**The Tweetables**

**Ada Bulgur, Amir Crutchfield, Sydney Johnson, Micaiah Steplight, Isaac Stinnette**

A blue bird in a circle with black text

AI-generated content may be incorrect.

**Test Plan**

**Table Of Contents**

[**1.0 Introduction 3**](#_heading=h.q8rbam1jphac)

[**1.1 Goals and Objectives 3**](#_heading=h.uracs7m8lhhf)

[**1.2 Statement of Scope 3**](#_heading=h.ppi1zsuyaovo)

[**1.3 Major Constraints 3**](#_heading=h.fte330lxybte)

[**2.0 Software Context 4**](#_heading=h.rhg2kq433d9e)

[**3.0 Software Testing 5**](#_heading=h.xo2e6tkv4en0)

[**3.1 Unit Testing 5**](#_heading=h.33iz1lynqhbv)

[**3.2 Integration Testing 5**](#_heading=h.7mnsy5ad3v89)

[**3.3 System Testing 5**](#_heading=h.jrs8x8s6f1z6)

[**3.4 Black-Box Testing 5**](#_heading=h.whrkil3wvajd)

[**3.5 White Box Testing 6**](#_heading=h.v9r9k3v8k1sa)

[**4.0 Testing Plan 7**](#_heading=h.w3ibn9hsnjor)

[**4.1 Test Items 7**](#_heading=h.e5bwbu6owcto)

[**4.5 Test Schedule 13**](#_heading=h.gldzi1cbnaxl)

[**4.6 Testing Resources and Staffing 14**](#_heading=h.mk7xycczsc2i)

[**6.0 Revision Log 16**](#_heading=h.65xt7x2uafhp)

# 

# **1.0 Introduction**

## **1.1 Goals and Objectives**

The goal of the project is to create a running application that will help corporations determine the emotional tone of a selected product. The application will do this by sorting people’s feelings for a product into three different categories. These categories are positive, negative and neutral.

## **1.2 Statement of Scope**

Tweetables plans to use a Lexicon-based approach in order to build a Sentiment Analysis application. This will be done by using APIs from X in order to build a Lexicon-Based dictionary that will be used to measure people’s emotional tone towards a product. The application will include components that will allow it to gather data, clean data, analyze data and then display the results of the data. Our application will not be using any ML-based techniques. A synopsis of the tasks that must be completed to create the application are:

* Utilize X’s APIs in order to gather data.
* Implement data cleaning so that the analysis focuses only on meaningful words.
* Implement Lexicon-Based sentiment analysis within the application so that certain words get tagged with the correct polarity (positive, negative, or neutral).
* An installation script be provided to install the app, so that the app can be executed with a single click on the installed file.

## **1.3 Major Constraints**

Implementation: Learning how to properly use X’s API. This is a major constraint because it is integral that we understand how the API works in order to get our application to run. It could take a lot of time to learn how to use X’s APIs which could lead to delays with our application.

Lexicon-Based Techniques: Not everyone is familiar with how to implement techniques related to Lexicon-Based searches. As a result, this could cause delays as people have to take the time to properly learn how to use Lexicon- Based techniques.

# **2.0 Software Context**

| **Software** | **Description** | **Type** |
| --- | --- | --- |
| Google Drive/Google Document | Cloud-based storage service that allows the team to store and collaborate on documents. | File Sharing |
| Visual Studio Code | An integrated development environment used for all of coding purposes for the project | Application |
| Python | The Programming language that the application will be created using. | Language |
| Jira | Allows tasks to be created and assigned to team members to be completed for each iteration. | Application |
| X (Twitter) | The social media platform where we will be using its APIs in order to build the program. | Application |

## 

## 

# **3.0 Software Testing**

## **3.1 Unit Testing**

Unit testing is when software components are isolated, and each segment is evaluated individually. By doing this we can ensure that the components function as intended individually. Unit tests are done before the integration and system testing of the software. As a result, unit testing is very important because the rest of the testing cannot be completed until the unit testing is finished. This makes unit testing the backbone of our testing strategy.

Since our application will consist of using multiple applications, unit testing is important to make sure the components of each application function as desired independently. To implement unit testing we plan on using the unittest package in Python. By doing this we will be able to accurately test our python code within Visual Studio Code. This testing will allow us to know whether components in the application are functioning properly. We also plan on implementing unit testing by going through code segments manually on visual studio code and checking its functionality.

## **3.2 Integration Testing**

Integration testing consists of testing whether components that are combined together can function properly as a single unit. This type of testing is done to make sure that components of the software function properly when they interact with one another. This is essential for our application because we have several components that have to interact together with each other. As a result integration testing is an important component of our test plan. To implement integration testing we plan on testing our API to make sure that it works properly. Our API will be used to connect data from X to our application. Therefore, it’s important for us that the API works properly so that the necessary components in our application can function as one unit.

## **3.3 System Testing**

System testing is the last component of our testing strategy. We will implement system testing once unit testing and integration testing are completed. System testing consists of testing the entire system to make sure that it functions properly. The purpose of this type of testing is to make sure that our application does all the tasks that we designed it to. This includes testing the usability of the application itself. To implement system testing we will run our application to see that it functions properly and that it’s easy for users to use. This will allow us to see if every component of our application works properly.

## **3.4 Black-Box Testing**

Black box testing is a simple but effective form of software testing. It involves testing the software without having any knowledge of the code. Black box testing focuses on how the software would behave from a user perspective. A tester would provide inputs into the software and see what type of output is generated. As a result the tester will be able see how the application responds to the user’s actions. To implement black box testing we plan to run our application and see if any issues arise when we input data into the user interface.

## **3.5 White Box Testing**

White box testing is a form of software testing that allows testers to check the inner workings of an application. White box testing involves using the knowledge you have about the internal structure of the application to look for any vulnerabilities or bugs. By doing this you can catch bugs that may not be found through other forms of testing such as black box testing. To implement white box testing we plan on checking that data is handled properly, loops operate efficiently and structures within the application operate properly.

## 

# **4.0 Testing Plan**

## **4.1 Test Items**

* Methods within the application
* X’s API

**4.2 Test Descriptions**

**Test Name:** **Check Rate Limit**

**Type Of Test:** Integration Test

**Test Description**: Checking that the X API’s have not caused the X account that we’re using to be rate limited.

**Test Instructions:**

1. Open Visual Studio Code
2. Open check\_rate\_limit.py file:

import requests

from twitter\_setup import BEARER\_TOKEN #import bearer token

def check\_rate\_limit():

#check twitter APIv2 rate limits manually

url = "https://api.twitter.com/2/tweets/search/recent?query=AI&max\_results=10"

headers = {

"Authorization": f"Bearer {BEARER\_TOKEN}"

}

response = requests.get(url, headers=headers)

if response.status\_code == 200:

print("Request successful! Checking rate limits...\n")

print("Headers:", response.headers) #print full headers(includes rate limits)

print(f"Requests Remaining: {response.headers.get('x-rate-limit-remaining', 'Unknown')}")

print(f"Rate Limit Resets At: Remaining: {response.headers.get('x-rate-limit-reset', 'Unknown')}")

else:

print(f"Error checking rate limit: {response.status\_code} - {response.text}")

check\_rate\_limit()

1. Use the terminal to go to the directory containing the python file.
2. Run the file in the terminal: python check\_rate\_limit.py

**Expected Outcome:** The program should return the status code of 200, showing that the X account has not been rate limited.

**Test Name:** **Fetch Tweets**

**Type Of Test:** Integration Test.

**Test Description:** Checking to see that the API properly fetches tweets from X.

**Test Instructions:**

1. Open Visual Studio Code.
2. Open the file fetch\_tweets.py:

import tweepy

from twitter\_setup import client # import the APIv2 client

def fetch\_tweets\_v2(keyword, count=10):

#fetching tweets using APIv2

try:

#ensure count is between 10 and 100 (twitter's limit)

count = max(10, min(count, 100))

response = client.search\_recent\_tweets(

query=keyword,

max\_results=count,

tweet\_fields=["author\_id", "created\_at", "lang"]

)

#debug: print the raw response from twitter

print("Raw Response:", response)

if response.data:

for i, tweet in enumerate(response.data, start=1):

print(f"{i}. {tweet.text}\n")

else:

print(f"No tweets found for the keyword:", keyword)

except Exception as e:

print(f"Error fetching tweets: {e}")

#test with a keyword

fetch\_tweets\_v2("action", count=10)

1. Use the terminal to go to the directory containing the python file.
2. Run the file in the terminal: python fetch\_tweets.py

**Expected Outcome:** The API properly fetched tweets from X.

**Test Name:** **Test X Access**

**Type Of Test:** Integration Test

**Test Description:** Checking to see that the application has access to X.

**Test Instructions:**

1. Open Visual Studio Code
2. Open test\_twitter\_access.py

from twitter\_setup import client #import your twitter API client

def test\_twitter\_access():

#try fetching your own tweets to check if API is still working

try:

response = client.get\_users\_tweets(id="1886950279301541888", max\_results=5)

if response.data:

for i, tweet in enumerate(response.data, start=1):

print(f"{i}. {tweet.text}\n")

else:

print("No tweets found.")

except Exception as e:

print(f"Error: {e}")

#run the test

test\_twitter\_access()

1. Use the terminal to go to the directory containing the python file.
2. Run the file in the terminal: python test\_twitter\_access.py

**Expected Outcome:** The application should have access to X through the API.

**Test Name:** **Testing Overall Sentiment**

**Type Of Test:** Unit Test

**Test Description:** Checking to see that the system properly analyzes terms in the sentiment dictionary.

**Test Instructions:**

1. Open Visual Studio Code

2. Open testsentiment.py

3. Open testsentiment1.py

4. Add tests that you want to run into the test block section of the sentimenttest() method. test block section: test = [

("Masterpiece", "positive"),

("brilliant", "positive"),

("okay", "neutral"),

("average", "neutral"),

("slow", "negative"),

].

5. Run the testsentiment1.py file in the terminal

**Expected Outcome:** The system should accurately assign either 'positive', 'negative' or neutral to selected words.

**Test Name: Testing Overall Sentiment Score**

**Type Of Test:** Unit Test

**Test Description:** Checking to see that the system properly assigns a score to the selected words.

**Test Instructions:**

1. Open Visual Studio Code

2. Open testsentiment.py

3. Open testsentiment1.py

4. Add tests that you want to run into the test block section of the sentimentscoretest() method. test block section: test = [

("Masterpiece", 5),

("brilliant", 4),

("okay", 0),

("average", 0),

("slow", -2),

].

5. Run the testsentiment1.py file in the terminal

**Expected Outcome:** The system should accurately assign a score of -5 - 5 to the words selected.

**Test Name: Testing Case Sensitivity**

**Type Of Test:** Unit Test

**Test Description:** Checking that the system is case insensitive when analyzing words.

**Test Instructions:**

1. Open Visual Studio Code

2. Open testsentiment.py

3. Open testsentiment1.py

4. Add tests that you want to run into the test block section of the casesentimenttest() method. test block section: test = [

("MASTERPIECE", positive),

("BRILLIANT", positive),

].

5. Run the testsentiment1.py file in the terminal

**Expected Outcome:** The system should still assign the words with the proper sentiment despite any uppercase characters.

**Test Name: Testing Invalid Inputs**

**Type Of Test:** Unit Test

**Test Description:** Check that the sentiment doesn't assign a 'positive' or 'negative' sentiment to invalid inputs

**Test Instructions:**

1. Open Visual Studio Code

2. Open testsentiment.py

3. Open testsentiment1.py

4. Add tests that you want to run into the test block section of the invalidtest() method. test block section: test = [

(" ", negative),

("......", negative),

].

5. Run the testsentiment1.py file in the terminal

**Expected Outcome:** System should assign neutral to invalid inputs

**Test Name: Testing Punctuation**

**Type Of Test:** Unit Test

**Test Description:** Check that the punctuation doesn't mess up overall sentiment score.

**Test Instructions:**

1. Open Visual Studio Code

2. Open testsentimentt.py

3. Open testsentiment1.py

4. Add tests that you want to run into the test block section of the punctuationtest() method. test block section: test = [

("Masterpiece!!!!", positive),

(".....brilliant....", positive),

].

5. Run the testsentiment1.py file in the terminal

**Expected Outcome:** System should still assign the proper sentiment to assigned words.

**Test Name: Check Valid Login**

**Type Of Test:** Integration Test

**Test Description:** Check that account credentials match in our.txt file.

**Test Instructions:**

1. Open Visual Studio Code.

2. Run Main.py

3. Select "sign in" and create username and password information

4. Select "login" and use fake username and password to see if application runs

5. After being denied access use accurate username and password to see if application runs

6. After entering correct credentials users will have full access to our application

**Expected Outcome:** The application denies access with incorrect credentials and gives access with correct credentials.

**Test Name: Checking Login Storage Functionality**

**Type Of Test:** Integration Test

**Test Description:** Check that the system properly stores log-in data.

**Test Instructions:**

1. Open ‘Tweetables\_latestversion’ folder.

2. Open Tweetables folder.

2. Run Main.py

3. Select "sign up" and create username and password information

4. Go back to Tweetables folder.

5. Click on the ‘users’ text file.

6. Look for the username and password that you just created.

**Expected Outcome:** The username and password that was just created should be listed in the ‘users’ text file.

**Test Name: Checking System Storage Functionality**

**Type Of Test:** Integration Test

**Test Description:** Check that the system properly stores tweet data.

**Test Instructions:**

1. Open ‘Tweetables\_latestversion’ folder.

2. Open Tweetables folder.

2. Run Main.py

3. Log in to the interface, if you do not have a username and password then follow the sign up instructions.

4. Once logged in, enter a keyword into the search bar.

5. Click fetch tweets.

6. Click analyze sentiment.

6. Go back to the Tweetables folder.

7. Click on the ‘raw\_tweets’ text file and click on the ‘cleaned\_tweets’ text file.

**Expected Outcome:** The same raw tweets that were shown in the GUI when you clicked fetch tweets should be stored in the ‘raw\_tweets’ file. Additionally, the data that you see in the GUI when you click analyze sentiment should be stored in the ‘cleaned\_tweets’ file.

**Test Name: Checking Executable Launch**

**Type Of Test:** Integration Test

**Test Description:** Check that executable launches properly.

**Test Instructions:**

1. Open ‘Tweetables\_latestversion’ folder.

2. Once inside that folder, open ‘Tweetables’ folder.

3. Once inside that folder, open ‘dist’ folder.

4. Inside the ‘dist’ folder, double click on the ‘main.exe’ file.

**Expected Outcome:** The GUI for Tweetables should pop up on the screen. It should display the log-in screen.

**Test Name: Checking Executable Functionality**

**Type Of Test:** Integration Test

**Test Description:** Check that GUI functions properly when launched from the executable file.

**Test Instructions:**

1. Inside the ‘dist’ folder, double click on the ‘main.exe’ file.

2. On the log-in screen press the sign up button.

3. Enter a username and password.

4. After that, enter that same username and password in order to log in.

5. Once logged in, enter a movie keyword into the search bar.

6. After that is done press the fetch tweets button.

7. Next, press the sentiment analysis button.

**Expected Outcome:** The GUI functions for signing up, logging in, fetching tweets and sentiment analysis works properly when launched from the executable file.

**Test Name: Checking Slang Function**

**Type Of Test:** Unit Test

**Test Description:** Checking that the algorithm used in analyze\_sentiment.py properly translates slang that may be found in tweets.

**Test Instructions:**

1. Open testsentiment2.py

2. Run code.

**Expected Outcome:** The system should be able to understand that slang words such as ‘u’ equals ‘you’.

**4.4 Testing Tools And Environment**

Tools:

* Visual Studio Code

Environment:

* Microsoft Windows 11 & Mac OS

## **4.5 Test Schedule**

| Date: | Test Done: |
| --- | --- |
| 2/25 | Fetch Tweets |
| 2/26 | Fetch Tweets |
| 2/27 | Test X Access |
| 2/28 | Test X Access |
| 3/3 | Check Rate limit |
| 3/4 | Check Rate limit |
| 4/15 | Checking Login Storage Functionality |
| 4/16 | Checking System Storage Functionality |
| 4/17 | Checking Executable Launch |
| 4/19 | Checking Executable Functionality |
| 4/19 | Checking Slang Function |

## **4.6 Testing Resources and Staffing**

* Amir Crutchfield, Testing lead.
* Isaac Stinnette, Assistant Testing

**5.0 Response to Failure**

If any software tests fail there will be several actions that will be done. The first action is to investigate what went wrong and fix the issue accordingly. This will be done by inspecting the error messages. By doing this we will have a better understanding of the root cause of the issue. After this is done, we will make the necessary updates to the test to solve the issue.

# 

# **6.0 Revision Log**

| **Revision** | **By** | **Date** | **Description** |
| --- | --- | --- | --- |
| 1.0 | The Tweetables | 2/13/2025 | Test Plan  Document for Iteration 2 |
| 2.0 | The Tweetables | 3/6/2025 | Test Plan Document for Iteration 3 |
| 3.0 | The Tweetables | 3/27/2025 | Test Plan Document for Iteration 4 |
| 4.0 | The Tweetables | 4/19/2025 | Test Plan Document for Iteration 5 |