SE 3XA3: Test Report
SocialPy

Team 1, SocialPy
Anando Zaman, zamana11
Graeme Woods, woodsg1
Yuvraj Randhawa, randhawy
Due April 12, 2021
Tag TR-Rev.1

April 12, 2021

Contents

List of Tables

List of Figures

Table 1: Revision History

Date	**Developer(s)**	**Change**
March 25, 2021	Anando Zaman	Copy template

This document contains all the results of the planned tests that were defined in the test plan.

1 Functional Requirements Evaluation

1. **FR-ST-2**

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

   Input: Email address and password

   Expected Output: Returns "Login Successful!" message and updates $self.user$

   Actual Output: Returns "Login Successful!" message and updates $self.user$

   Result: Pass

Register new users

1. **FR-ST-3**

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

   Input: Email address and password

   Expected Output: Returns "Account profile successfully made!" message and updates $self.user$ object

   Actual Output: Returns "Account profile successfully made!" message and updates $self.user$ object

   Result: Pass

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

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

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

   Result: Pass

1.0.1 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

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

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

   Result: Pass

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.

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

   Result: Pass

Delete post

1. **FR-ST-5**

   Type: Functional, 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 that says "post deleted".

   Actual Output: Returns a string that says "post deleted".

   Result: Pass

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.

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

   Result: Pass

Edit post

1. **FR-ST-9**

   Type: Functional, Manual.

   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.

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

   Result: Pass

Help commands

1. **FR-ST-10**

   Type: Functional, Manual

   Initial State: User is logged into the system.

   Input: User inputs "Help"

   Output: Returns a message with all the commands available.

   Actual Output: Returns a message with all the commands available.

   Result: Pass

1.0.2 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.

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

   Result: Pass

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.

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

   Result: Pass

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 "Name has been set to 'inputted name'" message. The database reflects the changes made.

   Actual Output: Returns a "Name has been set to 'inputted name'" message. The database reflects the changes made.

   Result: Pass

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 "Succussfully removed account!" message. Removes user profile and posts associated with the user from the database.

   Actual Output: Returns a "Succussfully removed account!" message. Removes user profile and posts associated with the user from the database.

   Result: Pass

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 FOLLOWINGS_DELETE username" string

   Output: Returns a "'username' was successfully removed from your followers list" message. Removes user from following list in the database.

   Actual Output: Returns a "'username' was successfully removed from your followers list" message. Removes user from following list in the database.

   Result: Pass

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

   Expected Output: Returns a "[username] was successfully added to your followings list!" message. Adds the username to follower list in the database.

   Actual Output: Returns a "[username] was successfully added to your followings list!" message. Adds the username to follower list in the database.

   Result: Pass

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 separate by new-line characters.

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

   Result: Pass

1.1 Tests for Nonfunctional Requirements

1.1.1 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

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

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

   Result: Pass

Style Requirements

1. NFR-LF-2

   Type: Manual, Usability Survey

   Outcome: All four users stated that the text and menus were easy to comprehend and self-explanatory.

   Result: Pass

2. NFR-LF-3

   Type: Manual, Usability Survey

   Outcome: All four users stated that the color scheme was good since it is a terminal program, so they had no complaints about the black and white color scheme.

   Result: Pass

1.1.2 Usability and Humanity Requirements

Ease of use Requirements

1. NFR-UH-4

   Type: Manual, Usability Survey

   Outcome: The user has agreed to our terms when signing up that they must be 13 or over to access the service.

   Result: Pass

2. NFR-UH-5

   Type: Manual, Usability Survey

   Outcome: The average user was able to navigate between tasks in around 5 seconds.

   Result: Pass

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.

   Actual Output: "Sorry, your command is invalid. Please type 'help' if you need to view the list of commands!" for russian words.

   Result: Pass

1.1.3 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

   Expected: Outputs the time required to query each branch

   Actual Output: Outputs the time required to query each branch

   Result: 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.

   Expected Output/Result: Outputs the time required to authenticate, boolean value indicating if less than 4 seconds.

   Actual Output: Outputted 1.354 seconds

   Result: 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.

   Expected Output/Result: Outputs the the timestamp data.

   Actual Output: Outputs the the timestamp data.

   Result: Pass

2. NFR-PR-10

   Type: Automated, Dynamic, UnitTest

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

   Input/Condition: Post string content

Expected Output/Result: 'Post "postContent" added'

Actual Output: 'Post "postContent" added'

Result: Pass

Reliability and Availability Requirements

1. NFR-PR-11

   Type: Manual, Dynamic

   Expected Output: System will have 99% uptime.

   Actual Output: The firebase cloud console shows statistics with currently zero down-time, thus satisfies the availability requirement thus far.

   Result: Pass

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.

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

   Actual Output: System returns the response at the time of output without having duplicate posts of the same post_id. Each postID is unique, thus no duplication on posts.

   Result: Pass

2. NFR-PR-13

   Type: Manual, Dynamic

   Initial State: No commands being made

   Input: Execute 20 Post(add,view) commands

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

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

   Result: Pass

1.1.4 Operational and Environmental Requirements

Expected Physical Environment

1. NFR-OE-14

   Type: Manual, Dynamic

   Outcome: The team was able to run the program using Python 3.7 without any errors.

   Result: Pass

2. NFR-OE-15

   Type: Manual, Dynamic

   Outcome: the team disabled the internet connection and received an error but when the connection was restored SocialPy functioned as usual.

   Result: Pass

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

   Expected Output: System returns the post

   Actual Output: System returns the post

   Result: Pass

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*

   Expected Output: Project successfully cloned and SocialPy login screen appears.

   Actual Output: Project successfully cloned and SocialPy login screen appears.

   Result: Pass

Installability Requirements

1. NFR-OE-19

   Type: Manual, Dynamic

   Initial state: Program cloned on atleast 3 computers.

   Input: double click *main.py*

   Expected Output: SocialPy login screen appears.

   Actual Output: SocialPy login screen appears.

   Result: Pass

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 Expected Output: Development team maintains feedback via a Gitlab issue tracker.

   Actual Output:

   Result:

1.1.5 Maintainability and Support Requirements

Maintenance Requirements

1. NFR-MS-21

   Type: Manual, Static

   Initial state: N/A

   Input: N/A

   Expected Output: The code is documented accurately with comments

   Actual Output: The code is documented accurately with comments

   Result: Pass

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

   Outcome: Our documentation has been sufficiently reviewed by the group and is easily understandable.

   Result: Pass

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

   Initial state: Doxygen documents have not been created

   Input: Generate Doxygen documents (HTML and pdf)

   Expected Output: Doxygen documents are generated successfully

   Actual Output: Doxygen documents are generated successfully

   Result: Pass

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 so the run the *help* command.

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

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

   Result: Pass

Longevity Requirements

1. NFR-MS-26

   Type: Manual, Static

   Outcome: The code has been sufficiently divided into its respective modules and classes to allow for longevity.

   Result: Pass

1.1.6 Security Requirements

Access and Confidentiality Requirements

1. NFR-SR-28

   Type: Manual, static, code inspection

   Output: No data inconsistencies or exposed API endpoints found. No confidential data is stored locally, and all communication is done through secure HTTP API requests.

   Result: Pass

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

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

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

   Result: Pass

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.

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

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

   Result: Pass

1. NFR-SR-31

   Type: Automated, 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..."

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

   Result: Pass

2 Comparison to Existing Implementation

When comparing the testing of the original open-source project and the current re-developed project (SocialPy), there are some notable changes. However, the architecture remains the same and thus the testing is quite similar. The original implementation had unit tests for all of the common modules to test for boundary and edge cases. The modules that were tested in the original implementation include AddPost, ViewPost, and CommandParser. Our re-implementation had several other unit tests due to additional features that were added to the program such as authentication, deletion of posts, view/modification of following/followers along with profile View/Edit functionality. Each of these were tested using a combination of unit tests and manual tests whereas the original implementation used only unit tests. 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. In summary, the structure of the tests was the same as each of the unit tests were put in the unit tests folder and were tested for boundary and edge cases. The differences lie on the number of tests due to extended functionality along with slight differences due to data querying as the original implementation did not have a database to persist the data.

3 Unit Testing

Unit tests were performed using the Python Unittest framework. Statement coverage was analyzed using the same testing library. Specific code for the unit tests can be found in
https://gitlab.cas.mcmaster.ca/zamana11/se3xa3-project/-/tree/master/src/Social%20Media%20App/

4 Changes Due to Testing

There was no changes to the code upon testing as the actual outcomes matched our desired targets. All functional and non-functional requirements were validated and showed no signs of inconsistencies. Thus, no changes were made to the source code upon analyzing the results of the tests.

5 Automated Testing

Automated testing was used for a large majority of the modules and functional requirements using the Python Unittest framework. However, there was also some parts that were infeasible for automated testing aswell due to requiring several user commands with the UI which could not be simulated easily. For these specific instances, manual testing was done verify these requirements.

6 Trace to Requirements

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

Table 2: 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 3: 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	

7 Trace to Modules

All module IDs were taken from the Module Guide Document. This is found on the next page.

Req.	Modules
FR-ST-1	M16
FR-ST-2	M15, M16
FR-ST-3	M15, M16
FR-ST-4	M4, M5
FR-ST-5	M4, M2
FR-ST-6	M15, M16
FR-ST-7	M11, M12
FR-ST-8	M2
FR-ST-9	M3, M4
FR-ST-10	M17
FR-ST-11	M13, M14
FR-ST-12	M9
FR-ST-13	M10
FR-ST-14	M7
FR-ST-15	M8
FR-ST-16	M11, M12
NFR-LF-1	M18, M5, M12, M14
NFR-LF-2	M17
NFR-LF-3	M18
NFR-UH-5	M17, M18
NFR-UH-6	M18, M5, M12, M14
NFR-PR-7	M4
NFR-PR-8	M15
NFR-PR-9	M4
NFR-PR-10	M1
NFR-PR-12	M1, M2, M3, M4, M5
NFR-OE-16	M16
NFR-OE-19	M18
NFR-MS-24	M17
NFR-MS-26	M17
NFR-SR-28	M15
NFR-SR-29	M15, M4, M5
NFR-SR-30	M15, M1
NFR-SR-31	M15

Table 4: Trace Between Requirements and Modules

8 Code Coverage Metrics

Total coverage for the system was 82% which met our goal of having atleast 80% code coverage. This is shown on next page.

Coverage report: 82%

Module ↑	statements	missing	excluded	branches	partial	coverage
CommandParser\CommandParser.py	98	28	0	52	11	67%
CommandParser\help.py	3	0	0	0	0	100%
Firebase\Authentication.py	97	69	0	36	3	26%
Firebase\firebase_creds.py	25	1	0	2	0	96%
Posts\AddPost.py	16	0	0	2	0	100%
Posts\DeletePost.py	14	4	0	2	1	69%
Posts\EditPost.py	16	0	0	4	0	100%
Posts\QueryPosts.py	36	5	0	10	3	83%
Posts\ViewPost.py	74	17	0	32	6	73%
Profile\AddFollowers.py	32	2	0	12	1	93%
Profile\DeleteAccount.py	43	33	0	12	0	18%
Profile\DeleteFollowers.py	25	0	0	6	0	100%
Profile\EditLocation.py	12	0	0	2	0	100%
Profile\EditName.py	12	0	0	2	0	100%
Profile\QueryFollowers.py	19	0	0	8	0	100%
Profile\QueryProfile.py	9	0	0	2	0	100%
Profile\ViewFollowers.py	31	8	0	16	4	74%
Profile\ViewProfile.py	22	2	0	10	1	84%
UnitTests\Function_NonFunc_UnitTests__init__.py	0	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_1.py	19	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_10.py	20	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_11.py	19	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_12.py	19	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_13.py	21	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_15.py	20	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_16.py	19	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_2.py	19	1	0	0	0	95%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_4.py	19	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_7.py	19	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_FR_ST_8.py	19	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_NFR_OE_16.py	22	2	0	0	0	91%
UnitTests\Function_NonFunc_UnitTests\test_NFR_PR_10.py	23	0	0	4	0	100%
UnitTests\Function_NonFunc_UnitTests\test_NFR_PR_7.py	26	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_NFR_PR_8.py	16	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_NFR_PR_9.py	17	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_NFR_SR_31.py	13	0	0	0	0	100%
UnitTests\Function_NonFunc_UnitTests\test_NFR_UH_6.py	19	0	0	0	0	100%
UnitTests\General__init__.py	0	0	0	0	0	100%
UnitTests\General\test_AddFollowers.py	28	0	0	0	0	100%
UnitTests\General\test_AddPost.py	23	0	0	0	0	100%
UnitTests\General\test_Authentication.py	18	1	0	0	0	94%
UnitTests\General\test_DeleteFollowers.py	25	0	0	0	0	100%
UnitTests\General\test_EditLocation.py	21	0	0	0	0	100%
UnitTests\General\test_EditName.py	21	0	0	0	0	100%
UnitTests\General\test_EditPost.py	25	0	0	0	0	100%
UnitTests\General\test_QueryFollowers.py	45	0	0	0	0	100%
UnitTests\General\test_QueryPosts.py	24	0	0	0	0	100%
UnitTests\General\test_QueryProfile.py	20	0	0	0	0	100%
UnitTests\General\test_ViewFollowers.py	34	0	0	0	0	100%
UnitTests\General\test_ViewPost.py	31	6	0	0	0	81%
UnitTests\General\test_ViewProfile.py	23	2	0	0	0	91%
UnitTests\General\test_firebase_creds.py	26	0	0	0	0	100%
UnitTests\TestRunner.py	3	0	0	0	0	100%
Total	1300	181	0	214	30	82%

Figure 1: Code Coverage Table