Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering

CS 319 Project: Instagram Data Analysis Application

Design Report

Orçun Gümüş
Cansu Demiryürek
Nihan Varilci
Melih Ekmekçi

Course Instructor: Hüseyin Özgür Tan

October 28, 2015

Table of Contents

Table of Figures

1. Introduction

Instagram Data Analysis Application is a data analyzing program which can make data analysis and draw graphs among the user interactions. It is a web scale data operation and the data are provided by the Instagram API. In this program a user can see his/her social Instagram environment such as people from high school, university friends, work friends etc. They are going to be clustered and easily identified. In addition to this, they can also get information about whom like their photo regularly and who don't. This will show who the most popular one among friends is in terms of the follower number and the amount of like they get. At the same time people can easily identify who are close to each other and interested in their photos. Users' can do this analysis for themselves and at the same time if they are curious about anyone else, they can get the same information for others. To make this operation, you have to log in to our program with your Instagram account. After this process, users' data will be automatically saved to the database and the analysis will be shown by a graph. In this graph, nodes are set as Instagram users, follows and likes are considered as edges.

In today's world, people are so curious about everyone's social environment and instead of what they share; they are mostly interested in who they know and how close they are. There are some mobile applications that just show their own data about whom they follow and who gave up following them. By Instagram Data Analysis Application without knowing any of the people,

users will be able to know which society they belong to and can know what kind of people they are.

2. Requirement Analysis

2.1 Overview

In Instagram Data Analysis Application several functional requirements, nonfunctional requirements and constraints will be included. After the introduction part, this part will start to give more specific information about the program such as functionalities and technical information. Possible scenarios while using the program will be shown. In addition to this, they will be explained by the use case model and in the end the user interface will be demonstrated.

2.2 Functional Requirements

1) Help: Users are able to get all kind of information about application as an instructive explanation.

This instructive explanation contains the information about:

- How to use this application.
- What is the advantage of this application
- How to interpret graphs
- How to save the graphs.

This help document mainly provides Users a chance to learn the process of this application. To use this application, users have to have Instagram account. Moreover, this application works only with Instagram.

2) Login: This function is required and the most basic feature of the application for the users. Firstly, the user fills the user name and password boxes, these boxes include Instagram username and password. Then user should press the login button to access the main page of the application. If user has not got Instagram account, he/she must sing in new account from Instagram.

3) Showing Follow Analysis

When users login the application, they see main page. Main page includes flows analyze button. Users can make an analysis over their follows. They can reach their follows graph and see their friends' communities via their follows information

4) Like Analysis

When user login the application, they see main page. Main page includes like analyze button. Users can make an analysis over their likes. They can reach their like graph and see their like communities via their like information

5) Showing Graph & Offering Suggestion

This application's aim is to produce graphs. Graphs show the users communities by like or follow. These communities include users' groups of various friends, their relatives and coworkers… etc. Moreover, users can see people who are more closed to them.

Users can reach people who are closed. According to graphs, this application makes suggestions to users about people who users want to follow. Analyzes produces graphs but system is always saving the graphs. They can reach their graphs whenever they want.

5) Saving Downloaded Items from external API in Database

When user login to program and produce a graph, their all information (follows, likes) saved in a database. Users can access these graphs whenever they want to look in the future. Moreover, system is also using this information. During analyses system is always controlling the database first. If the required item is existing in the database, the system is not making any external API request on redundant situations.

6) Changing Settings

Setting is the part of the main menu. It includes profile picture and background. When users click the setting button, setting page is opened. Users can choose picture from his or her photo library as profile picture. Moreover, users can choose background for application.

7) Logging out

When user wants to exit the application, he or she can click the logout button. Log out button will delete all temporarily view information.

8) Deleting account

User can delete their account from system. However system will delete only the information related to this user.

2.3 Non Functional Requirements

Usability

Without creating a new account user can directly continue with their Instagram Account. Users do not need to enter their user name and password due to their data will be saved.

Performance

When users start to analyze their account or any other one, it will be completed less than in 5 minutes.

Security

Only users can see their analysis and except the authorized user, no one can reach the analysis report. Analysis report privacy for each user will be provided.

Supportability

This program will work in all the internet browsers and graphs can also be shown in every browser. For the mobile side of this program, graphs will not be seen. Only the interaction numbers will be available.

2.4 Constraints

Planned server provider is Amazon Web Services.

AWS will provide elastic beanstalk.

AWS will provide distributed and increasable size relational database system.

AWS will provide scalable computable power for analyze computations.

Files will have distributed over machines thanks to S3 service of AWS.

Required RAM, Hard Drive Space and computational power will be handled by AWS.

Python 3.4 will run on the server.

MySQL 5.6 will exist as RDBS.

PHP 5.5 will accept web page requests.

NoSQL (Dynamo DB)

2.5 Scenarios

Scenario Name: Occurring Instagram Data Analysis Application

Melih Gökçek is phenomenon of social media such as Instagram. Melih Gökçek wants to determine friends who are closed than others because although he has lots of followers, he feels alone. He needs a program that is able to draw a graph. This program takes data from Instagram and creates community according to likes and followers. Melih shares this desire with us. We occur Instagram data analysis application and thus, when Melih looks graph, which friends are close, which communities have follows and likes. When we present this application, Melih could not know to use this application. Fortunately, there is help button. When Melih enter to use this application, he clicks the help button. Therefore, he did not have difficulty in this application. First of all, he logins this application with Instagram user name and password. He can observe these graphs and he reached real friends and communities.

Scenario Name: Who likes my photos more?

Meliz Ozenat is a college student. He has a Instagram account. Every time he has uploaded any photo his friends started to like them. He has hundred averages like on his photos. He is really curious about finding the most liker among his friend. He login with his Instagram account to the application and make a like analyze request. After 5 min, he can now see his best fans from his school and family. Thanks to see like communities. After that, he log off from the application.

Use Case Description

Use Case 1:

Use Case Name: Firstly, Use Application

Participating Actors: User

Pre-condition: User must not be using this application before and have an Instagram account.

Entry condition: User presses “Login” button in the main page.

Exit condition: User presses “Logout” button in the main menu.

Main Flow of Events:

1) User press the help button and read the information about the application.

2) User fills the user name and password boxes according to his or her Instagram account.

3) After filling boxes user press “Login” button.

4) User can see his or her follower numbers in the main page.

5) User can look the graph which is about his or her followers.

6) User can save his or her graph for future.

7) To exit the program user press “Logout” button.

8) System goes to Step 2.

Use Case 2:

Use Case Name: Use Application

Participating Actors: User

Pre-condition: User must have an Instagram account.

Entry condition: User presses “Login” button in the main page.

Exit condition: User presses “Logout” button in the main menu.

Main Flow of Events:

1) User fills the user name and password boxes according to his or her Instagram account.

2) After filling boxes user press “Login” button.

3) User can see his or her like and follower numbers in the main page.

4) User can look the graph which is about his or her likes and followers which he or she saved before.

6) To exit the program user press “Logout” button.

7) System goes to Step 2.

Use Case 3:

Use Case Name: Change Setting

Participating Actors: User

Pre-condition: User must have an Instagram account.

Entry condition: User presses “Login” button in the main page.

Exit condition: User presses “Logout” button in the main menu.

Main Flow of Events:

1) User presses the help button and read the information about the application.

2) User fills the user name and password boxes according to his or her Instagram account.

3) After filling boxes user press “Login” button.

4) User can see his or her like and follower numbers in the main page.

5) User can look the graph which is about his or her likes and followers.

6) User can save his or her graph for future.

7) User press “Settings” button in the main menu.

8) User can change his or her profile picture from photo library.

9) User also changes the background of application.

10) To exit the program user press “Logout” button.

11) System goes to Step 2.

2.6 Use-Case-Model

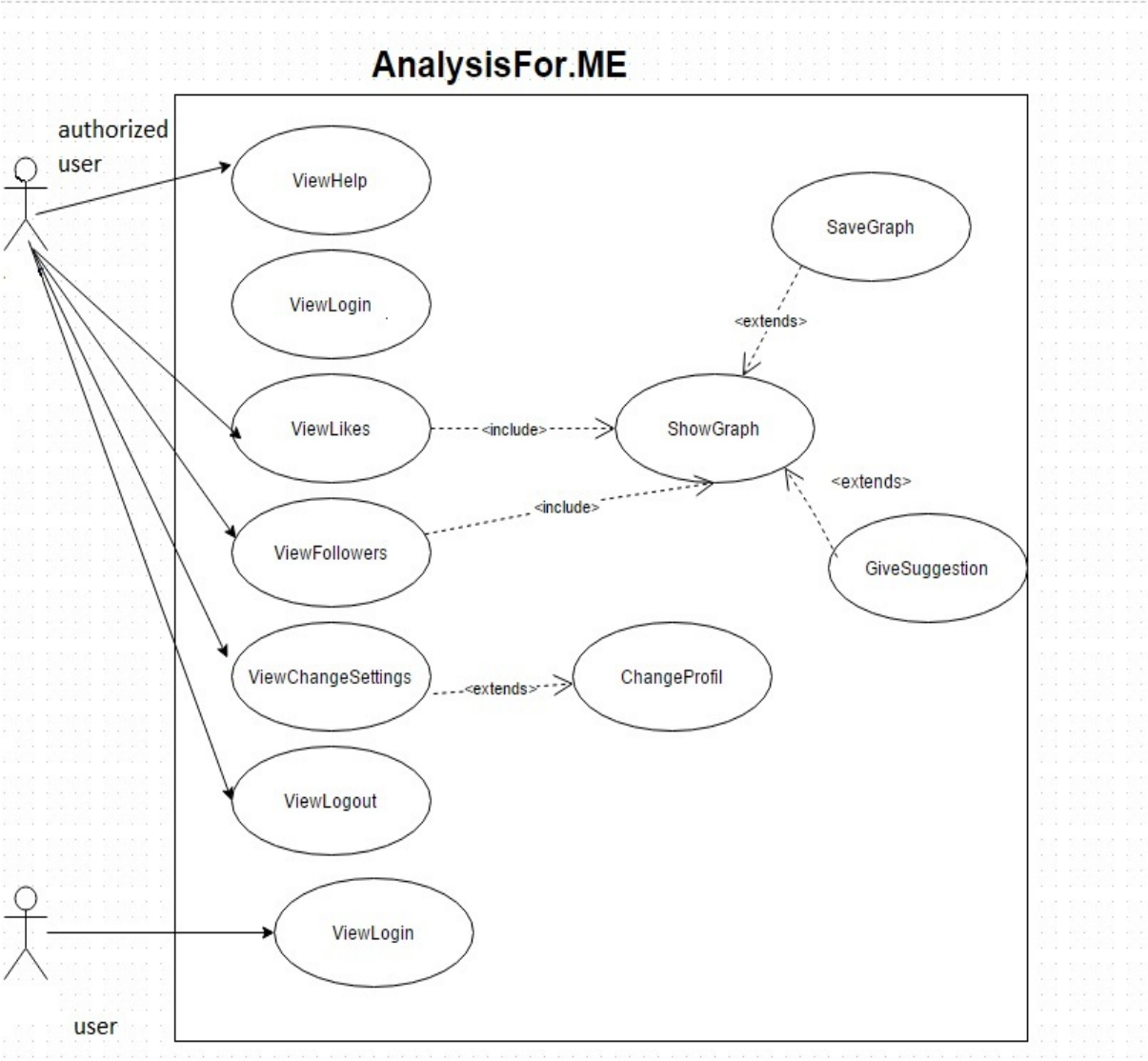

Figure 1: Use-Case Diagram

2.7User Interface

Figure 2: Welcome Page

User will be welcomed with a welcome page. We will basically give information about the application in this section.

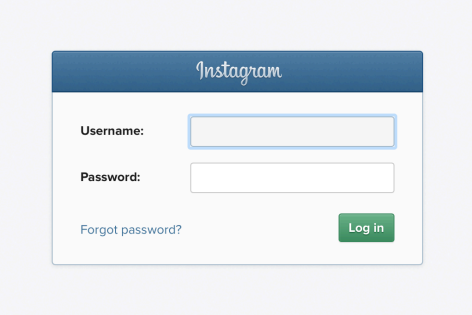

Figure 3: Login Page

User will login with a Instagram account to this application so login page will be provided by Instagram.

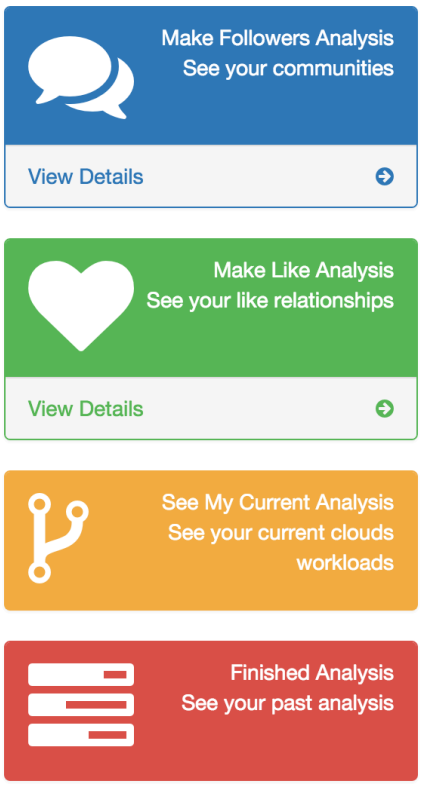

Figure 4: Job Selecting Page

User will select his analysis through a main menu. There will be always a possibility to see past analysis.

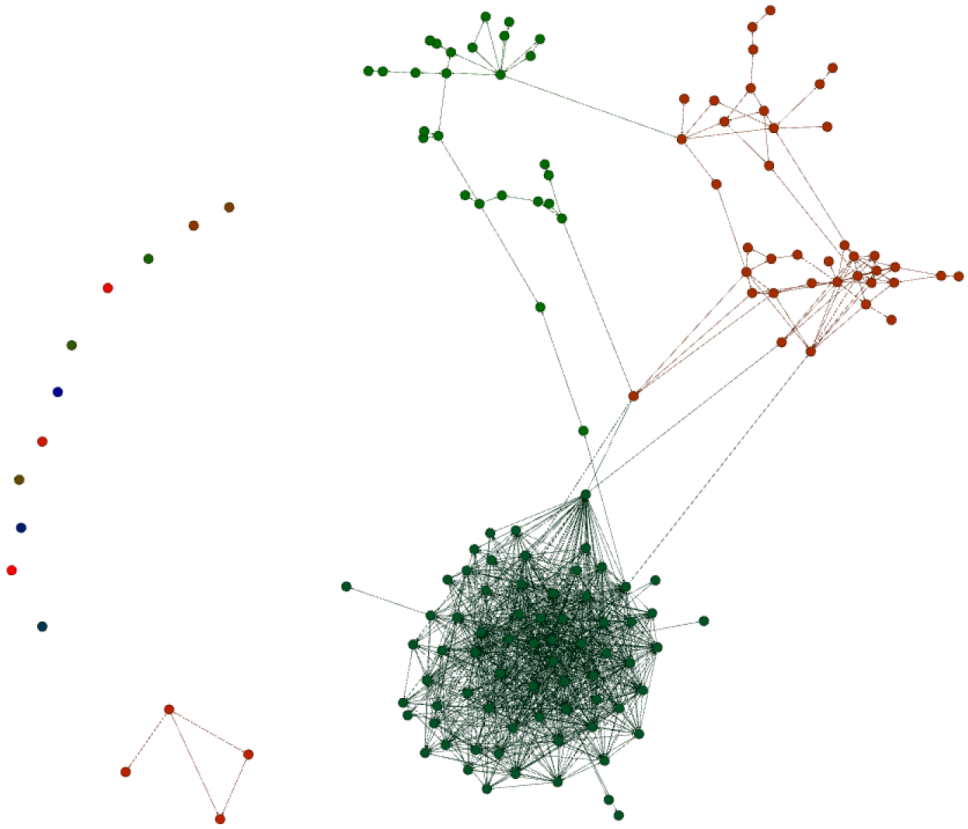

Figure 5: Follow Graph Example

Follow graph of follow information. User will see directly the communities that the person has. The graph will be drawn by ForceAtlas2 in server side and will be motionless in browser side but nodes will be clicked.

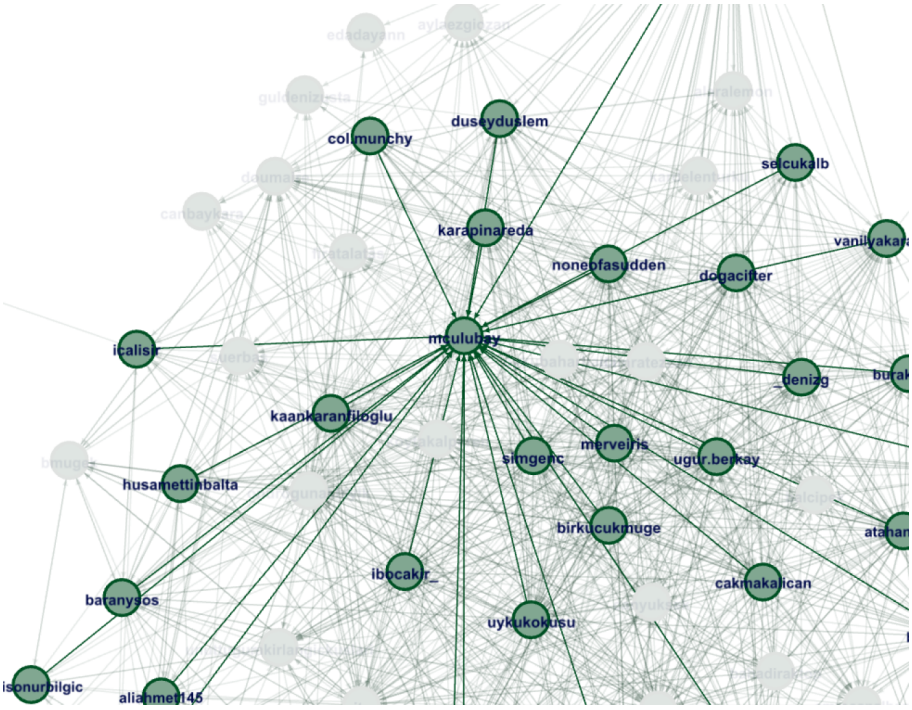

Figure 6: Follow Graph Zoomed, Focused to A User

When a node pressed, it will be possible to see the edges of this person more clearly.

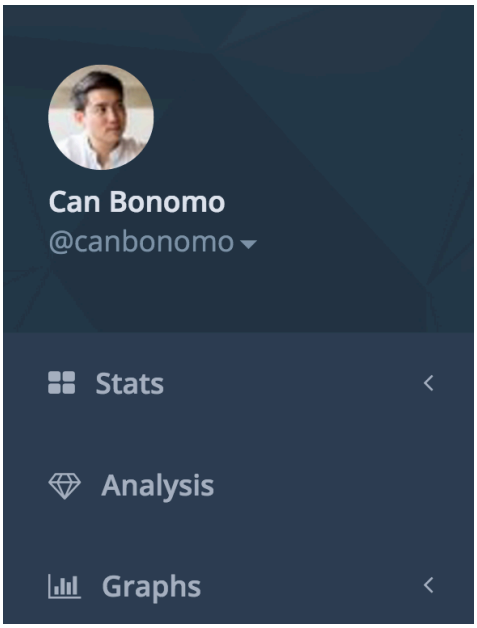

Figure 7: Sidebar Example

Sidebar will enable to user to jump over sections directly.

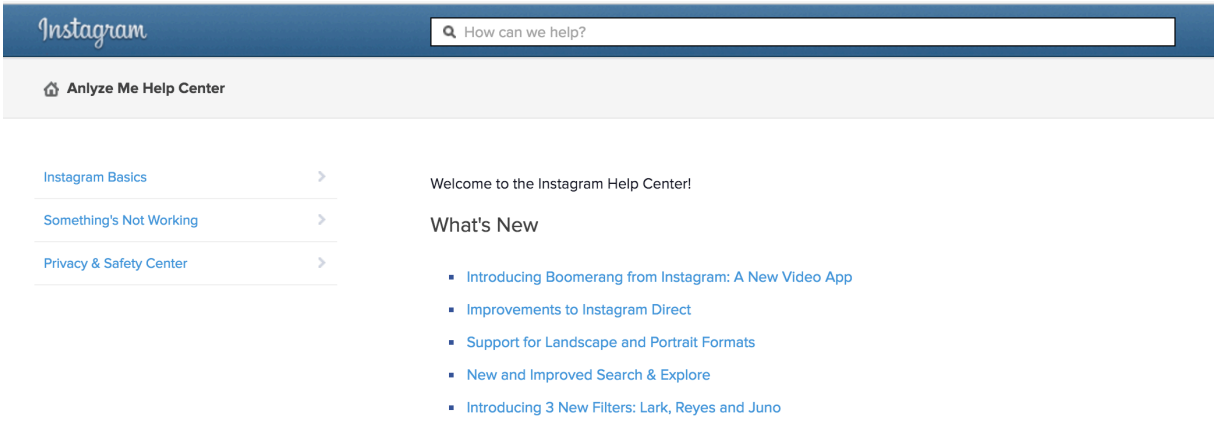

Figure 8: Help Section

Help section is a HTML page to help users to understand better application dynamics.

3. Analysis

3.1 Object Model

3.1.1 Domain Lexicon

SocialBaseObject: All database tables will have class equivalent in application. All columns of database tables will exist as attributes in python application. SocialBaseObject will be the parent class for all of them, and SocialBaseObject controller classes will have the ability of getting, inserting, modifying of SocialBaseObject. Thanks to this plan application will not have different controllers for all of the SocialBaseObjects.

User: A person who logs in the program and able to do his or her analysis. It will have access token to be able to make interaction with Instagram API.

InstaUser: A person who has profile on Instagram application. It has username, id, profile_picture etc.

Analyze: A standard analysis object which is the child of SocialBaseObject. It will hold the user id which wants to make this analysis, request time and etc.

FollowAnalysis: One of the analysis types. It is one of the children of the Analyze Class. This is only the ticket for the analysis. It is holding the analysis type, analysis purpose user, specific analyze properties.

InstaRelation: Child of SocialBaseObject. It is holding source_id and target_id of a following event.

InstaLike: Child of SocialBaseObject. It is holding source_id and target_id of a like event and also media_id of the liked media.

InstaUserGroup: It holds list of InstaUser but also it has attributes about gender information about the InstaUser group that it has.

Follows: Child of InstaUserGroup. Like its parent it has a list of InstaUser and information about genders. However, it has also attributes about follow.

Our class model divided in to three main parts. Main parts are subsystems which are named as InstaModel, InstaController, InstaView:

3.1.2 Class Diagrams

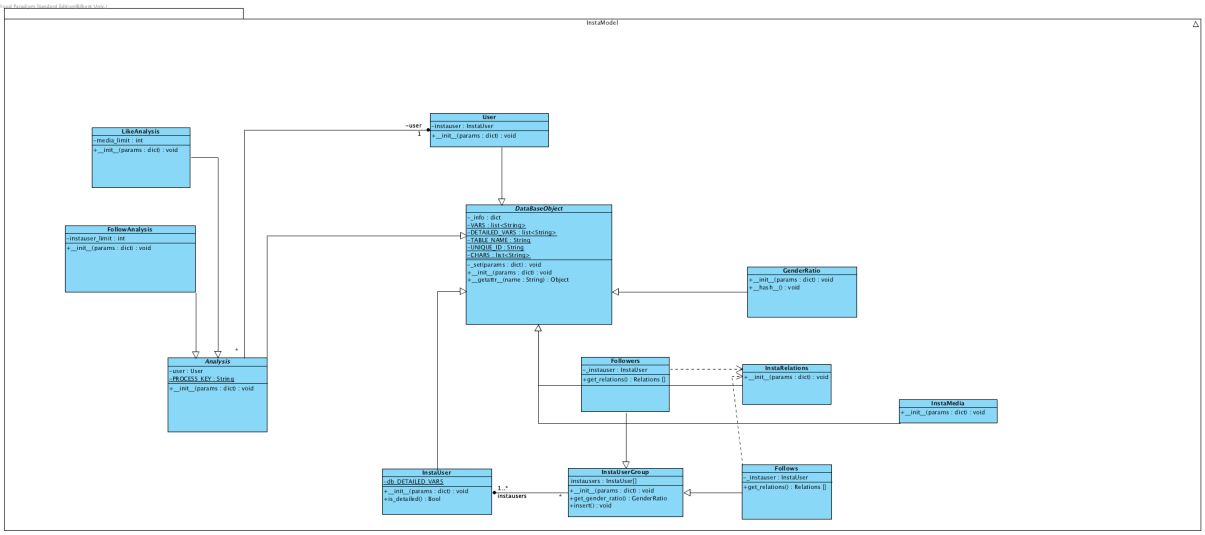

Diagram 1: Model Sub Package Class Diagram

SocialBaseObject which is common parent for model classes. All attributes can be accessible and settable by getter and setter which are not shown in UML diagram.

All attributes of model classes is saved in a –info attribute which is a dict(Python base class for Hashed Map).

For example, InstaUser will have

- 'instauser_uid',
- 'username',
- 'profile_picture',
- 'full_name',
- 'gender',
- 'is_public',
- 'total_followers',
- 'total_media',
- 'total_follows',
- 'is_loaded',

- 'total_media',
- 'total_followers',
- 'total_follows',
- 'follows_gr_uid'

These attributes will be saved inside *info* attribute like:

Example info attribute: {instauser_uid: 123123, username: ‘aylakadam’, profile_picture: ‘http://…/’, full_name: ‘Ayca Abancı’, gender: ‘f’, total_media: 56…}

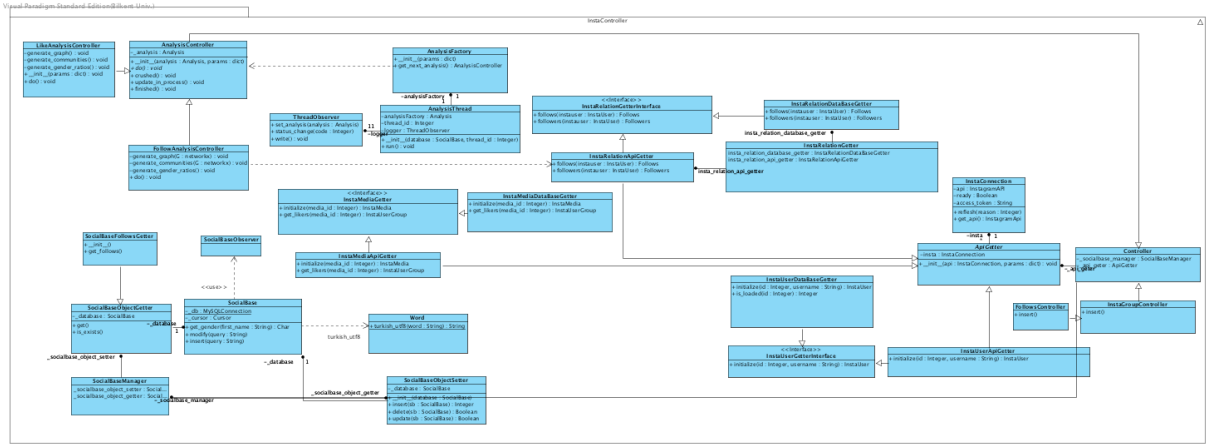

Diagram 2: Controller Sub Package Class Diagram

Controller objects are responsible for fulfill model objects by getting information from API, getting information from database or inserting them (SocialBaseObjects) to database.

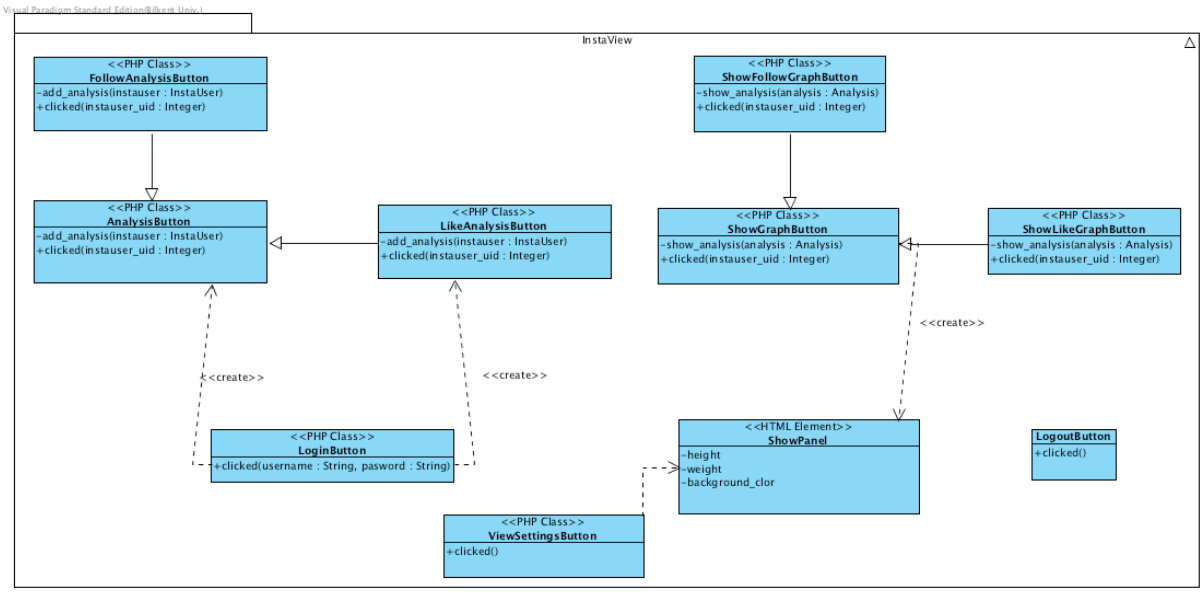

Diagram 2:View Sub Package Class Diagram

3.2 Dynamic Models

3.2.1 State Chart

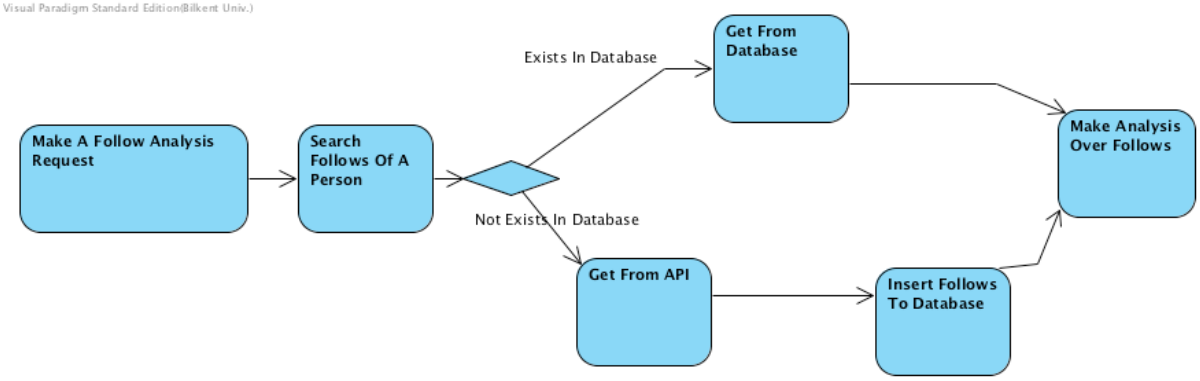

Diagram 3: Activity Diagram of Information Retrieval

In the server side Python modules is always working and idling by waiting any analysis request. If any request comes in database, one of the thread is taking it (by locking it for inhibit concurrency errors) and start an AnalysisController over the analysis. It can be understood by Diagram 4 that when it finished the

analysis. It runs finished method over the controller and seeking for any new analysis.

3.2.2. Sequence Diagram

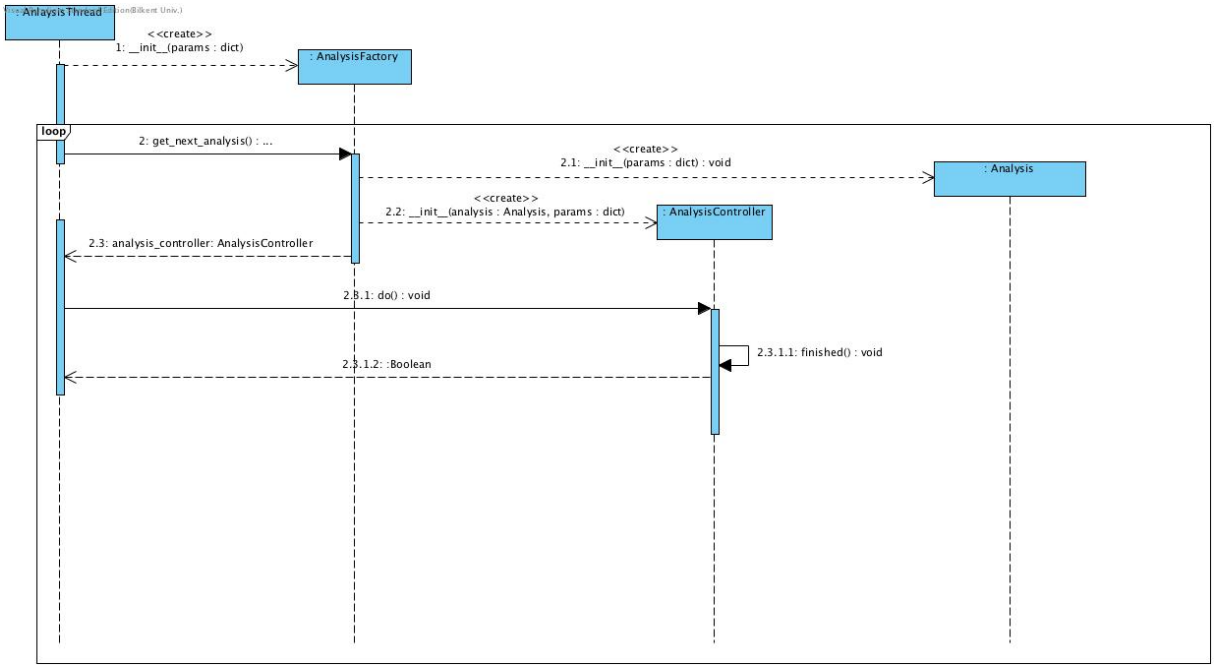

Diagram 4: Sequence Diagram in Server Side after received any Possible Analysis Request from User Interface

The application will prefer to use the information in the databases because of nonfunctional requirements. Therefore, SocialBaseControllers always search over the database before they do a request from the external Instagram API.

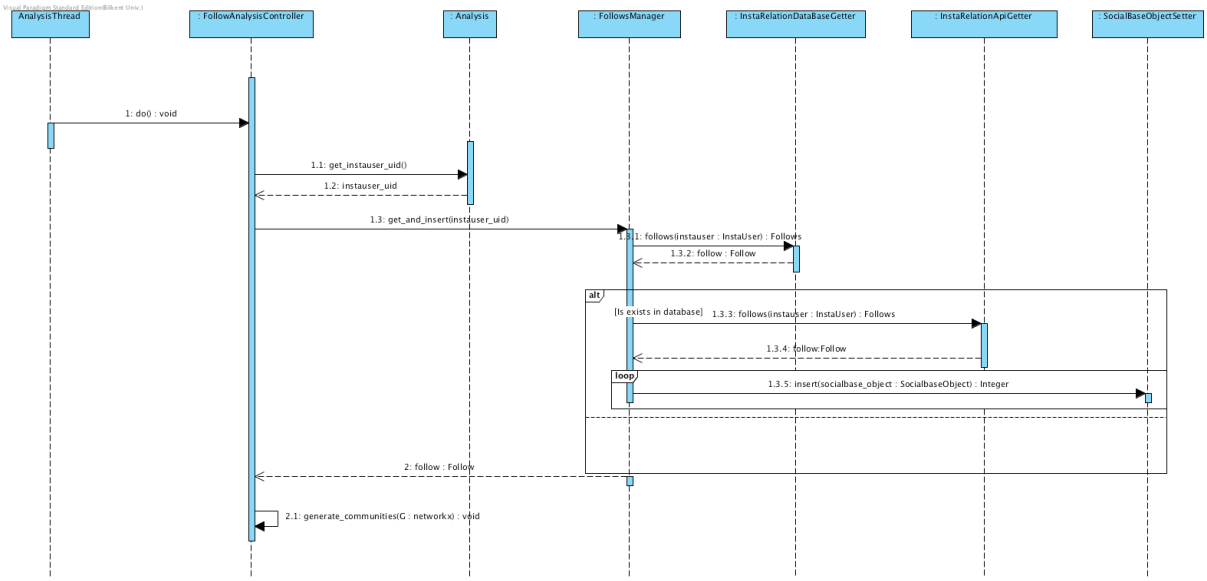

Diagram 5: Follow Analysis is Processing By Application

This behavior of application can be shown as a sequence diagram (Diagram 6). One of the analysis diagram calls do method of an AnalysisController. FollowManager is responsible for information retrieval and storage of follow information.

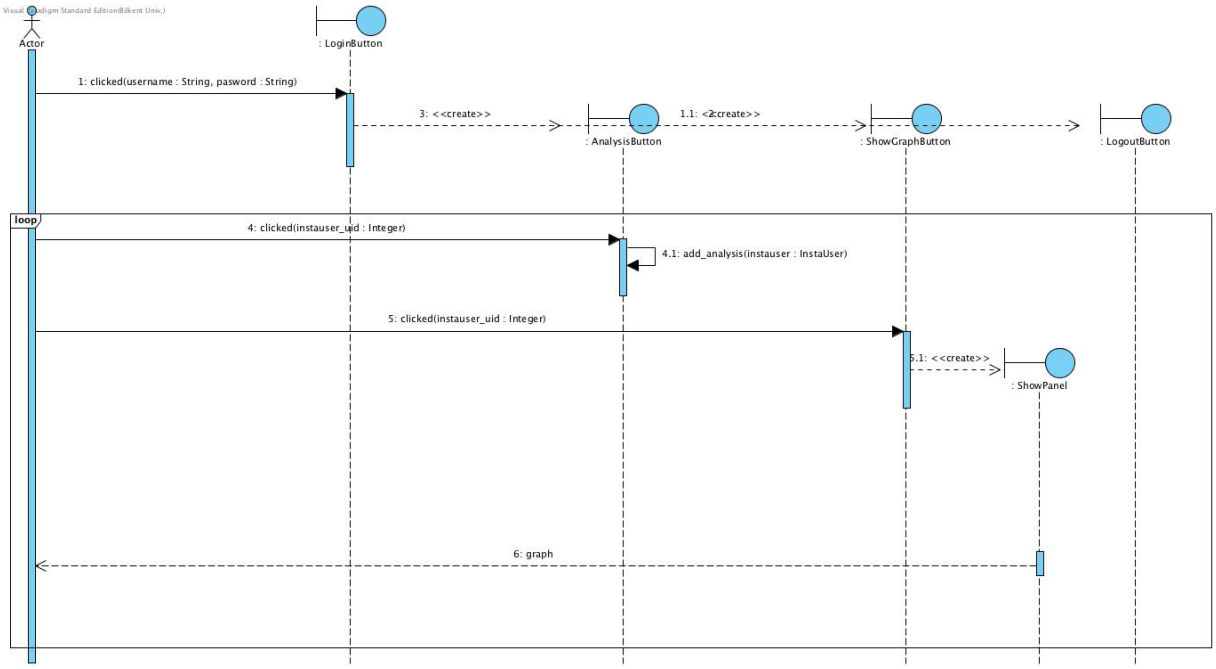

Diagram 6: Sequence Diagram for Standard Analysis Request from User Interface

User interface will contain login button. After pressed login button with correct authorization of username and password, the page will change to a main menu. Main menu have analysis button and show graph button with a panel and logout button.

4. Design

4.0 Introduction to Design

Our project is an Instagram Data Analysis Application (analyzefor.me) in other words a data-analyzing program. The purposes of this system make data analysis and draw graphs among the user interactions for every social media user. Instagram Data Analysis Application uses data form Instagram API. First of all, users have to log in to our program with their own Instagram account. After login, users' data will be automatically saved to the database. With the

information of database, our application occurs graphs. Thanks to graph, users are able to see social Instagram communities such as people from high school, university or work life. If the number of likes and follows increases, graphs have more edges. This application shows whom most popular one among friends in terms of the follower number and the amount of like they take. As a result of this, this system show user's social environment with graphs.

4.1 Design Goals

End User Criteria:

Ease of Learning: analyzefor.me has designed to provide an easy environment to do social media analysis. Our program's aim is to reach lots of people from young to adults. Understandability is the most important for program. Therefore, the system will be designed such that users are able to use application easily without any previous information. We want to provide fast learning process from users. If users cannot understand the system, we put the help button in main menu to users and thus users can use this program without any difficulties. The program includes a well-documented and tutorial to become understandable. In short, this application is user-friendly.

Utility: analyzefor.me has designed to provide a practical environment to do basic social media analysis for everyone. There is not any competitive application in the market. According to Lada Adamic from Michigan University (Adamic 2015), social media analysis for a individual can provide very useful information about her private life. Getting processed information from platforms such as Instagram, Facebook, Twitter can affect people positively. For instance, in our application (Instagram Data Analysis Application) users can do complex personal analysis over their network.

Extensibility of usage: Extensibility is important to provide continuity of application because users lose interest on application after a while. Our

application prevents this situation and provides to keep the interest alive. In terms overcame of this situation:

- Analyzfor.me is capable to many different analyses but they are not able to be chosen from the beginning. They are becoming available with time.
- End users can make analysis for their friends also – this section limited with a daily quota.

User side security: For analyzefor.me, people have to login with Instagram accounts. Users data will be saved database and the analysis will be shown with graph. One user's data which be saved by database cannot reach another user's data. Unauthorized users can not know other user passwords or nicknames.

Dependability Criteria:

Robustness: Our application is prepared with a variety of internal control points so as to determine unwanted problems. It is expected that analyzefor.me will be capable to handle with very popular persons by warning them or zero friend users. So it is expected that analyzefor.me will survive even after getting these extreme users.

Security: The system must be a secure from users' perspective and also for developer side. The security is fundamental stone for an application because system should not un wanted data violation by attackers.

Amazon web services provide cloud security. Different application parts have different accessible ports. For example for analyzefor.me there exist 3 different secure zones

- Databases

- Web Servers
- App Servers (Background Workers)

They will all have different accessible ports.

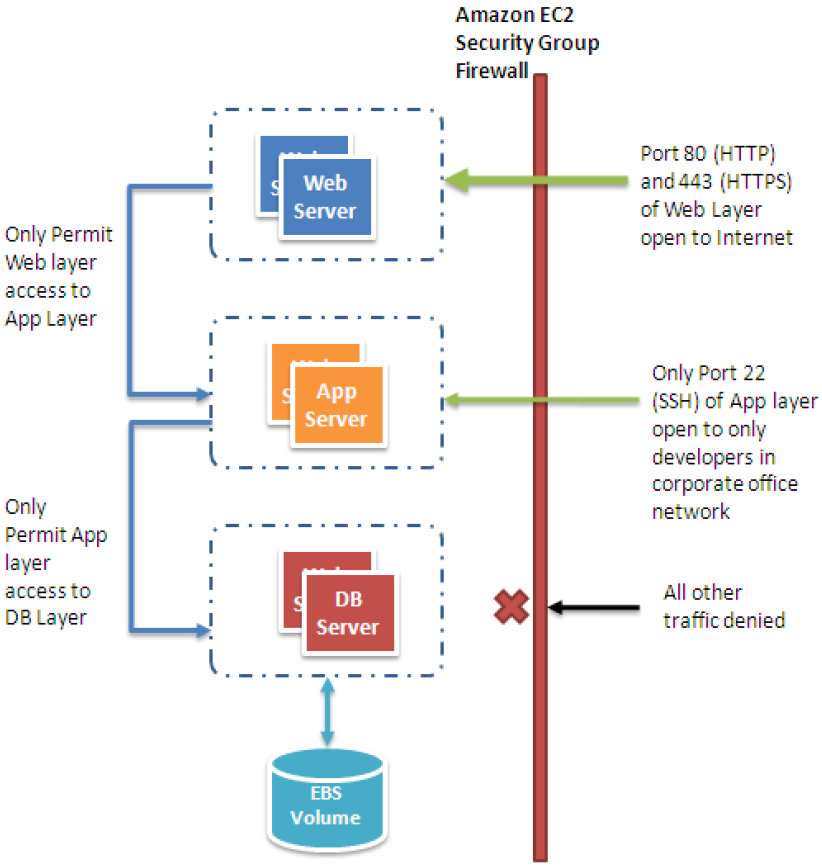

Figure 9: Security over ports

Reliability: analyzefor.me will provide same outputs for same analysis with the same data. Randomness in analysis algorithms will be excluded to make feel more reliable. End users will be able to see their expectations after their analysis selections.

Fault tolerance: analyzefor.me will deployed in Amazon Web Services with fault tolerance option. AWS provides a platform that is ideally suited for building fault-tolerant software systems. When a server crashes or a hard disk runs out of room in an on-premises datacenter environment, administrators are notified immediately.

- Crushed machines is being restarting automatically. (Amazon 2015)

- Amazon web services are capable to distribute domain to different IPs other than the crushed server. (Figure 1)

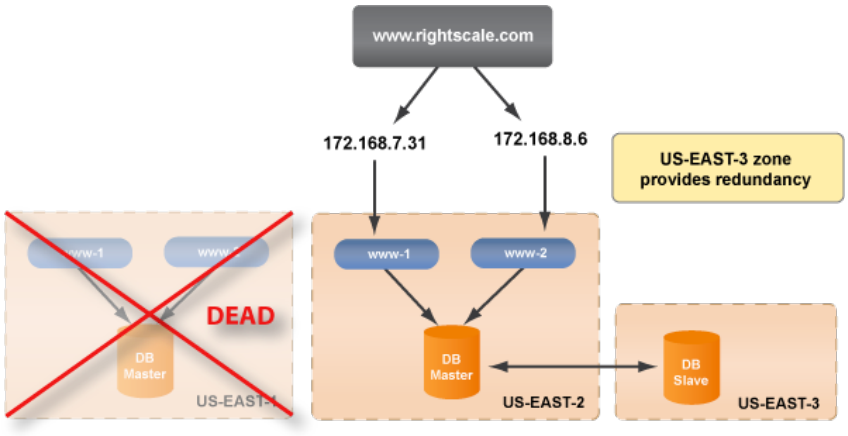

Figure 10: Elastic IP distribution via AWS in case of Server crash

Maintenance Criteria:

Extensibility: analyzefor.me designed to make able add more analysis with time. The overall application divided with sub programs. For example, adding new analysis classes is not affecting other sub programs such like interface.

Portability: Amazon Web Service Elastic Beanstalk service run applications usually in one static platform. Analyzefor.me background worker application is designed to work in CentOS which is a Linux distribution and a fork from Red Hat (RHEL) Linux distribution. However, in case of change in service provider background worker designed to run on RHEL and other RHEL core Linux distributions such as Oracle Linux (Red Hat Inc 2014).

Web service of analyzefor.me is capable to run on nearly all Linux distributions which have proper php distribution (Version 5.6).

Modifiability: analyzefor.me is designed to change continuously. Server application and background worker application divided also mini sub programs to make possible easy change in the future. Layer based separation between sub programs make possible to add one more level abstraction.

Performance Criteria:

Response Time: Response time of past analyzed data will be accessible in milliseconds over 80th port request which create only one request on only web server.

Response time of analysis requests is very important bottleneck for such an data process application. Analyzefor.me designed to answer analysis request as soon as possible and it also designed to provide cheap solutions. Maximum response times of analysis determined in case of over usage of the system:

- Follow Analysis and processing communities: 0.1 hour
- Like Analysis: 1 hour
- Gender Analysis for one person: 0.01 hour
- Popularity Analysis over Follow data: 0.1 hour

Trade-Offs:

Cost vs Response Time: More money is a solution most of the time for Amazon Web Services. Response time have direct relationship with number of computational units. In case of Amazon Web Services analyzefor.me is using EC2[1] instances in order to make analysis computations. Increasing over EC2 instances will dramatically reduce response time.

In case of the over usage of analyzefor.me AWS will increase EC2 instance number or it will redesign web server number. There is all in cloud, and a simple query over CPU usage and cost will determine these in-system dynamics (Figure 3).

[1] Amazon Elastic Compute Cloud (EC2) forms a central part of Amazon.com's cloud-computing platform, Amazon Web Services (AWS), by allowing users to rent virtual computers on which to run their own computer applications. EC2 encourages scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image to configure a virtual machine, which Amazon calls an "instance", containing any software desired.

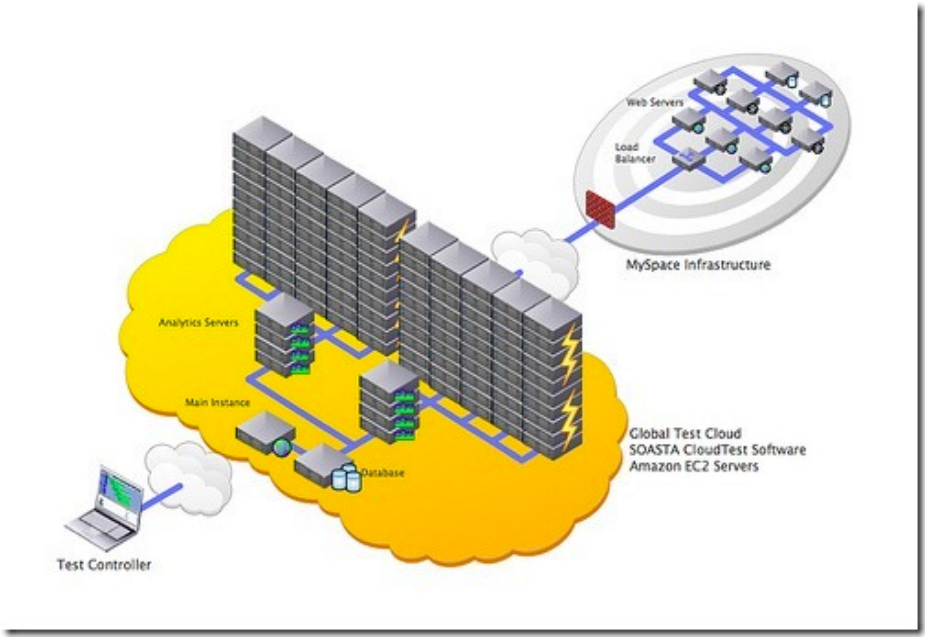

Figure 11: Web Services and Web Workers on AWS

Cost-Reliability: You pay only for the compute power, storage, and other resources you use; with no long-term contracts or up-front commitments so storing analysis results have a price. To show same graphs to users analyzefor.me should make chooses over price or reliability.

As a result of this, AWS may have high cost than others application servers but AWS service allows determining reliability, performance, scalability, and effectivity over cost.

4.2 Subsystem Decomposition

The main aim of decomposition is that occur recurrent interfaces that includes classes, which have similar method and states. Designers' aim is to reduce the coupling between the different subsystems. For a good design, high

coherence and low coupling are necessary factor. When designers want to occur program, which has good design, dependency between classes should be high; on the other hand, dependency between subsystems should be low because changes between subsystems will high impact on other subsystems.

In our program, analyzefor.me, we have identified main four subsystems but these four main subsystems include their own subsystems. These are User Interface, Worker Application Controller, Model Elements and Storage Elements. We try to write each subsystem with high coherence and low coupling.

User Interface: User Interface subsystem includes 3 own subsystems such as analysis interface, login interface and graph interface. The subsystem User Interface provides the connection between user and application.

Dependencies:

- Storage Elements
- Models Package

Worker Application Controller: Worker Application Controller package and its sub packages is designed to run in worker environment. They are designed to make analysis over users.

Worker Application subsystem includes four subsystems such as follows manager, like manager, comment manager and user manager.

Sub Packages:

- Thread Package
- Analysis Package
- Manager Package
- Adapter Package

Dependencies:

- Storage Elements
- Models Package

Model Package: Model package contains struck of expected data format. Model package designed to make enable usage of storage elements from controllers and user interface. It has no dependencies.

Storage Elements: Storage elements contain three different sub programs. In order to maintain low cost and high performance different type of databases and file systems is used.

Relational Database Management System (MySQL on AWS RDBM System): RDBMSs are a common choice for the storage of information. They are fast; in order to maintain fast service and minimum response time we are using this kind of service.

analyzefor.me's web server environment is commonly using this kind of database because of low latency time. Usually with the GB size data MySQL is capable respond in milliseconds.

- Fast and reliable system
- Capable to join tables
- All SQL command can runnable

Non-Relational Database (Dynamo DB on AWS): Application Server side (Worker Environment) of analyzefor.me application uses Dynamo DB of AWS. It is because some tables of storage level is more than storage abilities of Mysql.

File System (S3 buckets of AWS): Graph information is passing interface via S3 packages from controller to interface by JSON format.

- Have lower cost for bigger files than NoSQL
- Fast for one get request

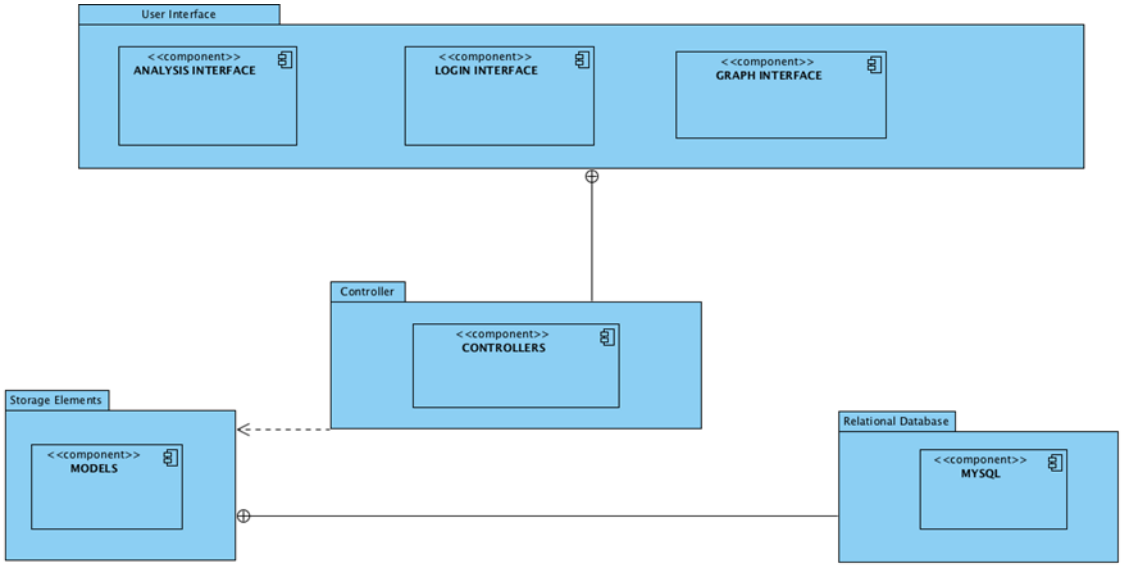

Figure 12: Subsystem Decomposition

4.3 Architectural Patterns

Architectural Patterns are helpful to have a pattern of system structure before design program. We use three patterns like Layers, MVC and Client/Server with Repository.

4.3.1 Subsystems as Layers

Subsystem is collection of classes, associations, operations, events and constraints. Subsystem interfaces are defined during object design. Designers divide system as subsystems to make comprehensible and implementable design. In addition to this, subsystems can be decomposed by many subsystems.

In analyzefor.me application, layers are used in order to prevent huge complexity of system. Subsystem layers designed to be free from their above and opaque for their below.

For example in analyzefor.me server side application have a system overview as follows:

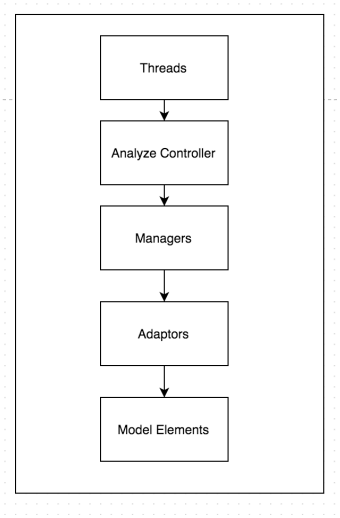

Figure 5: Layers of worker environment

Inside of the manager, adapter and model subsystems:

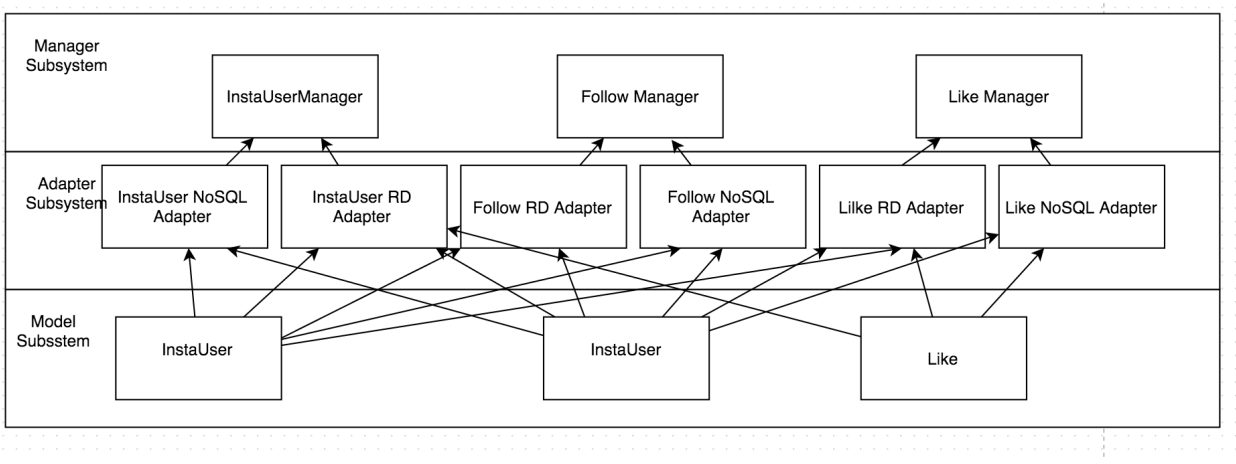

Figure 6: Inside of the layers detailed

4.3.2 MVC Architectural of Server-Side Program

The pattern of MVC (Model-View-Controller Pattern) includes user interfaces, transactions of user actions and data management. In this architectural pattern, the main aim is to separate the subsystems into three parts such as model, view and controller.

The reason of separating the subsystems is to isolate the domain information from the users interface by putting a controller department between them.

Normally Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. In analyzefor.me we use MVC to design separately our user interface and worker environment. However by a overall looking, all system have also a

Model: Model has the application domain knowledge Model which has higher coupling, is the data management part of system. The database interaction is most important part of model. Controller elements call the operations of model.

View: View part include graphs and buttons. There are many panels in program that are used by user. View is in charge of showing the model to user

Controller: Controller is the vital part of the system because controllers update and check classes. That is why, controller provides to communicate for user to the system.

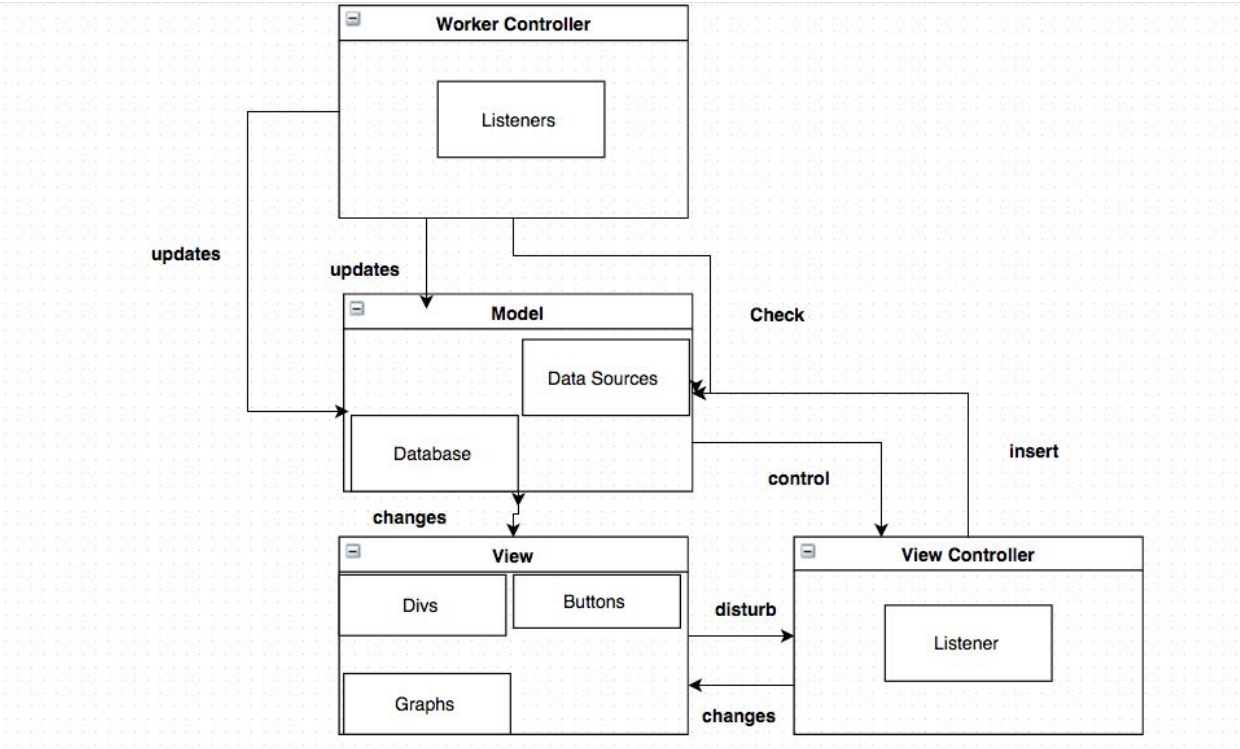

Figure 7: MVC of overall system

Our worker environment uses same model package with our user interface environment.

4.3.3 Client/Server with Repository

Distributed systems which use Client/Server pattern manage the large amount of data. Repository pattern is defined as a different pattern but they are bond together. A repository pattern involves a central data structure. In this structure the data of program is stored. For our application, the results of graphs are needed to keep in central data structure. Moreover, users, instausers, likes, follows etc. are stored in this database. The web server and the server of our application serve the database to numerous clients all over the world. The server

can easily handle follows, and likes but it is hard to find community and produce graph for server because throughput would be high.

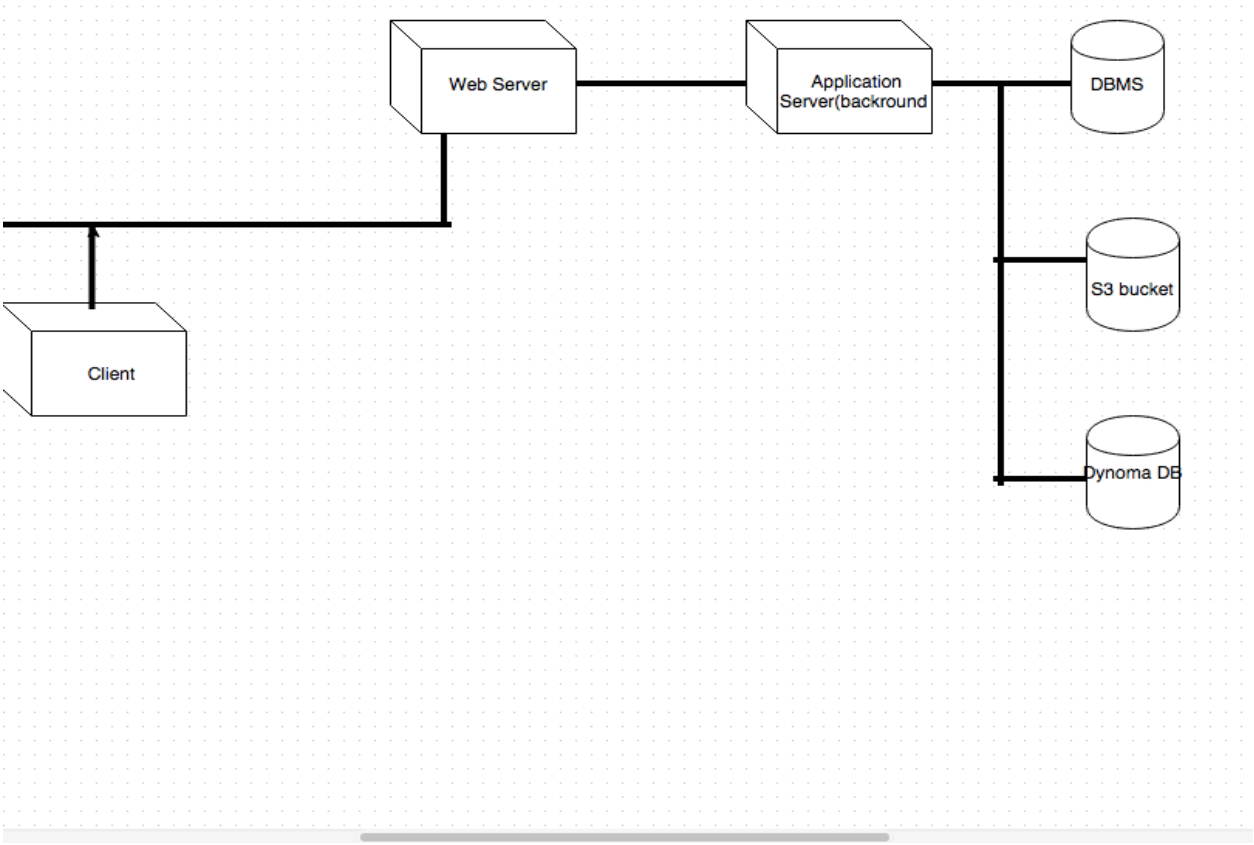

Figure 8: Client/Server with Repository

4.4 Hardware and Software Mapping

Instagram Data Analysis Application will be used from all kinds of people. Therefore, this program will work on every computer which is able to connect internet.

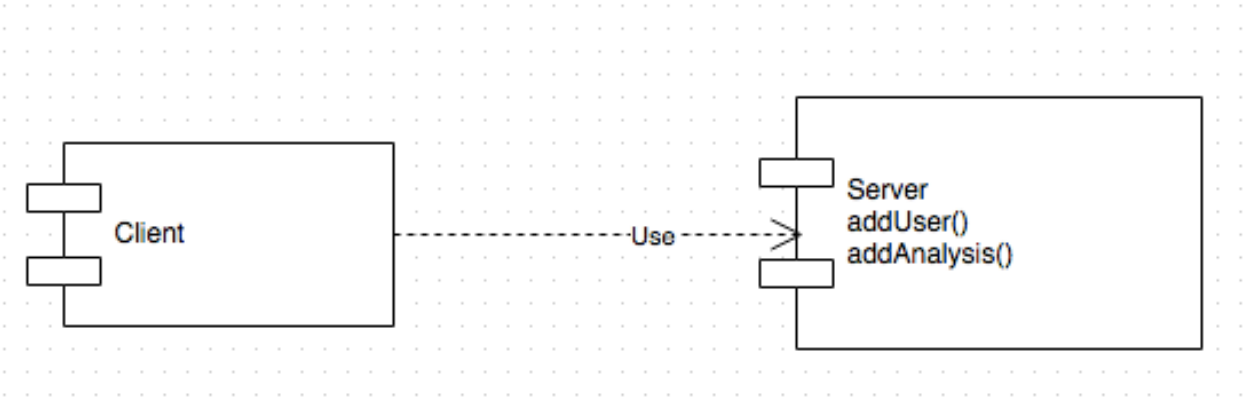

Figure 9: Client to server

When the user open the program, it will directly connect to server with the internet service and access to database will be provided by the server. In order to achieve the best performance results in our program, the database will be divided into three parts. In terms of the size of the data, it will be considered as big data, small data or ready data. Big data will be stored in Dynamo Database which is form of a NoSQL. This is the most efficient way to traverse the big data. For the small data MySQL database will be provided. For this database, in MySQL library client package will be used in Python. S3 AWS Bucket Service will provide the ready data for the program. Our database will be able to work on every Linux environment with respect to its properties. Python and PHP will be the developing languages. Deployment Diagram is as follows.

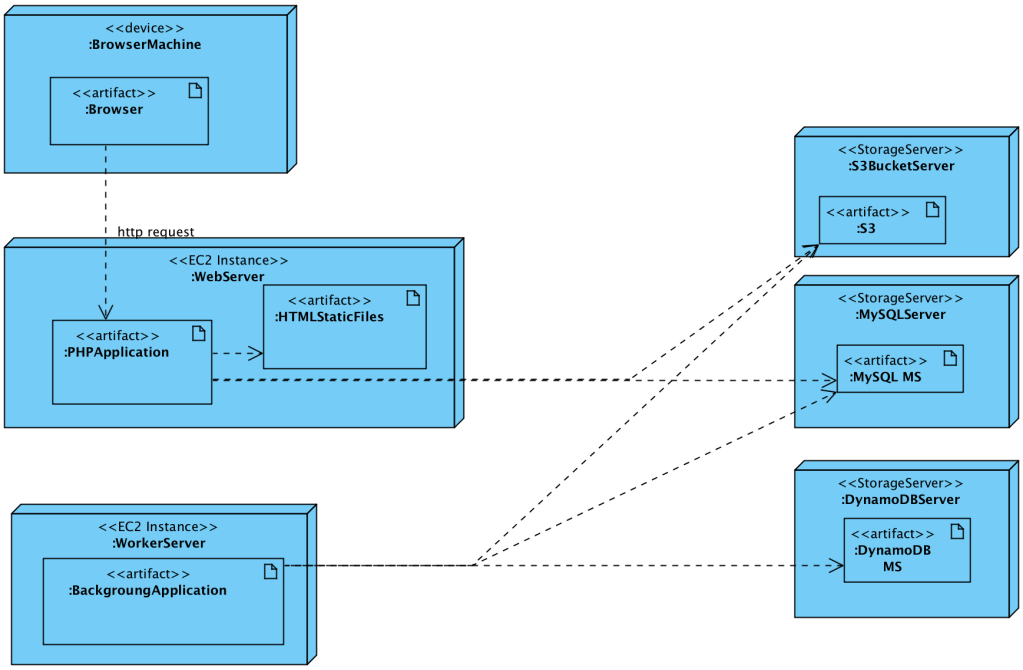

Figure 10: Deployment Diagram of analyzefor.me

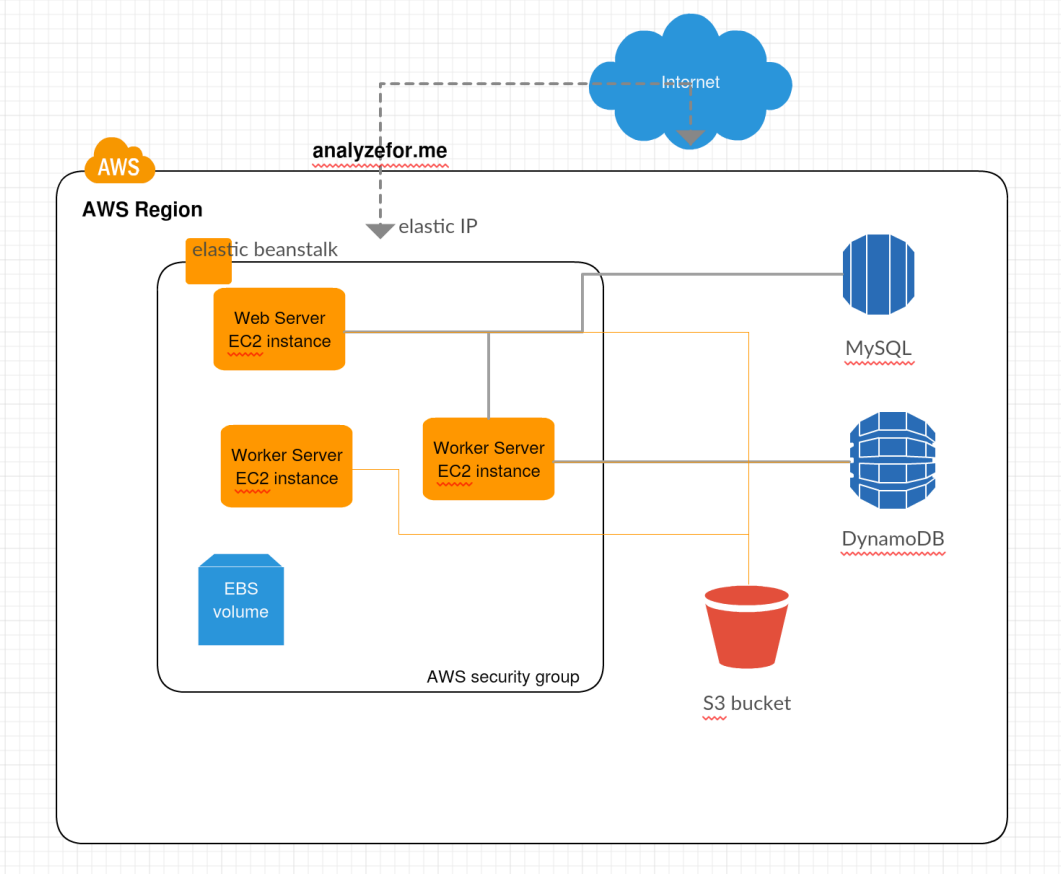

Figure 11: System Overview with AWS point of view

4.5 Addressing Key Concepts

4.5.1 Persistent Data Management

Instagram Data Analysis Application keeps the specific data for each individual. This data is only accessible for that user. There is not different kind of users and all the users should be kept in persistent storage because each person has a private account. If program doesn't keep them in the persistent data, all the users will be lost when they close the program. In addition to this, saved data analysis should also be kept in persistent storage, because there is an opportunity to see the old analysis. In order not to lose any information during each execution, the program must keep them in non-volatile memory.

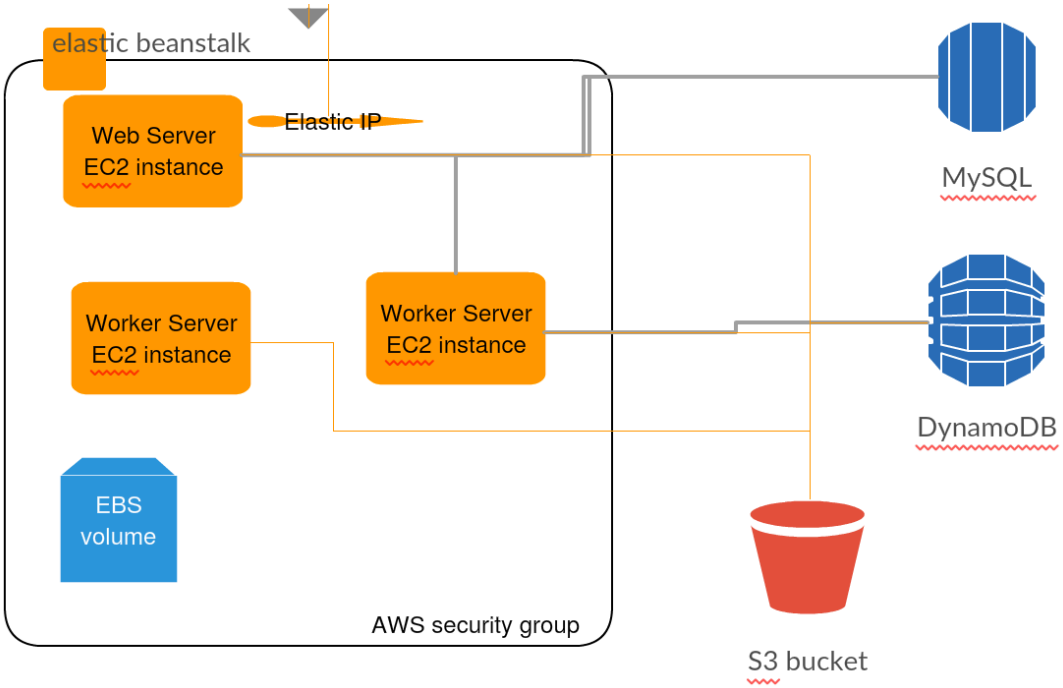

Figure 12: Overview of AWS storage system

Persistent data divides into three parts:

Dynamo DB - NoSQL: If the individual has so many followers, data for that person will be kept in Dynamo Database which is considered as a type of NoSQL in order to provide support bigger tables, low cost and indexing. RDM systems seem like have more disadvantages than NoSQL in terms of elasticity, flexibility and usability, it has decent amount of failure ratio. Because of we don't want to lose the data and our general audience will have reasonable amount of followers.

So the reason for usage of Dynamo Database can be clarified by these reasons:

- Have lower cost for bigger tables (compared to Relational Database Managements such as MySQL). Example price for indexed Dynamo DB per GB on Amazon is 0.25 $ monthly.

- Support indexing in one table, it can be very fast in only one table. Average service-side latencies are typically single-digit milliseconds. As your data volumes grow and application performance demands increase, Amazon Dynamo DB uses automatic partitioning and SSD technologies to meet your throughput requirements and deliver low latencies at any scale.
- It is supporting distributed tables. Data item collections distributed in the Amazon Cloud so it can support bigger tables.
- Dynamo DB supports reliability, fine-grained access control system, so it is more secure then saving data in files.

Figure 13: Used Dynamo DB icon in diagrams

MySQL – RDMS: However also some small data will be stored in MySQL server due to its typical SQL advantages. MySQL is also an undeniable option for persistent data. And sometimes analyzefor.me needs advantageous of complex SQL queries such as JOIN.

So the reason for usage of Amazon RDMS (with MySQL Engine) can be clarified by these reasons:

- MySQL engine can respond very fast without JOINs.
- It is easy scalable, and can support up to 244 GB data.
- It is durable with automated backups, database snapshots, and automatic host replacement

- Also RDMS is powerful; MySQL is capable to run complex SQL queries.

Figure 14:13 Used RDMS icon

S3 – File System: When the analysis applied on an account it will be saved into persistent data and in terms of efficiency S3 Bucket AWS will provide that analysis to other users. Analyzefor.me worker application produces big objects as result, which contains hundreds of nodes and thousands edges inside. These files will be saved in S3 buckets which is a kind of file system. It is cheap, and fast to read consecutive areas not random access.

So the reason for usage of S3 buckets can be clarified by these reasons:

- Fast to read one large file consequently.
- Very cheap to storage: Example price for S3 bucket per GB on Amazon is 0.03 $ monthly.

Figure 15: Used S3 bucked icon

4.5.2 Access Control & Security

In our application there is only one authorized level for users in the system, the authorized users can reach every feature of the application. For access to functions of the application, the user has to authenticate their information at the login screen of the application. The table which shows an application access control mechanism for users:

Actors	*User*	*InstaUser*	*FollowAnalysis*	*LikeAnalysis*	*Relations*
Authorized User		<<create>> createInstaUser() view()	<<create>> createFollowAnalyis() view() do() stop()	<<create>> createLikeAnalysis() view() do() stop()	<<create>> createRelations () view()
User	<<create>> createUser()				

Table 1: Access Matrix

We use Larevel to make implementing authentication very simple. We use the database authentication driver which uses the Laravel query builder. For storing passwords, we use the Laravel Hash class which provides secure Bcrypt hashing. When user log into our application, we use the Auth::attempt method. When the attempt method is called, the auth.attempt event will be fired. If the authentication attempt is successful, the user can be logged in to the application. Moreover, Laravel provides facilities for strong AES encryption via the 'mcrypt' PHP extension.

Users are not allowed to reach the system data files while they are using the application. The data file is only reachable by the system.

4.5.3 Global Software Control

Event-driven control is used in the application because the system needs to give reaction to the user's input and handle algorithm computations at the same time. Procedure-driven control mechanism maintains an update for application.

Event-driven control allows the system to be highly reactive to the status of the objects in the system. The input is centralized and the controller structure of the system will be basic. Therefore, when a new input is received, system will react to the input.

Procedure-driven control is necessary into program code, because our game is designed in such a way that if there is no user interaction to the system should refresh or update itself. AWS (Amazon Web Server) Bean Stack used browsers to update the application. While browsers are updating the application, they use the old version of the application. Therefore, system regularly updates itself.

4.5.4 Boundary Conditions

Main page is initialized by accessing their user interface parts in the initialization process. There is user logged in part in the main page. User should log in to the application to access main page. After the login process, the graphs that were produced by user will be loaded to the system. When user log in to program, he or she produce his or her graph according to his likes and follows.

The system creates its controllers and views on every log in and destroys them at every log off. The data is saved to the database before every log off, and the user data is loaded after every authentication. As a result of this, when the application is closed or logged off, no information is lost and also saved.

Startup: Enter www.analyzefor.me from a browser newer than

- Internet Explorer Version 8
- Chrome Version 22
- Opera Version 25
- Safari Version 7

Shutdown: Click on the "log off" icon on the main application screen.

Configuration: In this application configuration conditions are needed. View subsystem produces configurations with using analysis class. Background maker arranges configurations according to users.

Termination: This application can be closed or terminated from client side by clicking "Log out" button in main menu.

Exception handling: Try catch blocks are used for exceptions. If there is an exception in the application analysis class catches them and save them in to database. If there is exception, the application remembers it.

Software failure: If there is a wrong about the choice of user, exceptions will be handled by a warning.

Error conditions are as following:

Extreme Conditions:

- Users can have very few number of friend to make an analysis over them
- Users can have very mush number of friend to make an analysis over them

Logging in:

- Username or password boxes are not filled.
- Username or password is not matching.

User settings:

- Settings system does not give answer while editing.

5. Object Design

5.1 Pattern Applications

Façade Pattern

Façade is a type of structural design pattern in which a class makes all the specific type of operations. The aim of this pattern provides extendibility, reusability and maintainability because subsystems which create with Façade Design Pattern can provide a public interface for external classes to use so as to relation with the package/subsystem. That is why; connection between classes is secure and proper way. Another advantage is that developers are able to handle a subsystem from a façade class properly and conveniently because of the fact that the interaction between outside of this subsystem, provides from common interfaces. This pattern decreases the coupling because it can block many classes to attack on a single class.

In our design we used façade pattern in subsystems:

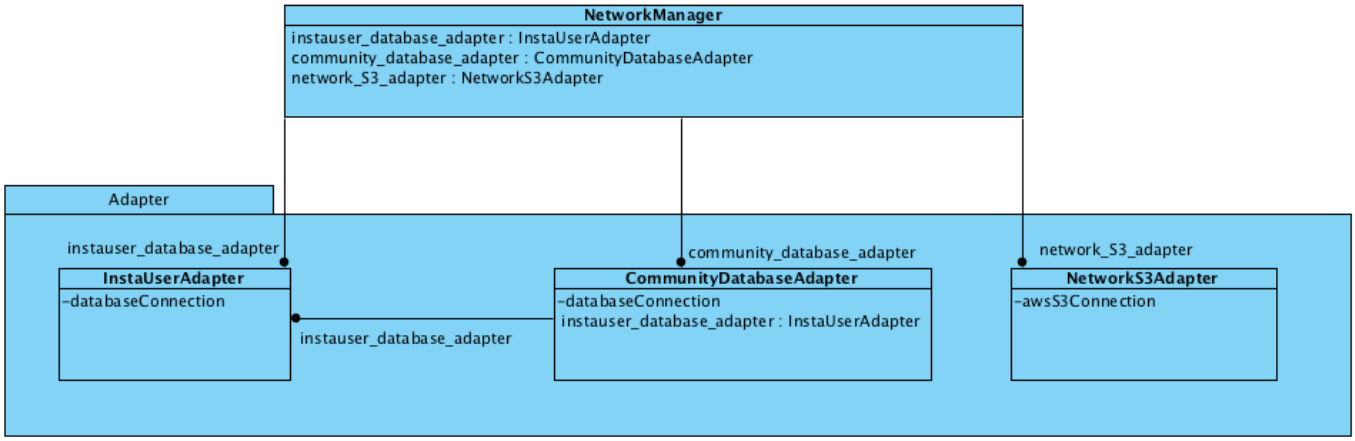

Figure 14: One example of Façade patterns

Observer Pattern

Observer pattern which is mentioned, it connects the state of an observed object, the subject with many observing objects to the observers. In order to provide, sending notifications at one time we use observer pattern. SendNotification class notifies all the users for any decision at one time instead of notifying the users one by one. This provides a quick response time when letting know about new feature.

In addition to this, observer pattern has a relationship with the Model View Controller. When changes occur in the observer pattern, all the model objects will be directly affected from that situation and all users are able to see the news about the program or they are able to know if something goes wrong.

In Instagram Data Analysis Application we use the three variants of maintaining consistency which are push notification, push/ update notification and pull notification. When state is changed for all the users it will be announced by this pattern. During the program if there is update, program will use the push update notification system in order to let people to know about the new feature. This enables to program work more efficiently. Otherwise all users should be informed by one by one and it means time loss and using the server much more. Pull notification is also other advantage of the system. Instead of answering users individually they all get the same information from the same place at one time.

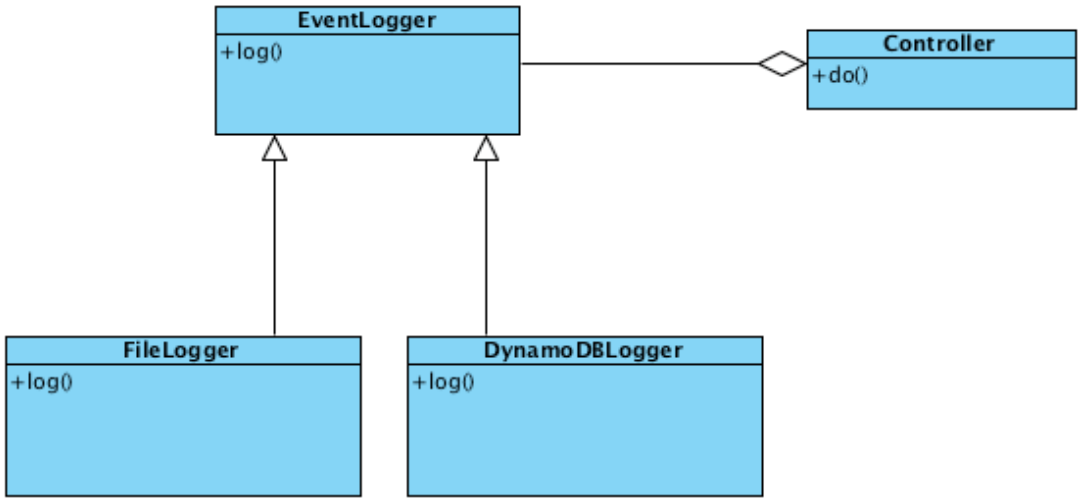

Figure 15: Observer Pattern Class Diagram

Composite Pattern

Composite design pattern is such a structural pattern and it creates a tree structure of group of objects. In our application, we used composite design pattern. We used a lot of group of objects. Composite design pattern provide us to treat a group of objects in similar way as a single object. Composite design is like tree structure. It shows whole hierarchy of objects. Composite Design pattern creates a class which is included group of its own objects. We can modify our objects with using this type of pattern in our application. Our application needs to control a hierarchical collection of "primitive" and "composite" objects. Processing of a composite object is managed differently than primitive objects because primitive objects is managed one way.

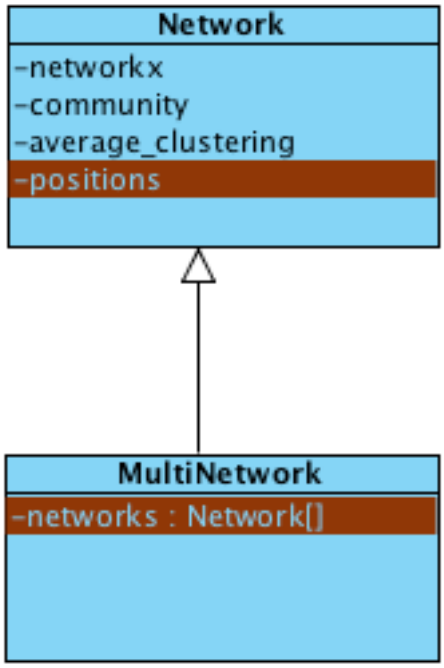

Figure 16: Composite Pattern Class Diagram

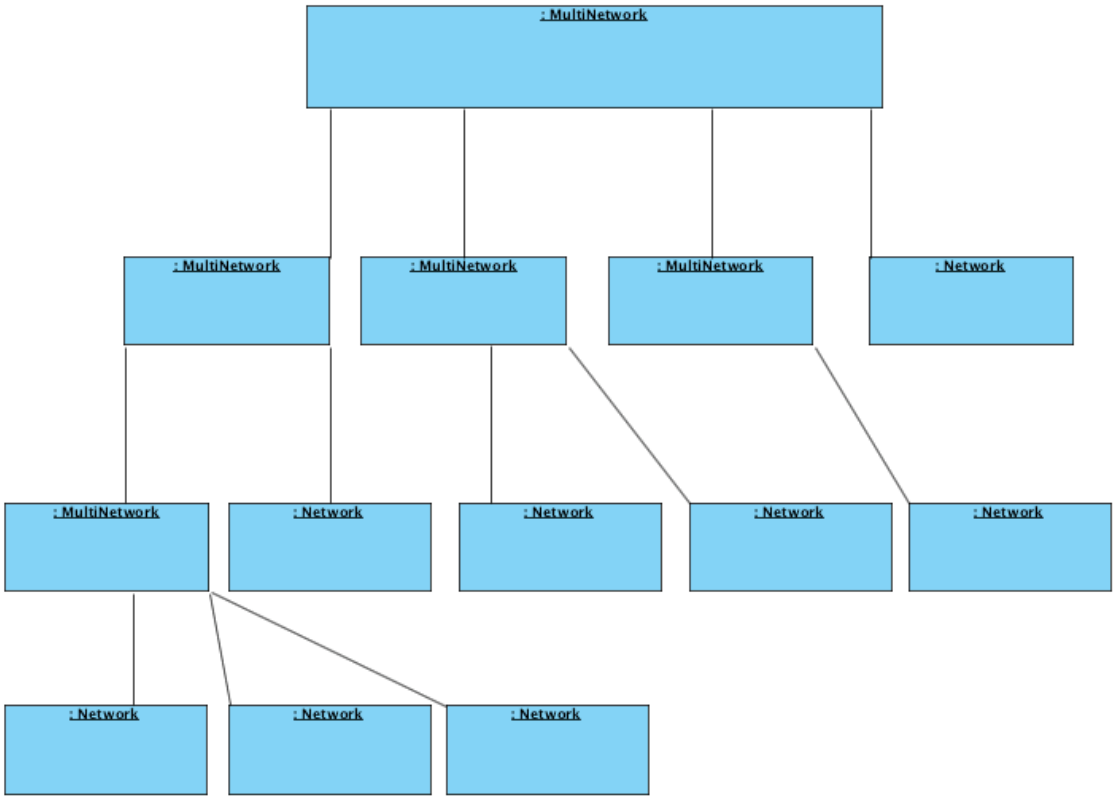

Figure 17: Composite Pattern Object Instance Diagram

5.2 Class Interfaces

5.2.1 Model Classes

5.2.1.1 Analysis Abstract Class

Attributes

Analysis class is a child of SocialBaseObject so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

Analysis_uid : int	Primary key for Analysis entity in database
In_process : int	Represents is in process or not
Finished : int	Represents is finished or not
crushed_msg : String	Represents crushed message
Instauser_uid : int	Represents user who ordered this analysis
Created_at : timestamp	Represents created time for analysis entity
Updated_at : timestamp	Represents update time for analysis entity

Table 26: Attributes of Analysis Class

Constructor

def __init__(self, vars__=None, params=None): Initializes an Analysis object with given parameters. Constructor takes current possible attributes and sends them with child attributes to SocialBaseObject constructer.

Methods

def finished(self, success=1): Modify finished attribute of the related object.

def update_in_process(self): Modify in_process attributes of the related object.

Community Class

Attributes

Community class is a child of SocialBaseObject so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

community_uid:int	Primary key for Community entity in database
source_instauser_uid:int	Represents its instauser source
clustering_coef	Clustering coefficient of community
s3location	Represents location

Table 27: Attributes of Community Class

*Constructo*r

def __init__(self, from_db=False, instauser=None, instausers=[], pageranks={}, community_type=CommunityType.community, vars__=None, params=None): Initializes a Community object with given parameters. Constructor takes current possible attributes and sends them with child attributes to SocialBaseObject constructer.

Methods

def add_instauser(self, instauser): Add new instauser to community object.

def set_clustering_coef(self, coef):

def set_pageranks(self, pageranks): Set page ranks in the community object.

CommunityRelation Class

Attributes:

CommunityRelationclass is a child of SocialBaseObject so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

community_relation_uid:int	Primary key for CommunityRalation entity in database
instauser_uid:int	Represents user who ordered this CommunityRalation
community_uid:int	Represents community who ordered this CommunityRalation
Pagerank:int	Represent pagerank

Table 28: Attributes of CommunityRelation Class

Constructor

def __init__(self, vars__=None, params=None): Initializes a CommunityRelation object with given parameters. Constructor takes current possible attributes and sends them with child attributes to SocialBaseObject constructer.

Methods

enLargerCommunityAnalysis Class

Attributes:

enLargerCommunityAnalysis is a child of Analysis so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

community_uid:int	Represents community who ordered this enLargerCommunityAnalysis

Table 29: Attributes of enLargerCommunityAnalysis Class

Constructor

def __init__(self, vars__=None, params=None): Initializes an enLargerCommunityAnalysis object with given parameters. Constructor takes current possible attributes and sends them with child attributes to Analysis constructer.

Methods

FollowAnalysis Class

Attributes:

FollowAnalysis is a child of Analysis so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

instauser_uid:int	Represents users who ordered this FollowAnalysis
Limit:int	Represents the limit of Follow Analysis

Table 30: Attributes of FollowAnalysis Class

Constructor

def __init__(self, vars__=None, params=None):): Initializes a FollowAnalysis object with given parameters. Constructor takes current possible attributes and sends them with child attributes to Analysis constructer.

InstaGroup Class

Attributes:

Follows class is a child of InstaGroup.

Constructor

def __init__(self, instausers=None, params=None): Initializes a InstaGroup object with given parameters.

Follows Class

Constructor

def __init__(self, from_db, instauser, instausers, params=None): Initializes a Follows object with given parameters. Constructor takes current possible attributes and sends them with child attributes to InstaGroup constructer.

InstaRelation Class

Attributes:

InstaRelation is a child of SocialBaseObject so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

relation_uid:int	Represents relations who ordered this InstaRelation
source_uid:int	Represents sources who ordered this InstaRelation
target_uid:int	Represents targets who ordered this InstaRelation

Table 31: Attributes of InstaRelation Class

Constructor:

def __init__(self, vars__=None, params=None): Initializes a InstaRelation object with given parameters. Constructor takes current possible attributes and sends them with child attributes to SocialBaseObject constructer.

InstaUser Class

Attributes:

InstaUser is a child of SocialBaseObject so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

instauser_uid:int	Represents users who ordered this

	InstaUser.
Username:String	Represents user name.
profile_picture:	Represents profile picture of users.
full_name:String	Represents full name of users.
Gender:String	Represents gender of users.
is_public:int	Represents users information is public or not.
total_followers:int	Represents total followers of users.
total_media:int	Represents total media of users.
total_follows:int	Represents total follow of users.
updated_at: timestamp	Represents update time for InstaUser entity
is_loaded:int	Represents is loaded or not

Table 32: Attributes of InstaUser Class

Constructor:

def __init__(self, is_exists_in_db=True, from_db=True, vars__=None, params=None):): Initializes a InstaUser object with given parameters. Constructor takes current possible attributes and sends them with child attributes to SocialBaseObject constructer.

Methods

def __str__(self): to String method for instauser

MultiNetwork

Attributes:

MultiNetwork is a child of Network.

Constructor

def __init__(self, networkx, community, positions=None): Initializes a MultiNetwork object with given parameters. Constructor takes current possible attributes and sends them with child attributes to Networkconstructer.

Methods

def generate_sub_networks(self): Turn network to multinetwork

def get_community_uid_of_instauser(self, instauser_uid): Get the community of UNIQUE_ID of given instauser

SeparateAnalysis

Attributes:

SeparateAnalysis is a child of Analysis so db_vars determine possible attributes in self._info. Possible meanings of parameters are below:

community_uid:int	Represents community who ordered this SeperateAnalysis
Limit:int	Represents the limit of SeparateAnalysis.

Table 33: Attributes of I SeparateAnalysis Class

Constructor

def __init__(self, vars__=None, params=None):): Initializes a SeparateAnalysis object with given parameters. Constructor takes current possible attributes and sends them with child attributes to Analysis constructer.

SocialBaseObject

Attributes:

SocialBaseObject is a child of object.

Constructor

def __init__(self, is_exists_in_db=True, from_db=True, vars__=None, params=None):): Initializes a SocialBaseObject object with given parameters.

Methods

def __getattr__(self, name): Get the identified attribute of SocailBaseObject.

def __hash__(self): !!!!!

def __eq__(self, other): Equalize the UNIQUE_ID of two attributes of SocailBaseObject.

def _set(self, params): Set the given attribute to parameter value.

def have_uid(self): Get the UNIQUE_ID of given attributes.

5.2.2 Analysis Controller Package Classes

AnalysisThread Class

Attributes:

AnalysisThread is responsible for graph analysis.

_factory : AnalysisFactory	It is a factory object
_database : database	It is a database object
_thread_id : thread_id	It is a thread_id object
_ready : true	It is a boolean object
_status : status.Ready	It is a status object
_analysis_controller:analysis_controller	It is a analysis_controller object

Table 14: Attributes of AnalysisThread Class

Constructor

def __init__(self, database, s3client, thread_id, analysis_controller=None): Initialize analysisController.

Methods

def run(self): Process goes on.

def stop(self): Process finish

SocialBase Class

Attributes:

SocialBase is responsible to people's social status.

_cursor : Cursor	Cursor of Mysql

Table 15: Attributes of SocialBase Class

Constructor

def __init__(self): This is Singleton Class for SocialBase communications.

Methods

def get_instance(): This method call the instance object.

def get_gender(self, name=None): This method return to gender of name like 'f' for female or 'm' for male 'u' for undefined.

def select_row(self, query): This method takes to Select Query and runs it in socialbase and returns the result of Select Query as list of dicts.

def modify(self, query): This method takes INSERT, UPDATE, DELETE queries and runs it in socialbase and return true for successful modification.

AnalysisController Class

Attributes:

This class control to the analysis object. AnalysisController is a child of SocialBaseObjectDatabaseAdapter and InstaConnection.

_analysis : analysis	All analysis will have api connection.
_insta_connection :InstaConnection(analysis.access_token)	It is an API caller object.

Table 16: Attributes of AnalysisController Class

Constructor

def __init__(self, database, analysis): Initializes a analysis database object with given parameters in params.

EnLargerCommunityAnalysisController Class

Attributes:

This class controller the enlarger community. EnLargerCommunityAnalysisController is a child of InstaUserManager, AnalysisController and FollowsManager.

_user_api_manager: InstaUserManager(database=database, insta_connection=self._insta_connection)	It is an API caller object.
_follows_manager : FollowsManager(database=database, insta_connection=self._insta_connection)	It is an API caller object.

Table 17: Attributes of EnLargerCommunityAnalysisController Class

Constructor

def __init__(self, database, analysis): Initializes a analysis database object with given parameters in params.

Methods

def do(self): start to do an enlarger analysis

FollowAnalysisController Class

Attributes

This class is controller the follow analysis. FollowAnalysisController class has a relation of InstaUserManager, AnalysisController, AnalysisManager, FollowsManager, NetworkManager, CommunityManager, Community, CommunityType and MultiNetwork classes.

_community_manager : CommunityManager	It is object of CommunityManager class
_instauser_manager : InstaUserManager	It is object of InstaUserManager class
_follows_manager : FollowsManager	It is object of FollowsManager class
_network_manager : NetworkManager	It is object of NetworkManager class
_analysis_manager : AnalysisManager	It is object of AnalysisManager class

Table 18: Attributes of FollowAnalysisController Class

Constructor

def __init__(self, s3client, database, analysis): Initializes a client, analysis, database object with given parameters in params.

Methods

def do(self): This method uses all relation which takes place in attribute.

def get(self, socialbaseobject):raise NotImplementedError

SeparateAnalysisController Class

Attributes

This class is controller the separate analysis. SeparateAnalysisController class has a relation of InstaUserManager, AnalysisController, AnalysisManager, FollowsManager, NetworkManager, CommunityManager and MultiNetwork.

_instauser_community_group_manager: CommunityManager	It is object of CommunityManager class
_instauser_manager : InstaUserManager	It is object of InstaUserManager class
_network_manager : NetworkManager	It is object of NetworkManager class
_analysis_manager : AnalysisManager	It is object of AnalysisManager class

Table 19: Attributes of SeparateAnalysisController Class

Constructor

def __init__(self, s3client, database, analysis): Initializes a client, analysis, database object with given parameters in params.

Methods

def do(self): This method uses all relation which takes place in attribute. This method has multi network which is a composite design pattern.

def get(self, socialbaseobject):raise NotImplementedError.

InstaConnection Class

Attributes

This class provides instagram connection. This class import InstagramAPI and InstagramAPIError Class.

_api : None	Api connection for Instagram api
ready :False	Is ready or not Boolean
access_token : access_token	String of token to access information

Table 20: Attributes of InstaConnection Class

Constructor

def __init__(self, access_token=""): initializes Instagram connection which holds an Instagram api and responsible for communication with Instagram. Param is access_token whichis a string to reach Instagram.

Methods

def refresh(self, reason=None): This method provide update apı.

def follows(self, instauser_uid): This method returns list of insta-users which follow.

def user(self, instauser_uid): This method returns list of insta-users.

ApiGetter Class

Attributes:

It provides to connect instagram

_insta : InstaConnection	It is an API caller object

Table 21: Attributes of ApiGetter Class

Constructor

def __init__(self, insta_connection): Initialize _insta_connection.

Attributes:

Instagram api getter is responsible for api connection for instagram relations like follows and followers informations. InstaRelation Class uses Façade pattern with instaconnection object.

_insta : InstaConnection	It is an API caller object

Table 23: Attributes of InstaRelationApiGetter Class

Constructor

def __init__(self, insta_connection): Initialize _insta_connection.

Methods

def follows(self, source_instauser): This methods return the follows of specified instagram uid. Instagram uid means instagram user unique.

InstauserApiGetter Class

Attributes:

Instagram api getter is responsible for api connection for instagram relations like follows and followers informations. InstaRelation Class uses Façade pattern with instaconnection object.

_insta : InstaConnection	It is an API caller object

Table 24: Attributes of InstaUserApiGetter Class

Constructor

def __init__(self, insta_connection): Initialize _insta_connection.

Methods

def follows(self, source_instauser): This methods return the follows of specified instagram uid. Instagram uid means instagram user unique.

Word Class

Attributes:

This class import EmojiDict.

Initialized: Boolean	İs initialized or not Boolean
word :word	String represatation of word
emotion : int	Emotion point
repatition : int	Count of repatition

Table 25: Attributes of Word Class

Constructor

def __init__(self, str_word):): Initialize str_word.

Methods

def __hash__(self): hasher for object

def __eq__(self, other): Compare two word object

def is_emoji(self): return is emoji or not

def extract_word(str_word): extract the string word

def turkish_utf8(word): change Turkish carecters to ascii

CommunityManager Class

Attributes:

CommunityManager is a Manager Layer class. It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

_communitiy_database_adapter : CommunityDatabaseGetter	Adapter Layer class to database operations of communities
_instarelation_database_adapter : InstaRelationDatabaseGetter	Adapter Layer class to database operations of instarelations

Table 26: Attributes of CommunityManager Class

Constructor

def __init__(self, vars__=None, params=None): Initializes a CommunityManager object with given parameters in params. It takes AWS s3 client and a database connection for give them to adapters

Methods

def get(self, community_uid): Gets and initialize community from a community_uid

def create_network(self, community, all_relations=None): Create a Network modal object from a list of relation and a community object. It can take all relations as parameter or it can also generate it.

FollowsManager Class

Attributes:

FollowsManager is a Manager Layer class. It has a Façade design pattern, so it has Adapter Layer objects, database adapter and InstaRelationApigetter use them in order to fulfilling their jobs.

_instauser_manager : InstaUserManager	User Manager classes that organizes and manages the users.
_follows_database_adapter : FollowsDatabaseAdapter	Adapter Layer class to database operations of instarelations

Table 1: FollowsManager Class Attributes

Constructor

def __init__(self, database, insta_connection):Initializes a FollowsManager object with given parameters in params. It takes database and insta_connection togive them to adapters

Methods

def get_and_insert(self, instauser_uid):Gets and inserts users to community from a instauser_uid

def insert(self, follows): This method is inserting Follows object to database if it is full and do not exists in the database.

def get_follows(self, source_instauser_uid): This method takes the follow information from instagram via API.

InstaUserManager Class

Attributes:

InstaUserManager is a Manager Layer class. It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

_instauser_database_adapter: InstaUserDatabaseAdapter	Adapter Layer class to database operations of communities
_instauser_api_getter: InstaUserApiGetter	Adapter Layer class to database operations of instarelations

Table 3: Attributes of InstaUserManager Class

Constructor

def __init__(self, database, insta_connection):Initializes a database object with given parameters in params. It takes connection for give them to adapters.

Methods

def get_and_insert(self, instauser_uid): Gets and inserts the instagram user from a community_uid

def insert(self, instauser): inserts the instagram user to the database.

def get(self, instauser_uid): gets the instagram user's id.

NetworkManager Class

Attributes:

NetworkManager is a Manager Layer class. It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

_communitiy_database_adapter: CommunityDatabaseAdapter	Adapter Layer class to database operations of communities
_ networkS3Adapter: NetworkS3ObjectAdapter	Adapter Layer class to database operations of instarelations

Table 3: Attributes of NetworkManager Class

Constructor

def __init__(self, s3client, database):Initializes a s3client objecta nd database with given parameters in params.

Methods

def get(self, community_uid): gets the community uid.

def insert(self, network): inserts the instauser graph

InstaRelationApiGetter Class

Attributes:

Istagram api getter is responsible for api connection for instagram relations like follows and followers informations. InstaRelation Class uses Façade pattern with instaconnection object.

_insta : InstaConnection	It is an API caller object

Table 3: Attributes InstaRelationApiGetter Class

Constructor

def __init__(self, insta_connection): Initialize _insta_connection.

Methods

def insert(self, community_uid): Gets and initialize community from a community_uid into to database

def get_without_instausers(self, community_uid): Gets a community without instausers from database by community_uid.

def get_with_instausers(self, community_uid): Gets a community with instausers from database by community_uid.

InstaUSerAPIGetter Class

Attributes:

InstaUSerAPIGetter is a Getter Layer class. It can only reach Model classes and other APIGetter Layer objects.

Constructor

def __init__(self, instaram_connection, params=None): Initializes instaram connection. Instagram api getter is responsible for api connection for instagram relations like follows and followers informations

Methods

def get(self, instauser_uid): This methods return the follows of specified instagram uid.

CommunityDatabaseAdapter Class

Attributes:

CommunityDatabaseAdapter is a Adapter Layer class. It can only reach Modal classes and other Adapter Layer objects.

_instauser_database_getter : InstauserDatabaseGetter	Adapter Layer class to database operations of instausers

Table 3: Attributes of CommunityDatabaseAdapter Class

Constructor

def __init__(self, database): Initializes a CommunityDatabaseAdapter object with given parameters in params. It takes only database connection.

Methods

def insert(self, community_uid): Gets and initialize community from a community_uid into to database

def get_without_instausers(self, community_uid): Gets a community without instausers from database by community_uid.

def get_with_instausers(self, community_uid): Gets a community with instausers from database by community_uid.

FollowsDatabaseAdapter Class

Attributes

FollowsDatabaseAdapter is adapter Layer class. It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

_database: database	Adapter Layer class to database operations of communities
_ instauser_database_getter: InstaUserDatabaseGetter	Adapter Layer class to database operations of instarelations

Table 2: FollowsDataBaseAdapter Class Attributes

Constructor

def __init__(self, database): Initializes the database getter class.

Methods

def insert(self, follows):Gets This method is inserting Follows object to database if it is full and do not exists in the database.

def get(self, source_instauser_uid): gets the user id via API from the database.

InstaGroupDatabseAdapter Class

Attributes

InstaGroupDatabseAdapter is an adapter Layer class. . It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

Constructor

def __init__(self, database):Initializes database connection.

Methods

def insert(self, instagroup):inserts the instagram users to database.

def get(self, socialbaseobject): gets the socialbaseobject.

InstaRelationDatabaseAdapter Class

Attributes:

InstaRelationDatabaseAdapteris an adapter Layer class. . It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

Constructor

def __init__(self, database):Initializes database connection.

Methods

def get(self, source_instauser_uid):inserts the instagram users id.

InstaUserDatabaseAdapter Class

Attributes:

InstaUserDatabaseAdapter is an adapter Layer class. . It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

Constructor

def __init__(self, database):Initializes database connection.

Methods

def get(self, instauser_uid):gets the instagram users id.

NetworkS3Adapter Class

Attributes:

NetworkS3ObjectAdapter is an adapter Layer class. It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

_communitiy_database_adapter: CommunityDatabaseAdapter	Adapter Layer class to database operations of communities
_ networkS3Adapter: NetworkS3ObjectAdapter	Adapter Layer class to database operations of instarelations

Table 3: Attributes ofNetworkS3Adapter Class

Constructor

def __init__(self, s3client):Initializes a s3client object and database with given parameters in params.

Methods

def get_network(self, community): Initiates the graph's network settings.

def insert(self, network): inserts the instauser graph.

def delete_all(self, network): delete all network between nodes.

def delete_graph(self, graph_type, graph_id): delete all thegraphs according to the user.

Constructor

def __init__(self, s3client):Initializes s3client.

SocialBaseObjectDatabaseAdapter Class

SocialBaseObjectDatabaseAdapter is adapter Layer class. It has a Façade design pattern, so it has Adapter Layer objects and use them in order to fulfilling their jobs.

Constructor

def __init__(self, database): Initializes the database getter class.

Methods

def get(self, socialbaseobject): Gets social base object with the name and the id.

def insert(self, socialbaseobject):inserts the user name and with the id

def update(self, socialbaseobject, set_values): updates the values of the user from the instagram.

def delete(self, socialbaseobject, delete_conditions): delete the social base object according to the conditions.

5.3 Specifying Constraints Using OCL

Community Database Adapter Class

Context: CommunityDatabaseAdapter::insert (self, community : Community)

Pre: MySQL::CheckCommunityIsExists(community_uid)

Post: SocialBase::insert(community)

Post: foreach community_relation : SocialBase::insert(community_relation)

In order to insert a community to database Mysql first check is it exists in database already.

After it insert community to Community table

After it insert all community relations

Context: CommunityDatabaseAdapter:: get_without_instausers(self, community_uid : Integer):

Pre: MySQL::CheckCommunityIsExists(community_uid)

Post: SocialBase::select_row(query)

In order to get a community from database Mysql first check is it exists in database already.

After it gets only community row without its instausers

Context: CommunityDatabaseAdapter:: get_with_instausers(self, community_uid : Integer):

Pre: MySQL::CheckCommunityIsExists(community_uid)

Post: SocialBase::select (query)

In order to get a community from database Mysql first check is it exists in database already.

After it gets only community row with its instausers

Follows Database Adapter Class

Context: FollowsDatabaseAdapter::insert (self, follow : Follow):

Pre: MySQL::CheckFollowIsExists(instauser_uid)

Post: foreach instauser_relation SocialBase::insert(instauser_relation)

Post: foreach instauser: SocialBase::insert(instauser)

In order to insert a follow to database Mysql first check is it exists in database already.

After it inserts instauser relations to Relations table

After it insert all instausers

Context: FollowsDatabaseAdapter:: get (self, instauser_uid : Integer):

Pre: MySQL::CheckFollowIsExists(instauser_uid)

Post: SocialBase::select (query)

In order to get a follow from database Mysql first check is it exists in database already.

After it gets follows with its instausers

InstaRelation Database Adapter Class

Context: InstaRelationDatabaseAdapter:: get (self, source_instauser_uid : Integer):

Pre: MySQL::CheckRelationsIsExists(instauser_uid)

Post: SocialBase::select (query)

In order to get a relations from database Mysql first check is it exists in database already.

After it gets relations without instausers.

Context: InstaUserDatabaseAdapter:: get (self, source_instauser_uid : Integer):

Pre: MySQL::CheckInstauserIsExists(instauser_uid)

Post: SocialBase::select (query)

In order to get a instausers from database Mysql first check is it exists in database already.

After it gets instauser.

SocialBase Object Database Adapter Class

Context: SocialBaseObjectDatabaseAdapter:: get (self, source_instauser_uid : Integer):

Pre: MySQL::CheckObjectIsExists(uid : Integer)

Post: SocialBase::select (query)

In order to get a socialbaseobject from database Mysql first check is it exists in database already.

After it gets socialbaseobject.

Context: SocialBaseObjectDatabaseAdapter:: insert (self, source_instauser_uid):

Pre: MySQL::CheckObjectIsExists(uid : Integer)

Post: SocialBase::insert (query : String)

In order to insert a socialbaseobject to database Mysql first check is it exists in database already.

After it insert socialbaseobject.

Context: SocialBaseObjectDatabaseAdapter:: update (self, source_instauser_uid):

Pre: MySQL::CheckObjectIsExists(uid : Integer)

Post: SocialBase::modify (query : Stirng)

In order to update a socialbaseobject to database Mysql first check is it exists in database already.

After it update socialbaseobject

Context: SocialBaseObjectDatabaseAdapter:: delete (self, source_instauser_uid):

Pre: MySQL::CheckObjectIsExists(uid)

Post: SocialBase::modify (query : String)

In order to delete a socialbaseobject to database Mysql first check is it exists in database already.

After it update socialbaseobject.

InstaRelation Api Getter Class

Context: InstaRelationApiGetter:: get(self, source_instauser_uid : Integer):

Post: self._insta.follows(source_instauser_uid)

Post: InstaUser.instauser_adapter (user)

In order to get a relation from api it uses InstagramConnection.follows method.

After it changes it to to own Instauser object type. **(Adapter Patern)**

After it return them.

Instauser Api Getter Class

Context: InstauserApiGetter:: get (self, instauser_uid : Integer):

Post: user = self._insta.get(instauser_uid)

Post: InstaUser.instauser_adapter (user)

In order to get a instauser from api it uses InstagramConnection.follows method.

After it changes it to to own Instauser object type. **(Adapter Patern)**

Analysis Factory Class

Context: AnalysisFactory:: get_last_analysis (self):

Post: MySQL::CheckNonFinishedAnalysissIsExists()

In order to get last analysis from database it is cheacking database first

After it returns the last analysis by instantiate it with correct class. **(Factory Pattern)**

Analysis Manager Class

Context: AnalysisManager:: finished (self, analysis_uid):

Post: MySQL::CheckNonFinishedAnalysisExists(analysis_uid)

It is checking is it exists in database and change finished attribute of it.

Context: AnalysisManager:: crushed (self, analysis_uid, msg):

Post: MySQL::CheckNonAnalysisExists(analysis_uid)

It is checking is it exists in database and change crushed attribute of it.

After it changes message attribute of it also

Context: AnalysisManager:: in_progress (self, analysis_uid, msg):

Post: MySQL::CheckNonAnalysisExists(analysis_uid)

It is checking is it exists in database and change in_progress attribute of it.

Community Manager Class

Context: CommunityManager:: get (self, instauser_uid : Integer):

Post: CommunityDatabaseAdapter : get(community_uid : Integer)

In order to get a community from database by using adapter layer class.

Context: CommunityManager:: createNetwork (self, instauser_uid : Integer):

Pre: CommunityManager : get(community_uid : Integer)

Post: Network::__init__(community : Community, networkx : networkx, positions : dict)

In order to get a community from database by using adapter layer class.

InstaUser Manager Class

Context: InstauserManager:: get (self, instauser_uid : Integer):

Post: InstaUserAdapter : get(community_uid : Integer)

In order to get a instauser from database by using adapter layer class.

Network Manager

Context: NetworkManager:: get (self, community_uid : Integer):

Post: NetworkAdapter : get(community_uid : Integer)

In order to get a network from database by using adapter layer class.

InstaConnection Class

Context: InstaConnection:: refresh (self):

Post: InstagramApi::__init__(access_token : String)

Retry to connect Instagram api.

Context: InstaConnection:: follows (self, instauser_uid: Integer):

Post: InstagramApi::user_follows(instauser_uid)

Return list of object of InstagramUser Package.

Context: InstaConnection:: user (self, instauser_uid : Integer):

Post: InstagramApi::user(instauser_uid)

Return object of InstagramUser Package.

EmojiDict Class

Context: EmojiDict:: add_emoji (self, text : String):

Post: emoji_basket.append(emoji)

Add emojis to a list and return them.

Context: EmojiDict:: extract_word (self, text : String):

Post: temp_word.decode(encoding='unicode_escape')

Clear word from emojis and other redundant meanless pieces.

Context: EmojiDict:: contains_emoji (self, text : String):

Post: if temp_word in self.emoji_list: return True

Return true if word contains emoji.

Analysis Thread Class

Context: AnalysisThread:: run (self):

Pre: AnalysisFactory:get_last_analysis() : AnalysisController

Post: AnalysisController:do(self)

AnalysisThread run gets analysis from database and run it in a thread.

Context: AnalysisThread:: stop (self):

Post: self._status = Status.Finished

AnalysisThread stop method stop the thread by a Enum.

6. Conclusions and Lessons Learned

Instagram Data Analysis Application is a data-analyzing program that can make data analysis and draw graphs among the user interactions. Users are able to see their community and thus, users social interaction increases because they can see followers and likes.

The first step was the project analysis. In this step, we explained requirements analysis and analysis models. In the requirements analysis, functional and non functional requirements, overview of the project, constraints, scenarios, use case models and user interfaces are included. Along with requirements analysis, analysis object model and dynamic model of the system are used in the analysis models part. Analysis part divides two parts such as object model and dynamic model. Object model include domain lexicon and class diagrams. Dynamic models include state charts and sequence diagrams. Second step, system design of the project occurs. This part includes design goals, subsystem decomposition, architectural patterns and other system design activities such as hardware/software mapping, persistent data management handling, access control and security, global software control and boundary conditions. We use many nonfunctional requirements to create effective design. Lastly, we use pattern applications such as façade, observer and composite. Using these patterns, we occur extendibility, reusability and maintainability system. We use python language in this project. We have known Python language but we didn't know fine details in Python language. With this project, we have an opportunity to improve ourselves.

During this project, we have gained many experience and knowledge about how to design project. First of all, we learn with together without knowing each other. People who do not know each other work together in these design projects in work life. We gain experience about this situation. We learn how to share project parts in this project. Sometimes, we have to face some problems about design project and overcome from these problems with critical and analytical thinking

Quality of the design has a big impact on implementation. If people prepare detailed design, part of implementation can be so easy for them. We

show to pay strict attention while we are making design. Therefore, our project implementation works properly.