Bilkent University

Engineering Faculty

Department of Computer Engineering

CS 319

Object-Oriented Software Engineering

Course Instructor: Can Alkan

Semester: 2015-2016 Fall

Final Report

Group 02

Twitter Analysis

Miraç Aknar
Buğrahan Kalkan
Osman Koyuncu
Emir Özbek

Table of Contents

Table of Figures

1. Introduction (Problem Statement)

The system which takes information of Twitter users and stores them in a database, then analyzes those data, object oriented, and is “Twitter Analysis”. This data analysis will be able to work by only taking user’s simple logging in to Twitter. After logging in the user will be able to choose different functionalities and get information about. After the process of taking, storing and analyzing data, the system will give its analysis result to user. This analysis is composed of followers’ and followings’ statistics, differentiated community analysis, like friend groups and families, finding popularity among user’s followers and followings. This system provides those by the analysis part as the control units of the system.

Twitter analysis, works in a way that user first will log in with his Twitter account, second he will pick a target user for analysis, it can be a friend of user’s or user himself, next he picks an analysis, such as sending tweet frequency statistics or community analysis etc. Then system does that analysis then give the result of it to user, by showing it on the website, our user interface.

In this report, after introduction an overview of project will be given to reader with functional and non-functional requirements, constraints, scenarios and use case models and user interface, which is requirement analysis. After that as analysis, both object models, domain lexicon and class diagram, and dynamic models, state chart and sequence diagram is included. Finally, it ends with a conclusion.

2. Requirement Analysis

2.1. Overview

2.2. Functional Requirements

· The user should be able to login the program through his/her Twitter id and password.

· The user should be able to logout the program.

· The user should see his Twitter followings and followers in his/her graph of social cycles.

· The system should have follow statistics of the user which are how many of followers that the user follows back and how many of followings that follow back the user.

· The system should analyze Twitter retweet statistics which are how many of the last a hundred tweets of user retweeted and whose tweets the user retweets most.

· The system should create Twitter community analysis whose output is the social cycle graph.

. User should see tweet statistics which are distribution of tweets in terms of days of weeks and distribution of the tweets in terms of hours of day.

. User should see favorites statistics which are which users that the user favorite most.

. User should see all statistics and analysis of any other user such as tweet, favorite, retweet, and follow statistics except the community analysis which provides the graph of social cycles.

2.3 Non-Functional Requirements

· The system shall be licensed.

· The system shall have firewall.

· There is “tutorial” document that explains how to use system to user.

· The system shall not store any data on the server or computer.

· The response time must be kept under 5 minutes otherwise there must be a warning message.

. In case of system crash the reason of the crash must be viewed to the user.

2.4 Constraints

· The program must be implemented in python.

· The program must be online.

· Twitter API must be uses

. Tweepy which is an open source Python library is used to connect with API

· d3 which is an open source JavaScript graphic library is used to draw social cycles graph.

2.5 Scenarios

Scenario Name: Learning statistics about followers and followings

İbrahim Melih Gökçek is a twitter user who posts tweets and spends time in twitter almost every day. He enjoys these activities but he wants to learn more about his activities such as at what rate he follows back his followers, what is the gender ratio of his followings, how many tweets does he post in a day on an average. He also wants to know these statistics of his followings or followers.
He heard about a website in which he can find out these statistics. After an ordinary daily usage of Twitter, he connects to the website and see the welcome page. He logins with his twitter id and password. He finally sees the main page of the website. All of his followings and followers are seen at a graph on that website. When he clicks one of the icons which belongs to one of his followings or followers, he can see many statistics that are mentioned above. Therefore he learned the information about his followers and followings that he wonders.

Scenario Name: Connection between social circles and users

Buğra is a twitter user who received his high school education in İstanbul and comes to Ankara for his undergraduate education. He has many friends from İstanbul and he makes new friends in Ankara during his University education. So he has two different social circles and there are some connections between these circles. Twitter is a popular social media website among both of his high school friends and university friends. He wants to know about these circles and connections. He is also curious about who is more active in Twitter and whose posts are retweeted more or who follows who among his friends.

He finds out a website where he can see his social circles and connections between these circles even he can observe who is popular in this groups. He connects the website through his twitter account and see a graph where he can observe connections and circles. So he satisfied his curiosity about his friends.

2.6 Use Cases

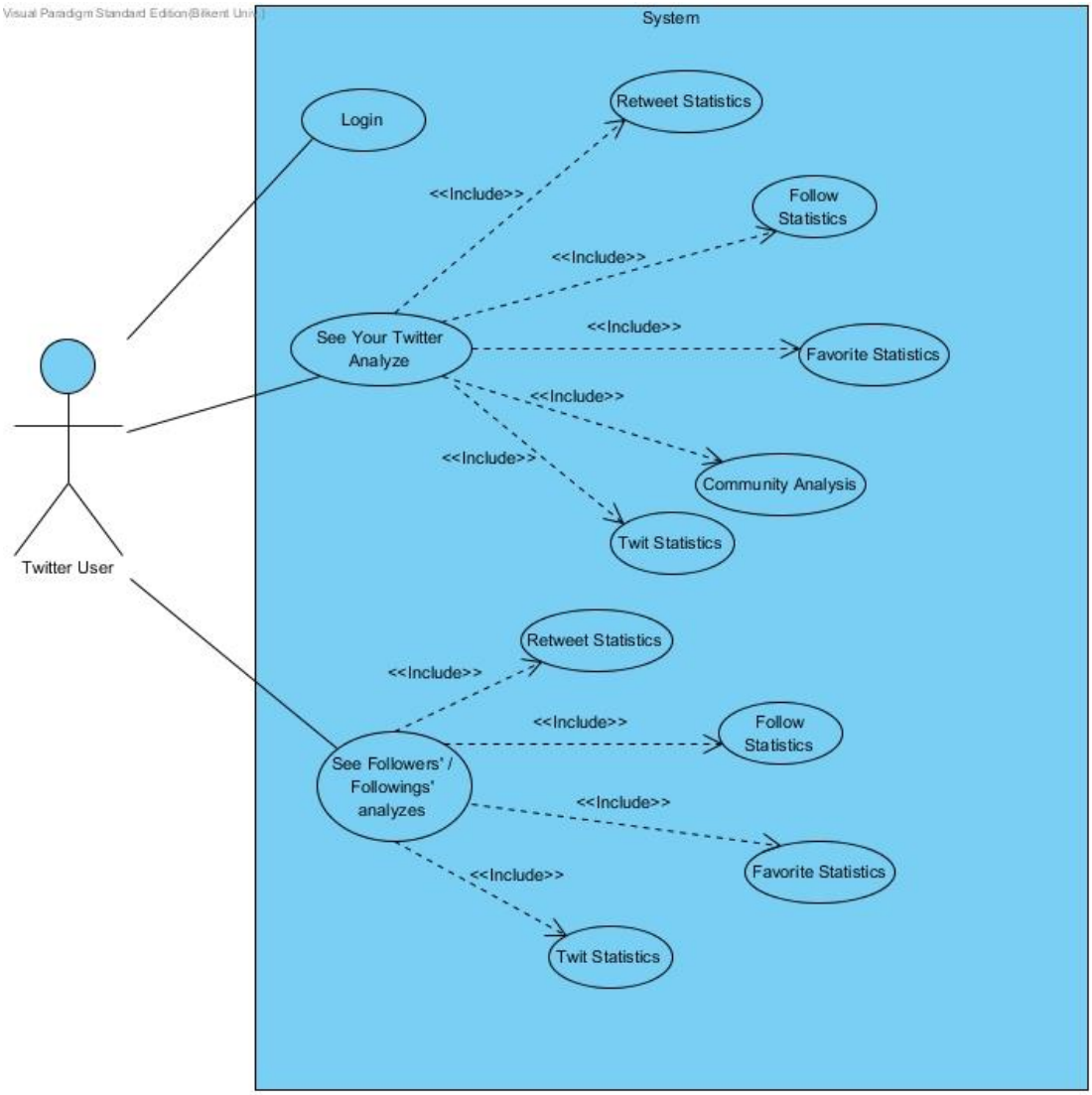

Figure 1 : Use case diagram

2.6.1.Use Case: Login

Participating actor: Eren Boğa (User)
Entry Condition: User access the website.

Exit Conditions: User exit the website.

Flow of Events:

1. User starts a browser.
2. User types the URL of the website on the address bar.
3. Welcome screen appears with a login panel.
4. User enters his/her twitter id and twitter password
5. User clicks the login button
6. User exit the website.

2.6.2.Use Case: Twitter Analysis

Participating actor: Leydi Boğa (User)
Entry Condition: User can choose either own Twitter analysis or follower's/ following analysis

Exit Conditions: User exit the website.

Flow of Events:

1. User can analyze his/her own personal Twitter information
2. User can analyze his/her follower's/ following Twitter user information

2.6.3.Use Case: Choose which person analyze

Participating actor: Randy Marsh (User)

Entry Condition: User click follower's/ following analysis button

Exit Conditions: User exit the website.

Flow of Events:

1. User click search follower/following button.
2. User choose a person from his/her follower's/ following.
3. User press Analyze button.

2.6.4.Use Case: followers/following analyze

Participating actor: Buğrahan Kalkan (User)

Entry Condition: User press Analyze button.

Exit Conditions: User exit the website.

Flow of Events:

1. User can see his/her follower's/ following user follow statistics.
2. User can see his/her follower's/ following user Tweet statistics.
3. User can see his/her follower's/ following user favorite's statistics.
4. User can see his/her follower's/ following user retweet statistics.
5. User exit the website.

2.6.5.Use Case: user own Twitter analyze

Participating actor: Buğrahan Kalkan (User)

Entry Condition: User click your Twitter Analysis button.

Exit Conditions: User exit the website.

Flow of Events:

1. User can see his/her own follow statistics.
2. User can see his/her own Tweet statistics.
3. User can see his/her own favorites statistics.
4. User can see his/her own retweet statistics.
5. User can see his/her own community analysis.

6. User exit the website.

2.7 User Interface

2.7.1 Login Screen

The initial screen of the program is the login screen. In this screen the user will enter his/her username and password to login. So the user is expected to enter correct username and password accordingly. After the user logins the second screen will appear.

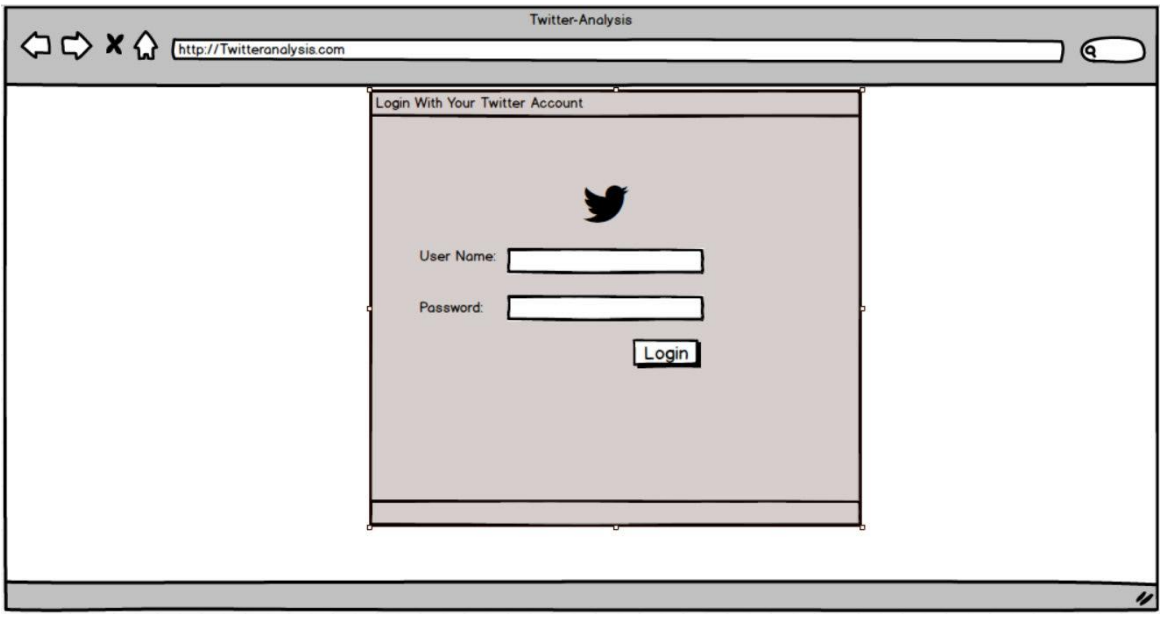

Figure 2: User interface main page

2.7.2 Analysis Main Screen

This screen will appear right after getting login information of the user. The user will be able to get his own twitter analysis or one of his friends' analysis. If the user chooses 'Your Twitter Analysis' option, he/she will be asked to choose which analysis types are desired. If the user chooses the other option which is 'Follower's/Following's Analysis', he/she will be asked to pick which aspects of their friend's analysis are desired. The user also shall be able to logout safely.

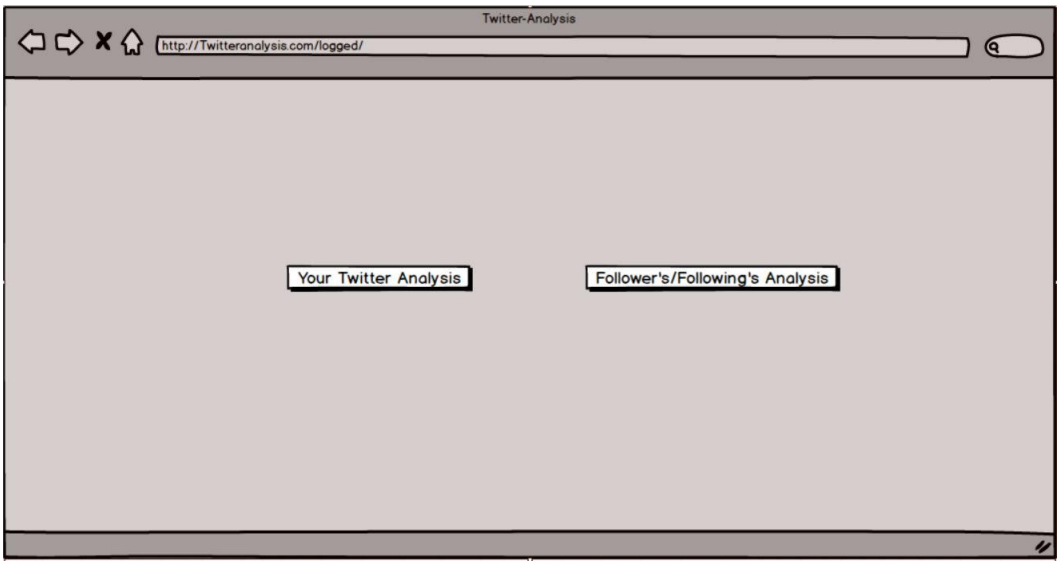

Figure 3: User interface main page

2.7.3 Search Screen

This screen will appear if the user clicks on follower's/following's Analysis which means he wants to see one of his/her friend's analysis. In this screen, the user will be provided with a text label and a simple list. The user will be able to type one of his/her friend's name and see the analysis for that particular person. If the user types only name of that person, all the followings/followers who have the same first name will be shown in the list. If the user types the full name of the user, only that particular person will be shown on the list so that the user can easily reach his/her analysis.

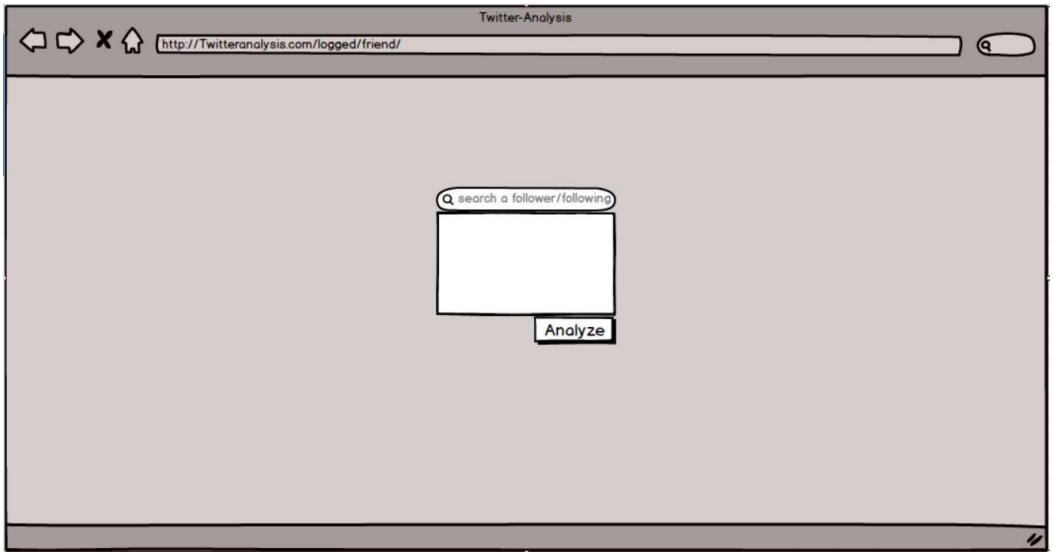

Figure 4: User interface search page

2.7.4 Twitter Statistics Screen (Friend's)

After the user determines which of his/her friend's analysis he/she would like to see, this screen will be shown on the screen. At the left top corner of the current screen, the user will be able to observe his/her friend's profile picture, username, number of followings and followers accordingly. The user also will be given with 4 different options to continue with on their analysis process. If the user picks 'Follow Statistics' option, he/she will be able to see the screen which demonstrates the statistics of one of his/her friend's followings and followers. If the user picks 'Tweet Statistics', the Tweet Statistics screen will appear. If the user picks 'Favorite Statistics', the Favorite Statistics screen will appear. Lastly, if the user picks 'Retweet Statistics', the Retweet Statistics screen will appear.

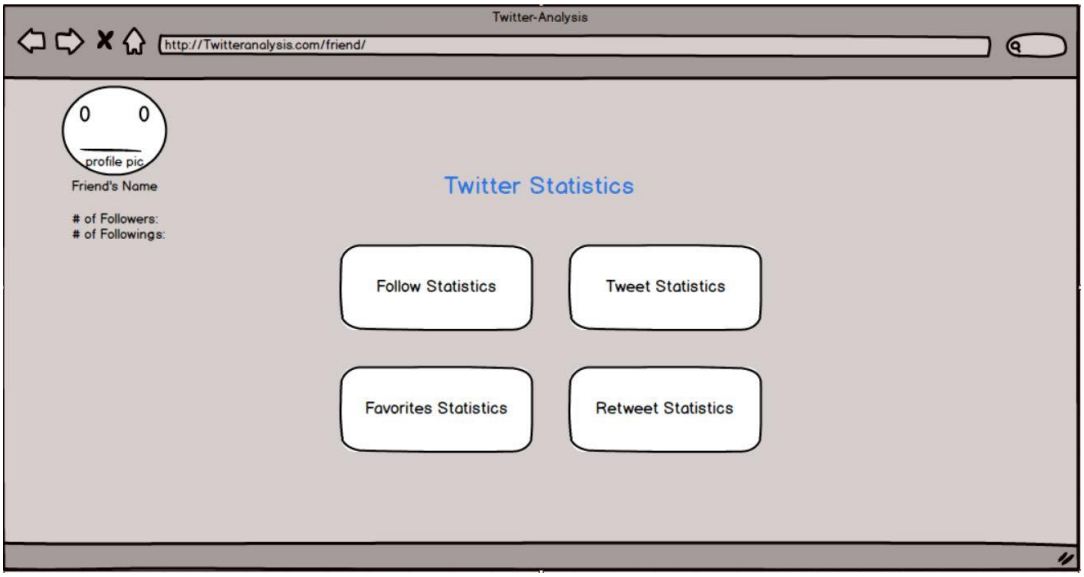

Figure 5: User interface friend's statistics page

2.7.5 Twitter Statistics Screen (User's)

If the user picks 'Your Twitter Analysis', this screen will appear on the screen. This screen has the same functionalities with the Twitter Statistics Screen(Friend's). But this screen also offers Community Analysis which is a very extensive type of analysis which analyzes following and followers and gives the information with a specific graph.

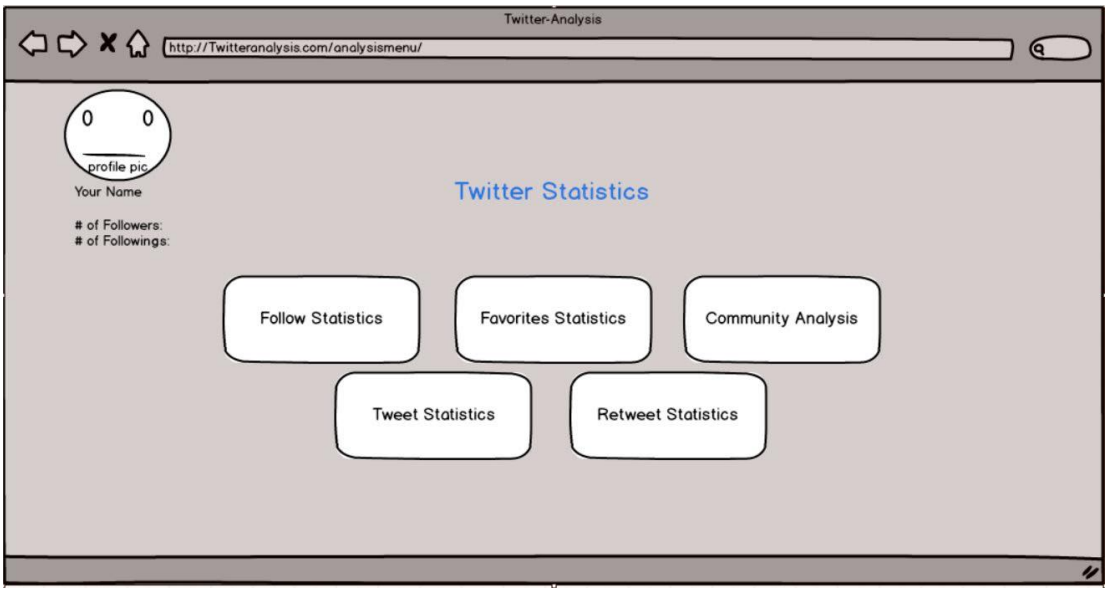

Figure 6: User interface the user's statistics page

2.7.6 Follow Statistics Screen

This screen allows the user to see both followings and followers statistics of a desired account. This screen has 4 diagrams (2 for followings and 2 for followers). In Followings section, the user will be able to see who follows and does not follow the desired account from his/her followings. The other graph shows how many of his followings are male and female. In Followers section, the user will be able to see who follows and does not follow the desired account from his/her followers. The other graph shows how many of his followers are male and female.

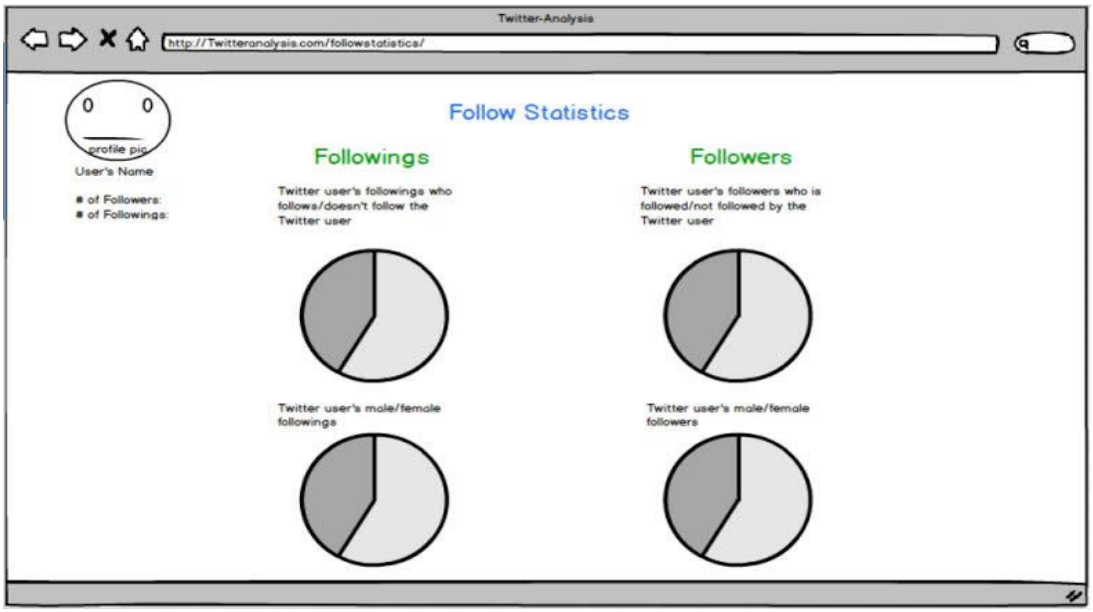

Figure 7: User interface follow statistics page

2.7.7 Tweet Statistics Screen

This screen demonstrates two similar graphs to the user. According to last 20 posts of the account, the graph arranges tweets to days of weeks they are posted. The other graph also arranges last 20 posts of the account so that the user will be able to see which hours of day they are posted. Therefore, distribution of the last 20 tweets can be examined easily.

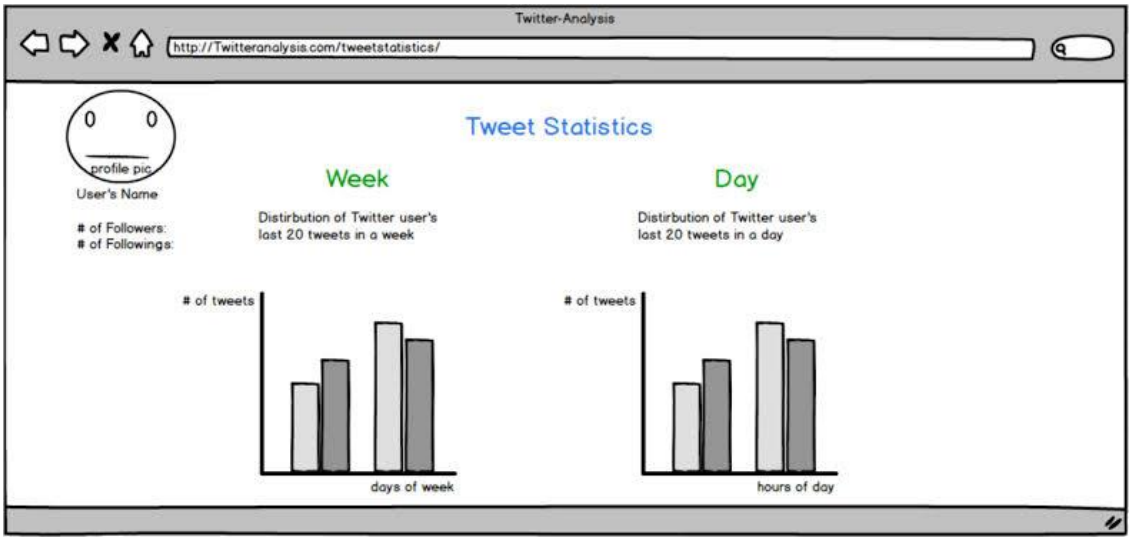

Figure 8: User interface tweet statistics page

2.7.8 Retweet Statistics Screen

This screen allows the user to see the last 10 tweets of the account and a graph which illustrates the frequency of followers who retweeted users' last 10 tweets so that the closest friends of the user can be seen easily. If a friend did not retweet one of the last 10 posts, he is included as never in graph. If a friend retweeted 1 or 2 tweets from last 10 post, he is included as rarely in graph. If a friend retweeted 3 or 4 tweets from last 10 post, he is included as rarely in graph. If a friend retweeted 5 or 6 tweets from last 10 post, he is included as often in graph. If a friend retweeted 7 or more tweets from last 10 post, he is included as frequently in graph.

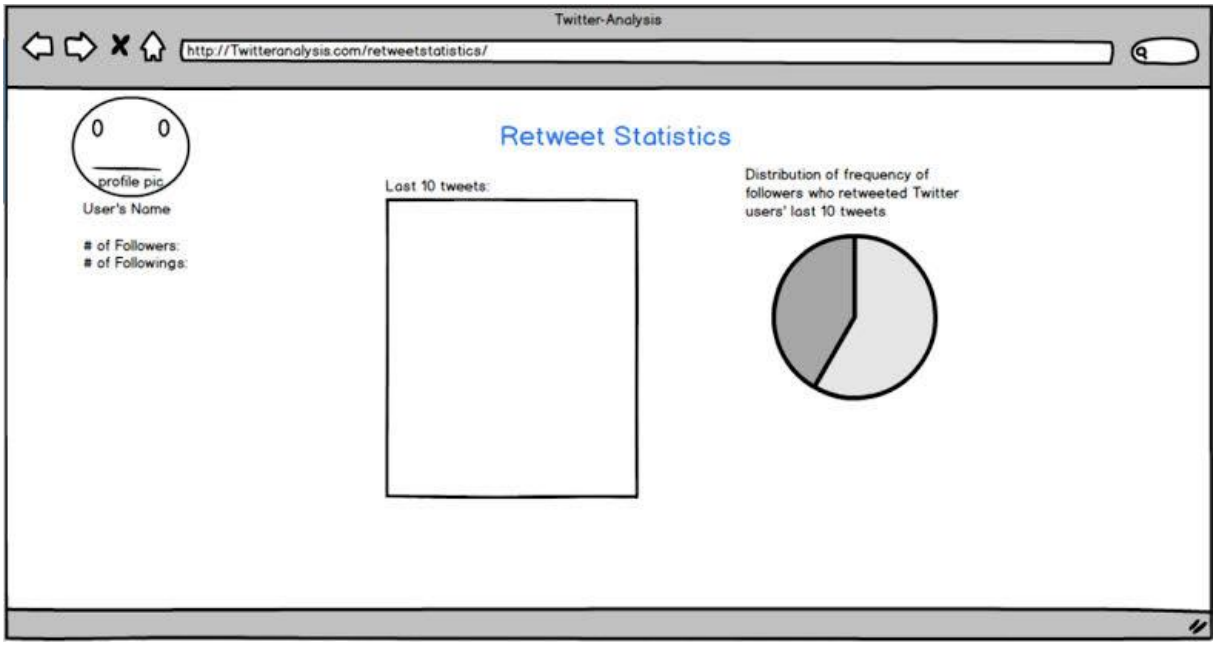

Figure 9: User interface retweet statistics page

2.7.9 Favorite Statistics Screen

This screen allows the user to see three of his friends who he/she favorite at most and a graph which illustrates the distribution of frequency of followings whose tweets the user favorite at most. This enables the user to figure out which of his/her friends he likes to favorite more than others.

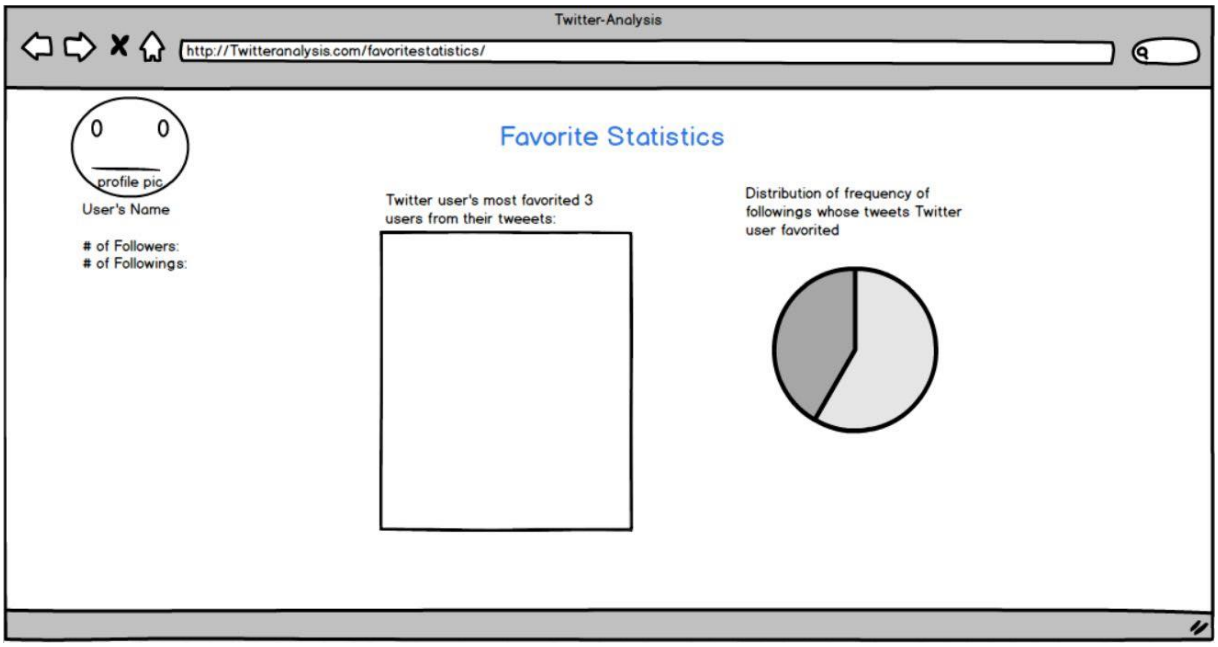

Figure 10: User interface favorite statistics page

2.7.10 Community Analysis Screen

This screen plots a graph which analyzes the user's friends in a way that close friend groups can be determined. Certain groups in his friends interact with each other more. Therefore they are treated as a social group in this graph. Interactions between friends are shown by lines. Friends are denoted by dots and their width and color determines how popular they are.

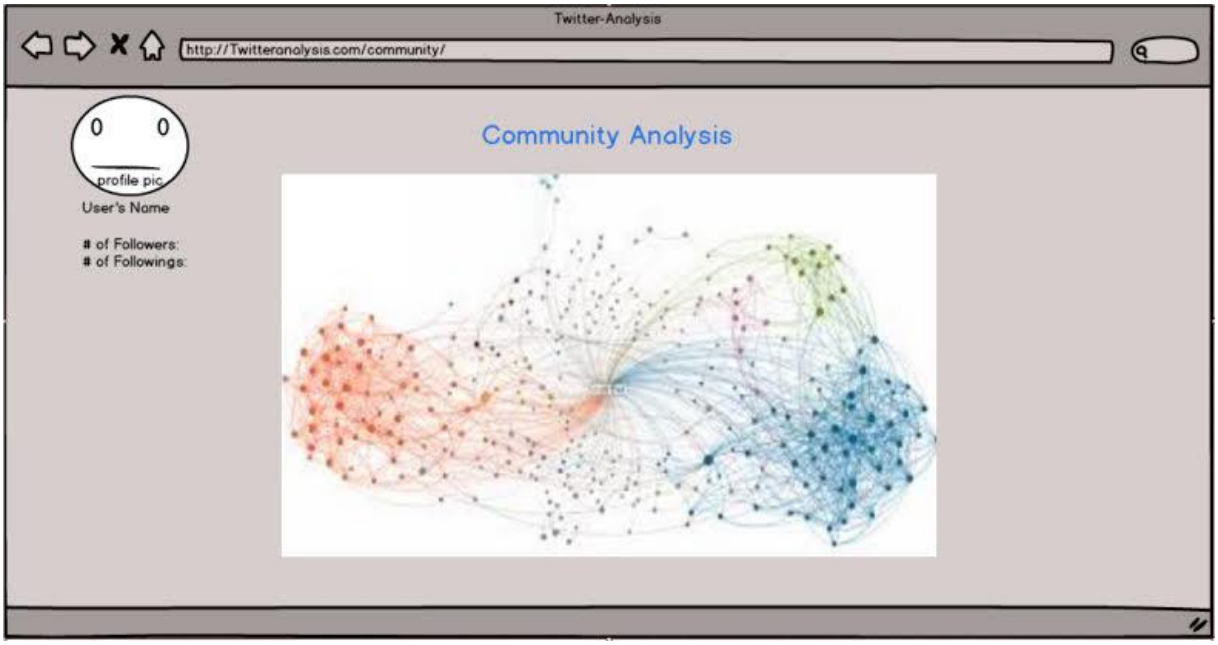

Figure 11: User interface community analysis page

3. Dynamic Models

3.1 Object Model

3.1.1 Domain Lexicon

TwitterUser: It is one of the model objects. It is defined to hold the data of any kind of user such as followings of the user or followers of the user, those types of user do not have separate classes for memory efficiency and not to possess unnecessary classes. It has twitter_id which is the unique id that twitter gives to all users. It also has username which is the users' twitter username. The other attributes are also follower_count and followers count and profile_picture which is the url of the user's profile picture.

Tweet: This is an also a model object. It is defined to hold all kind tweets like favorite, retweet or just an ordinary tweet, the reason of having one object for all type of the tweets is the same as having one user object for all type of users: memory efficiency and not to have unnecessary classes. It has a is_favorite attribute which will be true if the tweet is a tweet that is favorite by user, in this case the uid attribute will be the id of owner of the tweet that user favorite. Is_retweet attribute will be true if the tweet is a retweet and if it is, uid will be the id of the tweet owner. Uid will be zero if it is an ordinary tweet. Tweet class has also created_at which is a Datetime, so it is the time that tweet posted in terms of day and hour. Finally, text is the text of the tweet, it is holded since it is possible that the project may do some text analysis in the future.

TwitterConnection: It will have list of access keys and use them to make connection with Twitter through the library tweepy. If the key in process will exceed the request, it will change the key with another one in the list.

3.1.2 Class Diagrams

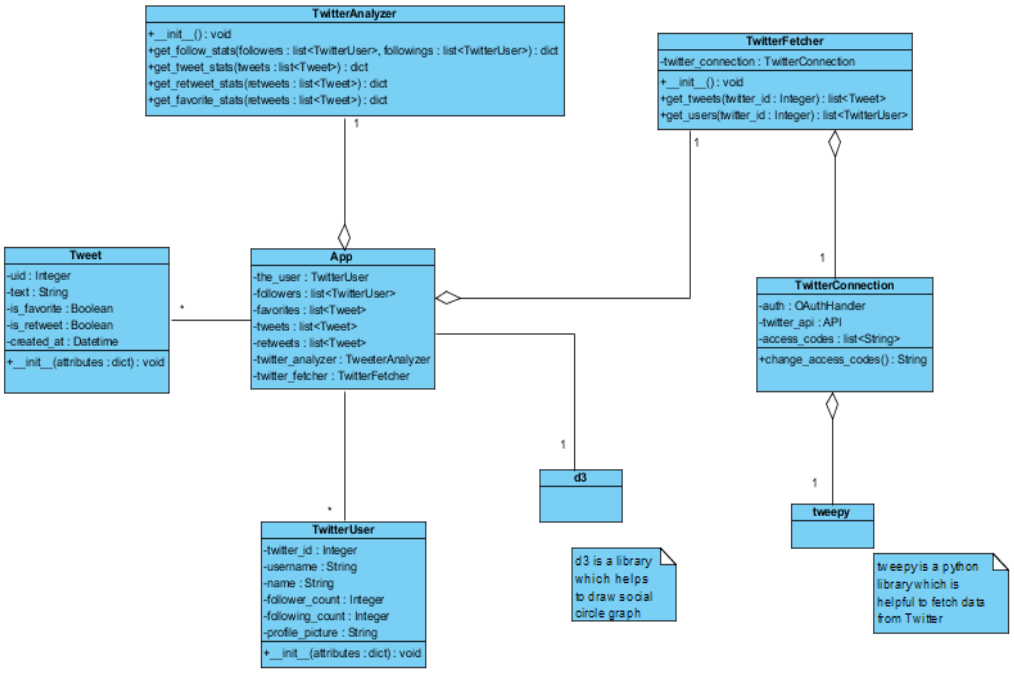

Figure 12: Class diagram

3.2 Dynamic Model

3.2.1 State Chart

The classes above are the model classes. The ones which are at the below are the control classes.

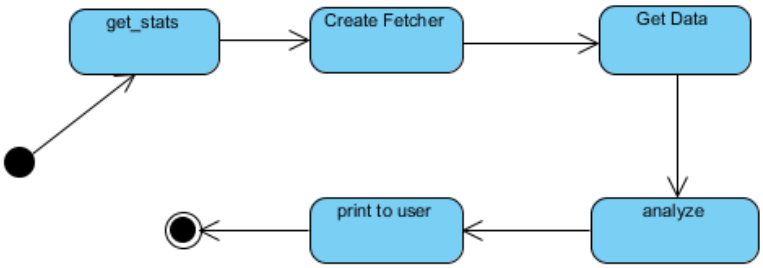

Figure 13: State chart

3.2.2 Sequence Diagram

Scenario Name: New Analysis

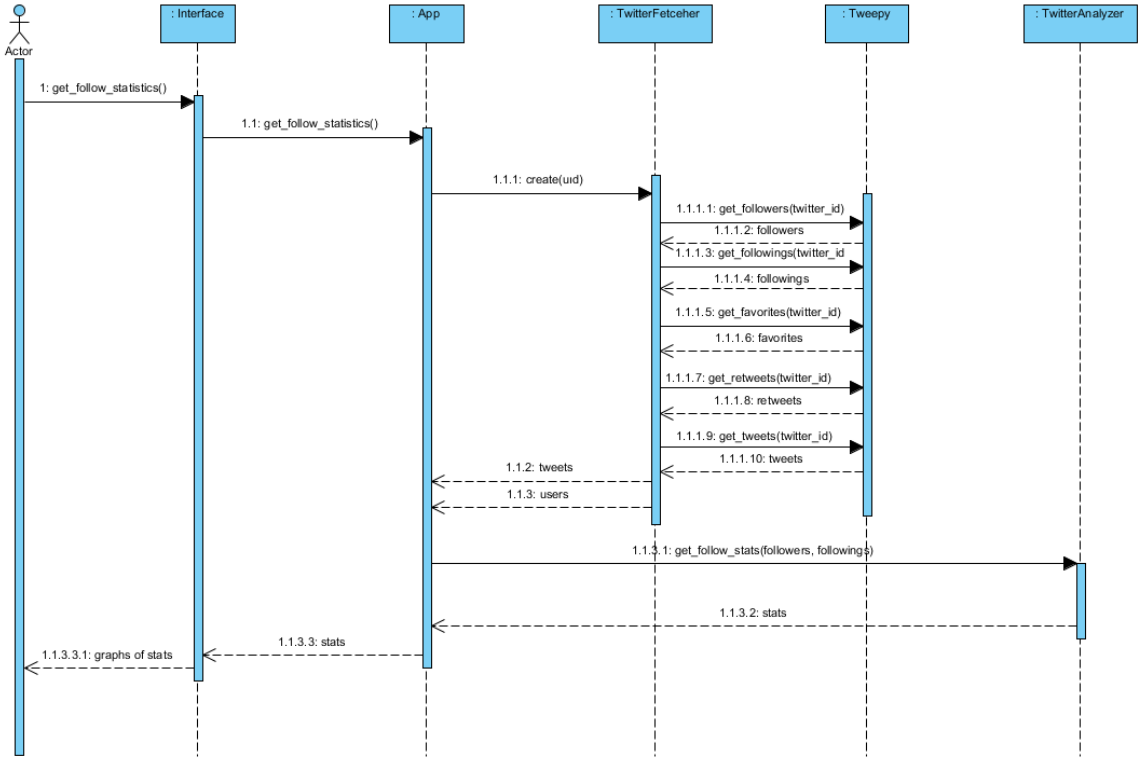

Figure 14: Sequence diagram

The actor interact with program through interface. When he wants his statistic, interface interacts with main class which is App in this program. App has an object of TwitterFetcher which is class for getting the needed data through Tweepy, which is the library we use to fetch data, by using the specified functions of it. Then it returns the data including followers and followings of the user and tweets, retweets and favorites of the user. TweeterFetcher group them into tweets and users and return it to the App as two arrays, one of them including tweets, favorites and retweets and the other contains followings and followers.

After needed data is fetched, it is needed to do analysis and obtain statistics from that data. So, this data is sent to the twitter Analyzer which is the class having analyze and statistic functions. Inputs of these functions are data that has been fetched and outputs of them are the results of the statistics and analyze. These results are sent to App to be posted on the Interface.

4 Design

4.1 Design Goals

Like any other engineering solutions, the system to be developed should also have prioritized objectives, and should optimize its resources to accomplish them. Since the system to be developed is a comprehensive analysis program, system should actually “inform” average computer users, who are not capable of dealing with the difficulties that may occur during process. Since this is a class project intended to be an example of efficient software engineering, the most appropriate methods of development should be applied.

Reliability: First of all, the system should be reliable. Since reliability is one of the fundamental design goals, it must be reached in Twitter Analysis as well. The system is supposed to run without any leak or data loss. For example, the data fetched from twitter must be correct so that it will not lead to frustration. The system fetches data through a library named Tweepy which uses Twitter API, so the data comes from the sources that Twitter shared. Tweepy is an open source library its functions and classes are public and can be examined by anyone. It can be seen that the data that system fetches fits the data that in the Twitter. Reliability involves a strong implementation and testing but it is intended to allocate just enough time for both.

Modifiability: In order to provide satisfying analysis experience to users, modifiability is a significant factor. It is possible to develop the program in the future, thanks to modifiability of our program. Through system patterns that is used in the project, the program can be modified easily. MVC pattern allows changing interface of the program or changing the analysis separately. Since Twitter shares many information about users and tweet, there is a great possibility to add new statistic and analysis to the program. For example, when a user is not satisfied with the result of the analysis, developers should be able to modify specific parts of the implementation so that the problem will not occur again.

High Performance: The system should not use more than anything what is allocated. Since, high performance is not mandatory, it is what makes a program distinguishable. Since Twitter updates by every user constantly, the data that we fetch and the statistics are valuable for a short time, any data or statistics are not stored. Storing data permanently may slow the system down, so our program is advantageous by this aspect.

Good Documentation: Documentation should be well organized and straightforward so that users will not have any difficulty in the use of program. Also, it is very important to have a detailed documentation of the analysis, design and development processes for other developers that like to improve the project. A detailed background of the analysis might be beneficial for the future improvements of the program.

Minimum Number of Errors: Our strongest goal was to create a reliable system. In order to develop a reliable program, number of errors should be lowered. Majority of analysis programs are unusable because of errors, so users prefer other options. We are aware of this fact and want to prevent any errors that may occur. Since the data that we use comes from Twitter API, we cannot prevent the errors that might occur in Twitter API, apart from that Teeepy is updated according to API in a suitable way, we do not expect any problem if there is no problem about API itself.

Ease of Learning: The use of options is very explicit. Some of the results of the analysis are complex, but the user also will be provided with necessary information. Other than that the program is thorough so user should not need any other assistance. The terms that we use are the general terms of Twitter, so it is very

easy to understand the statistics and analyses for any Twitter user. Since the system will be use through a Browser it is easy to connect to the system and see the statistics for an internet user.

Ease of Use: The interface of the program is easy to understand. User can recognize each buttons' functionality. The graphs may be hard to understand so we intend to make them as simple as possible. A short explanation about each graph can be added to their below to abolish the possibility of misunderstandings. Also the parts of screens are divided logically in to pieces.

Trade-Offs: Undeniably, all optimization algorithms have some drawbacks. Although we want to keep them at the minimum level, there are still some sacrifices listed below.

Cost (High Performance, Reliability): The system should be composed of high level coding and frameworks, which may lead to more cost than usual. As an engineering practice it is inevitable that cost might be sacrificed. In analysis programs, however, it is not unusual to sacrifice a system's cost to its high performance and reliability. In our program, The user that has more followers, followings or more number of retweets, favorites will be able to see his statistic slower since the data that system uses are composed of those followers retweets etc.

Functionality (Ease of Use, Ease of Learning): The system should appeal to all kinds of user groups, from a newbie to experts at computers. For this reason, it should be acceptable to sacrifice some of the advanced features for the sake of usage and easy learning phase. In our program, some of the graphs might be challenging to understand, but we expect the user to get accustomed to program in short time. It is obvious that the system with several confusing functions might cause lots of difficulties for the users.

Rapid Development (Good Documentation, Minimum Number of Errors): Although, in our case, the implementation stage is relatively short and does not require advanced coding, this process cannot be considered as the whole development of the system. Actually, development of a program means all the parts from its first designs to the further testing. Therefore, extending this time interval may conflict with the rapid development of a program. It is intended to minimize errors as

much as possible, from design to testing. For these reasons, a considerable trade-off is expected.

4.2 Subsystem Decomposition

Subsystems are needed to organize the whole program by dividing it slightly independent parts. This principle of divide and conquer allows us to extend or modify the system easily, make performance more efficient since unnecessary data transfer between classes is prevented. So it decomposition will be helpful while satisfying non-functional requirements of our system. The system is basically divided into three subsystem which are Model Controller and Interface.

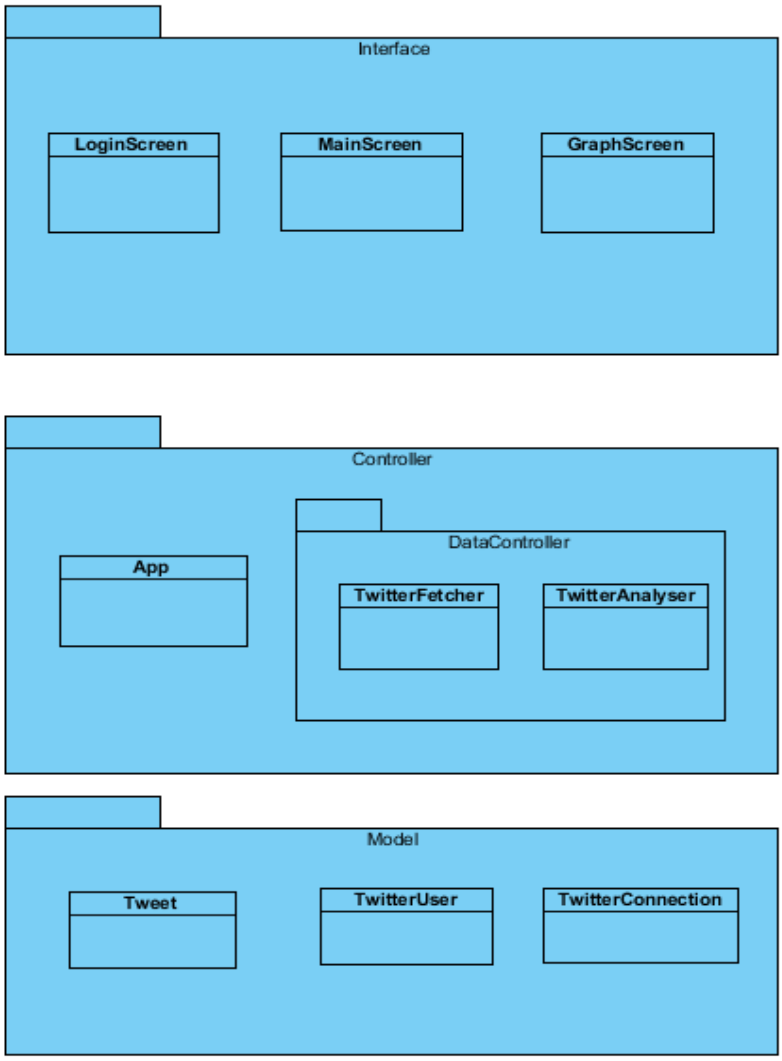

Figure 15: Basic subsytem decomposition

Interface subsystem is the part by which users interact the system. It is going to have three classes which are LoginScreen, MainScreen and the GraphScreen. LoginScreen and MainScreen is not dynamic, they will show same content for all users. GraphScreen is the one that is dynamic, it will take shape through the statistics and analysis data that come from APP class of the Controller subsystem, which is the only interactions between this subsystem and the other subsystems which makes modifying and extending both sides easier. GraphScreen may have different number of panels according to the content.

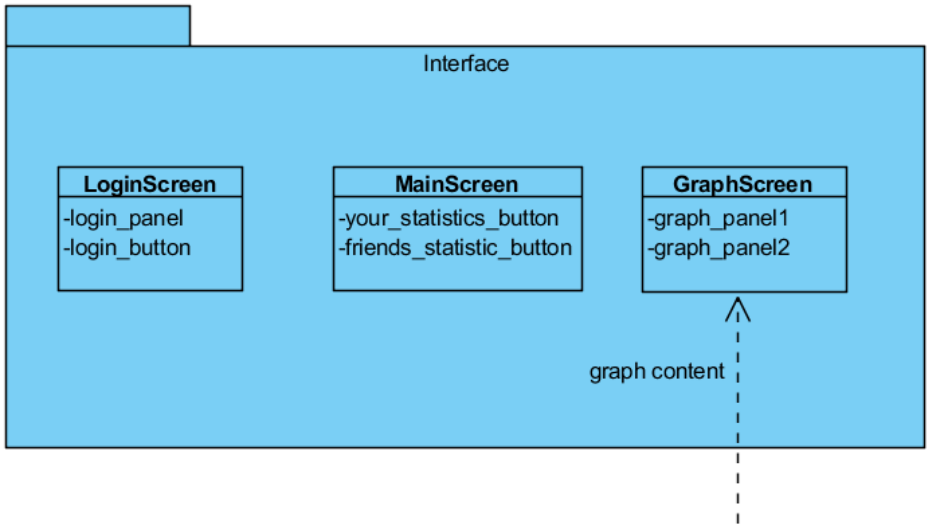

Figure 16: Interface subsystem

The controller parts which are the App, TwitterFetcher and TwitterAnalyzer. App is the main class and it has properties of main controller it has object of TwitterFetcher and TwitterAnalyzer. It send the statistics and analyses to Interface. TwitterFetcher fetches data from TwitterApi through Tweepy functions which are followers and followings of the user, besides favorites retweets and the tweets of the user. It groups these data into tweets and twitter_users, and returns these data as a list. Main controller which is App hold that lists and use them as parameter of the TwitterAnalyzer object's function. Twitter analyzer sends the results of that analyses and statistics to the App to be sent to Interface.

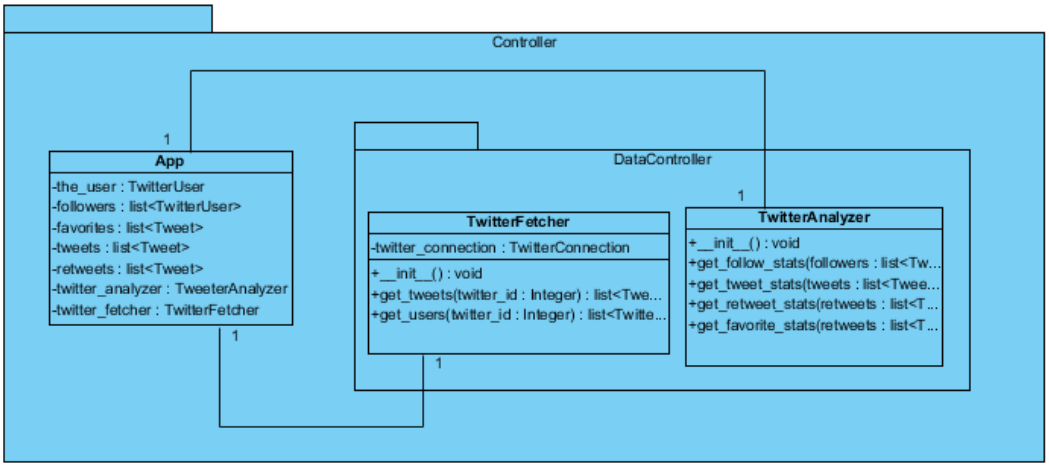

Figure 17: Controller subsystem

The model subsystem contains model objects for statistics and analyses which are TwitterUser, Tweet and TwitterConnection. TwitterUser has attributes such as id of the user, counts of followers and followings, url of profile picture of the user. It also has type attribute which specifies the user is a follower, following. Tweet has attributes like date time and tweet which will be needed for statistics and analyses. Tweet also has a type attribute specifying that tweet is a retweet, favorite or an ordinary tweet. App class of the Controller subsystem has list of tweets and users to be filled by TwitterFetcher and to be sent to TwitterAnalyzer. TwitterConnection is another class of this subsystem which makes arrangements to have a connection with Twitter API through Tweepy. It has a list of access code and it can change the access code in process in case the request limits exceed by the access code in process. It is owned by TwitterFetcher of the Controller subsystem.

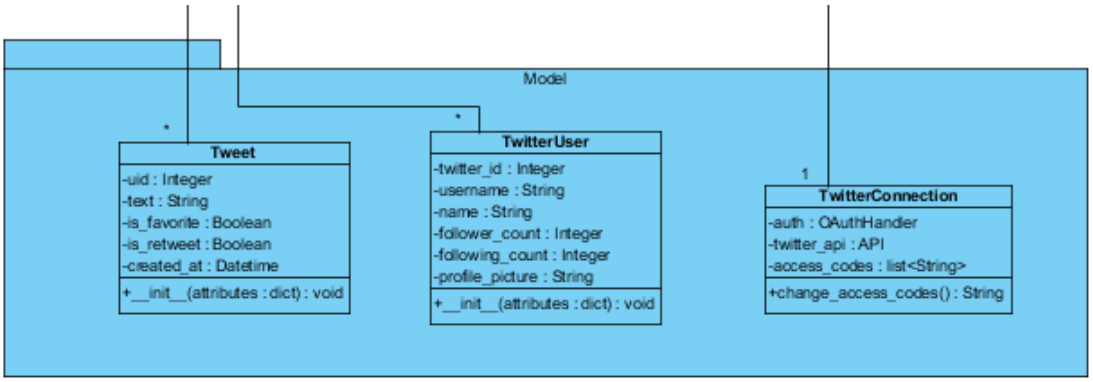

Figure 18: Model subsystem

4.3 Architectural Patterns

Our architecture of the system design composes of model view controller(MVC) architecture. However, this system does not allow a layer architecture, since our system process as a straight line, namely it passes by TwitterApi, gets the data, process and shows it. Thus, it is not possible to have a layer pattern on this system. For MVC, this system has a pattern exactly for it, in contrast to layers' not existing.

4.3.1 Model View Controller

A system with various objects always need two faces, first is the one users see, interact with and the second one is the software behind it. This software is the one that gives data to the interface user interacts, which is also known as view. This software, in itself, is also composed of parts. The organ that controls data flow, modify it, send it between other organs, the controller, and the organ that stores the data for usage or after usage in controller, the model. A system works with this architecture is required also for our project.

Model-View-Controller (MVC) is depended on our project, since the showing analysis results in a web browser, view, taking data from Twitter via TwitterApi processing that data on users, tweets, retweet, follow situations, analyzing them and then send them to view in order to show it, controller, and storing that data during this process, model, require this MVC architecture.

View part of MVC will be on a web browser which takes analyzed data and make it showable to user. That is the main goal of it, and also it takes user's login information with that information controller goes by TwitterAPI to take the wanted data of that user. Login screen, analyze choices are in the view of our project.

Model of MVC in our system, stores the data taken from Twitter itself by controller.

Controller of MVC, does all these mentioned jobs above and also does the analysis part of data from Twitter.

4.4 Hardware and Software Mapping

It is expected that many kind of users with different type of devices that have different operating systems and browsers will use Twitter Analysis. Twitter Analysis

can be used by those users, only requirements are a device with Internet connection and a browser that is installed in the device. So mobile users or desktop uses will be able to use it. When the user enter the website, he will connect to the service which is working in the web server which supports Python, the language that is used to implement Twitter Analysis. Our server cannot support Python directly, like most HTTP servers, so we need to make the server suitable for python by using an interface called WSGI stands for Web Server Gateway Interface. It is widely use to make connection between python applications and servers.

4.5 Addressing Key Concern

4.5.1 Persistent Data Management

Normally, persistent data management is a concern that its importance cannot be underestimated. Namely, keeping data, preserve it from corruption, taking precautions against corruption, like having a back-up storage is needed, for systems that has a part of saving data, with databases or files etc. This concern is important since, if the stored data corrupts and that data is needed, whole system can even face a failure. That is why this is an important concern. For our system, we store data of users and their information, for analysis, but only for a very short time, until user exits program.

Our project does not require a database, since data of Twitter change every second, there is no meaning to holding the data that might and most probably will change. Thus, at the moment user want an analysis by logging in with Twitter account of his, data for analysis will be hold by model classes after user exits, data will be gone. Therefore, if data gets corrupted it is not a problem, as it can be taken from Twitter again. This concern is very important but it is not one of our major problems for this system. If we face a problem associated with data management, it would be during the short process of taking data, analyzing and showing results, and it is highly unexpected.

4.5.2 Access Control and Security

Access control and security is one of the major problems of the system that works on a web browser, which is online all the time and anyone can access any time. For example, bank web sites will require a very high access control and security. Access control and security is again needed for protect the system’s data

and itself from outsider dangerous software or prevent any other illegal actions that might harm system or its clients and end users.

For this system has a network connection, it's on a web browser and users log in with their Twitter account. However, our system analyze data that can also be observed from Twitter's site, which is public to everyone. Therefore all those data are actually accessible by everyone on Twitter. We only take that information and analyze it and show it to user again. Therefore access control is not required. For security, we take Twitter login info of users, but that data is not stored and if he exits that info does not exist, basically, so security is also not a major concern for us.

4.5.3 Global Software Control

Our system works on a web browser, that is the why user can use web browsers buttons for going back, refreshing the page or exit the program; there will be alternative buttons to browser's buttons for navigation . Except for that, there is a button that make user to login with his Twitter account. Pick an analysis target, himself or a friend if friend pick the specific friend, by buttons and clicking. And which analysis result to show is with buttons and clicks, as well.

4.5.4 Boundary Conditions

Initialization: To use our system, user must log in with a Twitter account, therefore it is a must to use the system of ours, since we need to know which account to analyze. Then he will choose an analysis among 5-6 analysis for himself or for a friend from Twitter. After that, he can see anything he wants in the analysis.

Termination: Because of the fact that this system's user interface is on the web browser, user can quit any time or go back to the page he was before, with the close or go back buttons of web browser.

Error: User can enter a wrong account name or a password, if that happens user will have wrong password warning.

4.5.5 TwitterApi Limit

Unfortunately, Twitter limits data taking of TwitterApi, with a certain number of requests, then for an API it gives that API a break for 15 minutes. With an average Twitter account's analysis there is no problem, even data may corrupt and needed a retaking from Twitter. However, if a person with a huge number of followers becomes

our analysis target, like a celebrity with one million followers. This might become problematic for us, as, we don't have enough number of access codes, the reason of this situation is each API requires an account in Twitter with a valid phone number. Thus, we may need to make the user wait like 15 minutes or 30 minutes or maybe even more, because we can't take the wanted data with our number of access codes, likewise we cannot have enough number of access code for an analysis target like that with our current budget, which is zero. This is our major concern.

5 Object Design

5.1 Pattern Applications

5.1.1 Facade Pattern

Facade pattern is used in our system to encapsulate more complex classes into a class that can be used in main app directly. Facade pattern provides a clear interface to control the system. Without Façade pattern, it is needed to initialize list of entity objects in main app separately and give them the analyze objects as parameters for getting each statistic.

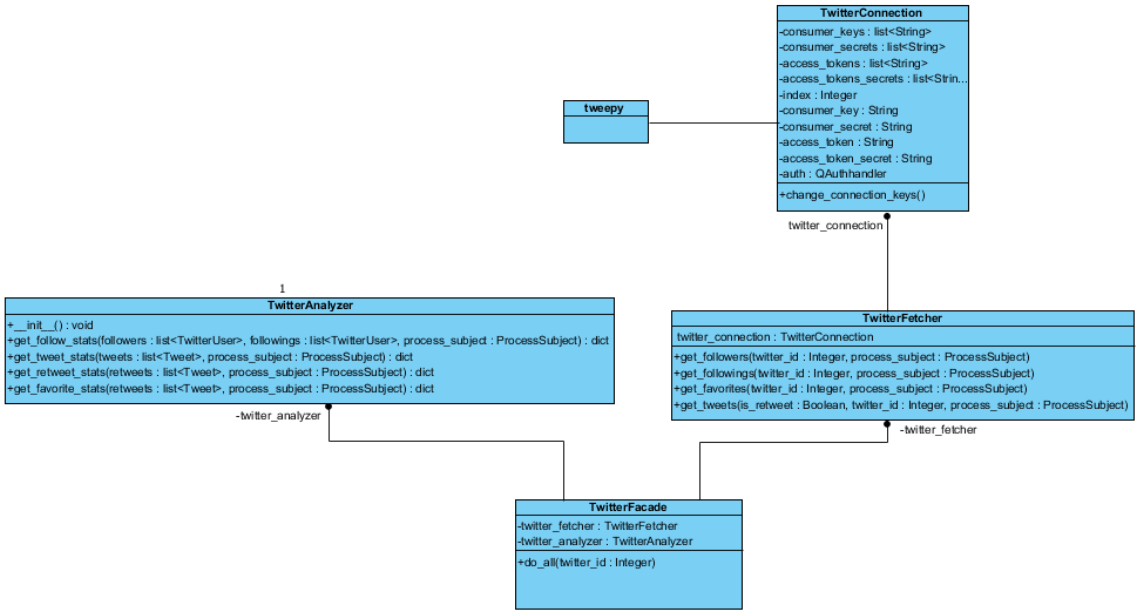

Figure 19: Facade Patter Class Diagram

The system will have a class named TwitterFacade that will have only one function that will get twitter_id to do all operations which are firstly fetching and secondly analyzing and lastly

returning for the main app. It will return a dictionary which contains all statistics and those statistics will be sent to website to be viewed. So the main app will only have one façade object and give it a twitter_id, which might be obtained from a twitter screen name provided by a user; that object will connect all other classes, which Façade Pattern should allow.

5.1.2 Strategy Pattern

Strategy pattern is a pattern required for choices of a system. When a system has to make a choice between two or more different ways according to another feature of program. There needs to have different algorithms for the wanted result in the system. This pattern is also known as policy pattern, since the name suggests choices between algorithms are made based on a policy. For instance, there are different kind of sorting algorithms exist in a program, like bubble sort, quick sort, system has a policy to choose which one based on the data structure's applicability to that sort, as performance, or user's wish or such. Like this the design pattern, strategy (policy) pattern exist in the system.

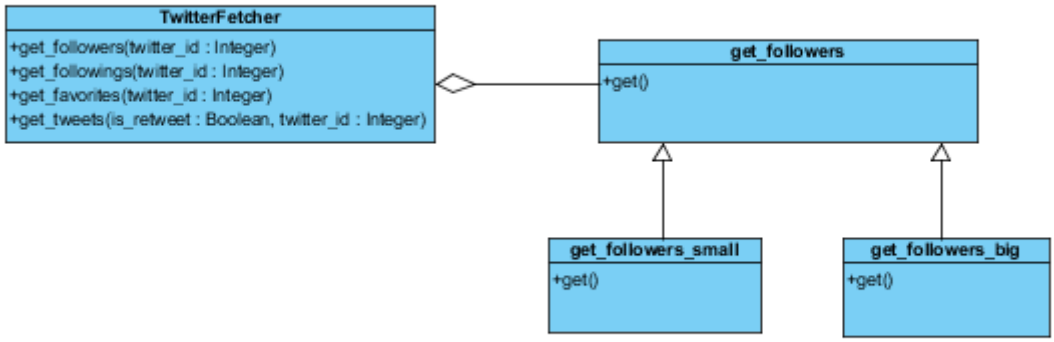

Figure 20 Strategy Pattern Class Diagram

Normally strategy patterns have choice policies between different classes. However, in our system the strategy pattern have choices between functions, in order not to have unnecessary classes in the system. Figure above shows that TwitterFetcher is class and get_followers is a function of that class and get_followers_small and get_followers_big are functions which are a part of the same class and also kind of function get_followers.

We have strategy pattern inside of the get_followers function. The pattern in this function is based on the number of followers the twitterUser has. We need list of the followers and we will have the list by get_followers. However, if the number of followers is too high, an access key is not enough to get all of the followers at once, so we need to get

them separately which can be done manually which is like getting first 20 followers then get next 20 followers. The number of followers can be got from twitterUser object's attribute directly. If number of followers is below 400 system uses get_followers_small function, if it is 400 or above the system uses get_followers_big function. The reason of it is the access key limit we have as an issue when taking data from twitter, if a user has more followers than 400 that will be problematic for us to take all that data, because of Twitter' limit for requests. That is the policy we have to choose, thus this is the strategy pattern in our system, and likewise get-followings have the similar policy pattern based on number of followings.

5.1.3 Observer Pattern

Some systems have information that constantly change and the system needs to keep the system' work the same even though data changes. To observe these changes, system needs a class that that observe those changes and update other related classes and their attributes. This design pattern is called observer pattern. For example, publisher and subscribers are entity objects and if publisher publish something subscribers will get a notification in an application. This notification will be sent via observer patterns, checking changes in publisher's new published product. Or getting a notification when a new e-mail is received is provided thanks to observer class.

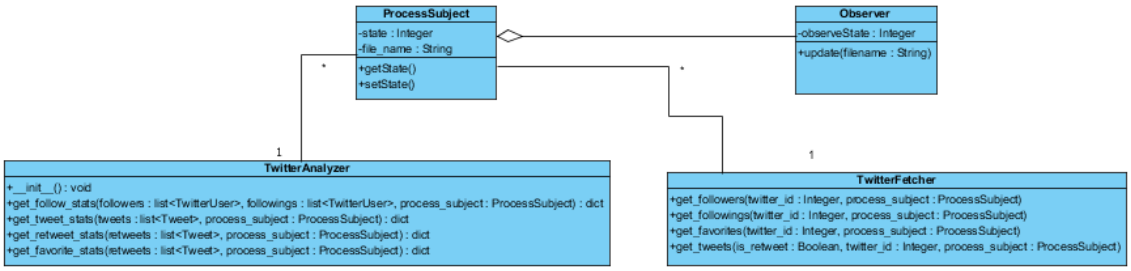

Figure 21 Observer Pattern Class Diagram

ProcessSubject is to check whether analysis is completed and whether an error occurred during the analyzing process or fetching. If an error has occurred, it will update the text file specified for writing process subject states and writing errors that might happen during the process. The text file will be updated after an analysis or fetching is done according to the state attribute in the ProcessSubject object and that attribute will be 1 if the analysis or fetching is completed successfully. Every function in TwitterAnalyzer and TwitterFetcher take an instance of that object as a parameter so every analysis can update the file.

5.2 Class Interfaces

Tweet

Tweet class has 5 attributes, which are source_id (Integer), target_id (Integer), text (String), created_at (Datetime). Its function is __init__(attributes: dict).

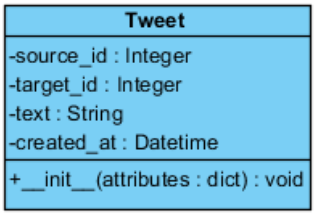

Figure 22 Tweet Class

TwitterUser

TwitterUser class has 6 attributes, which are twitter_id (Integer), username(String), name(String), follower_count (Integer), following_count (Integer), text (String)), profile_picture(String). Its function is __init__(attributes: dict).

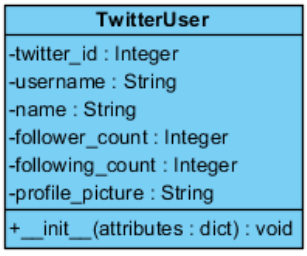

Figure 23 TwitterUser Class

TwitterConnection

TwitterConnection class has 10 attributes, which are the keys and tokens which allows us to take data from Twitter. Its function is change_connection_keys().

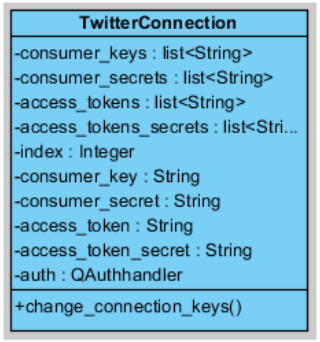

Figure 24 TwitterConnection Class

TwitterAnalyzer

TwitterAnalyzer class has 5 functions, which are __init__(), get_follow_stats(followers: list<TwitterUser>, followings: list< TwitterUser>, process_subject: ProcessSubject), get_tweet_stats(

tweets: list<Tweet>, process_subject: ProcessSubject), get_retweet_stats(retweets: list<Tweet>, process_subject: ProcessSubject), get_retweet_stats(favorites: list<Tweet>, process_subject: ProcessSubject).

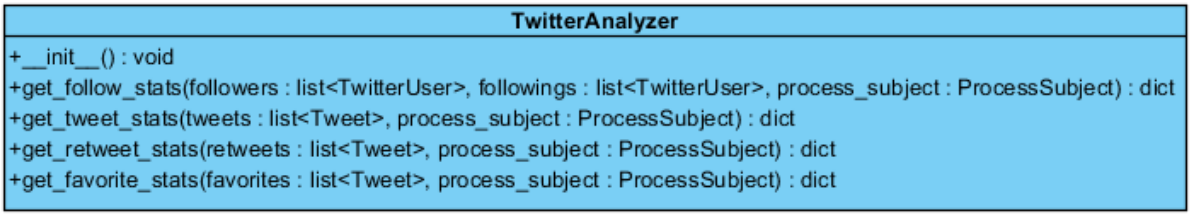

Figure 25 TwitterAnlayzer Class

TwitterFetcher

TwitterFetcher class has the attribute twitter_connection(TwitterConnection) and 4 functions, which are get_followers(twitter_id: Integer, process_subject: ProcessSubject), get_followings(twitter_id: Integer, process_subject: ProcessSubject), get_favorites(twitter_id: Integer, process_subject: ProcessSubject), get_tweets(is_retweet: Boolean, twitter_id: Integer, process_subject: ProcessSubject).

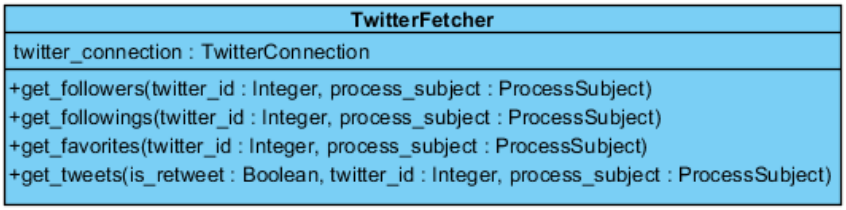

Figure 26 TwitterFetcher Class

TwitterFacade

TwitterFacade class has 2 attributes which are twitter_fetcher (TwitterFetcher), twitter_analyzer(TwitterAnalyzer) and has the function do_all(twitter_id : Integer).

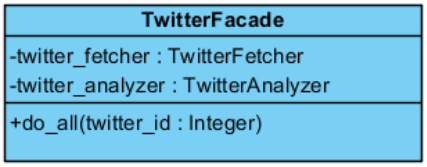

Figure 27 TwitterFetcher Class

Observer

Observer class has one attributes which observeState(Integer) and has the function update(filename : String).

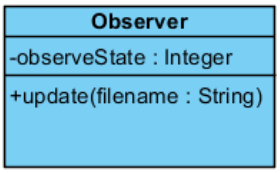

Figure 28 Observer Class

ProcessSubject

ProcessSubject class has 2 attributes which are state(Integer), filename (String) and has 2 functions getState(), setState().

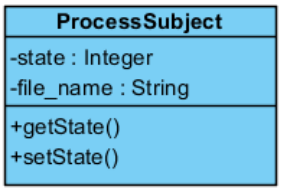

Figure 29 ProcessSubject Class

App

App class is the main class. It has 2 attributes which are app (Flask) and twitter_fecade (TwitterFecade). And has two functions homepage() and statistics_page().

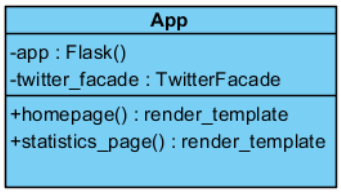

Figure 30 App Class (main)

5.3 Specifying Contracts

TwitterFetcher:

Note: All variables in post parts are in the function do_all() of the class TwitterFacade.

1. **Context** TwitterFetcher :: get_twitter_user(self, twitter_id : Integer, username : String) : TwitterUser

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: twitter_user = get_twitter_user(self, twitter_id : Integer, username : String) :TwitterUser

In order to get a twitter user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. twitter_user object will be initialized by this function.

2. **Context** TwitterFetcher :: get_followers(self, twitter_id : Integer) : list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: followers = get_followers(self, twitter_id : Integer)

In order to get followers of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. followers list will be filled by return data of this function.

3. **Context** TwitterFetcher :: get_followings(self, twitter_id : Integer) : list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: followings = get_followings(self, twitter_id : Integer)

In order to get followings twitter user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. followings list will be filled by return data of this function.

4. **Context** TwitterFetcher:: get_followers_small(self, twitter_id : Integer) : list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: followers = get_followers_small(self, twitter_id : Integer)

In order to get followers by using this method of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. followers list will be filled by return data of this function.

5. **Context** TwitterFetcher:: get_followers_big(self, twitter_id : Integer) : list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: followers = get_followers_big(self, twitter_id : Integer)

In order to get followers by using this method of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. followers list will be filled by return data of this function.

6. **Context** TwitterFetcher:: get_followings_small(self, twitter_id : Integer) :list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: followings = get_followings_small(self, twitter_id : Integer)

In order to get followings by using this method of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. followings list will be filled by return data of this function.

7. **Context** TwitterFetcher:: get_followings_big(self, twitter_id : Integer) :list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: followings = get_followings_big(self, twitter_id : Integer)

In order to get followings by using this method of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. followings list will be filled by return data of this function.

8. **Context** TwitterFetcher:: get_favorites(self, twitter_id : Integer) :list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: favorites = get_favorites(self, twitter_id : Integer)

In order to get followings by using this method of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. favorites list will be filled by return data of this function.

9. **Context** TwitterFetcher:: get_favorites(self, twitter_id : Integer) :list<TwitterUser>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: favorites = get_favorites(self, twitter_id : Integer)

In order to get followings by using this method of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. favorites list will be filled by return data of this function.

10. Context TwitterFetcher:: get_tweets(self, is_retweet, twitter_id): list<Tweet>

pre: TwitterFetcher.checkUserValid(self, twitter_id : Integer, username: String) :Boolean

post: tweets = get_tweets(self, is_retweet, twitter_id):
retweets = get_tweets(self, is_retweet, twitter_id):

In order to get retweets or tweets by using this method of a user, a user specified with the id or username must be valid in Twitter, if the user does not exist in Twitter, no data can be fetched. retweets and tweets list will be filled by return data of this function.

TwitterAnalyzer:

11. Context TwitterAnalyzer:: get_intersection (self, followers, followings): Integer

pre: TwitterFetcher :: get_followers(self, twitter_id : Integer) : list<TwitterUser>
TwitterFecther :: get_followings(self, twitter_id : Integer) : list<TwitterUser>

In order to perform get_intersections function, its parameters which are followers and followings must be filled by functions belonging to TwitterFetcher.

12. Context TwitterAnalyzer:: get_post_days (self, tweets): List<Integers>

pre: TwitterFetcher :: get_tweets(self, twitter_id : Integer) : list<TwitterUser>

In order to perform get_post_days function, its parameter which is tweets must be filled by functions belonging to TwitterFetcher.

13. Context TwitterAnalyzer:: get_post_hours (self, tweets): List<Integers>

pre: TwitterFetcher :: get_tweets(self, twitter_id : Integer) : list<TwitterUser>

In order to perform get_post_hours function, its parameter which is tweets must be filled by functions belonging to TwitterFetcher.

14. Context TwitterAnalyzer:: get _favorites (self, favorites): Dictiobary

pre: TwitterFetcher :: get_favorites(self, twitter_id : Integer) : list<Tweet>

In order to perform get_favorites function, its parameter which is favorites must be filled by functions belonging to TwitterFetcher.

15. Context TwitterAnalyzer :: get _retweets (self, retweets): Dictiobary

pre: TwitterFetcher :: get_retweets(self, twitter_id : Integer) : list<Tweet>

In order to perform get_favorites function, its parameter which is retweets must be filled by functions belonging to TwitterFetcher.

TwitterConnection:

16. Context TwitterConnection :: change_connection_keys (self): void

pre: TwitterConnection: __init__(self) : void

In order to perform change_connection_keys, an istance of TwitterConnection must be initialized by the constructor of the class.

TwitterFacade:

17. Context TwitterFacade :: do_all(self, username):

post: analyses = results : Dictionary

This method fills the analyses variable which contains results of the analysis, and the analyses variable goes to the website.

TwitterUser:

18. Context TwitterUser :: setTwitter_id (self, twitter_id: Integer): void

post: self.twitter_id = twitter_id : Integer

This method sets the twitter_id variable to Integer parameter.

19. Context TwitterUser :: setUsername (self, username: String): void

post: self.username = username : String

This method sets the username variable to String parameter.

20. Context TwitterUser :: setName (self, name: String): void

post: self.name = name : String

This method sets the name variable to String parameter.

21. Context TwitterUser :: setFollower_count(self, follower_count: Integer): void

post: self.follower_count = follower_count : Integer

This method sets the follower_count variable to Integer parameter.

22. Context TwitterUser :: setFollowing_count(self, following_count: Integer): void

post: self.following_count = following_count : Integer

This method sets the following_count variable to Integer parameter.

23. **Context** TwitterUser :: setProfile_picture (self, profile_picture: String): void

    post: self.profile_picture = profile_picture: String

    This method sets the profile_picture variable to String parameter.

Tweet:

24. **Context** Tweet :: setSource_id(self, source_id: Integer): void

    post: self.source_id = source_id: Integer

    This method sets the source_id variable to Integer parameter.

25. **Context** Tweet :: setTarget_id(self, target_id: Integer): void

    post: self.target _id = target_id: Integer

    This method sets the target _id variable to Integer parameter.

26. **Context** Tweet :: setText(self, text: String): void

    post: self.text = text: String

    This method sets the text variable to String parameter.

27. **Context** Tweet :: setCreated_at(self, created_at: Datetime): void

    post: self.created_at = created_at: Datetime

    This method sets the created_at variable to Datetime parameter.

28. **Context** Tweet :: setIs_retweeted(self, is_retweeted: Boolean): void

    post: self. is_retweeted = is_retweeted: Boolea

6. Conclusions and Lessons Learned

The project is conducted with a great risk from the beginning: we tried to improve an efficient analysis program, Twitter Analysis. Since there are plenty of analysis programs on the market, our project had the risk of repeating what is already done and considered as imitation. To prevent that, our project group mostly focused on different analysis types. It can be said that previous examples of this kind of program lacked efficiency and interesting results. The algorithm of the program was hard to design since it must be a correct analysis tool. At this point, existing programs are studied and their weaknesses tried to be avoided.

The project was a team work practice, after 2 years of studying computer science, it was a good opportunity to share and communicate with others. Like other group works, our also required participation and dedication which we did our best to fulfill.

The design process was pretty tough since it is first time we encounter this strict design process. Key concepts and algorithm took some time but it was worth it. The biggest challenge, in these terms, was to understand the concepts and applying them in implementation simultaneously, without any prior experience. It would be more convenient to exemplify the key concepts with real life project.

Reports and documentations were useful in many ways. However, it was a big challenge to achieve them on time before each deadline. We realized that designing phase is way more important than implementing phase.

Some patterns are learned in class and they were very useful when we formed the documentations. After design process, it was easier to implement rather than only implementing. Anyone who wants to continue our project, can follow the reports and add new features desired.

When, we decided to do this project we should have to examine the capabilities of the libraries and the Twitter API, because we had difficulties fetching the data we need. We needed to eliminate some analyses and statistics since Twitter limits the data. At the beginning we wanted to draw social cycle of the user, but it is needed to fetch followers and followings of the users' followings and followers to do that. We needed hundreds of access keys to do that, thus we decided not to draw social cycle.

We created about 30 classes at the beginning of the project to perform our functionalities, however after meeting with the TA, we reduced the number of classes. We could perform everything we did with lesser classes which provides us simplicity and regularity. Namely, reduction of the classes helps us to write clearer and simpler reports, likewise our system became easier to deal with.

Eventually, the project was a hard and also very educating experience. Project development cycle and individual processes are clearly learned with a team work together. The significance of documentation and applying patterns to easier solutions were key concepts of the project. Last but not least, we realized the interaction between developer and client is much more important than any other phase of project.