FLOAT

Test Plan Document

Aaron So 17150137
Clarence(Junwei) Su 36387132
Ming Hin Matthew Lam 33056145
Rachel Yeo 30032122
Richard Xie 55786140
Samuel Farinas 17721144
Selina Suen 14022140

Introduction

The concept of Float is to create a fundraising system where users can view, contribute to and start charity campaigns. As a consequence of how these charity campaigns are structured, the project itself is to rely heavily on the user locations and secure payments. Our implementation of Float will be in the form of an application targeted towards Android devices, with the previously mentioned functionalities supported by the following: a Facebook login system for users, a PayPal interface for receiving payments, a Firebase database for storing information regarding users and campaigns, as well as Google Maps to connect users and campaigns to locations. An extensive user interface will also be created to connect all subsystems together.

Verification Strategy

In the beginning stages of verification and testing, we plan to talk to charity organizations regarding the idea of the app and our stated requirements, as to receive feedback on if the idea itself is something that will be of interest to the common population. If this proposed meeting goes well, we will also aim to have the chosen charity organizations be part of later user acceptance testing. We have chosen to focus on charity organization workers as our first point of verification as, even though they may not cover the entirety the project's target audience, they themselves are well versed in the manner of charity work and what motivates people to become involved and donate. Thus, talking to them will give us a sense of if our requirements for the project are valid.

One of the main sources of feedback that we will be heavily relying on is the weekly instructor meetings. Feedback received from these meetings will let us know if we are on the correct track to fulfill the requirements we initially set up for the project. Additionally, we plan to further verify that the application meets user's real needs by running multiple instances of beta testing at the end of the term. Such testing will be directed towards our main target audience of reasonable well-off young to middle aged adults.

With regards to the multiple instances of user acceptance testing, we are currently planning to have three testing/feedback sessions which are as follows: testing of the prototype, testing of the pre-release of the application as well as testing of the actual release of the application. During these testing sessions, we will strive to receive detailed feedback from all participants. Though we have not fully developed the questionnaire we plan to provide to the participants, we will be looking to find feedback concerning if the app is something the public would be interested in, and what changes/add-ons should be made to the idea of the project. On top of such questions regarding the general overview of the project idea, we will also seek feedback regarding the app's user interface and any bugs that are apparent. Upon receiving this feedback, we will then do our best to integrate suggested changes into the application.

Non-functional Testing and Results
Reliability testing: We plan to test our application for reliability amongst our group members. We will first verify that all the information on the application is agreed upon when run by different users. To do this, we will have our group members run the application, and then have one member update the campaign information (either by spreading or creating a campaign within the area). The other group members will then check to see that they have obtained the updated information on their application. Secondly, different group members will run through the use cases from our design document to ensure there are no crashes or errors that occur.

<u>Usability testing (with specific testing plan):</u>

1. **Select users for testing**
   As stated in the requirements document, the primary targeted users of our application are those who wish to promote charity awareness and donations by sharing stories and images. With this in mind, we will give priority to people who participate in regular charity activities when selecting a testing user. Additionally, to get a more representative sample, we will consider the diversity of their background when choosing our sample. Given the size of our project and constraints on resources, we will start with a small sample, around 5~10 people. A small sample size is more manageable and allows us to spend more time with each testing user, thus extracting more useful feedback from each one.

2. **Design feedback process**
   Usability is a subjective dimension of the application. One major challenge for designing usability testing is collecting feedback from testing users such that the feedback genuinely reflects their experience with the application (the testing user themselves may not even know how they feel when using the application). With this being said, we would prefer feedback that is quantifiable, objective and interpretable. Moreover, given the nature of our application, we will focus on designing tests for learnability and efficiency, since safety is not a big concern in our current situation.

3. **Learnability test**
   To determine how much time a user requires in order to learn how to use the application, we will give testing users some one-sentence tasks and ask them to finish the tasks without any aid from instructions. An example of one of these tasks is: create a new campaign.
   We will record the time each user takes as well as the steps they take to finish the task. In this way, we will be able to find out how intuitive our GUI is and how long the user needs to figure out how to use our GUI.

4. **Efficiency test:**
   Similar to the previously mentioned learnability test, we will again give testing users some tasks to finish. This time, instruction will be given in order to aid the testing user to finish the tasks.
   An example of a such a test is as follows:
      Follow the steps below to create a campaign:
   1. Click the button on the bottom of the screen to navigate to map page
   2. Click the create campaign button
   3. Fill in necessary information
   4. Click save campaign button

   Again, we will record the time as well as the steps each user takes. In this way, we will be able to find out how efficient our application is, given that the user knows the steps required to perform the desired action.

5. **Analyzing results**
   Comparing the result from the two tests above, we will then analyze how steep the learning curve for first-time user is and how efficient our GUI is for regular users. If the times or steps needed to complete a task is too large, this is a clear indication that we need to improve our UI to make it more intuitive and easy to use. Additionally, analyzing time needed for individual tasks, we could identify the bottleneck of our application and improve on it.

Load testing: We will simulate load in our application by starting a large cluster of campaigns in an area. A user should log into the application without being impacted by the volume of the campaigns (there should not be a significant impact on the performance of the application such as lags, and the user should be able to still easily navigate through the campaigns).
Compliance testing: We must ensure that our application stores data safely and securely.

Functional Testing Strategy

Testing categories

Tests performed for this project will fall into two major categories: integration testing and unit testing. As showed in the design document, the application can be divided into 6 sub-systems. Therefore, having unit and integration testing is important to ensure the subsystems are working and communicating properly.

To be more specific, we will have test suites for the Google API, Facebook API, Firebase database, and GPS location services (refer to the implementation for the test suit). It is difficult to write unit test cases for Paypal because the PayPal testing environment requires the user to manually check that charges have been incurred on their account through the PayPal website. Therefore, it is hard to assert that a transaction has been executed properly by automated unit testing. To test Paypal, we will use exploratory testing. We will have a team member act as an user for the Paypal system and adapt the tour approach which is to document the events by the following categories: goal of the actions, context, steps taken to finish and result/feedback from the system

Testing frequency

If time permits, we will set up a testing server like Jenkins to have test cases run on the implementation once per week (which is roughly half of the sprint cycle). Otherwise, the test suite will run before any new code is pushed to Github. In this way, we utilize regression testing to make sure that the application fails early and to minimize the scope we need to inspect in order to find the bug.

Testcase plan

The main strategy we will use to come up with test cases for the test suite will be domain partitioning. We will partition the input domain and have test cases inside each interval as well as on the boundary. We argue that domain partition is an effective testing strategy when developing application with lots of features, and function.

For crucial methods such as `searchNearbyCamp` (a method that returns campaigns within the visible radius of the current location), we will aim for a high statement and branch coverage (i.e covers all essential statements and branches of the production code). We understand that high coverage does not necessarily mean a better test suite, but it is one of the necessary conditions for a test suite to be good. For methods with complex boolean expressions, we will employ MCDC (Modified Condition Decision Coverage) to cover as many possibilities as we can. In the end, we will run our test suite against the mutation test to guarantee the ability of our test cases to detect bugs. We will do that by using PIT.

Adequacy Criterion

The adequacy criteria list for our application is as follows:

- At least one test exists for each subsystem
- Test cases are to be focused on error-prone points of the system
- Tests exist that target all possible points of interactions between subsystems
- At least one test exists for each requirement and use case

Amongst each subgroup, members will initiate their own testing to ensure all methods written are tested and functional before combining the components together. This includes separate testing for the Facebook login, the database, PayPal payment, and the GUI.

The use cases and requirements will be tested upon combining the application together. We will cover each use case we have provided in our Project Requirements documentation.

During unit testing, the coverage of our tests will be checked through Android Studio's IDE. EclEmma also provides a library called JaCoCo that can be added as a plugin on Android Studio, which will produce a report indicating the coverage of the test suite. With the current upgrades to Android Studio however, third party plugins are not necessary to check code coverage for JUnit tests.

Test Cases and Results

Test #	Requirement Purpose	Action/Input	Expected Result	Actual Result	P/F	Notes
1	To ensure the user has an account	User logs into the application using Facebook	The user successfully logs on and is brought to the main page of the application. The information provided by their Facebook account can be accessed by our application for processing			
2	Payments can be made to charities	User enters the amount of money they wish to donate to a campaign	The user is prompted to submit their preferred payment option and the transaction successfully goes through via PayPal	Using the PayPal sandbox to test, the payment option and transaction is able to be successfully processed.	P	
3	User can start a campaign	The user presses the “Start Campaign” button	The user is brought to a new page in the application where they can fill a form to start a campaign			
4	User can select a charity through the map	The user taps a marker on the map	A window fills half of the screen displaying the campaign’s picture, title, user, charity. A ‘details’ button and a deselect ‘x’ button is also in the window.			

5	User can view charity information	The user taps the “list” button in the bottom-right of the screen. The user scrolls and taps on a campaign.	The map is replaced by a list of campaign, each entry containing same elements as in test 4. The list scrolls to reveal more charities. The list view is replaced by the campaign page with same info as in test 4 and a text description.			
6	User can view charity information	After selecting a campaign, the user clicks on the charity’s logo/name button.	The campaign page is replaced by the charity page, containing the charity logo, a “Verified charity” status, statistics, and a blurb (small write-up).			
7	User can view account information	At any page, the user clicks the person icon in the top-right of the screen. User slides finger up and down the screen	The present page is replaced by the user’s page containing the user’s username, thumbnail picture, statistics, blurb (small write-up), and settings (revealable through scrolling).			
8	We obtain the user’s consent for a future payment	When the user creates a campaign, they make a pledge that brings them to a consent page which allows us to bill them in	By clicking on agree, we can grab necessary information from the user to create a future payment	Using Sandbox to test, we were able to process a refresh token from the server which can be used to create a payment in the future	P	Server response may be slow at times - need to wait a few seconds for a response

		the future if they click “agree”				
9	When a campaign succeeds, we charge the user the amount they pledged	A campaign expires and reaches its goal destination.	The user is charged the amount they pledged at the start of the campaign	Using Sandbox to test, we were able to process a payment from a previously obtained refresh token, and the total account balance of the test account decreased by the amount we specified	P	Server response is slow at times, so the payment needs to wait in order to fully process