Testing Plan

Inclusive Solutions

Members:

Connor Kilgore, Olive Price, Monika Beckham, and Ethan Green

Sponsor:

Susan Purrington

Mentor:

Italo Santos

Apr 7, 2023

Overview:

This document details the activities intended to ensure that the project's implementation demonstrates the required functional and non-functional attributes.

Version 1.0

Table of Contents

Introduction

Our app, welcomed. serves the purpose of maintaining a crowd-sourcing based system capable of determining more specific necessities that are shown within local businesses and services. These needs follow along the lines of disability accommodation, and being safe from discrimination. We intend to do so by allowing users to review or report their own experiences within our app and determine if their specific needs were met. When others look to confirm specific data, it will be reflected based on those reviews and reports.

In order to ensure the success of this project we will be implementing tests that will need to be met before we can deem the application to work properly. The areas we need to test the most are:

1. If the map is able to accurately portray data to the user based on information collected from the database.
2. If the application is able to successfully communicate with the database from any use case.
3. If the user account is able to successfully verify account information and store it within the database.
4. Is the user interface and features of the application easy to use and understand for a new user that may be less familiar with our application and technology in general.

We mean to test the first three areas the most because they cover the three critical functions of the application. The apps review, business, and profile managers that exist within the backend. Particularly step two should be focused the most because it covers a wide range of cases in which the application may not work as intended. Step four will be additional testing on the front end and allow us to spot any hidden bugs from other users.

Unit Testing

First comes unit testing. Unit testing specifically targets individual operations performed by parts of our program to ensure they work as expected. These tests generally utilize sample data to simulate a run of individual units or methods in our program. The benefit of having these tests cannot be overlooked. For one, these allow us - and any future collaborators - to properly test code without interacting with live data outside a testing environment. Secondly, they can identify issues that have been overlooked much quicker - especially with regards to issues that may arise from unintended side-effects of modifications of the code. Consequently, the less time spent on identifying issues that have been identified by unit testing, the less money spent and more effort can be put into improving the application.

The main tools which will be used to construct our unit tests will be the Java libraries JUnit and Mockito. JUnit will allow us to construct code which is designed to run our tests and designate the expected output from each test. Mockito will allow us to broaden our unit testing particularly with regards to Android specific methods which cannot be accessed from within the unit testing environment. This will be crucial for us to establish tests that emulate live behavior and, thus, sufficient tests.

Our unit tests will largely focus on sections of our code that receive and send code to our database and Google's Firebase platform. This primary includes our profile management and review management sections of our code. Units related to mapping largely cannot be covered by unit testing as the relevant methods require definitive input from Google's map platform, which can become costly if run more than necessary. We also won't be able to simulate this behavior in a useful way due to the unit's reliance on proprietary code and calls. Database calls can largely be simulated by mock objects and won't provide any more financial burden incurred by sending and receiving data. We will also be focusing on testing the more crucial methods in each unit rather than testing whole units. Many of our methods on the client-side focus on user

interface interactions which fall better under the other testing scopes, and we do not want to make our unit tests too broad as to take away from the time and importance of these other testing scopes.

First, we'll focus on testing related to our localized profile management. The first component of this is sign-up. Sign-up processes as a whole expect two results: either a user was signed up successfully or not. We can further subdivide these results into expected cases. First, a failure condition will occur if a user provided sign-up information that does not meet our expectations. This can include any form of erroneous input, whether that be some input in an email field which is not an email, or a password which doesn't meet our standards. Next, a failure condition will occur if a user provided appropriate information, but the client's call to Firebase to create the user entry failed. Finally, we have a success condition if and only if the user's information matches the expected input and the client successfully sends this information to the Firebase platform. These results exist largely as a binary rather than a spectrum, that is, there's not much in the way of boundary values for each of these cases. Part of this is due to a large part of the error checking occurring on the Firebase side which eases the amount of cases we have to deal with in our own testing.

Next, we have login procedure related testing. This unit is very similar to our sign-up related testing, but we have a few additional cases to consider. Regarding erroneous input, we need to consider the case in which the user has provided the right email for an account, but has also given an erroneous password. This is important as with all systems with a similar login system, we may wish to inform the user that they have the right email but the wrong password. Since our login system has more moving parts than our sign-up, we will also need to consider cases where credentials are correct and Firebase calls completed successfully but our code still provides an error situation. For the most part, it is safest for us to assume that the error was proper and clear any credentials, but we will need to keep this case in mind in our testing.

We then must consider our localized profile manager code. This is related to user input for demographics and preferences while using our application. There are three cases we should consider while developing our testing. First, there is the case where we can retrieve user input correctly, it is formatted as we expect, and we successfully make the exchange of information with our database. Second is the case wherein the user input everything as expected, but the client is unable to relay this information to the database properly. Last, the user has input information that is incompatible with our database. Database interactions like this - as mentioned - can be simulated by mock objects to lessen any extraneous load on our server and allow us to test extreme cases without potentially harming our production environment. Edge cases we will need to consider here are as follows. One, we have a success condition where all data was exchanged as expected, but the server handles the data improperly and saves erroneous data to the database. Two, the user inputs potentially dangerous strings into text input boxes, which could potentially be used to alter our SQL database in a dangerous way.

Finally comes our review management and report management units. These are essentially the same for the purposes of unit testing as the profile management unit as they perform similar functions, taking user input and sending it to the database. We will want to construct individual tests for each of these due to the different database tables they reference as well as their individual importance. Largely, though, the implementation and considerations of this testing is identical to our profile management.

Integration Testing

Integration testing is a type of software testing that focuses on *interactions* between major modules and components of our system. The goal of integration testing here is to ensure that these modules work together as intended and that data is exchanged correctly between them. Integration testing is particularly important for a complex system such as ours, where many different modules must work together seamlessly to achieve the goals of ‘welcomed’.

In our case, we will be using a Client-Server architecture for our system, with six major client-side modules that interact with server components. To test the integration of these modules, we will focus on the boundaries between these modules and their interactions with the AWS server.

Our approach to integration testing will involve a combination of manual testing and automated testing. We will use manual testing to ensure that the system is functioning correctly from a user's perspective and to identify any unexpected behaviors. For automated testing, we will use test harnesses to simulate interactions between modules and to verify that data is exchanged correctly. To test the integration of the major modules in our code, we will follow the following plan:

Login Module Integration Testing:

a. Test harnesses: we will use Firebase Authentication emulator to simulate the interaction between the Login Module and Firebase Authentication service.

b. Test objectives: we will verify that users can log in and create new accounts as intended and that the authentication token and user ID are passed correctly to the Profile module.

Profile Module Integration Testing:

a. Test harnesses: we will use Mockito and JUnit to simulate interactions with the User Account Manager module and to verify that the user's attributes, identities, and accommodation needs are stored and retrieved correctly.

b. Test objectives: we will verify that the Profile Module correctly displays the user's information and allows them to select their attributes, identities, and accommodation needs. We will also ensure that the user's information is stored correctly on the server side.

User Profile Manager Module Integration Testing:

a. Test harnesses: we will use Mockito and JUnit to simulate interactions with the server-side User Profile Manager module and to verify that the user's attributes, identities, and requested business tags are exchanged correctly.

b. Test objectives: we will verify that the User Profile Manager Module correctly establishes a user's access on the server side and that the user's information is stored and retrieved correctly.

Review Manager Module Integration Testing:

a. Test harnesses: we will use Mockito and JUnit to simulate interactions with the database and to verify that reviews and business information are inserted and read correctly.

b. Test objectives: we will verify that the Review Manager Module correctly sends and receives review information in the expected JSON format and that it can retrieve and display reviews for a business with a given Google Maps Place ID.

Place Details Module Integration Testing:

a. Test harnesses: we will use Mockito and JUnit to simulate interactions with the Place Search Module and to verify that place details are displayed correctly.

b. Test objectives: we will verify that the Place Details Module correctly defines the model for representing place data and that it can display reviews and tags for a place.

Place Search Module Integration Testing:

a. Test harnesses: we will use Google Maps API to simulate interactions with the Google Maps service and to verify that places are auto-filled and displayed correctly on the map.

b. Test objectives: we will verify that the Place Search Module correctly communicates with the Google Maps API and that it can generate a map view of nearby places that have been reviewed in a Google Map.

By following this plan, we will be able to thoroughly test the integration of the major modules in our code and ensure the best possible user experience.

Usability Testing

Usability testing is the process of selecting representative beta testers to see if our application is able to work intuitively and is actually user friendly. These testers need to be walking into this application almost completely blind and come from a variety of backgrounds to make sure that the proposed application makes sense to our target audience visually and mechanically. We intend to work with our client to release a small beta test to the android play store for users to test the apps usability and collect information for our database.

We would mostly like to focus on users of a variety of backgrounds that fit within our target audience. This will include individuals of minority groups (including but not limited to race, ethnicity, religion, etc.). We are also looking for individuals with disabilities that require accommodation. Lastly we want a general user who may or may not qualify for the previous two sections, but is able to give an idea of how intuitive the app is for less and more tech savvy users.

Our regime will mostly center on how well our application is able to communicate with the backend. In order to do so we will have put aside a few days at a time in which the backend will be untouched by us so that we can log information from beta users to see at what times and use cases our application doesn't work as intended. Once the testing period has concluded, we will review the logs and attempt to fix the issues that were encountered, this will massively decrease the likelihood of a user accidentally breaking the applications functionality.

Next we will be asking for information from the user's object experience as well. If a minor bug is found (i.e. a visual bug) it will be reported on what page and where. This will be trivial for us but allows us to have extra sets of eyes for things such as that. For any major hidden issues that may compromise the functionality of the application, we will ask the beta user to recount the exact use case that led up to said issues. This will start from when they opened

the app all the way up to issues encountered. That information will be sent back to us so that after the testing period is concluded we may sniff out and fix those bugs as well.

Lastly, we will ask users for a recount of their experience over the testing experience and to tell us in a brief description what they liked and disliked about our application. By getting a larger sample size we will be able to better understand what features are worth improving and what features may want to even be removed altogether. We can also better understand how intuitive the user interface is and how much users may enjoy it visually.

We intend to conduct usability testing between April 9th - 23rd in burst intervals of 2-3 days to allot periods in which we are not toggling with the backend in a way that may compromise the applications integrity. The beta testers will be friends, colleagues, and family of our client as well as ourselves. We as testers will be able to look for more technical breaks, while the other users can focus more on blind-testing and feedback.

Conclusion

Our testing plan has detailed our efforts to ensure that our product will have a strong foundation of maximized error-free performance, resulting in a smooth experience for our client, future users, and possible future developers. We have shown our plans for covering testing for individual code components, testing that each of these components interact in an expected manner, and testing the side of our coding efforts that will be shown to end-users.

Unit testing will ensure that the fundamental components used to establish the individual functionality of localized account handling, as well as user feedback in the form of reviews and reports are handled properly. Implementation testing will ensure that each component of our application will interact with each other sufficiently and in a way that is integral to our application being a proper product rather than solely a proof of concept. Usability testing will bring it all together, to show the results of our efforts and provide useful feedback and new perspectives on the functionality and usability of our end product.

In short, we have outlined our work to make our product the best it can be for our client and beyond.