SE 3XA3: Software Test Plan
SocialPy

Team 1, SocialPy
Anando Zaman, zamana11
Graeme Woods, woodsg1
Yuvraj Randhawa, randhawy
Due March 5, 2021
Tag TP-Rev.1

April 8, 2021

Contents

List of Tables

List of Figures

Table 1: Revision History

Date	Developer(s)	Change
February 23, 2021	Anando Zaman	Copy template
February 24, 2021	Anando Zaman	Completed General Information section
February 24, 2021	Anando Zaman	Completed Plan & system description sections
February 25, 2021	Anando Zaman	Made some progress on NFRs, Traceability matrices, & usability survey.
February 25, 2021	Anando Zaman	Completed Unit testing internal functions section.
March 1, 2021	Yuvraj Randhawa	Maintenance Requirements
March 1, 2021	Yuvraj Randhawa	Supportability Requirements
March 1, 2021	Yuvraj Randhawa	Longevity Requirements
March 2, 2021	Graeme Woods	Tests for Proof of Concept
March 3, 2021	Graeme Woods	Comparison to Existing implementation

1 General Information

1.1 Purpose

The purpose of this document is to outline the testing, validation, and verification process of the functional and non-functional requirements, for the SocialPy project. This will ensure that each of the modules and Database API calls are functioning as expected for an overall consistent and smooth application.

1.2 Scope

The scope of the test plan is to provide a basis for testing the functionality of the SocialPy application. This plan will provide a means of outlining tests for each of the individual modules via Python Unittest library. The unit tests will cover partition testing, fuzz testing, specification-based testing, and boundary testing. The objective is to satisfy all the functional and non-functional requirements provided in the SRS document.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: Table of Abbreviations

Abbreviation	**Definition**
FR	Functional Requirement
API	Application programming interface
FireBase	Cloud Database provider
NFR	Non-functional Requirement
POC	Proof of Concept
SRS	Software Requirements Specification
UTC	Coordinated Universal Time
APP	Application

Table 3: Table of Definitions

Term	Definition
Acceptance Testing	A method of testing which is conducted to determine if the requirements of the specification are met
Boundary Testing	A method of testing where values are chosen on semantically significant boundaries
Checklist	A method of testing through inspecting the code
Code Inspection	A method of static testing where developers walk through the code
Dynamic Testing	A method of testing where code is executed
Exploratory Testing	A method of testing where the tester simultaneously learns the code while testing it. It approaches testing from a user's viewpoint
Fuzz Testing	A method of testing where random inputs are given to attempt to violate assertions
Integration Testing	A method of testing where individual software modules are combined and tested as a group
Partition Testing	A method of testing where the input domain is partitioned and input values are selected from the partitions
Unittest	a Python Unit testing framework.

1.4 Overview of Document

The main objective of the this document is to fully encompass tests for all the requirements of SocialPy in order to verify correctness and validation. This document contains relevant information on the type of test cases to be conducted and their rationale.

2 Plan

2.1 Software Description

Social Media is a relatively new and quickly growing form of communication being used all over the globe. Gone are the days of expensive and inconvenient long-distance phone calls just to be able to communicate with family and friends. SocialPy is a Python terminal-based software application that further enhances upon an existing open-source java-based social media application. The goal of SocialPy is to provide an intuitive, pleasant, and smooth user-experience such that users completely transition to a digital form of communication. This application will allow users to post their current status/emotions/thoughts and share other content in the form of short 3 to 4 sentence messages. These messages will be visible under the users "activity wall" which other users will be able to view. Users will also be able to follow other users to view their post activity. Additionally, the app will feature account authentication to ensure the integrity of the users posts.

2.2 Test Team

The test team for this project consists of the following members who are each responsible for writing and and executing tests for their respective modules:

- Anando Zaman
- Graeme Woods
- Yuvraj Randhawa

2.3 Automated Testing Approach

Testing shall all be automated as most of the user-interface is just a terminal window. Thus, automated testing of the back-end code will be done. Each module will have a test class which will be commited to the Gitlab repo. These tests can also be updated in the future if more edge cases arise.

2.4 Testing Tools

The unit tests will be written using the Python Unittest framework since the team is familiar with the library and provides a comprehensive documentation. Additionally, manual testing will be done by the test team for cases that cannot be automated.

2.5 Testing Schedule

All the tests such as Unit testing, integration testing, performance testing, other NFRs, User Acceptance testing are done in parallel.

Gantt chart provided here: https://gitlab.cas.mcmaster.ca/zamana11/se3xa3-project/-/tree/master/ProjectSchedule.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Registration/login subsystem tests

Authenticate/login a user

1. **FR-ST-2**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: Existing account with same email exists in database.

   Input: Email address and password

   Output: Returns "Login Success" message and updates $self.user$

Exception: if sign in fails, error message stating "sign-in failed"

How test will be performed: Unittest will run internal commands to login with a test/dummy account to check login success.

Register new users

1. **FR-ST-3**

   Type: Functional, Dynamic, Manual.

   Initial State: Existing account with same email does NOT already exist in database.

   Input: Email address and password

   Output: Returns "Registration Success" message and updates $self.user$ object

   Exception: if registration fails, error message stating "Signup failed. Account already exists or password is not long enough"

   How test will be performed: Unittest will run internal commands to register with a test/dummy account to check registration success.

Reset account password

1. **FR-ST-6**

   Type: Functional, Manual.

   Initial State: Existing account with same email already exists in the database

   Input: Email address

   Output: Firebase sends a email to reset the users' password in which they will enter their new password.

   Exception: If email does not exist, system displays "Password reset failed. Please enter a valid email or type 'exit' to escape.".

   How test will be performed: A member of the team will enter a dummy email to be reset. The person will then reset the password via the automated email provided from Firebase. The user will then attempt to login using the new password to check if password changes have been applied.

3.1.2 Post subsystem tests

Query all posts data

1. **FR-ST-1**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: Firebase contains predefined JSON structure already completed

   Input: "POST VIEW ALL" string

   Output: Returns string containing all the posts from the database.

Exception: If no posts available, system displays "no posts to display".

How test will be performed: Unittest will run internal command "POST VIEW ALL" in SocialPy to execute to query the data and run assertions for the expected outcome.

Query posts by username

1. **FR-ST-4**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: Database contains atleast one post and atleast one username.

   Input: "POST VIEW username"

   Output: Returns a string containing all the posts for a given user along with the timestamp of when post was created.

   Exception: If user does not exist, system displays "There is no user with the username daniel".

   How test will be performed: Unittest will run internal command "POST VIEW username" to query the data. It will then assert with the expected outcome.

Delete post

1. **FR-ST-5**

   Type: Functional, Dynamic, Manual.

   Initial State: Database contains atleast one post for the current logged in user with the matching post_id.

   Input: "POST DELETE post_id"

   Output: Returns a string containing all the posts for a given user along with the timestamp of when post was created.

   Exception: If postID does not exist as a valid ID for the user, system displays "invalid postID".

   How test will be performed: The team will manually run internal command "POST DELETE post_id" to delete the post. Then, it will query the DB to check if the posts no longer exists.

Add post

1. **FR-ST-8**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: Database contains a valid "posts" branch for the post to be added to.

   Input: "POST ADD post_id"

   Output: Returns a message informing the user that "post content has been added". The database reflects the changes made in the users' post branch.

Exception: If post content is blank, systems outputs "post content cannot be empty".

How test will be performed: Unittest will run internal command "POST ADD content" to add a post. Then, it will query the DB to check if the posts now exists.

Edit post

1. **FR-ST-9**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: Database contains a valid "posts" branch for the post to be added to.

   Input: "POST EDIT post_id" followed by the user entering the modified post.

   Output: Returns "post has been updated" message and post content becomes updated in the database.

   Exception: If postID does not exist for the user, systems outputs "invalid postID".

   How test will be performed: Unittest will run internal command "POST EDIT post_id" to add a post. Then, it will query the DB to check if the updates have been applied.

Help commands

1. **FR-ST-10**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: User is logged into the system.

   Input: User inputs "Help"

   Output: Returns a message with all the commands available.

   Exception: No exceptions.

   How test will be performed: Unittest will run an assertion on the output of the internal command "HELP", which will return a string containing all the available commands. Assertions will be made for the expected help commands.

3.1.3 Profile subsystem tests

Query Profile

1. **FR-ST-11**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: The user profile exists in the database.

   Input: "PROFILE VIEW ALL" string

   Output: Returns string containing all the profile information(location, name, followers) from the database.

   Exception: If postID does not exist for the user, systems outputs "Sorry, profile for the given username does not exist".

   How test will be performed: Unittest will run assertions on the internal command "PROFILE VIEW ALL" with that of the expected output.

Edit Profile Location

1. **FR-ST-12**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: The user profile exists in the database.

   Input: "PROFILE EDIT_LOCATION location_name" string

   Output: Returns a "Successfully updated location" message. The database reflects the changes made.

   Exception: If location string is empty, print "Location field cannot be empty...".

   How test will be performed: Unittest will run assertions on the internal command "PROFILE EDIT_LOCATION location_name" that SocialPy will execute to update the database.

Edit Profile Name

1. **FR-ST-13**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: The user profile exists in the database.

   Input: "PROFILE EDIT_NAME name" string

   Output: Returns a "Successfully updated name" message. The database reflects the changes made.

   Exception: If name string is empty, print "name field cannot be empty...".

   How test will be performed: Unittest will run assertions on the internal command "PROFILE EDIT_NAME name" that SocialPy will execute to update the database.

Delete Account

1. **FR-ST-14**

   Type: Functional, Dynamic, Manual.

   Initial State: The user profile exists in the database.

   Input: "PROFILE DELETE ACCOUNT" string

   Output: Returns a "Successfully deleted account" message. Removes user profile and posts associated with the user from the database.

   Exception: None

   How test will be performed: Unittest will run assertions on the internal command "PROFILE DELETE ACCOUNT" that SocialPy will execute to update the database.

Delete Follower

1. **FR-ST-15**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: The follower to be deleted exists in the users following list. The user is also logged in.

   Input: "PROFILE FOLLOWERS_DELETE username" string

   Output: Returns a "Successfully removed follower" message. Removes user from follower list in the database.

   Exception: if entered invalid username, display "User to follow cannot be blank. You must enter a valid username to follow". If entered a username that is not in your followers list, display "Sorry, username does not exist in your following list".

   How test will be performed: Unittest will run assertions on the internal command "PROFILE FOLLOWERS_DELETE username" that SocialPy will execute to update the database.

Add Follower

1. **FR-ST-7**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: The follower to be added does NOT already exist in the users following list. The user is also logged in.

   Input: "PROFILE FOLLOWINGS_ADD username" string

   Output: Returns a "Successfully added follower" message. Adds the username to follower list in the database.

   Exceptions. If follower is not a valid username, display "There is no user with the username [Name here]". If follower is yourself, display "You cannot follow yourself. Please try following a valid user other than yourself." If follower is blank or empty, display "User to follow cannot be blank. You must enter a valid username to follow"

   How test will be performed: Unittest will run assertions on the internal command "PROFILE FOLLOWINGS_ADD username" that SocialPy will execute to update the database.

View Followers

1. **FR-ST-16**

   Type: Functional, Dynamic, Unit, Automated.

   Initial State: The user is logged in. The username to view followers is valid/exists on database.

   Input: "PROFILE FOLLOWERS_VIEW username" string

Output: Returns a string containing all the followers for a given username seperate by new-line characters.

Exception: If username entered is invalid, display "There is no user with the username [Name here]". If a user has no followers, display "[Name here] has no followers".

How test will be performed: Unittest will run assertions on the internal command "PROFILE FOLLOWERS_ADD username" that SocialPy will execute to query from the database.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Requirements

Appearance Requirements

1. NFR-LF-1

   Type: Manual, Dynamic, Checklist

   Initial State: SocialPy is active and running

   Input/Condition: Set of all SocialPy commands entered

   Output/Result: Outputs the results of the commands in the terminal as text format.

   How test will be performed: The test team will run commands in the SocialPy application and check if the text format returned is in accordance to the english language.

Style Requirements

1. NFR-LF-2

   Type: Manual, Usability Survey

   How test will be performed: The test team will conduct a questionaire where the users would rank how easy it is understand the product text labels. Refer to Appendix

2. NFR-LF-3

   Type: Manual, Usability Survey

   How test will be performed: The test team will conduct a questionaire where the users would rank their preference on the color scheme as "good, neutral, or bad".

3.2.2 Usability and Humanity Requirements

Ease of use Requirements

1. NFR-UH-4

   Type: Manual, Usability Survey

   How test will be performed: The test team will provide a questionnaire asking the user to input their age and then ask follow up questions about their level of difficulty using the application. The following options will be provided to let the user decide their experience: "intimidating", "complicated", "neutral", or "excellent". The fit criterion is if 80% of the users who are 13+ years old voted neutral or excellent, then the test has passed and NFR has been satisfied.

2. NFR-UH-5

   Type: Manual, Usability Survey

   How test will be performed: The test team will provide a questionnaire asking the user to input the average time it takes for them to transition between tasks.

3. NFR-UH-6

   Type: Manual, checklist

   Initial State: SocialPy is waiting for input

   Input/Condition: English words and Russian words.

   Expected Output/Result: Outputs "Sorry, your command is invalid. Please type 'help' if you need to view the list of commands!" if in Russian input. Outputs no error message if valid command in English.

   How test will be performed: The test team will input letters in another language to check if they get rejected. They will then compare english words to check if it passes.

3.2.3 Performance Requirements

Speed Requirements

1. NFR-PR-7

   Type: Automated, Dynamic, UnitTest

   Initial State: Data to query exists in DB

   Input/Condition: set of data branches to query

   Output/Result: Outputs the time required to query each branch

How test will be performed: The test team will run automated tests using Python Unittest module where the system will keep track of the execution time ensure that it is less than 2 seconds in order to pass.

2. NFR-PR-8

   Type: Automated, Dynamic, UnitTest

   Initial State: Waiting for user credentials. User account already exists in the DB.

   Input/Condition: email and password credentials.

   Output/Result: Outputs the time required to authenticate

   How test will be performed: The test team will run automated tests using Python Unittest module where the system will keep track of the execution time ensure that it is less than 4 seconds in order to pass.

Precision or Accuracy Requirements

1. NFR-PR-9

   Type: Automated, Dynamic, UnitTest

   Initial State: Atleast one post exists in DB for the given user.

   Input/Condition: Query post for the given user.

   Output/Result: Outputs the the timestamp data.

   How test will be performed: The test team will run automated tests using Python Unittest module where the system will run assertions to check if the at most 2 decimal places are used.

2. NFR-PR-10

   Type: Automated, Dynamic, UnitTest

   Initial State: The user is logged in prior to posting.

   Input/Condition: Post string content

   Output/Result: 'Post "postContent" added'

   How test will be performed: The test team will run automated tests using Python Unittest module where the system will run assertions to check if a post has been posted with 200 characters or less.

Reliability and Availability Requirements

1. NFR-PR-11

   Type: Manual, Dynamic

   How test will be performed: The test team will maintain a log file containing the days when the server will be down. If it 3 days or less out of 365 days or less, then it ensures that the system is active for 99% out of the 365 days.

Scalability or Extensibility Requirements

1. NFR-PR-12

   Type: Manual, Dynamic

   Initial State: No commands being made

   Input: Post(add,view) commands being made by 5 users at the same time.

   Output: System returns the response at the time of output without having duplicate posts of the same post_id.

   How test will be performed: The test team will manually run tests and check if the firebase DB tree shows inconsistencies(ex; duplicate posts of the same post_id).

2. NFR-PR-13

   Type: Manual, Dynamic

   Initial State: No commands being made

   Input: Execute 20 Post(add,view) commands

   Output: System returns the success response message of each post.

   How test will be performed: The test team would manually post 20 different posts and capture screenshots of Firebase console if the storage capacity is increasing to compensate as more posts are being added.

3.2.4 Operational and Environmental Requirements

Expected Physical Environment

1. NFR-OE-14

   Type: Manual, Dynamic

   How test will be performed: The test team will manually run the commands on the PC without the correct version of Python to check if errors arise as expected or if the program somehow continues to run correctly. The team will then use the correct version of Python(3.7) and check if it runs correctly as expected. If so, then our test case has passed.

2. NFR-OE-15

   Type: Manual, Dynamic

   How test will be performed: The test team will manually run the commands on SocialPy with the internet disabled to see if errors arise as expected. The team will then check if the program works once the internet is turned back on. If so, then test case passed.

Requirements for interfacing with Adjacent Systems

1. NFR-OE-16

   Type: Automatic, Dynamic, Unittest

   Initial State: No commands being made, SocialPy waiting for command. username to be queried already exists in Firebase.

   Input: "POST VIEW username" command

   Output: System returns the post

   How test will be performed: The test team will use Unittest Python library and the Firebase API to execute a post view command via SocialPy. Test is a success if the posts for the given username are returned, implying that the system is able to communicate with Firebase.

Installability Requirements

1. NFR-OE-17/18

   Type: Manual, Dynamic Initial state: Program does not exist on the users computer.

   Input: "git clone gitlab_project" and double click *main.py*

   Output: Project successfully cloned and SocialPy login screen appears.

   How test will be performed: The testing team will manually clone the project from Gitlab and attempt to run the *main.py* file to execute the program. If this works, then test passes as no installation was required. Just a simple git repo "clone".

Installability Requirements

1. NFR-OE-19

   Type: Manual, Dynamic

   Initial state: Program cloned on atleast 3 computers.

   Input: double click *main.py*

Output: SocialPy login screen appears.

How test will be performed: The testing team will manually execute the *main.py* file while keeping track of the time via a stopwatch. This will be done 4 times on atleast 3 computers. The average time will be used to determine whether it is under 2 seconds.

Release Requirements

1. NFR-OE-20

   Type: Manual, static, code inspection Initial state: Program deployed to users in production Input: Users provide feedback and feature requests Output: Development team maintains feedback via a Gitlab issue tracker.

   How test will be performed: The testing team will keep track of feature requests via Gitlab monthly. Additionally, the testing team will conduct code inspections to identify the faults for bug-fixes.

3.2.5 Maintainability and Support Requirements

Maintenance Requirements

1. NFR-MS-21

   Type: Manual, Static

   Initial state: N/A

   Input: N/A

   Output: The code is documented accurately with comments

   How test will be performed: The code will be inspected by the team to determine if it has been accurately documented. The documentation will be manually inspected to ensure all aspects of the code have been documented.

2. NFR-MS-22
   Type: Manual, Static

   Initial state: N/A

   Input: N/A

   Output: The documentation for the code is easily understandable

   How test will be performed: The code documentation will be inspected by the group to verify that it is easily understandable and does not contain any gaps in communicating the code.

3. NFR-MS-23
   Type: Manual, Static

   Initial state: Doxygen documents have not been created

Input: Generate Doxygen documents (HTML and pdf)

Output: Doxygen documents are generated successfully

How test will be performed: The team will attempt to generate the Doxygen documents (HTML and pdf). Through the use of various commands the team will assess whether the documents have been created successfully or not.

Supportability Requirements

1. NFR-MS-24

   Type: Manual, Dynamic

   Initial state: SocialPy is actively running

   Input: User would like to view the commands that they are able to execute

   Output: List of all the commands is displayed for the user

   How test will be performed: The team will attempt to utilize the 'help' command to display all the commands that they are able to execute.

Longevity Requirements

1. NFR-MS-26

   Type: Manual, Static

   Initial state: N/A

   Input: N/A

   Output: All the code is divided into various classes

   How test will be performed: The code structure will be inspected to ensure that there are several classes and modularity to allow for longevity. The modularity of the code will be assessed to determine whether it will be easily modifiable in the future.

2. NFR-MS-27
   Type: Manual, Dynamic

   Initial state:

   Input:

   Output:

   How test will be performed:

3.2.6 Security Requirements

Access and Confidentiality Requirements

1. NFR-SR-28

   Type: Manual, static, code inspection

   Initial state: N/A

   Input: N/A

   Output: N/A

   How test will be performed: All connections between the system and the database use HTTP requests via the Firebase API. The test team will inspect the code and check if the password is stored locally anywhere on the system. If not, then the system is secure since the password information is not exposed to the public.

2. NFR-SR-29
   Type: Manual, Dynamic,

   Initial state: Database does not have users' email already registered.

   Input: User tries to login using a set of invalid credentials and symbols

   Output: System outputs "invalid credentials" message and prevents them from proceeding any further.

   How test will be performed: The unit testing team will manually try executing a set of random unregistered email and password combinations to try to get past the login screen of the system. If they are unable to, this means the test has passed since only registered users can get past the login screen and conduct post/profile commands.

Integrity Requirements

1. NFR-SR-30

   Type: Manual, Dynamic.

   Initial state: Database contains at least one user account

   Input: Email and password credentials for login. "POST ADD content" command afterwards.

   Output: Success message for logging in and adding a post. Failure message if unable to login due to invalid credentials.

   How test will be performed: The unit testing team will manually run commands to login and post content. They will use unregistered accounts and registered accounts to see if the program lets them past the login screen and allows them to post content. If the user recieved a failure message due to invalid credentials, then the test has passed as it prevents unregistered users from using the core components(Post/Profile commands) of the application.

1. NFR-SR-31

   Type: Manual, Dynamic.

   Initial state: Zero timeouts and attempts

   Input: Email and password credentials for login.

   Expected Output: "Too many failed login attempts. Please try again after 10 seconds..."

   How test will be performed: The unit testing team will use Python Unittest library to automatically execute incorrect login attempts 5 times in a row. The test passes if the system times out. Otherwise, the test has failed.

...

3.3 Traceability Between Test Cases and Requirements

Test IDs correspond to the tests found in this document. Requirement IDs correspond to the requirements found in the SRS.

Table 4: Traceability Matrix: Functional Requirement

Test IDs	Requirement IDs															
	FR1	**FR2**	**FR3**	**FR4**	**FR5**	**FR6**	**FR7**	**FR8**	**FR9**	**FR10**	**FR11**	**FR12**	**FR13**	**FR14**	**FR15**	**FR15**
FR-ST-1	X															
FR-ST-2		X														
FR-ST-3			X													
FR-ST-4				X												
FR-ST-5					X											
FR-ST-6						X										
FR-ST-7							X									
FR-ST-8								X								
FR-ST-9									X							
FR-ST-10										X						
FR-ST-11											X					
FR-ST-12												X				
FR-ST-13													X			
FR-ST-14														X		
FR-ST-15															X	
FR-ST-16																X

Table 5: Traceability Matrix: Non-functional Requirement

Test IDs	Requirement IDs																													
	NFR1	NFR2	NFR3	NFR4	NFR5	NFR6	NFR7	NFR8	NFR9	NFR10	NFR11	NFR12	NFR13	NFR14	NFR15	NFR16	NFR17	NFR18	NFR19	NFR20	NFR21	NFR22	NFR23	NFR24	NFR25	NFR26	NFR27	NFR28	NFR29	NFR30	NFR31
NFR-LF1	X																														
NFR-LF2		X																													
NFR-LF3			X																												
NFR-UH4				X																											
NFR-UH5					X																										
NFR-UH6						X																									
NFR-PR7							X																								
NFR-PR8								X																							
NFR-PR9									X																						
NFR-PR10										X																					
NFR-PR11											X																				
NFR-PR12												X																			
NFR-PR13													X																		
NFR-OE14														X																	
NFR-OE15															X																
NFR-OE16																X															
NFR-OE17/18																	X														
NFR-OE19																		X													
NFR-OE20																			X												
NFR-MS21																				X											
NFR-MS22																					X										
NFR-MS23																						X									
NFR-MS24																							X								
NFR-MS25																								X							
NFR-MS26																									X						
NFR-MS27																										X					
NFR-SR28																											X				
NFR-SR29																												X			
NFR-SR30																													X		
NFR-SR31																														X	

4 Tests for Proof of Concept

4.1 View/Add Posts

Create and view post

1. test-POC1

   Type: Functional, Dynamic, Unit, Automated

   Initial State: User is currently logged in.

   Input: Post content

   Output: The newly created post

   How test will be performed: Unittest will run internal commands to create a post under the current logged in user and then retrieve that post to verify it has been posted.

View other user's post

1. test-POC2

   Type: Functional, Dynamic, Unit, Automated

   Initial State: Database contains at least 1 post under the specified username

   Input: Username

   Output: Returns posts of the specified user

   How test will be performed: Unittest will run internal commands to return all posts by the specified user.

View all posts

1. test-POC3

   Type: Functional, Dynamic, Unit, Automated

   Initial State: Database contains multiple posts under multiple users.

   Input: N/A

   Output: Return all of the posts currently in the database.

   How test will be performed: Unittest will run internal commands that will return all posts for all users including the current user.

4.2 Authentication

Log in to existing user

1. test-POC4

   Type: Functional, Dynamic, Unit, Automated

   Initial State: Database contains a login for the specified email and password

   Input: Email address and password

   Output: Returns "Registration Success" message and updates the $self.user$ object.

   How test will be performed: Unittest will run internal commands to login with the specified credentials.

Create new user & log in

1. test-POC5

   Type: Functional, Dynamic, Unit, Automated

   Initial State: Database does not contain a user account with the specified email address

   Input: Email address and password

   Output: Displays "Registration Success" message and updates the $self.user$ object.

   How test will be performed: Unittest will run internal commands to register a new user account with the specified credentials.

5 Comparison to Existing Implementation

When comparing the the current product (SocialPy) with the original open-source Social Media application, there are some notable changes. However, a large part of the architecture and layout still remains similar. Both applications use the command line terminal to execute commands without any graphical user-interface. The code structure and layout between both remains largely the same by separating posts, profile, and the command_parser into their respective code packages. Each of these folders and functions has it's own suite of test cases in an accompanying function. The open-source application separated all its tests into a separate directory structure, whereas our tests are integrated in to their respective files. The open-source application had a specific time formatting function so they had a test suite for that which we did not include. They also had an exit command which our app does not, so no tests were needed in that regard. Our app has many of the same features as the open-source application as well as additional features such as deleting posts, viewing followers, and authenticating users. These are just a few of the activities that were NOT included in the original implementation which we had to build test suites for. Additionally, SocialPy has different commands for posts and profiles as well as a "help" option to get further information on these commands. We added tests for these features as well to ensure they worked as expected. Finally, our implementation includes data persistence via a NoSQL database whereas the original implementation has no way to save application data. Without

much setup (other than authentication), our tests can login and run tests on existing users and posts that are already in the database. The open-source application had to create "dummy" posts and users before running the tests since it had no data persistence.

6 Unit Testing Plan

Unit testing will be performed through the use of the Unittest Python framework

6.1 Unit testing of internal functions

Unit testing of internal functions will be performed for each of the functions for a given module to ensure robustness. These tests will consist of a combination of boundary/partition testing and Fuzz testing in order to detect any faults or potential error states causing the system to crash. All of these unit tests will ensure that the final product deployed for production produces behaviour that is predictable.

Every internal function will be tested with the following cases, where applicable:

- Normal case - input that it is expected to handle correctly
- Edge or boundary cases
- Abnormal/Negative path test cases resulting in error states.

Additionally, the functions will undergo integration testing once unit testing is complete. This is to verify compliance with the SRS document and ensure that each of the module functions are able to communicate with each other and output the expected system behaviour. During this phase, stubs and drivers will be created using the Unittest "Mock" library. In regards to test coverage, the team will aim for a minimum of 85% coverage.

6.2 Unit testing of output files

SocialPy will not create any output files, and as such, no testing on output files will be necessary.

References

7 Appendix

7.1 Symbolic Parameters

Not applicable. There are no added resourced that are included in this document.

7.2 Usability Survey Questions?

These survey questions will be used to test usability requirements.

1. How easy is it to understand the text labels in the app on a scale from 1-10.
2. Do you agree that SocialPy is appropriate for English speakers? [Y/N]
3. Is the documentation and help commands easy to interprete? [Y/N]. If not, explain what makes it difficult.
4. Do you find SocialPy commands unncessarily complex? If so, explain.
5. Rate your preference on the product color scheme as "good", "Neutral", or "bad".
6. Does the current color scheme seem distracting or cause you physical issues such as eye-strain? [Y/N]
7. Are you 13+ years old. [Y/N]
8. Please rate your experience using the application. (Intimidating. complicated, neutral, or excellent).
9. Given the current complexity and state of the application, would you use this application on a regular basis? [Y/N] explain why
10. On average, how many seconds does it take for you to navigate and find what you are looking for on the app? (1 second, 2 second, 4 seconds, 5-10 seconds, 10+ seconds). [Y/N] explain why
11. Are you content with the amount of time it takes for you to navigate through different parts of the app such as Profile view or Posts view pages? [Y/N] explain why
12. Are you content with the amount of time it takes for the system to add/delete/update/view your posts? [Y/N] explain why
13. Do you believe that the time it takes to authenticate credentials in the system is sufficient? [Y/N] explain why