TEST PLAN FOR A FACEBOOK-LIKE SOCIAL MEDIA PLATFORM

ChangeLog

Version	Change Date	By	Description
version number	Date of Change	Name of person who made changes	Description of the changes made
1	11-02-22	Robert Dilworth	Initially Populated the Document

1 Introduction

The most important aspect of our web application, given its *raison d'être*, is its functional capabilities. Rather, the mark of a poor website—disregarding styling and layout issues—often stems from the absence or poor execution of features. In this way, our testing posture will be more unit-oriented, focusing on our code's operability.

To this end, our testing procedure will simulate and send faux inputs to our web application while also capturing and verifying the presence of expected outputs. Said outputs will be refined and derived from the requirements elucidated by our stakeholders. The tasks spawned from this undertaking will reflect not only the software's objectives but also our clients' intentions.

We will actualize our goal by leveraging Python's testing framework: PyTest. Specifically, we will engineer various test cases to ensure that our deployed software encapsulates the vision of our constituents. The testing itself will follow an assertion-detection archetype, where we claim *x*—where *x* represents the presence or dearth of content—and confirm *y*—where *y* represents anticipated or known response values.

1.1 Scope

1.1.1 In-Scope

The following functional and non-functional requirements will be assessed during our testing session:

- The ability to login and logout of the service.
- The ability to retrieve forgotten credentials.
    - A credential, in this context, represents a user's forgotten password.
    - Do note, there is not a mechanism in place to recover abandoned or misplaced identifiers.
        - An identifier is the user's unique username and email address. Both can be used interchangeably to classify a user.
- The ability to confirm whether an account has been made.
    - Due note, that this requirement is related to but is distinct from the act of creating an account.
- The ability to compose statuses and post them to the user's timeline.
- The ability to attach media to a created post.
    - Testing for this requirement also ensures that the file uploaded is of the correct type, checking the extension appended to the stream of data feed to the service.
- The ability to edit a status, altering the text and media attributed to a post.
- The ability to search for existing members of the service.

- The ability to send friend requests to users accessed via search.
- The ability to accept or decline friend requests.
    - Testing for this requirement also examines the state of the users' friend pages before and after the acceptance or rejection of a friend request.
- The ability to like posts contained within a friend's timeline.
    - Testing for this requirement ensured that the user's like counter toggled when clicked, alternating between incrementing and decrementing the value.
- The ability to comment on posts contained within a friend's timeline.
    - Testing for this requirement also implicitly ensures that the user's comment counter increments and decrements upon composing and/or deleting a comment.
- The ability to share posts contained within a friend's timeline.
    - The act of "sharing"—as defined by our web application—is the process of taking both the data and metadata ascribed to a friend's post and appending it to the timeline of the user who initiated the share. Think of Twitter's concept of "retweeting," but with a limited range of capabilities.

1.1.2 Out-of-Scope

The following functional and non-functional requirements will not be assessed during our testing session:

- The account disable feature triggered after 20+ unsuccessful login attempts.
    - A login attempt is defined as the transmission of user credentials with a valid identifier, a username or email address, and an invalid password.
- The ability to create an account with the service.
    - This subroutine is not explicitly tested because the procedure itself contains error handling via *FlaskForms*. Specifying a separate test would be redundant or derivative, which is why we choose to neglect it.
- The incrementing and decrementing of a user's comment and share counters.
    - This functionality, namely evaluating the presence or absence of comments and new posts, is assessed, while the counter is not examined. We elected to not engage this area because we have detected a bug within the process. We are currently addressing the issue, and, as a result, we have temporarily removed that portion of the test to enable traffic to and from the repository.

1.2 Quality Objective

With our testing we aim to achieve the following:

- To provide our customers with a cohesive user experience that is not only streamlined but also intuitive.

- To satisfy the fundamental requirements indicated by our stakeholders and clientele.
- To guarantee that our software conforms to the functional and non-functional requirements specified by our stakeholders.
- To identify and address bugs before deployment.

1.3 Roles and Responsibilities

Software Development and Software Testing Roles and Responsibilities:

(1) Collaborator: a member of the software development team
   a. Attends scrum meetings
   b. Provides input when discussing the creation of software
   c. Engages in polls and votes regarding software

(2) Tester: interacts with the software in an attempt to break or exploit its features
   a. Brainstorms and executes informal test cases
   b. Documents the steps taken to perform or exhibit bugs

(3) Software Architect: the engineer who describes the architectural aspects of a system
   a. Defines the functional and stylistic design choices made throughout the project
   b. Outlines how key components of the system should operate

(4) Team Lead: the person tasked with accepting the leadership role of a project
   a. Delivers or submits project assets on behalf of the software development team
   b. Coordinates discussions and moderates polls to make software-related decisions
   c. Periodically inquires about the development of code

(5) Automation Tester: derives and implements automated test cases
   a. Interfaces with continuous integration (CI) services to run tests when a trigger is detected
   b. Crafts formal, standalone test cases that will be reused to assess code

(6) Full-stack Developer: a developer with a non-specialized skill set that can address a broad range of software-related issues
   a. Takes on a myriad of responsibilities, satisfying both Full-stack and Back-end workloads
   b. Supplements the project at every stage of development, serving as a replacement when necessary

(7) Software Tester: derives and implements formal testing scenarios while generating code
   a. Performs manual testing of software
   b. Documents and reports defects when they are discovered
   c. Analyzes bugs and reviews code

(8) Lead Back-end Developer: responsible for generating the code that handles the hidden aspects of a system, establishing the logic and routines that underlie the visible mechanisms of a system

a. Develops and fleshes out the features specified for each Sprint

Name	Net ID	GitHub username	Role
Brett Baltz	blb820	BrettBaltz	Lead Back-end Developer, Software Tester
Robert Dilworth	rkd103	rkd103	Full-stack Developer, Automation Tester, Team Lead, Software Architect
HeFeng Ou	ho170	ho170	Tester
William Neeley	wtn36	Weeyumn	Collaborator

2 Test Methodology

2.1 Overview

For this project, we adopted an agile testing framework.

We decided upon this methodology because we anticipated frequent interactions with our stakeholders. As we communicated with our patrons, we also periodically demonstrated prototypes of our code. During these presentations, we would often receive feedback, suggesting major or minor changes that would later be refactored.

Given our circumstances, the agile framework was a fitting approach due to the stringent time constraints of the project, the introduction of new features, and the close relationship between the developers and stakeholders. Above all else, we needed to be flexible, which is a quality that is synonymous with the agile method.

2.2 Test Levels

Given that we developed the components of our project using the “big bang model”, we exercised little to no integration testing; rather, we implemented a feature, tested it, and passed off the code for the next collaborator to build upon. We found that taking this approach, as opposed to a more parallel technique, reduced the likelihood of integration mishaps.

Moreover, we did not incorporate aspects of acceptance testing. We forsook this area of scrutiny because our software was expected to be deployed to a limited audience. We did not account for the potential effects of our software on the operating systems or hardware hosting the web application nor did we assess the portability or scalability of our code. For all intents and

purposes, we aimed to deliver a functional product, anything thing beyond that was given a lower priority.

We did, however, extensively utilize unit and system testing. Namely, the limited scope of the project coupled with the tight schedule only permitted limited unit and system testing. Granted, we did not set up a mechanism to log the results of our tests as our tool of choice, PyTest, does that for us. Data-flow analysis, on the other hand, was executed manually through rigorous testing.

2.3 Bug Triage

Our "Defect Triage" stemmed from our initial User Story assessment. That is, when we originally derived the functional requirements at the outset of the project, we assigned the following attributes to each feature: (1) priority, (2) risk, (3) category, and (4) story points. We then catered and developed our Defect Triage based on the values assigned to those attributes. For example, we designated User Story A.1, create an account, as having high priority, elevated risk, and inflated story points. As a result of this classification, we allocated more time to get those features operational.

However, our general Defect Triage to the following approach: (1) ponder a temporary patch to amend the bug, (2) determine a definitive solution when more information is gathered, (3) record the steps used to produce the erroneous behavior, and (4) prioritize the defects and arrange an itinerary to address the bugs.

2.4 Suspension Criteria and Resumption Requirements

Suspension criteria define the conditions that suspend all or part of the testing procedures currently in development. Resumption criteria, on the other hand, determine when testing can resume after it has been suspended.

The suspension criteria for our project center around the automated test cases ingrained into our code's repository. If an upload to either the master or work environment branches fails a test, the repository should reject the submitted code. This gatekeeping or barring of upload privileges captures the suspension criterion that should halt testing. Conversely, successfully passing a triggered sanity check would facilitate further testing, which encapsulates resumption criteria.

2.5 Test Completeness

The following assignments must be addressed before our testing is considered complete:

- All tasks labeled as "code refactoring" must be repaired.
    - Code refactoring denotes a subset of bugs that possess an elevated priority over ordinary defects.
    - At the very least, all other bugs need to be addressed by the next release of software; however, at present, they do not impose an immediate issue.

- All automated test cases must be executed.
    - Moreover, each instance of submitted code should pass every test, of which there are twelve.
- All formal and informal manual test cases must be carried out.
    - Verification of manual testing acts as a resumption criterion that facilitates further testing or code development

3 Test Deliverables

The following artifacts will be delivered throughout the testing lifecycle.

- Test Plan Document
- Automated Test Case Scenarios
- Bug Reports (by way of GitHub Issues)
- Stakeholder Sign Off (by way of digital approval)

4 Resource & Environment Needs

4.1 Testing Tools

The following external tools are needed to test the project:

- Access to our team's GitHub repository
    - The GitHub repository can be retrieved at the following link: https://github.com/rkd103/Group-11.
- Access to the account tied to our continuous integration service: CircleCI
    - The pipeline can be retrieved at the following link: https://app.circleci.com/pipelines/github/rkd103/Group-11.
        - Requires "rkd103's" authentication to view.
- Access to the most recent version of Python and pip: Python >= 3.10.2 & pip >= 22.3
    - Reduces the area of variation that could lead to further issues
- Access to a web browser
    - The browser used during development will be specified later
- Access to the packages listed in the "requirements.txt" document hosted on our group's GitHub repository
    - Install the packages using the command: $ pip install -r requirements.txt

4.2 Test Environment

The following operating systems, software, and applications denote the minimum requirements needed to test the application.

Do note that this list is just a reflection of the devices used to generate and assess the code. You may find that you can host the project and test it on systems with specifications that are inferior to or exceed our own.

- Windows 10 and above
    - Including the latest version of Windows 11
- macOS 12 Monterey and above
    - Including the latest version of macOS 13 Ventura
- A Chromium-based web browser
    - Including Google Chrome, Brave, etc.
- A code editor
    - Including Visual Studio Code 1.73.0 and above
        - Or some other equivalent software such as macOs' Terminal
- GitHub
- Circle CI

5 Terms/Acronyms

Make a mention of any terms or acronyms used in the project

TERM/ACRONYM	DEFINITION
CI	Continuous Integration
Collaborator	See Section 1.3
Tester	See Section 1.3
Software Architect	See Section 1.3
Team Lead	See Section 1.3
Automation Tester	See Section 1.3
Full-stack Developer	See Section 1.3
Software Tester	See Section 1.3
Lead Back-end Developer	See Section 1.3