figure 10. Gui\_Graph\_Chart:** the value of the network by month

## Screen Objects and Actions

**1. Login screen**Email: The email is an identifier of the user.   
Password: Password can be ranged from 4 to 20 letters.

**2. Signup screen**

Network name: the owner of the network will enter his network name there.

Email: with that, the user will be able to connect to the software.

Password: Password can be ranged from 4 to 20 letters.

**3. Network screen**

Members: the number of members.

Active members: number of active members in the network.

Pages: the number of pages in the network.

Likes: number of likes.

Shares: number of shares.

Advertisers: number of active advetisers in the network.

#Ads: number of active ads

Profit by ads: the profit of the ads.

Profits: profits in general.

Employees: the number of employees.

Expenses: the amount of expenses per month

**4. Visualization screen**1. Refresh: refresh the data and the graph.

2. Evaluate Network: it gives an evaluate value of the network

3. Save picture: picture of the graph.

4. Shortest Path: returns the shortest path in the graph.

5. Longest Path: returns the longest path in the graph.

6. Reliability: returns how much the network is reliable, depens of the levels.

**5. Manage-users screen**  
1. Grant permission: the owner grants a permission to a user in the list.

2. Revoke permission: the owner revokes the permission from the user.

3. New User: to add a new user to the list.

4. Remove User: to remove the user from the list.

5. Items: list of users.