

A Project Report

On

“Twitter Wizard”

Submitted in partial fulfillment of the requirement of
University of Mumbai

For the Degree of
Bachelor of Science

Submitted By
**Vrutika
Gaikwad**

Under the Guidance of
**Prof. Priyanka
Gawhane**

Final Year Computer Science



Department of B.Sc. Computer Science
**Ramniranjan Jhunjhunwala College of Arts, Science &
Commerce**

Affiliated to University of Mumbai,
Mumbai
(2019-2020)



Ramniranjan Jhunjhunwala College
Ghatkopar(W), Mumbai 400-086

CERTIFICATE

This is to certify that *Vrutika Gaikwad* has

successfully completed the project titled

“Twitter Wizard”

under our guidance towards the partial fulfillment of degree of Bachelors
of Science (Computer Science – SEM VI) submitted to Ramniranjan
Jhunjhunwala College, Ghatkopar (W) during the academic year 2018 –
2019 as specified by UNIVERSITY OF MUMBAI.

Prof. Priyanka Gawhane
Project Guide
Dept. of Computer Science

Prof. Anita Gaikwad
In-Charge
Dept. of Computer Science

Index

1. Acknowledgement	5
2. Declaration	6
3. Feasibility Study	7
3.1.1. Technical Feasibility	7
3.1.2. Economic Feasibility	7
3.1.3. Operational Feasibility	7
3.1.4. Social Feasibility	8
4. Scope of the System	9
5. Existing System and it's Disadvantages	10
6. Proposed System and Its Advantages	10
7. Requirements	11
8. Software Development Model	12
9. Gantt Chart	17
10. System Analysis	21
10.1.1. E-R Diagram	21
10.1.2. Class Diagram	23
10.1.3. Object Diagram	25
10.1.4. Activity Diagram	27
10.1.5. Sequence Diagram	29
10.1.6. Use Case Diagram	32
11. System Design	34
11.1.1. Component Diagram	34
11.1.2. Deployment Diagram	36
11.1.3. Table Design	38
12. System Implementation	39
12.1. System Coding	39
12.1.1. design.kv	39
12.1.2. main.py	41
12.1.3. visualizer.py	44

12.1.4. analyzer.py	47
12.1.5. streamer.py	48
12.2. Screen Layouts	51
13. System Testing	54
13.1. White Box Testing	54
13.2. Black Box Testing	56
14. Future Scope	62
15. References	62

Acknowledgement

I have great pleasure for representing this project report entitled “**Twitter Wizard**” and I grab this opportunity to convey my immense regards towards all the distinguished people who have their valuable contribution in the hour of need.

I take this opportunity to thank “**Dr. Mrs. Usha Mukundan**”, **Principal of Ramniranjan Jhunjhunwala college, Ghatkopar(W)** for giving me an opportunity to study in the institute and the most needed guidance throughout the duration of the course.

I also like to extend my gratitude to “**Prof. Anita Gaikwad**”, Head of the Department for their timely and prestigious guidance.

I would also like to thank “**Prof. Priyanka Gawhane**” provided the guidance and necessary support during each phase of the project.

I also owe to my fellow friends who have been constant source of help to solve the problems and also helped me during the project development phase.

Thanking You,
Vrutika Gaikwad

Declaration

I declare that this written submission represents my own ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/fact/data/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Vrutika Gaikwad

Feasibility Study

Feasibility study is to check the viability of the project under consideration. Theoretically various types of feasibilities are conducted, but I have conducted four types of feasibilities explained as under.

Technical Feasibility:

- This system do not require much maintenance work. The only issue is that it needs a good bandwidth.
- An intermediate skilled individual can maintain it. Upgrades and updates are always welcomed since this desktop application can be used in different sectors of industries.
- Since this desktop application needs to search through all tweets based on user inputs, one can develop an algorithm for quick searching.

Economic Feasibility:

- As previously mentioned, this desktop application needs decent amount of bandwidth, which in turn tells us the only thing where we need to invest our money into, a decent internet connection.
- For this desktop application to run smoothly, an estimated price of 200-300 INR is required per month.

Operational Feasibility:

- This tool has wide range of applications. For example, we can get the sentiments of people all over the world on specific individual, organizations, etc. And use

them to build a proper business strategy to optimize and achieve the result.

- It can be used as polling in elections for specific individual or alliance.
- It can help people form views on various current affairs or global problems like Climate Change or Global Warming, etc.

Social Feasibility:

- This tool will be useful for people to form opinion(s) about products, organizations, etc. Thus there is no reason to make tool socially unfeasible.
- It is a good way to assess business or industrial organizations' behavior or response towards society

Scope of the system

The work and resources that go into the creation of the product or service are essentially the things that frame the scope of the project. The scope of the project outlines the objectives of the project and the goals that need to be met to achieve a satisfactory result.

The desktop application would fetch the data from twitter using it's API also known as tweepy. This data is in json format which is extracted according to the need. For my project I have fetched metadata of the tweet. Since the desktop application is using ready-made API from twitter then that API will be out of scope of this project. When these data are fetched they are fed into pandas data frame, and finally all this data are pulled into the desktop application and processed according to the user's input. The desktop application and back-end processing is solely developed by me.

This desktop application helps in building business strategies for individual, organizations by giving them the analysis of peoples' view on the said individual, organizations. The graphical representation of the processed data is more helpful to form a quick overview of the said individual, organizations.

Existing System and it's Disadvantages

Most of the System that are developed tend to give crude analysis that are basically the processed data being classified in positive or negative. Taking inspiration from one of the desktop application i.e., SocioSense, it helped me to visualize this desktop application. SocioSense even though provides better GUI then this desktop application, lacks the detailed analysis.

Proposed System and Its Advantages

- The proposed system processes the real-time tweets.
- When a user inputs a term in search field it is passed as key to Tweet streamer and then streamed tweets are put into data frame. Then based on user's input graphs are displayed.
- User can also perform analysis of individual twitter user by providing username. When username is provided the tweets of that user's home timeline will be fetched and put into a data frame.
- I have used pandas library for optimized search and management of the data.
- User can view sentiment analysis in pie or line graph for searched keyword tweet.
- It has easy to use and easy to understand GUI.
- It doesn't saves the streamed Twitter data the saves memory space in the system.

Requirements

- **Programming Language:**
 - Python
- **Software Requirements:**
 - Linux/Windows OS
 - Python3+
 - Python libraries
 - kivy
 - pandas
 - tweepy
 - matplotlib
 - nltk
 - textblob
 - PyCharm
- **Hardware Requirements:**
 - Generic Mouse, keyboard, monitor, CPU, etc.
- **Network Connectivity:**
 - A better and fast network is required to implement this project, since we are downloading the real time tweets. Network with speed of minimum 1mbps is recommended.

Software Development Model

V-Model

The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

Phases of V-model:

a. Requirement Analysis

This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business

requirements can be used as an input for acceptance testing.

b. System Design

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

c. Architectural Design

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

d. Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**. It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development

process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

e. Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

f. Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

i. Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

ii. Integration Testing

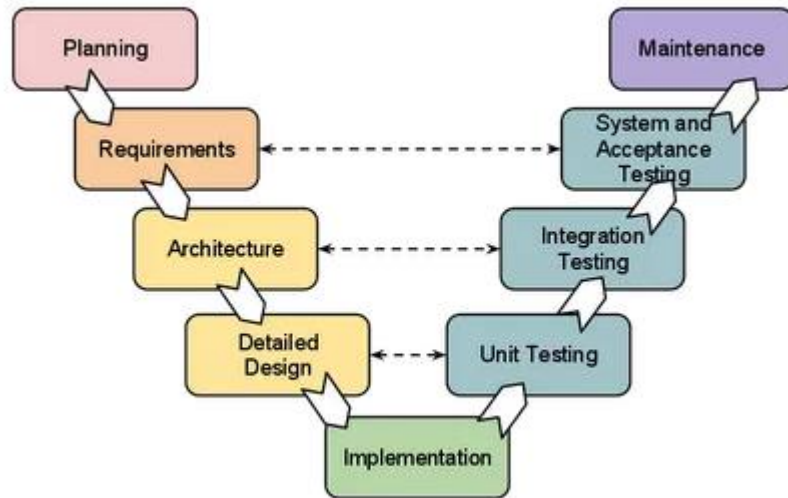
Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

iii. System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

iv. Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.



Gantt Chart

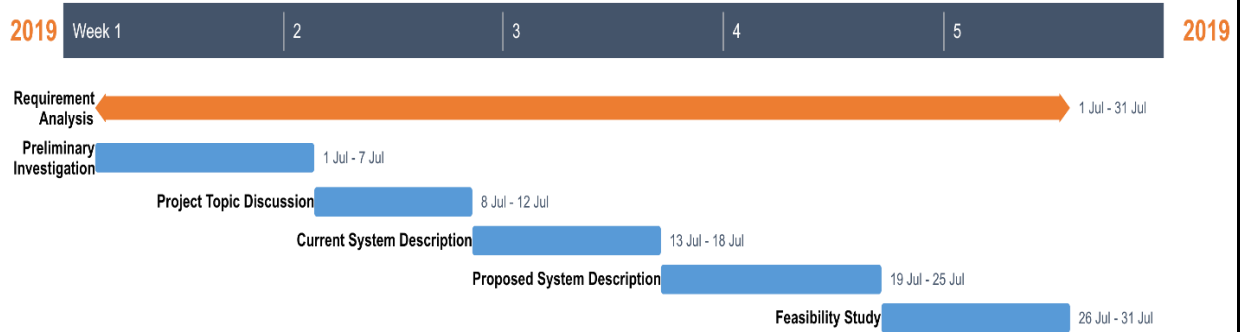
A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities(tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

- What the various activities are
- When each activity begins and ends
- How long each activity is scheduled to last
- Where activities overlap with other activities, and by how much
- The start and end date of the whole project

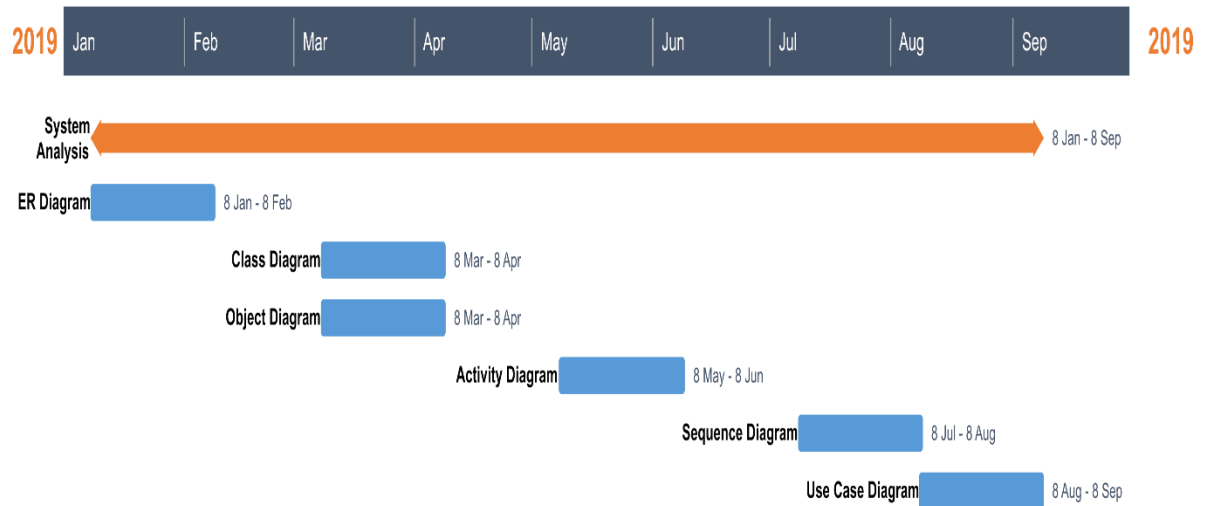
To summarize, a Gantt chart shows you what has to be done(the activities) and when(the schedule).

<u>Title</u>	<u>Start date</u> (mm-dd-yyyy)	<u>End date</u> (mm-dd-yyyy)	<u>Duration</u> (in days)
Requirement Analysis			
Preliminary Investigation	07-01-2019	07-07-2019	5
Project Topic Discussion	07-08-2019	07-13-2019	5
Current System Description	07-13-2019	07-18-2019	4
Proposed System Description	07-19-2019	07-25-2019	5
Feasibility Study	07-26-2019	07-31-2019	4
System Analysis			
ER Diagram	08-01-2019	08-02-2019	2
Class Diagram	08-03-2019	08-04-2019	1
Object Diagram	08-03-2019	08-04-2019	1
Activity Diagram	08-05-2019	08-06-2019	2
Sequence Diagram	08-07-2019	08-08-2019	2
Use Case Diagram	08-08-2019	08-09-2019	2
System Design			
Component Diagram	08-08-2019	08-09-2019	2
Deployment Diagram	08-12-2019	08-13-2019	2
Table Design	08-14-2019	08-15-2019	2
System Implementation			
System Coding	09-17-2019	01-01-2020	106
White Box Testing	09-22-2019	01-08-2020	108
Black Box Testing	01-08-2020	01-20-2020	13

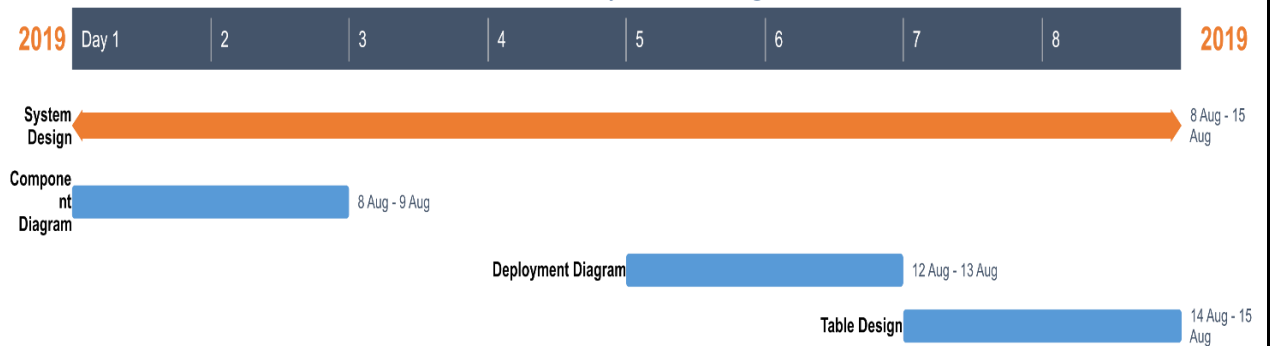
Gantt Chart 1: Requirement Gathering



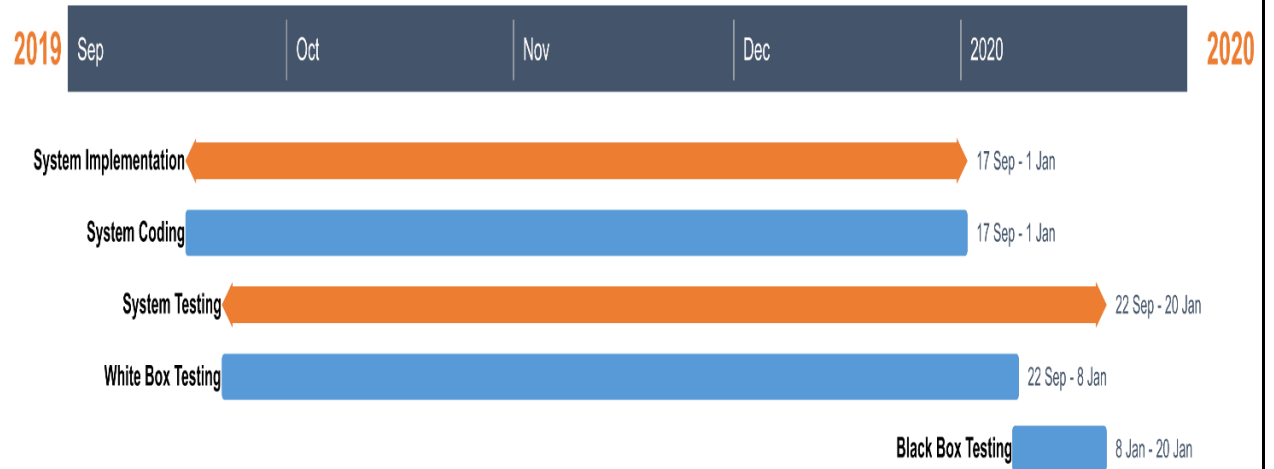
Gantt Chart 2: System Analysis



Gantt Chart 3: System Design



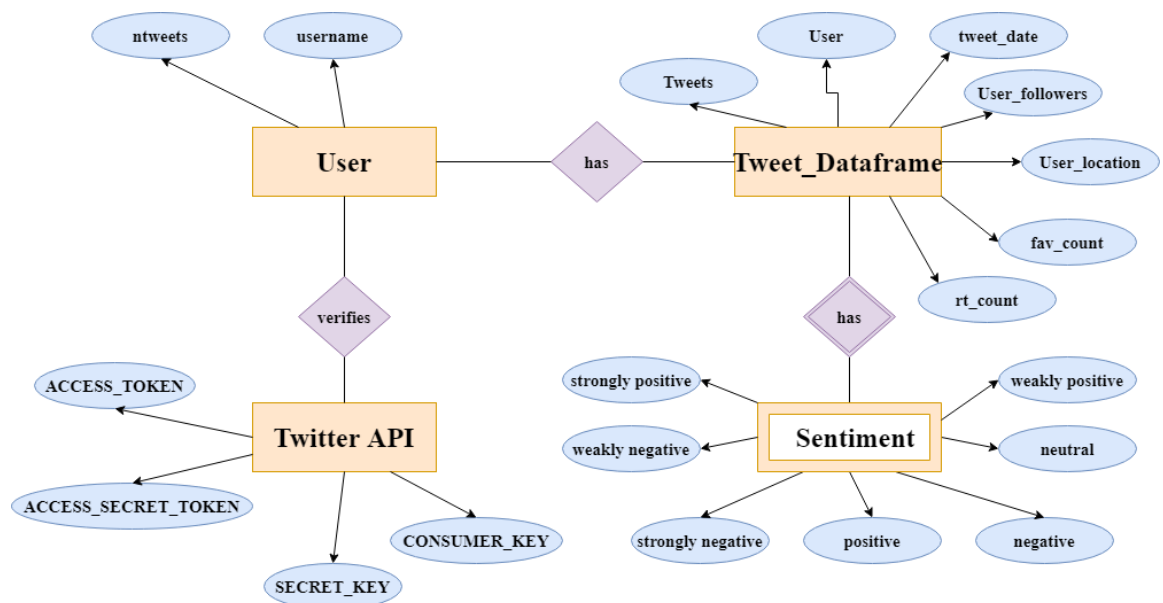
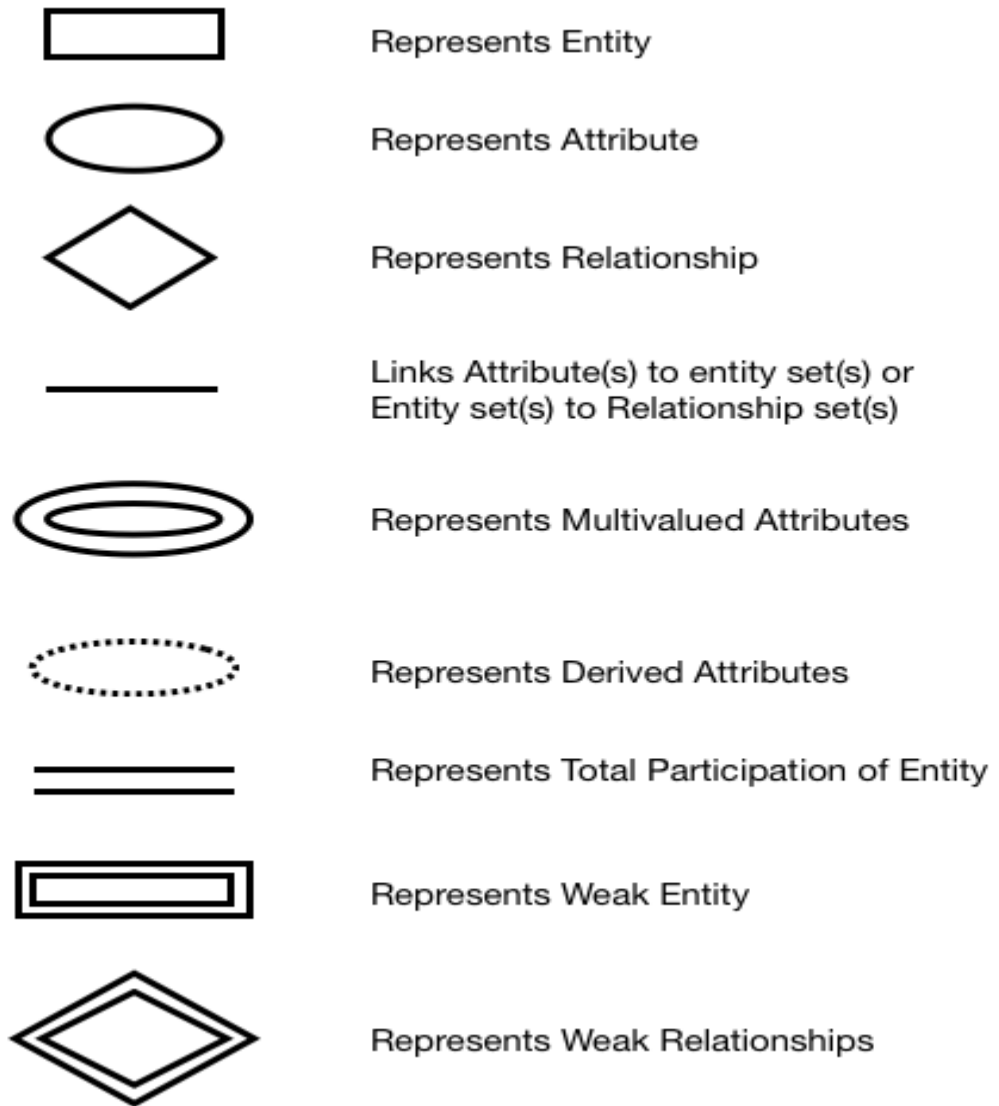
Gantt Chart 4: System Implementation and Testing



System Analysis

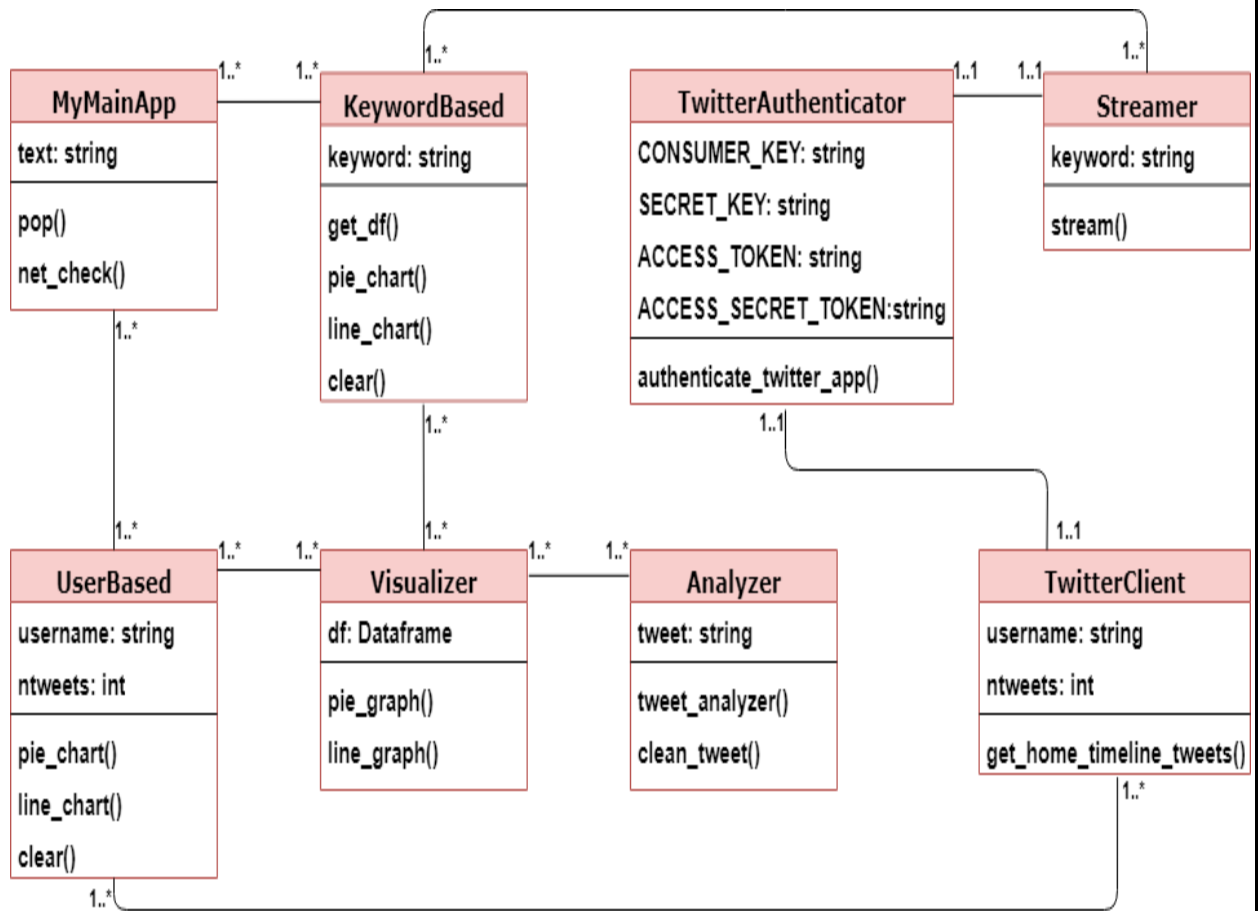
E-R Diagram

- An entity-relationship diagram (ERD) is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. An ERD is a conceptual and representational model of data used to represent the entity framework infrastructure.
- The elements of an ERD are:
 - Entities
 - Relationships
 - Attributes
- Steps involved in creating an ERD include:
 - Identifying and defining the entities
 - Determining all interactions between the entities
 - Analyzing the nature of interactions/determining the cardinality of the relationships
 - Creating the ERD
- Symbols used in ER Diagram:



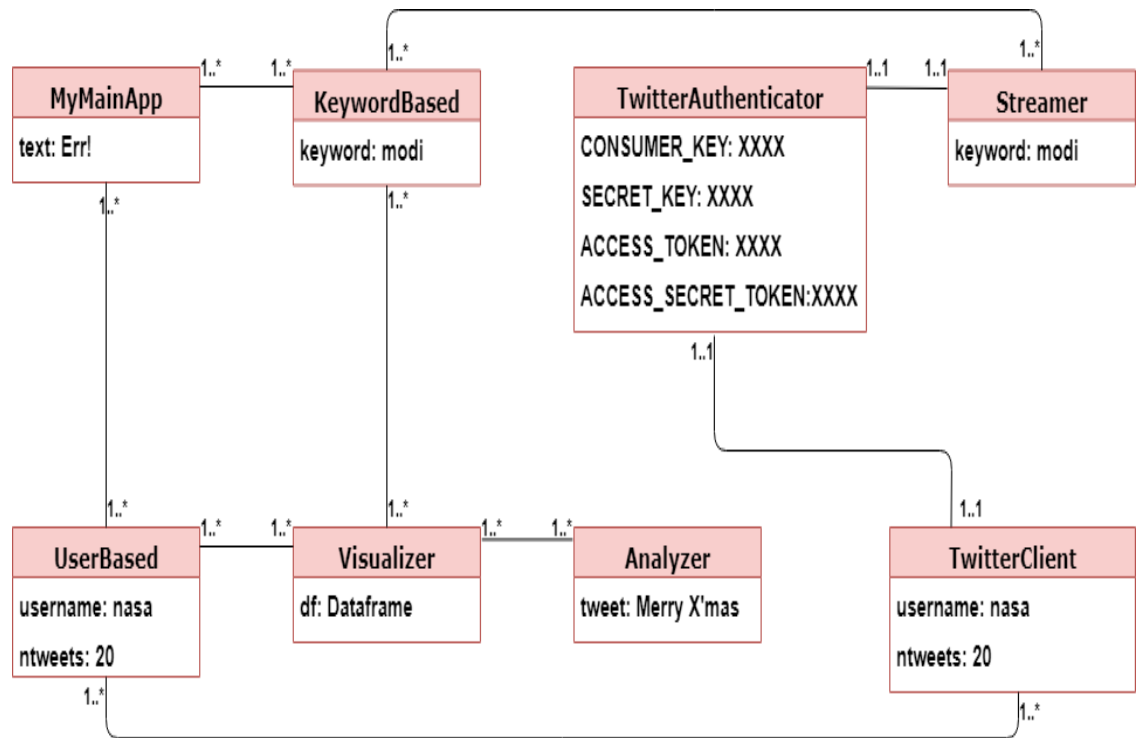
Class Diagram

- A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML).
- In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity.
- Class diagrams are useful in all forms of object-oriented programming (OOP). The concept is several years old but has been refined as OOP modeling paradigms have evolved. In a class diagram, the classes are arranged in groups that share common characteristics.
- A class diagram resembles a flowchart in which classes are portrayed as boxes, each box having three rectangles inside. The top rectangle contains the name of the class; the middle rectangle contains the attributes of the class; the lower rectangle contains the methods, also called operations, of the class.
- Lines, which may have arrows at one or both ends, connect the boxes. These lines define the relationships, also called associations, between the classes.



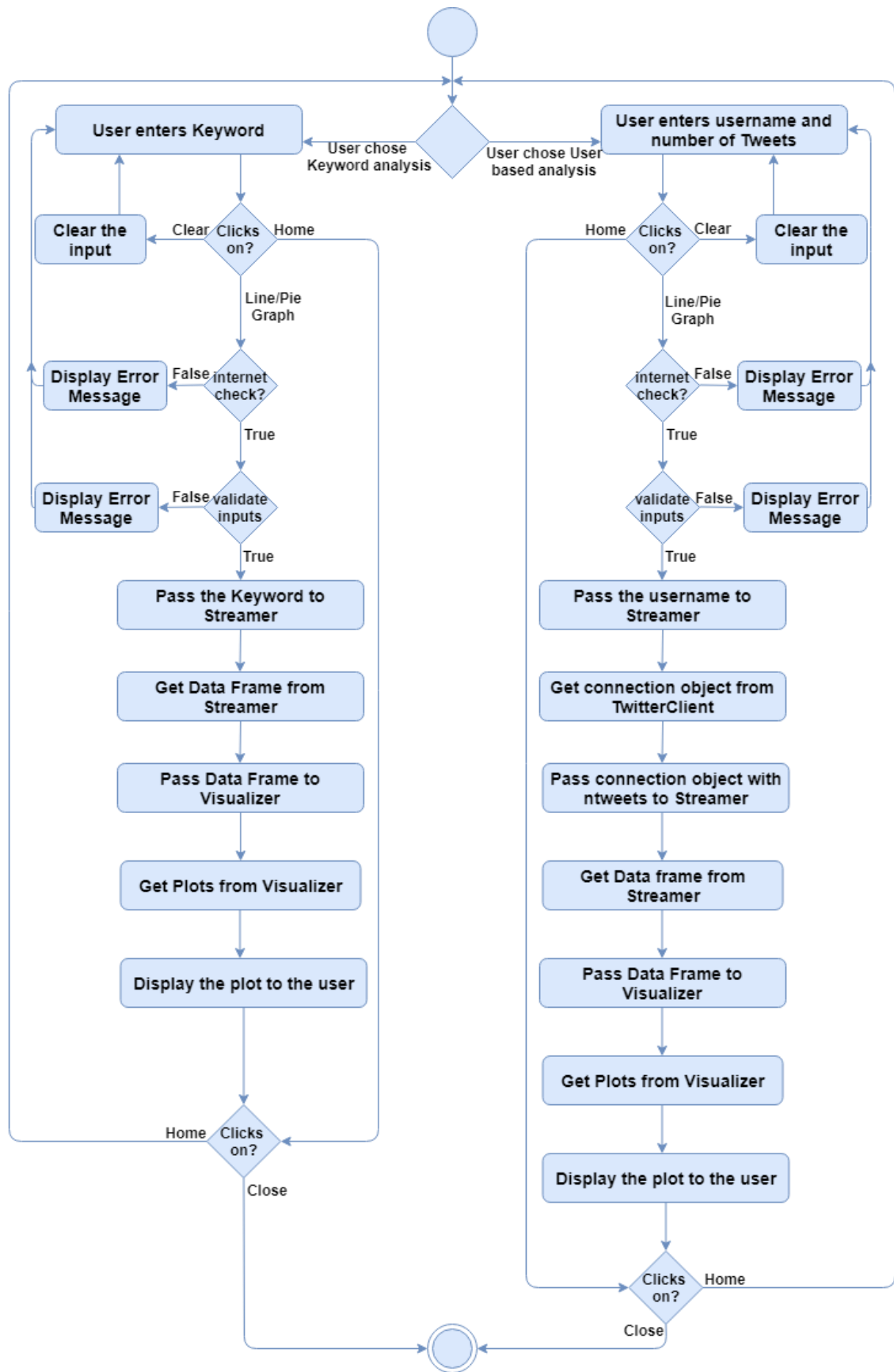
Object Diagram

- Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.
- Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams.
- Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment. Object diagrams are used to render a set of objects and their relationships as an instance.
- The purpose of a diagram should be understood clearly to implement it practically. The purposes of object diagrams are similar to class diagrams.
- The purpose of the object diagram can be summarized as –
 - Forward and reverse engineering.
 - Object relationships of a system
 - Static view of an interaction.
 - Understand object behavior and their relationship from practical perspective



Activity Diagram

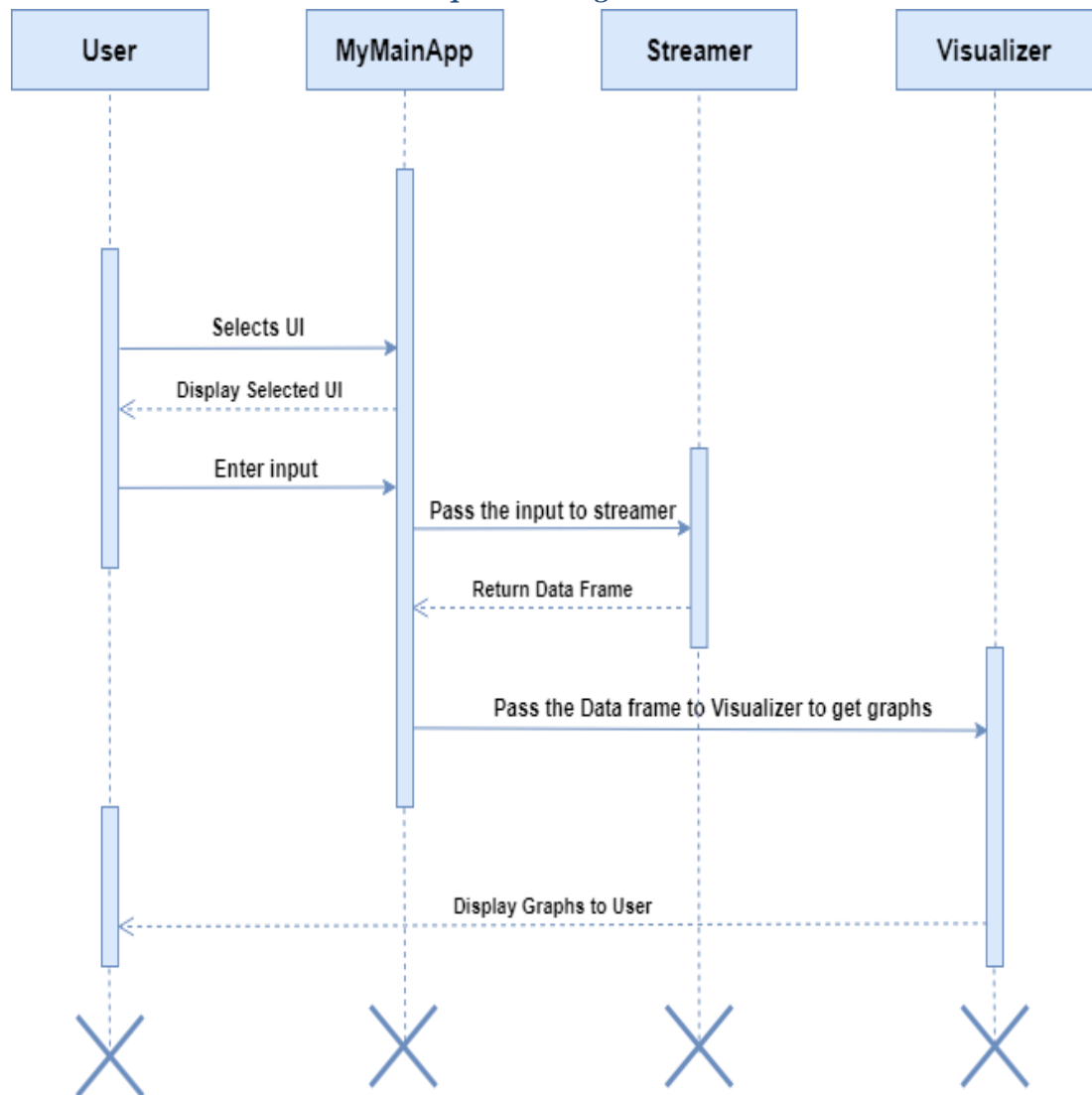
- We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.
- UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams. An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system.
- An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modeling where their primary use is to depict the dynamic aspects of a system.



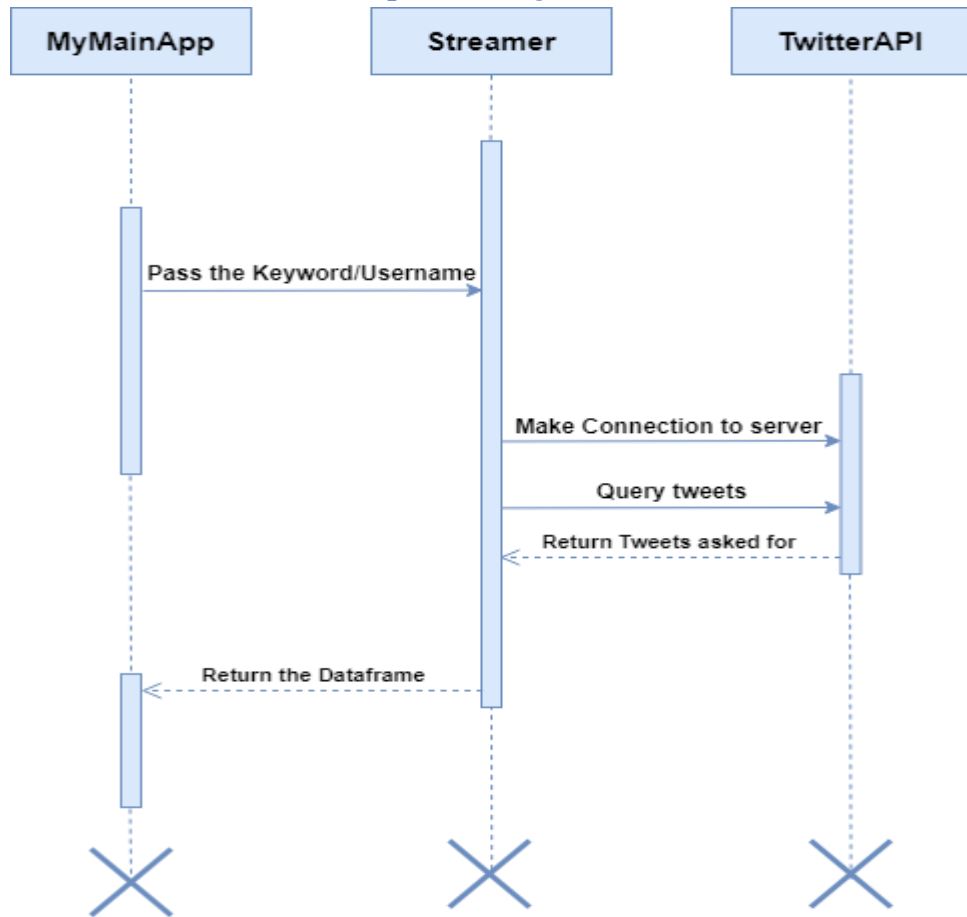
Sequence Diagram

- Unified Modelling Language (UML) is a modeling language in the field of software engineering which aims to set standard ways to visualize the design of a system. UML guides the creation of multiple types of diagrams such as interaction , structure and behavior diagrams.
- A sequence diagram is the most commonly used interaction diagram.
- **Interaction diagram** –An interaction diagram is used to show the interactive behavior of a system. Since visualizing the interactions in a system can be a cumbersome task, we use different types of interaction diagrams to capture various features and aspects of interaction in a system.
- **Sequence Diagrams** –A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

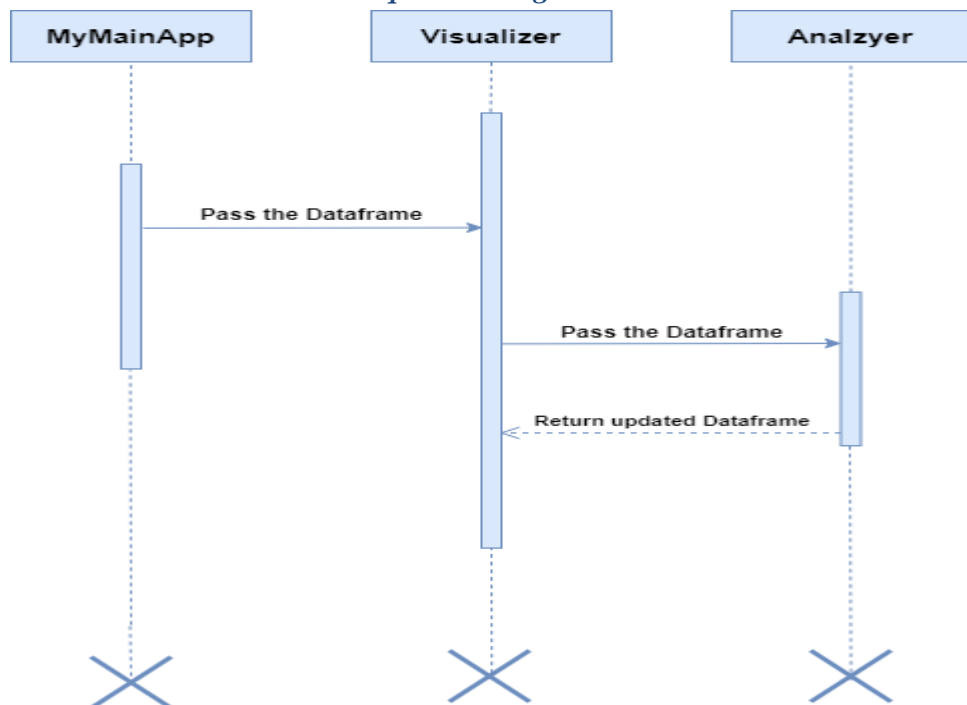
Sequence Diagram 1



Sequence Diagram 2

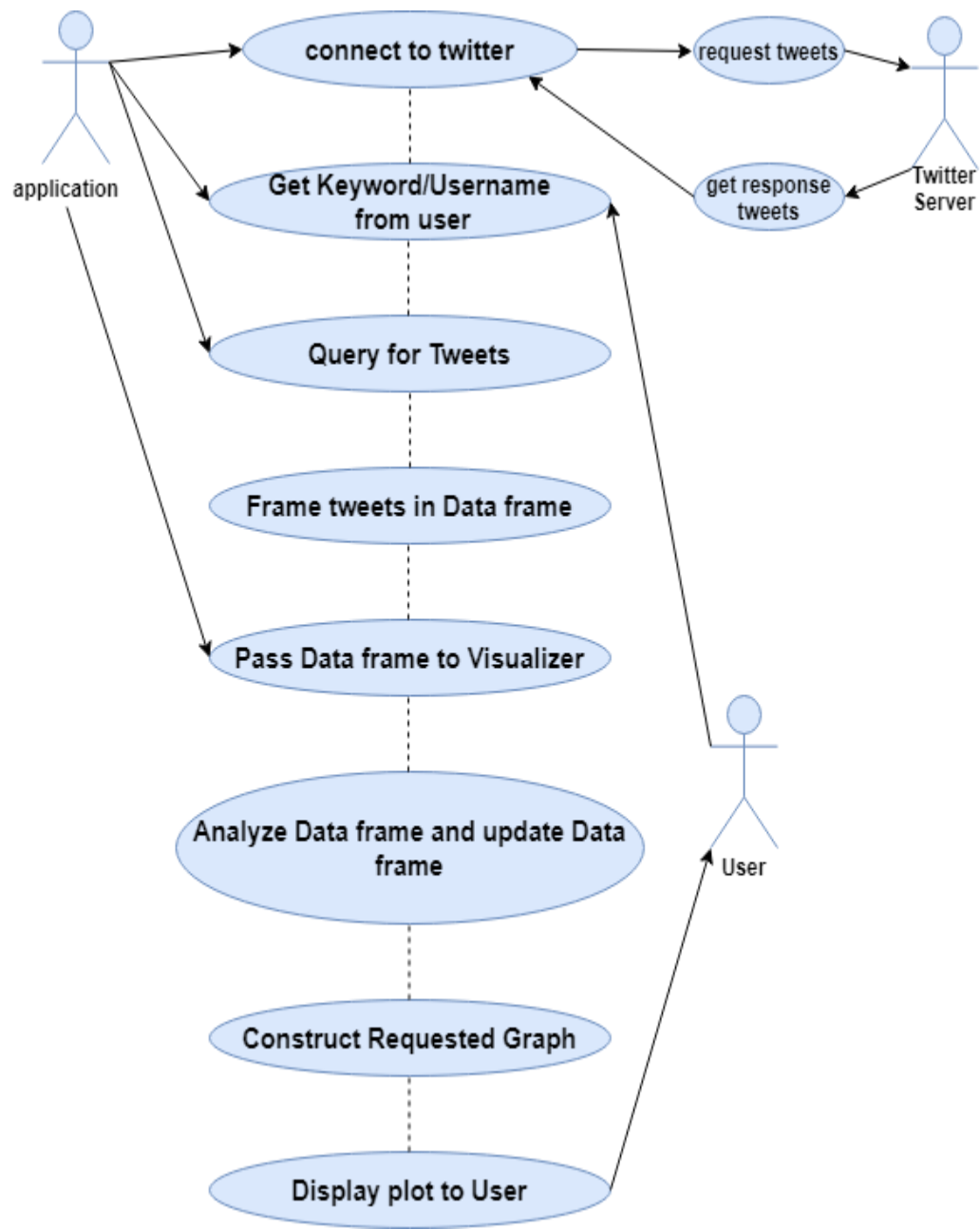


Sequence Diagram 3



Use-Case Diagram

- A use case diagram is a graphic depiction of the interactions among the elements of a system.
- A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to something being developed or operated, such as a mail-order product sales and service Web site. Use case diagrams are employed in UML (Unified Modeling Language), a standard notation for the modeling of real-world objects and systems.
- System objectives can include planning overall requirements, validating a hardware design, testing and debugging a software product under development, creating an online help reference, or performing a consumer-service-oriented task. For example, use cases in a product sales environment would include item ordering, catalog updating, payment processing, and customer relations. A use case diagram contains four components.
 - The boundary, which defines the system of interest in relation to the world around it.
 - The actors, usually individuals involved with the system defined according to their roles.
 - The use cases, which are the specific roles played by the actors within and around the system.
 - The relationships between and among the actors and the use cases.



Component Diagram

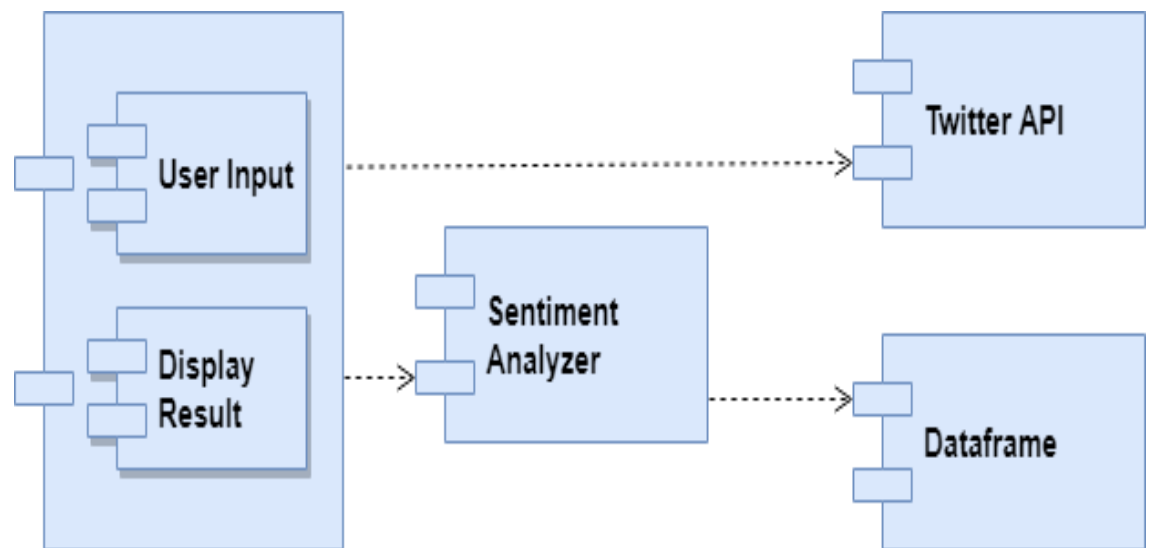
- Component Diagrams describe the organization of components, including source code, run-time (binary) code, and executable.

Component Diagrams:

- Give the physical view of the system in terms of implementation aspect. This is important for reusability and performance purpose.
- Constitute the Components, their interfaces and realizations, and dependencies between components.
- Component Diagrams are used:
 - To depict organizations and dependencies among Component type.
 - To show allocation of “Classes” and “objects” to components in the physical design of the system.
 - To indicate the “physical layering” and “partitioning” of the system Architecture.

A component typically encompasses:

- Structure and behavior of a “Collaboration of classes” from the system design.
- Interfaces that describe a group of operations implemented by components.



Deployment Diagram

- Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets.
- Artifacts represent concrete elements in the physical world that are the result of a development process. Examples of artifacts are executable files, libraries, archives, database schemas, configuration files, etc.
- Deployment target is usually represented by a node which is either hardware device or some software execution environment. Nodes could be connected through communication paths to create networked systems of arbitrary complexity.
- Note, that components were directly deployed to nodes in UML 1.x deployment diagrams. In UML 2.x artifacts are deployed to nodes, and artifacts could manifest (implement) components. Components are deployed to nodes indirectly through artifacts.
- Deployment diagrams could describe architecture at specification level (also called type level) or at instance level (similar to class diagrams and object diagrams).
- Specification level deployment diagram shows some overview of deployment of artifacts to deployment targets, without referencing specific instances of artifacts or nodes.

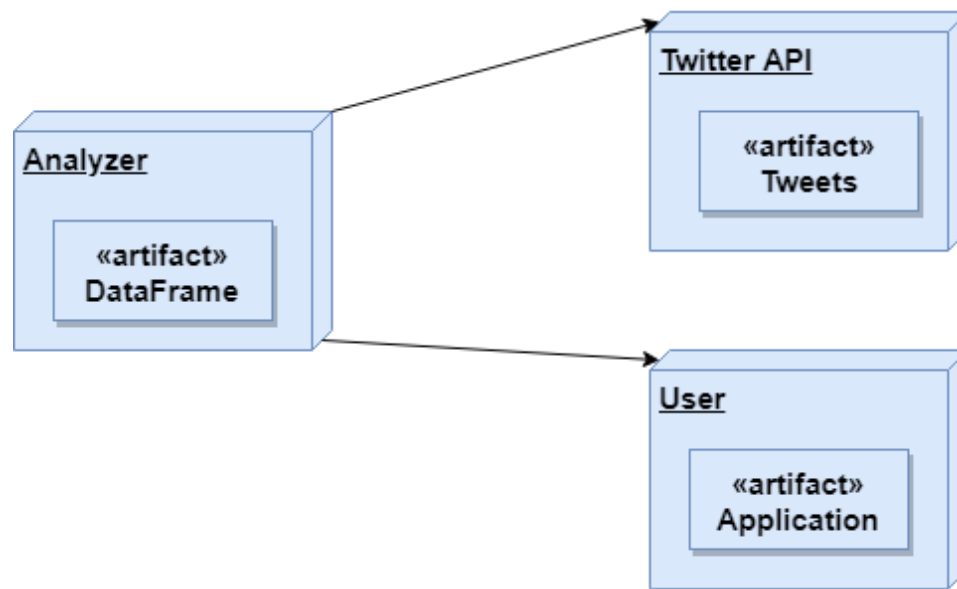


Table Design

Frame for storing tweets:

Sr. No.	Attribute	Data Type	Description
1.	Tweets	string	Text of the tweet
2.	User	string	Name of user who tweeted the tweet
3.	User_followers	int	Number of followers
4.	User_location	int	Location from which tweet was treated
5.	User_verified	boolean	Whether the user is a verified user or not
6.	fav_count	int	Number of likes received for the tweet
7.	rt_count	int	Number of times tweet is retweeted
8.	tweet_date	date	Date on which tweet was tweeted

System Implementation

System Coding

design.kv

WindowManager:

Home:

UserBased:

KeywordBased:

<Home>:

name: "home"

GridLayout:

cols:1

Button:

text:"User Based Analysis"

on_press:

app.root.current = "userbased"

root.manager.transition.direction = "left"

Button:

text:"Keyword Based Analysis"

on_press:

app.root.current = "keywordbased"

root.manager.transition.direction = "left"

<UserBased>:

name: "userbased"

username : username

ntweets : ntweets

GridLayout:

cols:1

GridLayout:

cols:2

Label:
text: "Username"

TextInput:
id:username
multiline: False

Label:
text: "Number of Tweets"

TextInput:
id: ntweets
multiline: False

Button:
text: "Pie Graph"
on_press: root.pie_chart()

Button:
text: "Line Graph"
on_press:root.line_chart()

Button:
text: "Clear"
on_press:
root.clear()

Button:
text: "Home"
on_press:
root.clear()
app.root.current = "home"
root.manager.transition.direction = "right"

<KeywordBased>:
name: "keywordbased"
keyword: kword

GridLayout:
cols: 1


```

GridLayout:
    cols: 2
    Label:
        text: "Keyword"
    TextInput:
        id: kword
        multiline: False

    Button:
        text: "Pie Graph"
        on_press: root.pie_chart()

    Button:
        text: "Line Graph"
        on_press: root.line_chart()

    Button:
        text: "Clear"
        on_press:
            root.clear()

    Button:
        text: "Home"
        on_press:
            root.clear()
            app.root.current = "home"
            root.manager.transition.direction = "right"

```

main.py

```

from urllib.request import urlopen
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
from kivy.properties import ObjectProperty
from kivy.uix.popup import Popup
from kivy.uix.screenmanager import Screen, ScreenManager
from visualizer import Visualizer
from streamer import Streamer, TwitterClient

```

```

from matplotlib.pyplot import show

def pop(text):
    pw = Popup(title="Err..!", content=Label(text=text),
size_hint=(None, None), size=(400, 400))
    pw.open()

def net_check():
    try:
        urlopen("https://www.google.com/")
        return True
    except:
        return False

class WindowManager(ScreenManager):
    pass

class Home(Screen):
    pass

class UserBased(Screen):
    username = ObjectProperty(None)
    ntweets = ObjectProperty(None)

    def pie_chart(self):
        if net_check():
            if self.username.text == "":
                pop("Please Enter A User Name!")
            elif self.ntweets.text == "":
                pop("Please Enter Number Of tweets to load!")
            elif not self.ntweets.text.isnumeric():
                pop("Please Enter Numbers Only!")
            else:
                tc = TwitterClient(self.username.text)
                df =
tc.get_home_timeline_tweets(int(self.ntweets.text))
                plt = Visualizer.pie_graph(df)

```

```

        plt.show()
    else:
        pop("Please check your Internet")

def line_chart(self):
    if net_check():
        if self.username.text == "":
            pop("Please Enter A User Name!")
        elif self.ntweets.text == "":
            pop("Please Enter Number Of tweets to load!")
        elif not self.ntweets.text.isnumeric():
            pop("Please Enter Numbers Only!")
        else:
            tc = TwitterClient(self.username.text)
            df =
tc.get_home_timeline_tweets(int(self.ntweets.text))
            Visualizer.line_graph(df)
            #plt.show()
    else:
        pop("Please check your Internet...")

def clear(self):
    self.username.text = ""
    self.ntweets.text = ""

class KeywordBased(Screen):

    keyword = ObjectProperty(None)

    def clear(self):
        self.keyword.text = ""

    def get_df(self):
        df = Streamer.stream(self.keyword.text)
        return df

    def pie_chart(self):
        if net_check():
            if self.keyword.text == "":
                pop("Please Enter A Keyword!")
            else:

```

```

        plt = Visualizer.pie_graph(self.get_df())
        plt.show()
    else:
        pop("Please check your Internet...")

def line_chart(self):
    if net_check():
        if self.keyword.text == "":
            pop("Please Enter A Keyword!")
        else:
            Visualizer.line_graph(self.get_df())
            #plt.show()
    else:
        pop("Please check your Internet...")

kv = Builder.load_file("design.kv")

class MyMainApp(App):
    def build(self):
        return kv

if __name__ == "__main__":
    MyMainApp().run()

```

visualizer.py

```

import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import style
import pandas as pd
import numpy as np
from analyzer import Analyzer as ay

def percentage(part, whole):
    temp = 100 * float(part) / float(whole)
    return format(temp, '.2f')

```

```

class Visualizer():

    def __init__(self):
        pass

    def pie_graph(df):
        polarity = 0
        positive = 0
        wpositive = 0
        spositive = 0
        negative = 0
        wnegative = 0
        snegative = 0
        neutral = 0
        sentiment_array =
np.array([ay.tweet_analyzer(ay.clean_tweet(tweet)) for tweet in
df['Tweets']])
        # print(sentiment_array)

        for i in range(len(sentiment_array)):
            if sentiment_array[i] == "neutral":
                neutral += 1
            elif sentiment_array[i] == "wpositive":
                wpositive += 1
            elif sentiment_array[i] == "positive":
                positive += 1
            elif sentiment_array[i] == "spositive":
                spositive += 1
            elif sentiment_array[i] == "wnegative":
                wnegative += 1
            elif sentiment_array[i] == "negative":
                negative += 1
            elif sentiment_array[i] == "snegative":
                snegative += 1

        positive = percentage(positive, len(sentiment_array))
        wpositive = percentage(wpositive, len(sentiment_array))
        spositive = percentage(spositive, len(sentiment_array))
        negative = percentage(negative, len(sentiment_array))
        wnegative = percentage(wnegative, len(sentiment_array))
        snegative = percentage(snegative, len(sentiment_array))

```

```

neutral = percentage(neutral, len(sentiment_array))

labels = ['Positive ' + str(positive) + '%', 'Weakly Positive ' +
str(wpositive) + '%',
          'Strongly Positive ' + str(spositive) + '%', 'Neutral ' +
str(neutral) + '%',
          'Negative ' + str(negative) + '%', 'Weakly Negative ' +
str(wnegative) + '%',
          'Strongly Negative ' + str(snegative) + '%']
sizes = [positive, wpositive, spositive, neutral, negative,
wnegative, snegative]
colors = ['yellowgreen', 'lightgreen', 'darkgreen', 'gold', 'red',
'lightsalmon', 'darkred']
patches, texts = plt.pie(sizes, colors=colors, startangle=90)
plt.legend(patches, labels, loc="best")
plt.axis('equal')
plt.tight_layout()
#plt.show()
return plt

def line_graph(df):
    style.use("ggplot")
    fig = plt.figure()
    ax1 = fig.add_subplot(1, 1, 1)
    sentiment_array =
np.array([ay.tweet_analyzer(ay.clean_tweet(tweet)) for tweet in
df['Tweets']])

    def animate(i):
        xar = []
        yar = []
        zar = []
        x = 0
        y = 0
        z = 0

        for sentiment in sentiment_array:
            x += 1
            if sentiment == "positive" or sentiment == "wpositive"
or sentiment == "spositive":
                y += 1

```

```

        elif sentiment == "negative" or sentiment ==
"wnegative" or sentiment == "snegative":
            z += 1
        else:
            pass

        xar.append(x)
        yar.append(y)
        zar.append(z)

    ax1.clear()
    ax1.plot(xar, yar, zar)

ani = animation.FuncAnimation(fig, animate, interval=1000,
save_count=1000)
plt.show()

```

analyzer.py

```

import re
from vaderSentiment.vaderSentiment import
SentimentIntensityAnalyzer
from textblob import TextBlob

class Analyzer():

    def __init__(self):
        pass

    def clean_tweet(tweet):
        return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z
\t])|(\w+:\w+\S+)", " ", tweet).split())

    def tweet_analyzer(tweet):
        analysis = TextBlob(tweet)
        # print(analysis.sentiment) # print tweet's polarity
        # polarity += analysis.sentiment.polarity # adding up
        polarities to find the average later

        if (analysis.sentiment.polarity == 0): # adding reaction of
        how people are reacting to find average later

```

```

        return "neutral"
    elif (analysis.sentiment.polarity > 0 and
analysis.sentiment.polarity <= 0.3):
        return "wpositive"
    elif (analysis.sentiment.polarity > 0.3 and
analysis.sentiment.polarity <= 0.6):
        return "positive"
    elif (analysis.sentiment.polarity > 0.6 and
analysis.sentiment.polarity <= 1):
        return "spositive"

    elif (analysis.sentiment.polarity > -0.3 and
analysis.sentiment.polarity <= 0):
        return "wnegative"
    elif (analysis.sentiment.polarity > -0.6 and
analysis.sentiment.polarity <= -0.3):
        return "negative"
    elif (analysis.sentiment.polarity > -1 and
analysis.sentiment.polarity <= -0.6):
        return "snegative"

```

streamer.py

```

import tweepy
import pandas as pd
import twitter_credentials

class TwitterAuthenticator():

    def authenticate_twitter_app(self):
        auth =
tweepy.OAuthHandler(twitter_credentials.CONSUMER_KEY,
twitter_credentials.CONSUMER_SECRET)

auth.set_access_token(twitter_credentials.ACCESS_TOKEN,
twitter_credentials.ACCESS_TOKEN_SECRET)
        return auth

```



```

df = pd.DataFrame(columns=['Tweets', 'User',
'User_statuses_count', 'User_followers', 'User_location',
'User_verified', 'fav_count', 'rt_count', 'tweet_date'])
udf = pd.DataFrame(columns=['Tweets', 'User',
'User_statuses_count', 'User_followers', 'User_location',
'User_verified', 'fav_count', 'rt_count', 'tweet_date'])

class Streamer():

    def __init__(self):
        pass

    def stream(data):
        i = 0
        api =
tweepy.API(TwitterAuthenticator().authenticate_twitter_app())
        for tweet in tweepy.Cursor(api.search, q=data, count=100,
lang='en').items():
            #print(i)
            df.loc[i, 'Tweets'] = tweet.text
            df.loc[i, 'User'] = tweet.user.name
            df.loc[i, 'User_statuses_count'] =
tweet.user.statuses_count
            df.loc[i, 'User_followers'] = tweet.user.followers_count
            df.loc[i, 'User_location'] = tweet.user.location
            df.loc[i, 'User_verified'] = tweet.user.verified
            df.loc[i, 'fav_count'] = tweet.favorite_count
            df.loc[i, 'rt_count'] = tweet.retweet_count
            df.loc[i, 'tweet_date'] = tweet.created_at
            i+=1
            if i == 500:
                break
            else:
                pass
        return df

class TwitterClient():

    def __init__(self, twitter_user):
        self.auth = TwitterAuthenticator().authenticate_twitter_app()
        self.twitter_client = tweepy.API(self.auth)

```

```

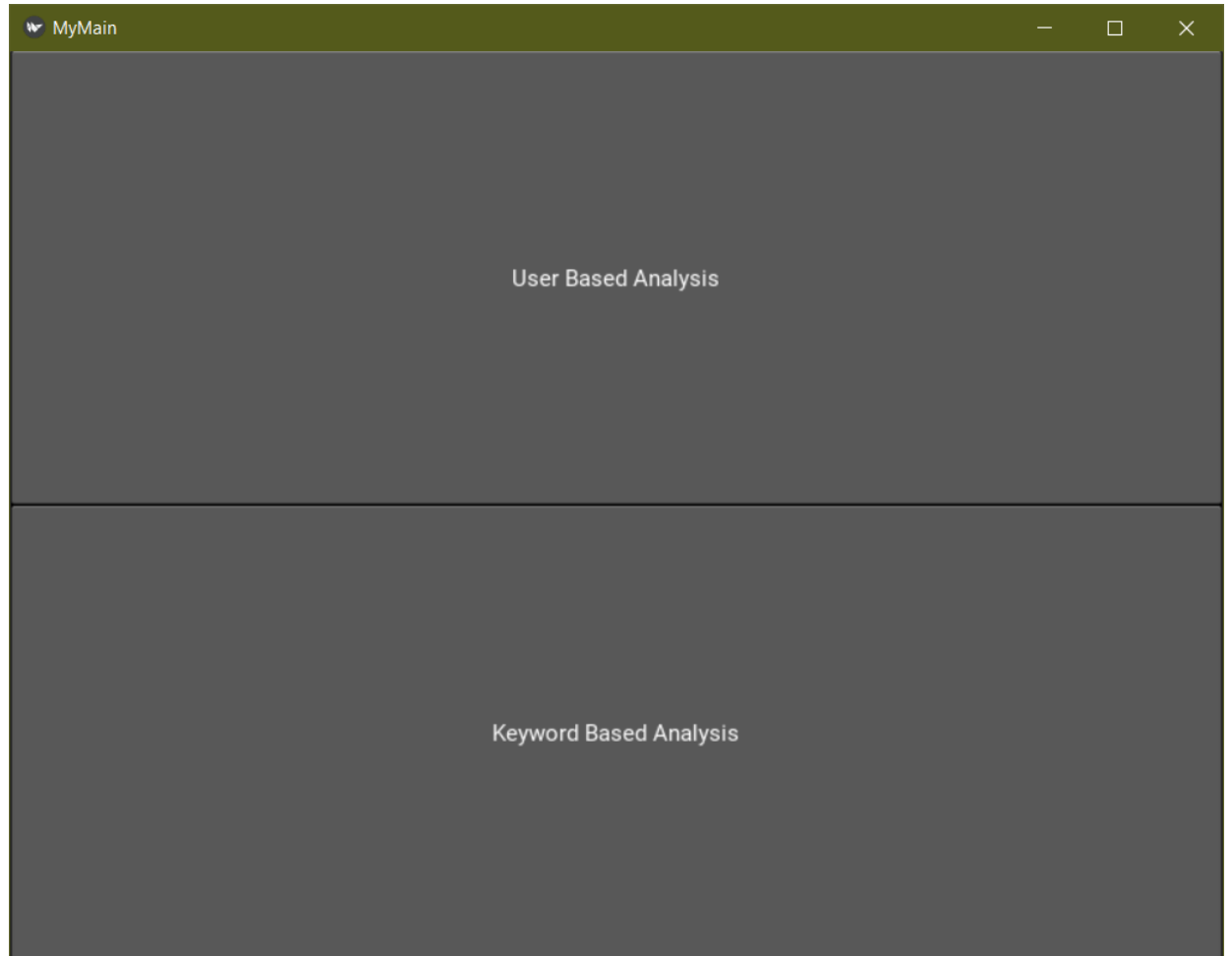
self.twitter_user = twitter_user

def get_home_timeline_tweets(self, num_tweets):
    i=0
    #home_timeline_tweets = []
    for tweet in tweepy.Cursor(self.twitter_client.user_timeline,
id=self.twitter_user).items(num_tweets):
        #home_timeline_tweets.append(tweet)
        #print(i)
        udf.loc[i, 'Tweets'] = tweet.text
        udf.loc[i, 'User'] = tweet.user.name
        udf.loc[i, 'User_statuses_count'] =
tweet.user.statuses_count
        udf.loc[i, 'User_followers'] = tweet.user.followers_count
        udf.loc[i, 'User_location'] = tweet.user.location
        udf.loc[i, 'User_verified'] = tweet.user.verified
        udf.loc[i, 'fav_count'] = tweet.favorite_count
        udf.loc[i, 'rt_count'] = tweet.retweet_count
        udf.loc[i, 'tweet_date'] = tweet.created_at
        i+=1
    if i == num_tweets:
        break
    else:
        pass
    #print(udf)
    #print(home_timeline_tweets)
    return udf

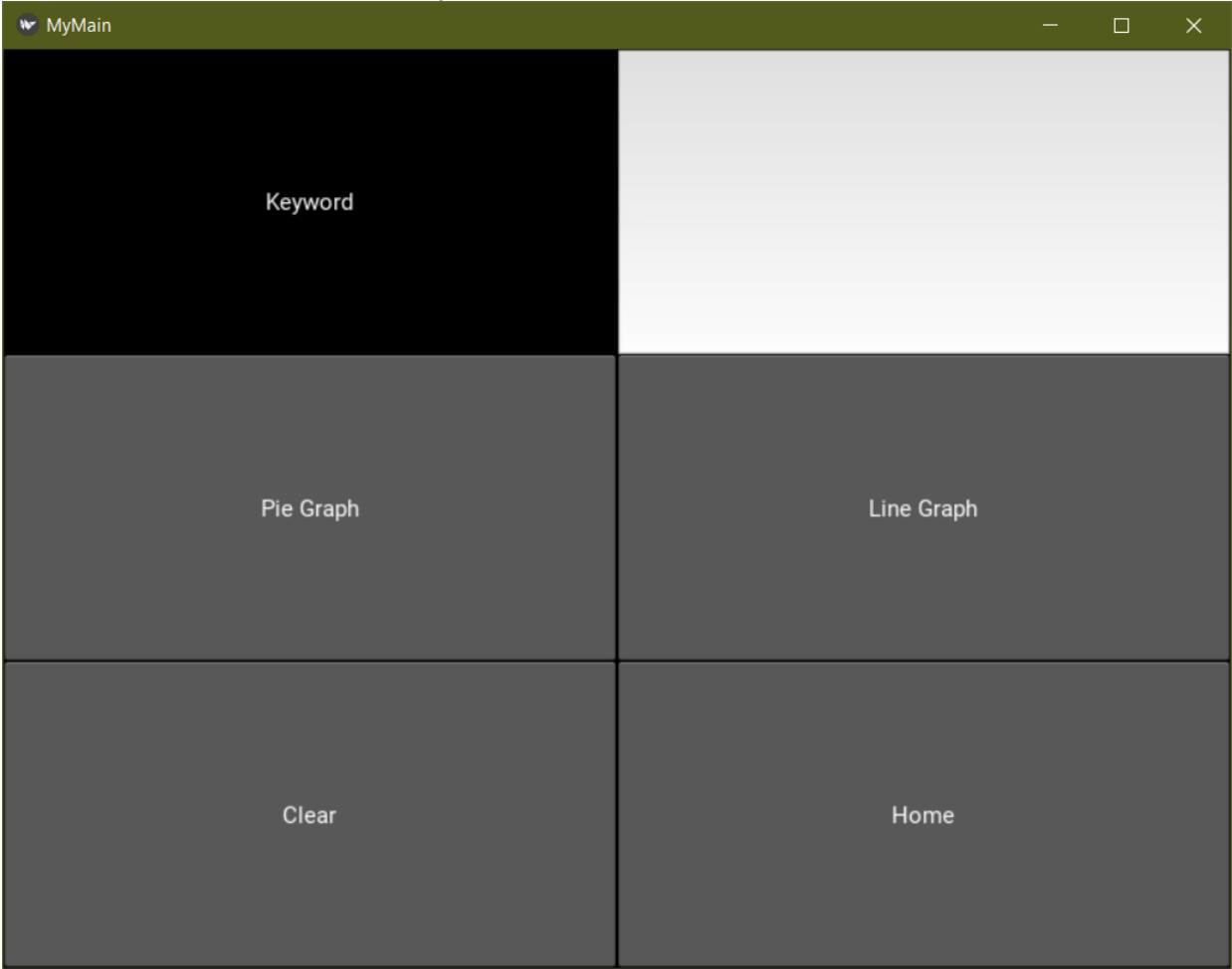
```

Screen Layouts

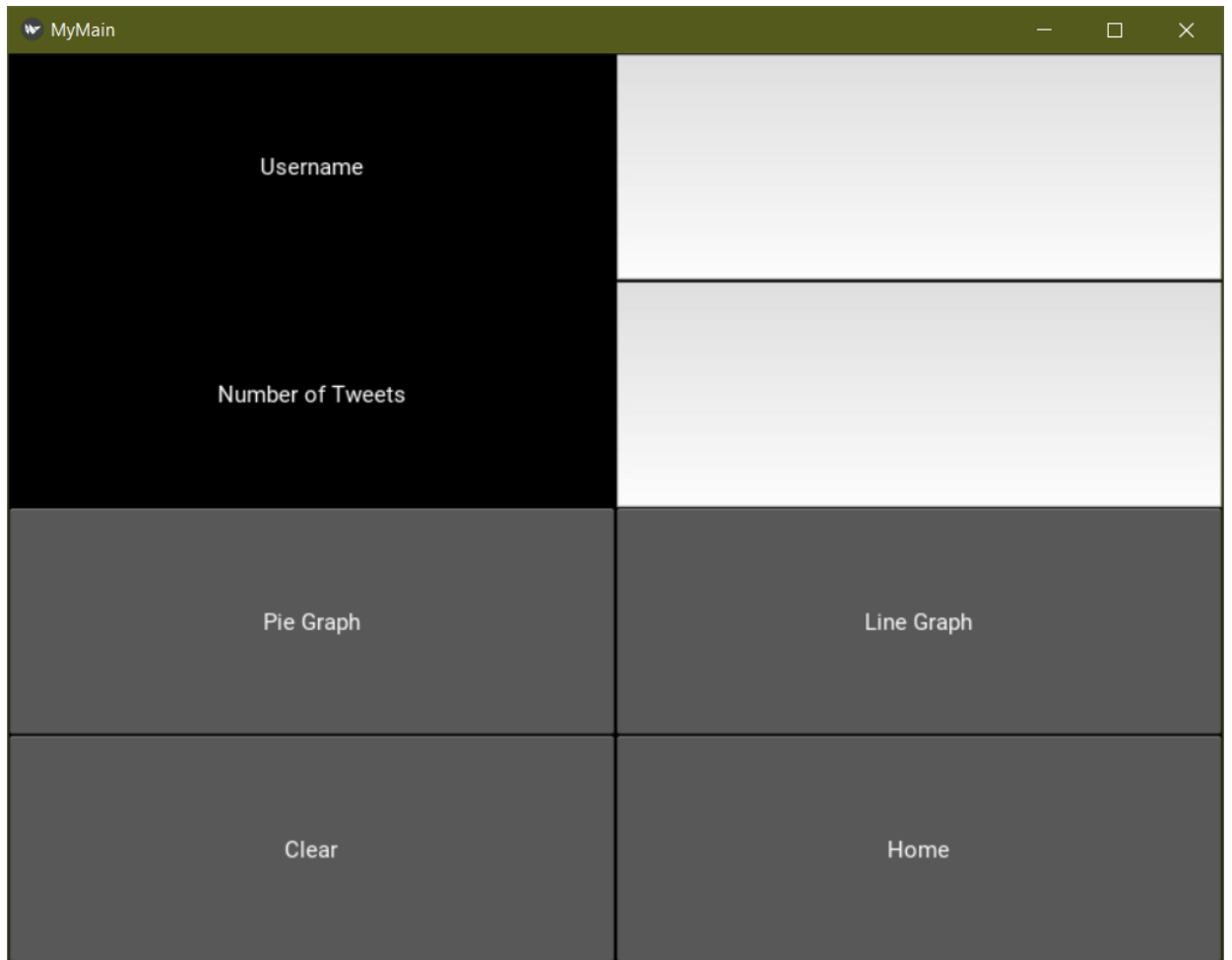
Screen 1: Home Screen



Screen 2: Keyword Based Search Screen



Screen 3: User Based Search Screen



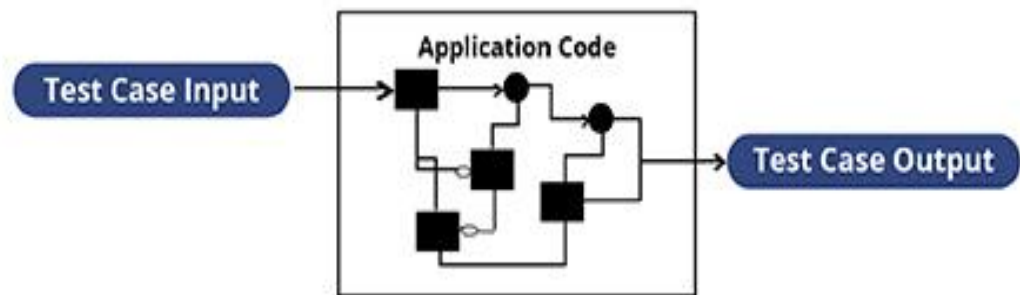
The image shows a user interface for a search screen. It features a dark green header bar with a Twitter bird icon and the text "MyMain" on the left, and standard window control buttons (minimize, maximize, close) on the right. The main content area is divided into a grid of six rectangular buttons. The top-left button is black and contains the text "Username". The button directly below it is also black and contains the text "Number of Tweets". To the right of these two buttons are two light gray buttons, one above the other, which are currently empty. The bottom row consists of four dark gray buttons arranged in a 2x2 grid. The bottom-left button is labeled "Pie Graph", the bottom-right button is labeled "Line Graph", the bottom-left-most button is labeled "Clear", and the bottom-right-most button is labeled "Home".

Username	
Number of Tweets	
Pie Graph	Line Graph
Clear	Home

System Testing

White Box Testing

- White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of software testing that tests internal structures or workings of an desktop application, as opposed to its functionality (i.e. black-box testing).

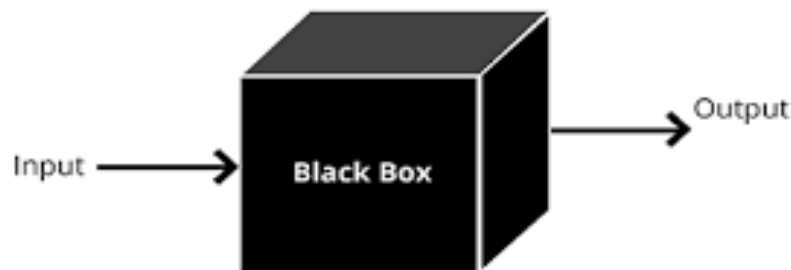


- White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today.
- **Advantages of White Box Testing:**
 - Side effects of having the knowledge of the source code is beneficial to thorough testing.
 - Optimization of code becomes easy as inconspicuous bottlenecks are exposed.
 - Gives the programmer introspection because developers carefully describe any new implementation.

- Provides traceability of tests from the source, thereby allowing future changes to the source to be easily captured in the newly added or modified tests.
- **Disadvantages of White Box Testing**
 - On some occasions, it is not realistic to be able to test every single existing condition of the desktop application and some conditions will be untested.
 - The tests focus on the software as it exists, and missing functionality may not be discovered.
- **Testing:**
 - **Unit Testing:** Every functions in all of the fore mentioned modules were tested separately to be thorough of its functionality.
 - **Integrated Testing:** Functions and modules were tested in different combination of two and three for assurance of compatibility with each other
 - **System Testing:** Structure and Code of entire system was tested to make sure there is no internal error

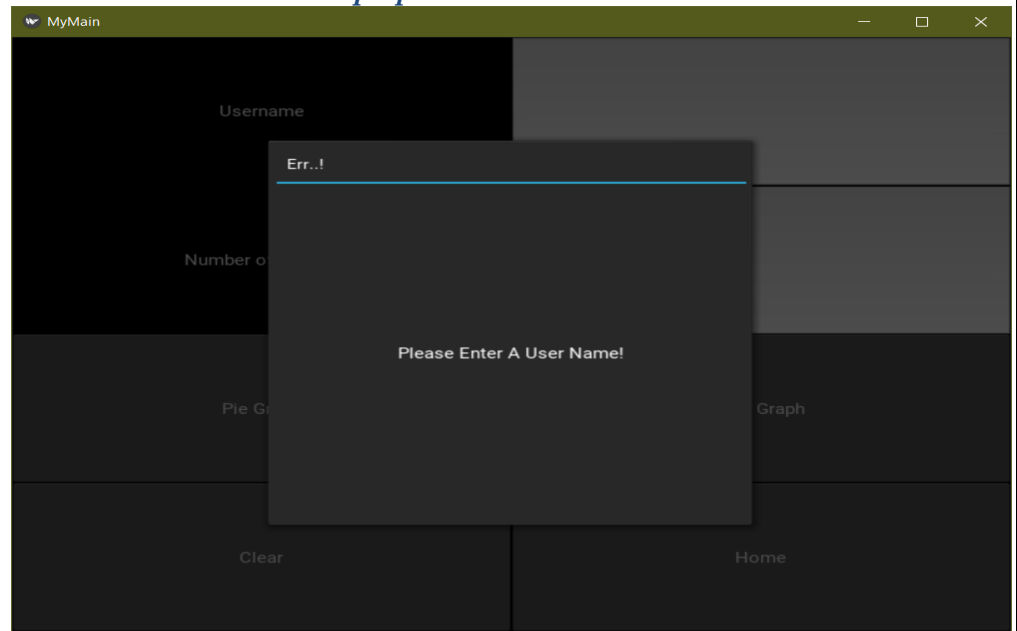
Black Box Testing

- Black-box testing is a method of software testing that examines the functionality of an desktop application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system and acceptance. It is sometimes referred to as specification-based testing.

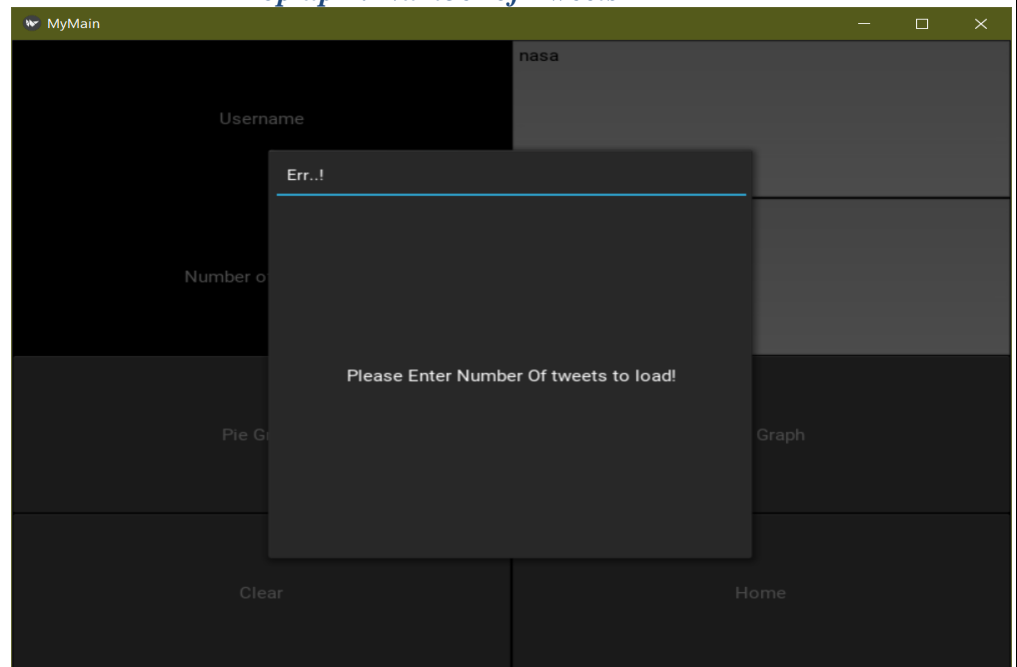


- Test cases are built around specifications and requirements, i.e., what the desktop application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used.
- **Test Cases:**
 - **Case 1: Input Validation**
 - Purpose: To check whether system has proper input validation's or not
 - Method: To press Button without entering any input and In case or integer inputs only try entering string input
 - Results:

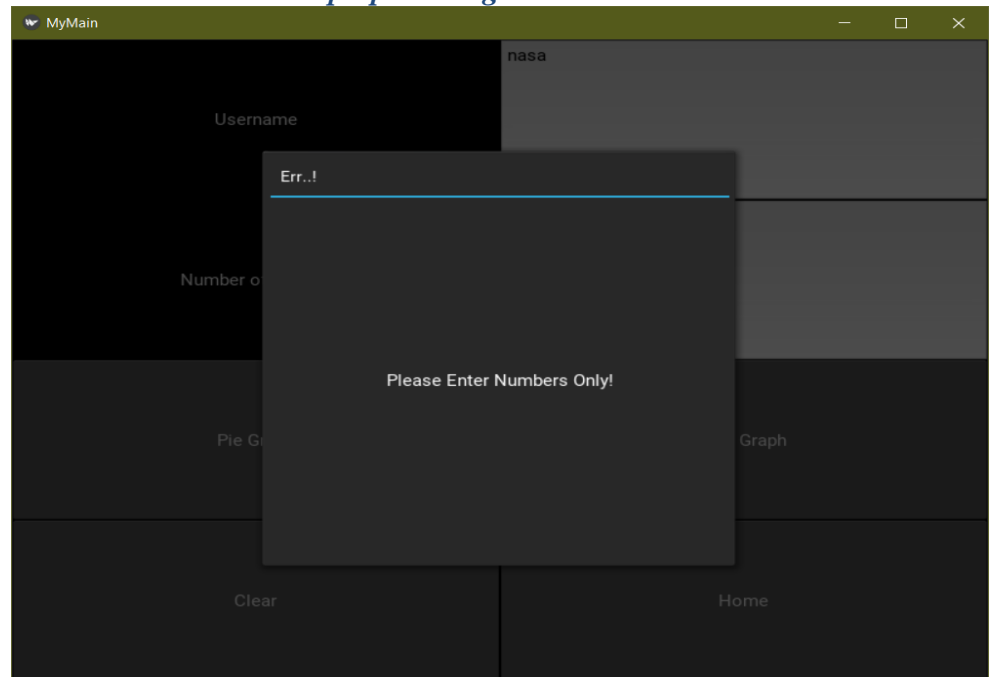
Pop up 1: Username



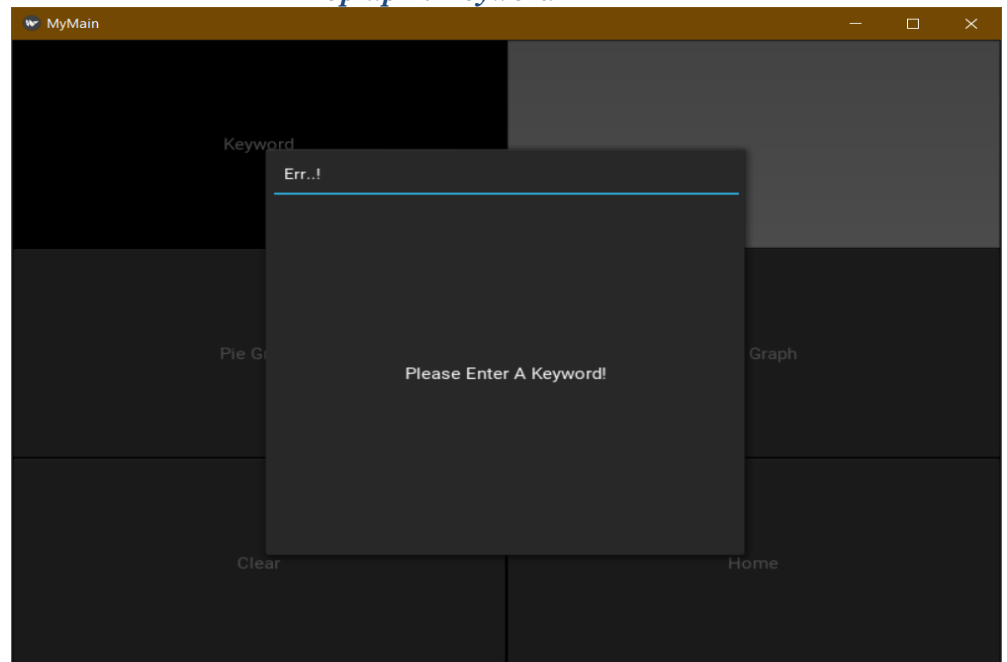
Pop up 2: Number of Tweets



Pop up 3: Integer Check



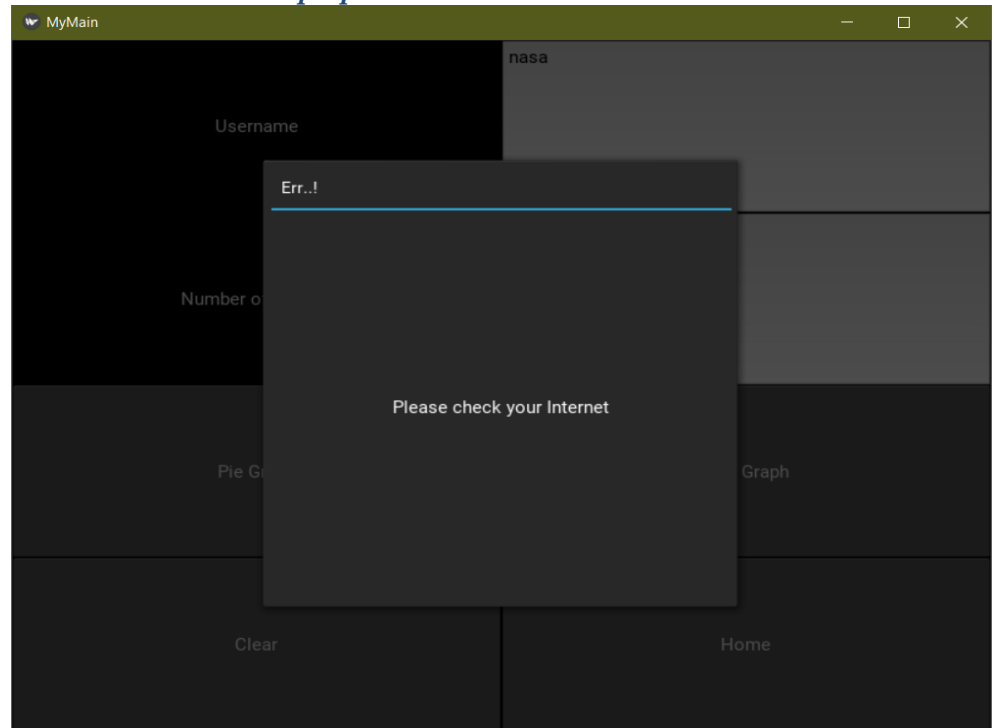
Pop up 4: Keyword



- **Case 2: Internet Connection Check**
 - Purpose: To check whether system performs expected pop up on no internet connection

- Method: Turn off Internet connection and press on button for output
- Result:

Pop up 5: Internet Check



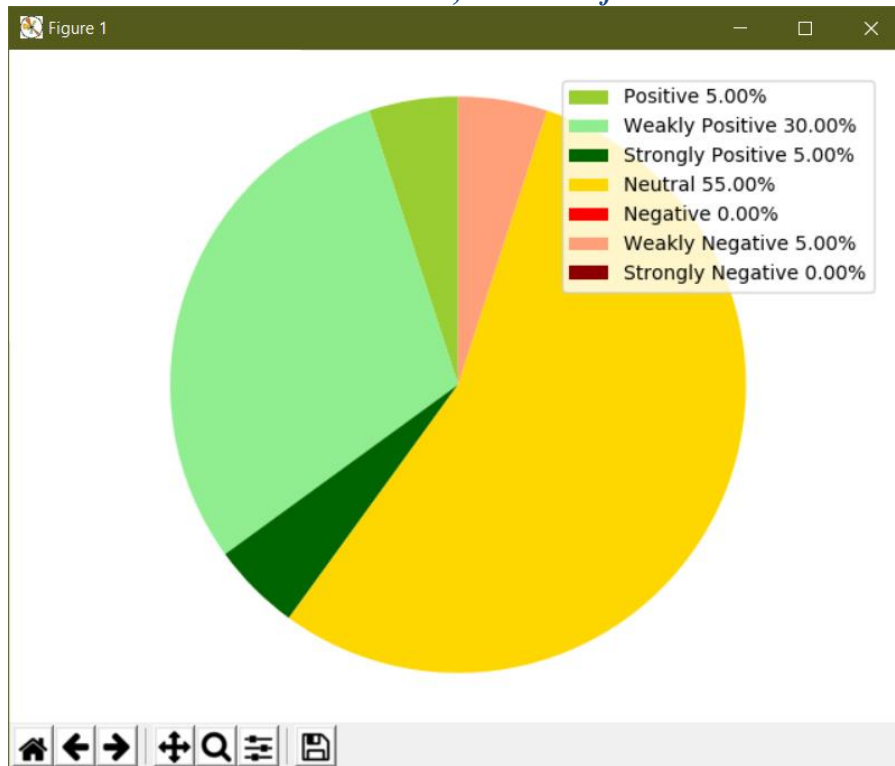
Case 3: Valid Inputs

Purpose: To check whether we get expected output or not.

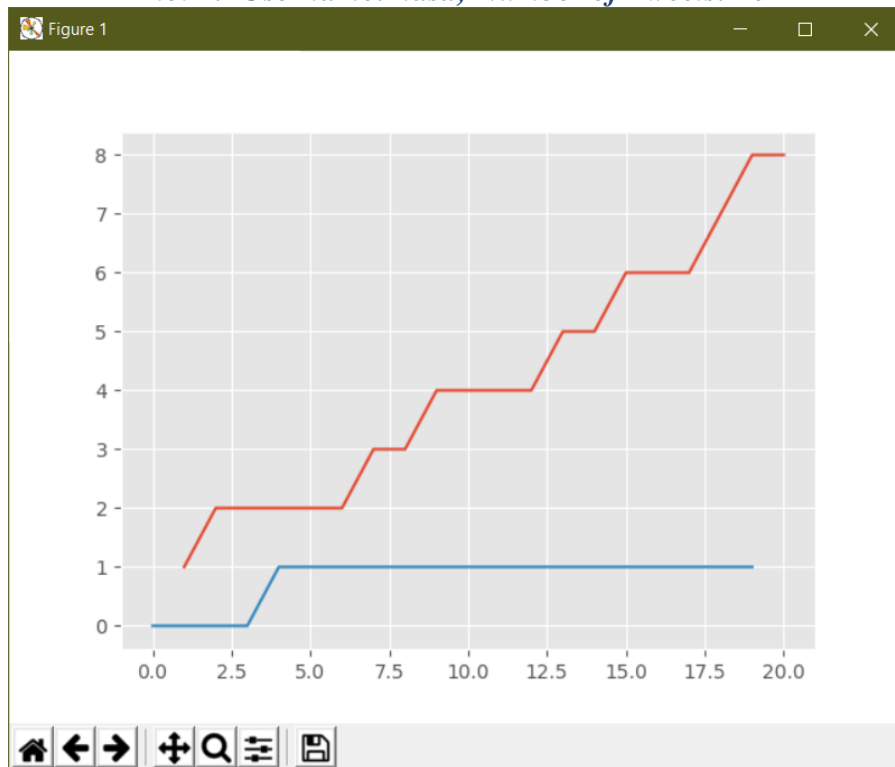
Method: Provided proper inputs in all the fields and press the buttons

Results:

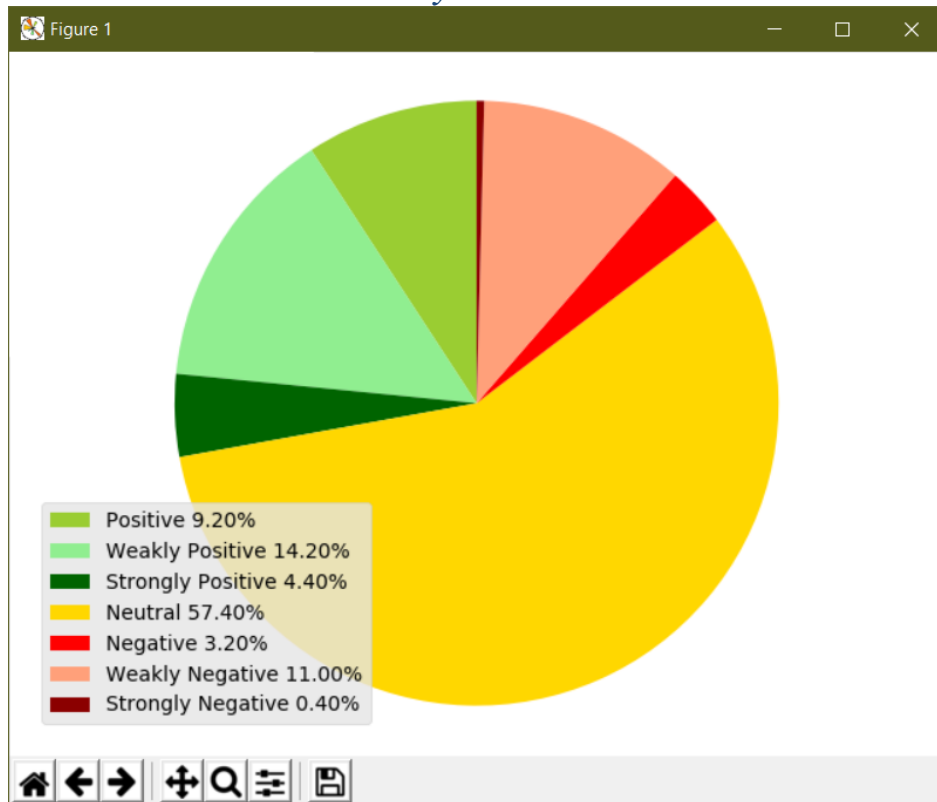
Plot 1: Username: nasa; Number of Tweets: 20



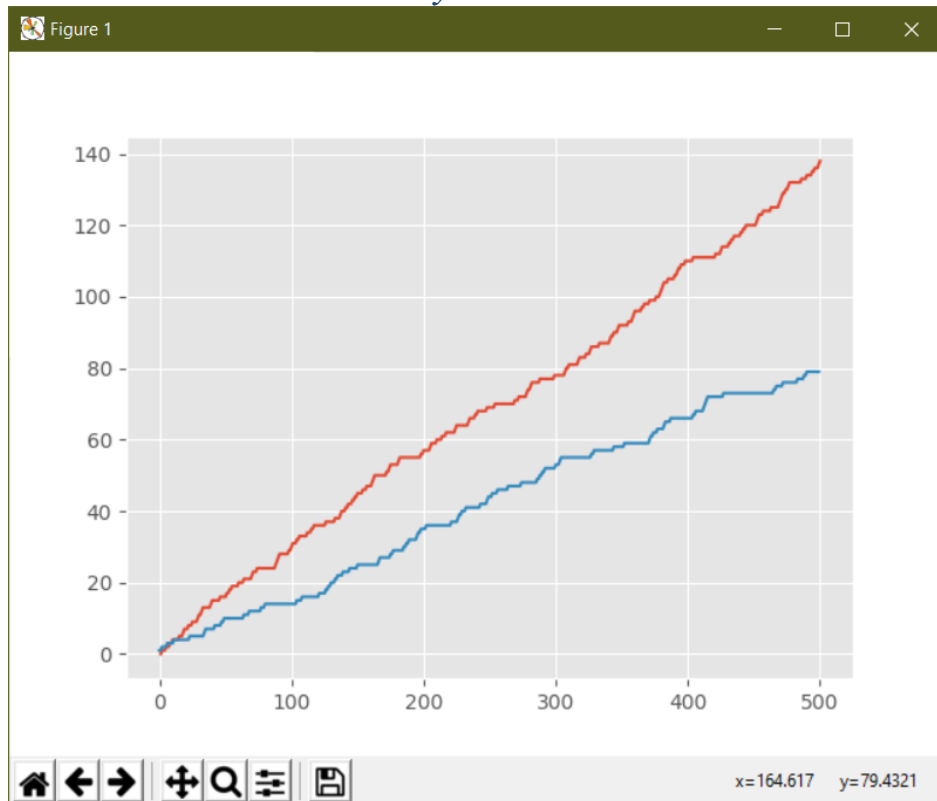
Plot 2: Username: nasa; Number of Tweets: 20



Plot 3: Keyword: Modi



Plot 4: Keyword: Modi



Future Scope

- Since, this project lacks the ability of quick response, I would like to optimize it more.
- Also, I would like to add live tweet streaming functionality. This will eventually make the desktop application more attractive and versatile.
- Adding geolocation functionality to the desktop application will make it more useful and power tool for performing sentiment analysis
- Related terms to the input and trending terms all over the world can also be added for better visualization purpose.

References

- <https://pandas.pydata.org/pandas-docs/stable/>
- <https://kivy.org/doc/stable/>
- <https://matplotlib.org/3.1.1/tutorials/index.html>
- <https://textblob.readthedocs.io/en/dev/quickstart.html>
- <https://docs.python.org/3/>
- <https://kivy-designer.readthedocs.io/en/latest/>
- <https://www.youtube.com/watch?v=1gQ6uG5Ujiw&t=317s>
- <https://techwithtim.net/tutorials/kivy-tutorial/>