**System Design Document**

**For**

**COVID-19 Tracking App**

**Team Name: COVID Tracking**

**Team Members:**

**Justin Andrews**

**Bryce Cole**

**Victoria Jordan**

**Jacob Preseau**

**Chandra Teja Tiriveedhi**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Authors** |
| **V1** | **2/18/2021** | **Justin Andrews, Bryce Cole, Victoria Jordan, Jacob Preseau, Chandra Teja Tiriveedhi** |
|  |  |  |
|  |  |  |
|  |  |  |

# **Table of Contents**

[**Table of Contents** 2](#_Toc64581542)

[**1 Introduction** 3](#_Toc64581543)

[**1.1 Purpose and Scope** 3](#_Toc64581544)

[**1.2 Project Executive Summary** 3](#_Toc64581545)

[**1.2.1 System Overview** 3](#_Toc64581546)

[**1.2.2 Design Constraints** 5](#_Toc64581547)

[**1.2.3 Future Contingencies** 5](#_Toc64581548)

[**1.3 Document Organization** 5](#_Toc64581549)

[**1.4 Glossary** 5](#_Toc64581550)

[**2 System Architecture** 5](#_Toc64581551)

[**2.1 System Hardware Architecture** 5](#_Toc64581552)

[**2.2 System Software Architecture** 6](#_Toc64581553)

[**2.3 Internal Communications Architecture** 6](#_Toc64581554)

[**3 Human-Machine Interface** 6](#_Toc64581555)

[**3.1 Inputs** 6](#_Toc64581556)

[**3.2 Outputs** 7](#_Toc64581557)

[**4 Detailed Design** 8](#_Toc64581558)

[**4.1 Hardware Detailed Design** 8](#_Toc64581559)

[**4.2 Software Detailed Design** 8](#_Toc64581560)

[**4.3 Internal Communications Detailed Design** 15](#_Toc64581561)

[**Required** 15](#_Toc64581562)

[**Optional** 15](#_Toc64581563)

[**5 External Interfaces** 16](#_Toc64581564)

[**5.1 Interface Architecture** 16](#_Toc64581565)

[**5.2 Interface Detailed Design** 16](#_Toc64581566)

[**6 System Integrity Controls** 16](#_Toc64581567)

# **1 Introduction**

## **1.1 Purpose and Scope**

This document will describe the system overview, system design, inputs, outputs, and interface of the COVID-19 Tracking App project.

## **1.2 Project Executive Summary**

Section 1.2 gives a high-level overview and introduction to the system and its constraints.

### **1.2.1 System Overview**

Figure 1 is a use case diagram that gives an overview of how the user and the software interact.

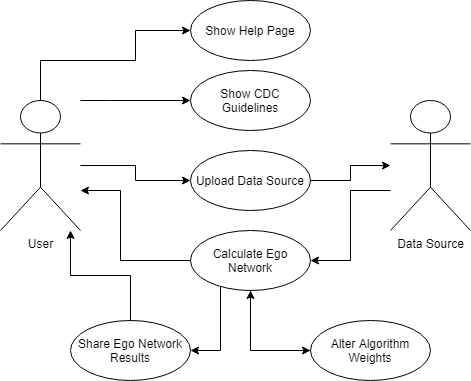


Figure 1: Use Case Diagram for COVID-19 Tracking App.

Figure 2 gives a high-level overview of the system process that calculates the ego network of the user. This figure includes inputs, like the user uploading their communication data, and outputs, such as displaying the user’s ego network that has been calculated.

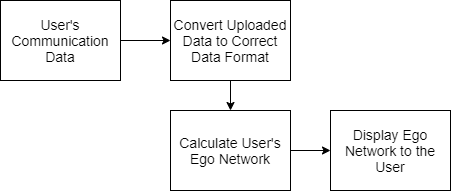


Figure 2: Overview of the process of the system

### 1.2.2 **Design Constraints**

The system is limited most by the data received in the social media export. Because the data only pertains to the user it puts constraints on how we can utilize location data. Another constraint is how we access said user data. In a perfect world we could hook up to an API and download everything that way. Because of the API’s limited feature set we instead have to download the data onto the phones file system. This reduces the ease of use of the product.

### **1.2.3 Future Contingencies**

There are currently no future contingencies due to the lack of interfaces that are needed to communicate to provide issues with compatibility.

## **1.3 Document Organization**

* System Requirements Specification Document (SRS)
* System Design Document (SDS)
* Test Plan

## **1.4 Glossary**

* Closeness - This term refers to the closeness score that the ego network algorithm calculates that defines how close it perceives that specific person to be to the user. This calculation is based on the quantity and frequency of interactions between the person and the user. The higher the closeness number, the closer the algorithm perceives these two individuals to be.
* Friendship Level - This term refers to the different groups that the ego network groups people into (Serious Friends, Good Friends, Friends, Distant Friends). The level that a person is placed into by the algorithm is based on their closeness score.
* Contact free Tracing- The term contact tracing is used to describe the process of identifying potential people who have been exposed to the virus due to a close contact with another person.
* Entry - A piece of data that impacts a friendship level. See Instagram DM, received Snapchat message, etc.

# **2 System Architecture**

## **2.1 System Hardware Architecture**

This system is purely software, so there is no system hardware architecture needed for this project.

## **2.2 System Software Architecture**

The app has three major components: accepting the user's uploaded data, calculating the user's ego network, and displaying the ego network and its details to the user. The user uploads their communication data to the app which the app then cleans so that the data is in the format that the ego network algorithm can work with. The app can then use this data to calculate their ego network. Finally, the app displays the ego network to the user. The user can then select different levels of their ego network to get more information about that level, such as who of their friends falls within that level. This is shown below in Figure 3.

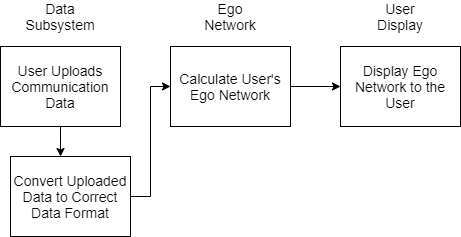


Figure 3: System Architecture Overview

## **2.3 Internal Communications Architecture**

# **3 Human-Machine Interface**

## **3.1 Inputs**

First, the user must go to their social media accounts to download their communication data. Once this has been done, unzip the folder you receive and upload it to your mobile device where the app is installed. To then link their data to the app first the user must go to the Upload Data tab of the app and then select the Upload Instagram or Upload Snapchat button. After selecting the button, they must use the device’s file explorer to navigate to where their data is stored and then select the folder they wish to link to the app. Once linked the app will remember the file path to the data and will call that path anytime it needs to find the data again. After the data has been linked, when the user selects the button to calculate their ego network the linked JSON files will automatically be parsed to the apps required format. This process is shown below in Figure 4.

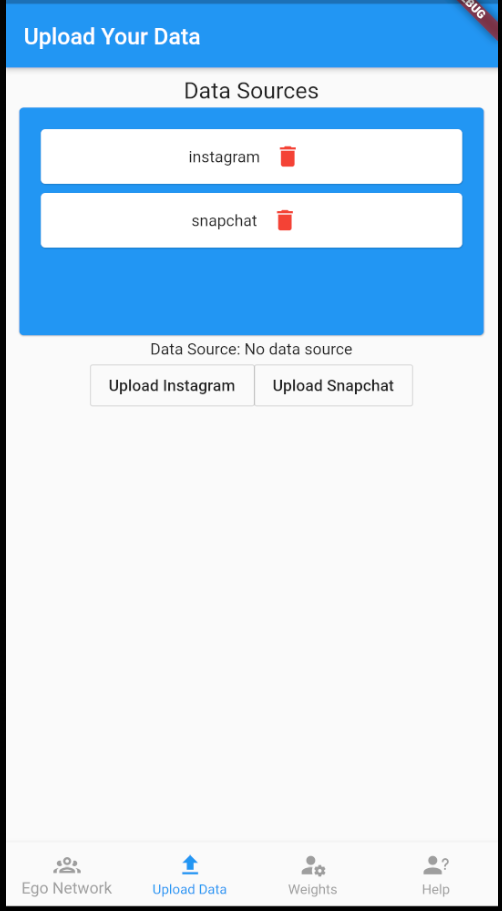


Figure 4: Upload Data Page

## **3.2 Outputs**

After the app has calculated the user’s ego network the app shall display the ego network to the user in an interactive way. The ego network algorithm will place the user’s friends into specific friendship levels based on how close the algorithm determines they are. The friends that are the closest will be placed into level 0 with the least close friends being placed in level 3. The user shall be able to see how many people are in each friendship level of their ego network. Additionally, if they select a level, they will be able to see a more detailed view of which of their friends are in that specific level of their ego network. This display is shown below in Figure 5. The left side of Figure 5 shows the user’s ego network as a whole. The right side of Figure 5 shows the view the user would see if they selected a level, in this case Level 2. It would show the people in that level and their friendship score. Ego network can be helpful with contract tracing with COVID 19 illness.

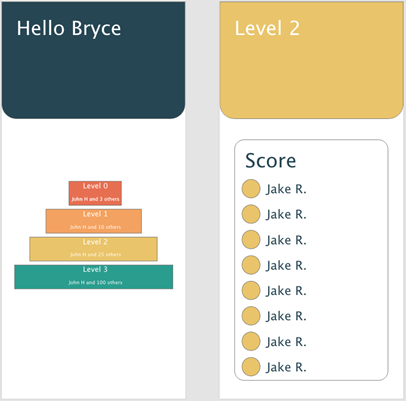


Figure 5: Output Displayed to the User

# **4 Detailed Design**

## **4.1 Hardware Detailed Design**

This project is purely a software project so there is no hardware design that needs to be discussed in this document.

## **4.2 Software Detailed Design**

The app has 3 main subsystems: the display subsystem, the data subsystem, and the ego network subsystem. The display subsystem displays the app to the user. This includes displaying the input and output of the other subsystems to the user. Next is the data subsystem, this subsystem deals with the user’s data once it is uploaded. This system accepts the data path, then finds the data and cleans it up so that it can be used by the ego network algorithm. Finally, the ego network subsystem takes the data outputted from the data subsystem and uses it as the input to the ego network algorithm. The algorithm then calculates the user’s ego network, which the display subsystem outputs to the user. An overview of how these systems communicate can be seen below in Figure 6.

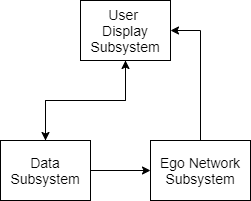


Figure 6. Software Architecture Communication: This figure gives a visual representation of how the 3 main subsystems of this app communicate.

The first subsystem the user will interact with is the display subsystem. This subsystem will display the app to the user and allow them to navigate to the different tabs of the app. These tabs include the Ego Network tab that allows the user to calculate and view their ego network, the Upload Data tab where the user links their data sources. It also includes the Weights tab where the user can customize the weights the algorithm will use to determine closeness scores, the Help tab that the user can view for more information about how to use the app, and the CDC Guidelines page with information on fighting COVID-19. See Figure 7 below for an example of a page shown using the display subsystem.

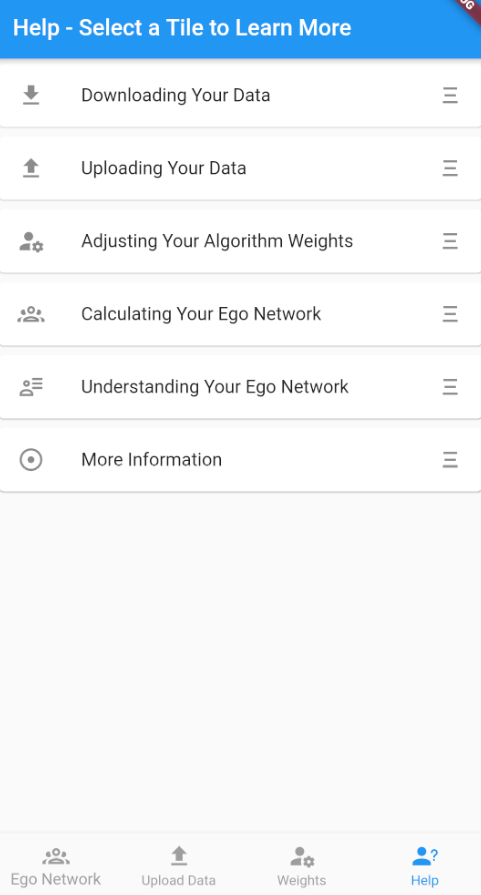


Figure 7: Display Subsystem Example

Next, the user will interact with the data subsystem. This subsystem will allow the user to select a button corresponding to the data source they wish to link with the app. After the button is selected the user will navigate to the correct folder using the device’s file explorer and then select the folder, they wish to link using the USE THIS FOLDER button at the bottom of the screen. When the user links a folder to the app the app will remember the file path to the data which the app can then call when it needs to access the data. If the user wishes to unlink the data source, they can use the trash can icon beside the data source they wish to unlink in the Data Sources section at the top of the screen. See Figures 8 and 9 below.

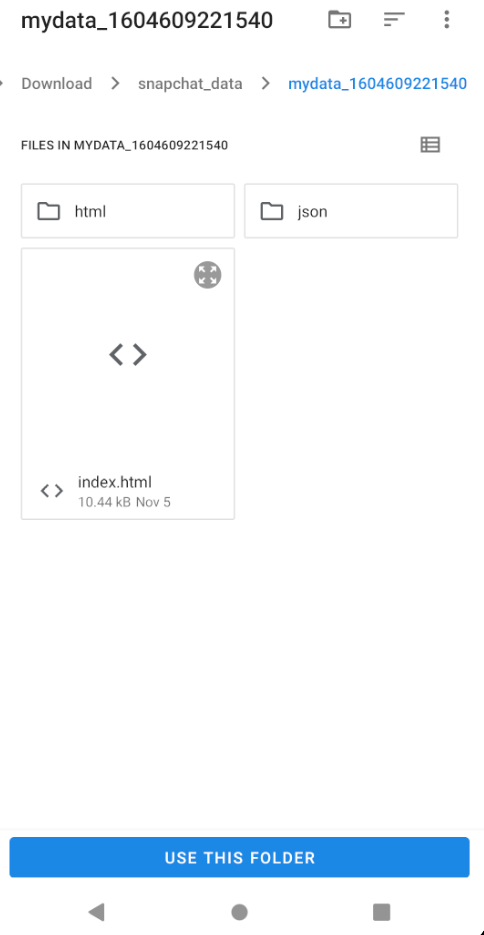
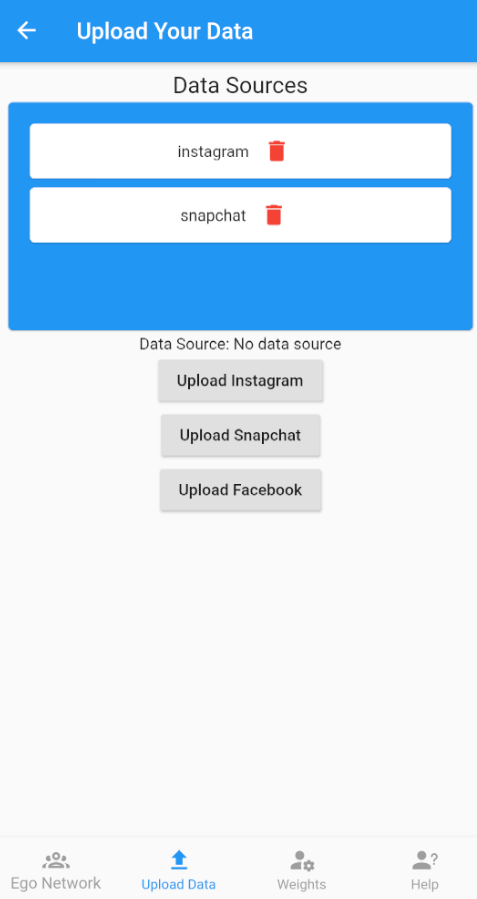


Figure 8 (left): The Upload Data tab. This tab is where the user can select which data source they would like to upload.

Figure 9 (right): This screen shows what the file picker system looks like when the user is tasked with selecting the folder containing their social media data.

Next, the user can use the display subsystem again to calculate their ego network. Before doing this the user has the option to go to the Weights tab in the app and alter the weights that the algorithm will use when calculating closeness scores. To see what the Weights tab looks like see Figure 10 below. Each weight option has a name that gives an idea of what that weight is and has a weight value, the higher this value the more emphasis the algorithm will give this parameter. The user can use this to make the algorithm more specific to their communication habits. For instance, if the user only sends snapchat images to their closest friends, they can give that weight a higher value so that they algorithm will have a better idea that people who receive snapchat images are closer friends. However, if the user has already calculated their ego network and then alters their algorithm weights the user will have to recalculate their ego network for the weight changes to take effect.

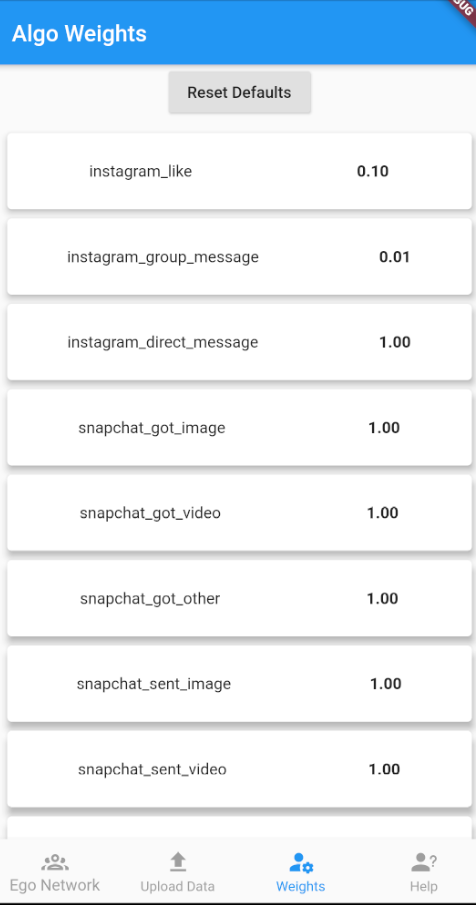


Figure 10: The Weights tab.

If the user does not wish to alter the weights or has already done so they can then move on to using the ego network subsystem. This is done by navigating to the Ego Network tab and selecting the Calculate or Re-Calculate button. Due to the variety of sources used by the app, the algorithm requires them to be in the same format. We created a custom json format we refer to as the universal entry format. Data is converted by the data subsystem from the default format you get when you download the data to the universal format by custom written parser classes. These custom classes all inherit from a parent parser class to make things more flexible on the app side. The algorithm then reads all entries and scores based on a simple formula.

*Where is the weight for the element type, is the time weight, and is the time difference from the present.*

Scores are totaled for each unique user and then stored inside the device’s documents directory as lastCalculatedAlgo.json. We store the results in the file because running the algorithm is very resource intensive, so caching helps with the user experience. It also allows the app to always display the last ego network calculated, so even if the user closes the app and then re-opens it, they will be able to see their last ego network. From there the levels are calculated using a new K-Means Clustering algorithm. The algorithm runs many times and scores each run to find the most fit set of parameters. An excerpt from the core code is below.

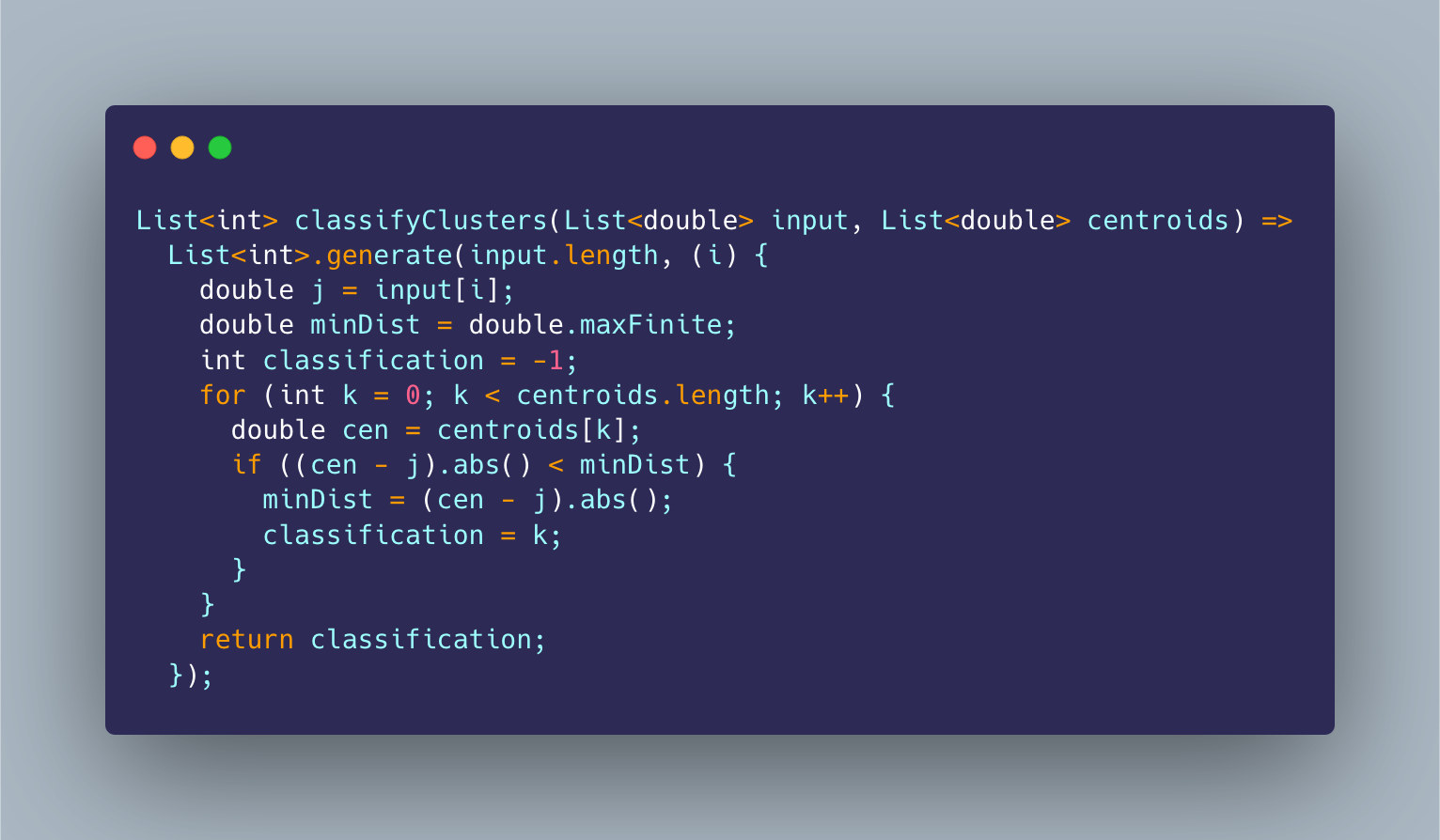


Figure 11: Code snippet from algorithm

A new subsystem that has been added is the emotion classifier. This is one of the larger new additions to the project this semester and consists of a neural network that attempts to classify an emotion based off of pre trained data. The result is two percent values that describe how confident the network is that it is a positive entry and how confident that it is negative. This result will get passed into a simple algorithm that will add that result to the score from previous semester. This algorithm will be based off the topics described in the previously mentioned research papers.

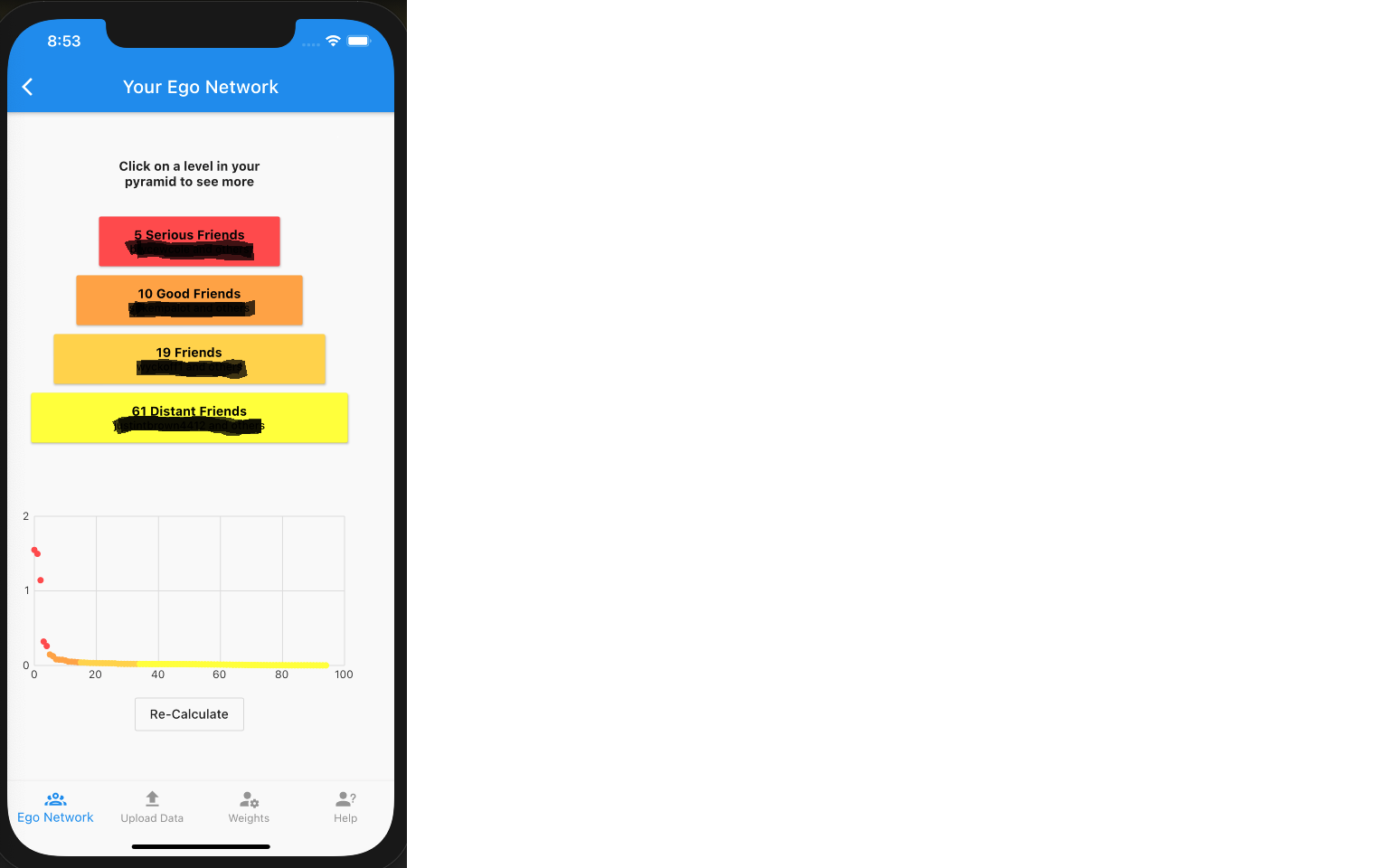


Figure 12: Ego Network results.

After the ego network subsystem has completed the display subsystem once again takes over. It will display the newly calculated ego network by reading the file lastCalculatedAlgo.json, seen above in Figure 12. It will display each friendship level and state how many friends are in that level, if the user selects a level then they will be able to see an expanded view of everyone in the level alongside their closeness score. Below the friendship levels it displays a graph of the user’s friends. Along the X axis is the friends rank and along the Y axis is the closeness score of each friend. The dots shown are also color coordinated to reflect the friendship level they belong to.

## **4.3 Internal Communications Detailed Design**

Our COVID-19 Tracking app deals with data from an ever-expanding pool of data sources. In order to properly analyze all this incoming data, we need to have a singular format that all data is in.

1. All data must be in JSON format, each file should be a list with each child being its own data entry.
2. A **source** is a platform that provides data, example Instagram or Snapchat.
3. An **entry** is a more flexible definition. It is anything that could link one person to another. This could be an Instagram DM, the fact that person A follows person B or even a view on someone’s snapchat story.
4. Each entry consists of **fields**, these fields are split into required and optional. All required fields must be filled out for the entry to be valid.

### **Required**

* Source - The source that the entry came from.
* Time - The time that the interaction took place, in UNIX timestamp format.
* Person - The other person, **not** the user who installed the app.
* Type - The type of entry data, this could be Instagram DM, Follower, etc. The weight of the communication will be pulled from a separate JSON file.

### **Optional**

These will change based on the source. The idea behind these optional fields is to provide information that is needed but it is not guaranteed to be there.

**Data Communication**

Data communication occurs between all three of the subsystems of the app. Data communication begins once data has been linked using the data subsystem by the user by utilizing the display subsystem. This data is then accessed when the ego network subsystem is called. The ego network subsystem communicates to gain access to the data, gets the data subsystem to convert the data into the form readable by the algorithm, and then calculates the user’s ego network. After the ego network subsystem is done it creates a file that stores the results of the ego network algorithm. Then the display subsystem communicates with the ego network subsystem to display the saved ego network results to the user. Throughout this process of data communication all three subsystems must work together.

# **5 External Interfaces**

## **5.1 Interface Architecture**

The project has no interfaces that need to be described.

## **5.2 Interface Detailed Design**

This project has no interfaces that need to be described.

# **6 System Integrity Controls**

The app has built-in controls in case the data that it tries to access is not found or has errors. The app will alert the user that there was an error relating to the data link and that they need to relink their data. To protect the privacy of the user, the app will hash the file that contains the output of the ego network algorithm so that it is better protected and not stored in plaintext in case someone tries to read the file without using the app. Additionally, to protect the user’s privacy this app will keep all data local on the device. This will protect the user from having their data in the cloud where it is more vulnerable to security threats.