# MoGreen

## Software Testing Plan

Version 3

April 4, 2019


Project Sponsor:
Ellen Vaughan and Brock Brothers

Team Mentor:
Ana Paula Chaves Steinmacher

Team Members:
Cassie Graham, Jennie Ryckman, Chase Mosteller, and Justin Shaner


NAU
NORTHERN ARIZONA UNIVERSITY

# Table of Contents

# 1. Introduction

The NAU campus strives to be as green and sustainable as possible, but as the NAU student population grows so do instances of overflowing trash bins, full dumpsters, and campus litter. Basic maintenance problems occur more often as well, like broken sprinklers and blocked trash cans. There are few and poor mechanisms in place for students and community members to engage in caring for their campus themselves and alerting the appropriate facility managers for problems outside the scope of what any community member can handle. The current system in place is outdated, unknown, and unusable. This becomes more problematic as the size of the NAU community grows.

Our Capstone project is the "Clean My Campus" mobile application and companion web application, designed to address all the above problems. The mobile application will make reporting minor issues easier for students and the general community and allow NAU facilities to receive and track these reports faster via the web application. It will also allow students and community members to take initiative in engaging with their communities by personally cleaning up litter. This application will help further NAU's sustainability efforts and keep our campus clean and green.

All the key requirements for our mobile application and web portal have been implemented, and now our software will need to be tested thoroughly to ensure our requirements are being met and that our software is robust. Software testing allows the creators of software and those invested in it to ensure its quality. There are a few different ways to measure the quality of a piece of software. Generally, important areas to test are that a piece of software is easy to use, is doing everything it's supposed to, and has minimal errors. These are the key areas that we plan on testing our software for.

Our team will be doing three types of testing: unit testing, integration testing, and usability testing. Unit testing ensures that each module produces the desired results with minimal errors and will be how we ensure that all our key requirements are being met. Integration testing will allow us to make sure that all the modules and components of our software are interacting correctly, and that data is being stored and retrieved correctly. Usability testing is how we will check that our software is easy to use for the intended audience of each component, and where we will be putting a higher emphasis. The purpose of our software is to replace an outdated system, and so our software will not be effective if it is not usable as intended.

Following is a specification of the types of testing we will be doing and a detailed plan for each.

# 2. Unit Testing

Unit testing for software tests the correctness of individual pieces of code, and how those pieces of code will behave given different inputs and interactions. Unit testing can also help ensure the modularity of software, drawing attention to existing units, and pieces of code that are inappropriately coupled.

For us and our software, we will be using unit testing to ensure primarily that our application is behaving appropriately while meeting key requirements. They key requirements for our mobile application are:

- The ability to view a map with designated “zones”, colored to indicate their status
- Make quick reports about maintenance or litter problems that affects the color of
- zones
- Make full, more detailed reports about issues that require staff attention

Unit tests for the application will involve performing different actions on parts of the application interface so we can note the results. Android Studio, the environment our application is built in, also has built-in support for unit testing that we can take advantage of.

Unit testing for our web application will be considerably lighter than for the mobile application. Our web application is written with HTML and JavaScript, neither of which are particularly intuitive to write unit tests for. However, we can still check how our scripts handle different input and will be testing those areas thoroughly.

## 2.1 Mobile Application

There are multiple areas of the mobile application that we will not be testing with unit testing. Most of the functions we will not test are implementations of an existing API, and thus our control for error handling is limited. For example, our implementation of user authentication is using Firebase and the error codes it gives will need to be handled specifically. Other features are redundant to test, such as every action bar functionality. It is included with testing the menu bar functionalities, so we are not going to note it specifically. Action bar features are one-line functions that are controlled by APIs or are basic functionalities, obvious when they are defective.

Some features, such as the phone rotating and keeping the maps status, can only be tested manually as the view seen before and after the change do not have consistent changes to the information. These features are controlled using the manifest for the application and are therefore features covered in the Gradle build and are not our responsibility to test. For some functions, we have implemented try/catch blocks to ensure that any errors do not crash our application. These functions mainly contain Firebase actions, which are not our responsibility to test specifically. We will be testing that functions with try/catch produce appropriate results, but do not consider them full “units” in this context.

The following functions will be tested thoroughly, as they have created multiple errors and crashes in their creation. Most of these involve the flow of information as it passes through our application, and as such would crash the application if they were to contain bad data. Tests that involve picture storage, report contents, and user information (as provided in the main menu) are required to pass for our application to perform as intended.

The functions we will be testing are detailed in the following section.

### 2.1.1 Unit Tests

**Unit Name:** sendPicture() is a function responsible for taking, storing, and displaying an image for the application.
**Description:** To test our image storage we will need to have a sample bitmap image thumbnail that will be tested against what is stored from our image storage function.
**Successful Path:** The flow of this method takes the result data from our camera instance and returns a thumbnail sized bitmap that is then sent to the database upon completion.
**Expected Outcome:** Normally this function will try to open the camera, take a picture, and return to the previous activity. When successful the previous activity will have a smaller image appear.
**Unsuccessful Path:** If the function is unsuccessful in starting, completion or is cancelled the application will return to the previous active Activity. The activity will continue with the previously used bitmap information or none if it didn't exist.
**Expected Outcome:** Bad input for this case would involve no bitmap being stored or a bitmap that is not condensed to a thumbnail sized image. The image is required to be compressed to this size to save space in the database.

**Unit Name:** onCreate() for both Pop.java and Quick_Report.java. Both classes are responsible for displaying the popup windows for the reports, organizing the information and sending this information to the database.
**Description:** Testing for this method involves pulling the entered information from a user's report and sending this information to the database. If any required information is not present, the database will be missing vital information, and this will affect the performance of the website.
**Successful Path:** Information that is entered contains all the required information in the correct format. The required information for this test is any string containing report content that is not blank, a timestamp that is in the requested format, a Boolean value that marks the report as incomplete, and the logged in users' email. All this information must be stored in a hashmap.
**Expected Outcome:** The function sends the report content to the database and the report activity is closed.
**Unsuccessful Path:** Invalid information that will fail the test will be a timestamp or string timestamp that does not follow Firebase standards, or blank report content or a blank hashmap being pushed into the test.

## 2.2 Web Portal

For the administrative website, we cannot test HTML or CSS as error handling is native to individual browsers. The web pages will either work as we wrote them, or the browser will not display anything. We are using JavaScript within our website which is possible to write unit tests for, but impractical. Nearly all our JavaScript functions do not take any parameters from the user. We can test each function but for most there would be little reason to do so, since they receive input that we define and only return outputs that we specifically request using well established API's. Specifically, we define the input and there is an expected output when using an API- there is no field on our website where the user can enter anything non-predefined. For example, if there are any issues with the way we implement the Google Maps JavaScript then the map will simply fail to load. Our code is written in a way where testing the API functions with different inputs is meaningless, and it is not our responsibility to test the logic of a function in an existing API.

We can choose not to test functions that store or retrieve data from Firestore, our database, since the Firebase API requires specific instructions to work properly. Writing to the database has built in error handling, while reading from the database just requires the correct name of the document we wish to read from. For our website functionalities to work at all, the Firebase API must be implemented correctly.

There are two functions that the user of the website does interact with. One of those is to filter dates which the user can select a date range from a predefined list of month, day, and year to another predefined list of month, day, and year. The other function is allowing our administrative users the ability to set the length of time a zone will be a particular color. These two functions can be tested manually by us. Below is the testing process for these two functions.

### 2.2.1 Unit Tests

**Unit Name**: selectTasksFromTimeRange() is a function that allows the user to display tasks based on a defined date range. The inputs of the start time and end time are selectable from a predefined list of months, days, and years. Upon selecting the desired month, day, and year for both the start date and end date, the user then clicks the Select Button to submit the input. The task list will then populate with the tasks that fall within the time range.
**Description:** To test this function, we can change the start date and end date to varying ranges. We select months from January to December, day 1 to 31, and years 2019 to 2022. The user cannot input, only select.
**Successful Path:** Date ranges are selected properly, meaning the start date is earlier than the end date and the day of month exists within that given month.

**Expected Outcome:** Tasks are displayed from within the time range selected. No tasks are displayed from outside that range, and all tasks within that range are displayed.
**Unsuccessful Path:** A date that does not exist is inputted (February 31) or the lower and higher ends of the date range are switched in input order (May 1 to January 1).
**Expected Outcome:** No data is displayed, since those dates or ranges are invalid.

**Unit Name:** setTimeInterval() is a function that allows the user to set the length of time every zone will be of a color. It will take a number within an input box and store that number for later use. Upon clicking the submit button, the time duration for each zone color will be set.
**Description:** To test this function, we will use different inputs into the input boxes such as integers, floats, and text.
**Successful Path:** The user enters an integer and clicks submit.
**Expected Outcome:** The zones are updated with the integer that was given and zone colors are changed based on new input.
**Unsuccessful Path:** The user enters a float or a letter.
**Expected Outcome:** The function will not be able to store the input that was entered. No changes will be made to zone color duration.

# 3. Integration Testing

Integration testing for software checks that all the modules and components are interacting with each other correctly, and ensures integrity of data flow. For our software this mainly means testing that our database is storing data with the correct fields, that our web application is reading and writing data appropriately, and that our application is reading and writing data appropriately. Most of the mobile application aspects will be covered by unit testing, so it is important that the web application is tested with a heavier hand here. The web application is where most of data that gets written to our database comes from and determines how our application looks and will be used.

## 3.1 Integration Tests

### 3.1.1 Mobile Application

Our integration testing for the mobile application will be simple since most interactions are using API calls. We need to test that each page interacts with the others correctly, and that the application reads and writes to the database correctly.

Testing the application navigation is handled nearly automatically by Android Studio. We can test the application by manually choosing different places to navigate to. Specifically, when the user selects an option from the menu, the application creates a new intent and will have a different context- the information related to the application's current environment. So, if we get to where we selected within the application, it passes. This test will fail if a blank context object is passed, or the context object of another activity is passed. Generally, these features will not fail in normal usage of the application because the

context object being used is created with the new intent, so it is impossible for different activities to have inappropriate results. Simply trying to navigate between pages on the application is sufficient to test that the navigation is working correctly.

As far as the interaction between the application and the database specifically, the same problems arise as with the website. The only potential for errors here is caused by a programmer using incorrect data fields, which we will address in the next section.

### 3.1.2 Website

Integration testing for our administrative website will be basic. We can test links between other HTML pages by simply clicking each link within the webpage. If we go to the desired webpage, it has passed. We also need to test that our website is correctly reading and displaying data from our database.

As far as reading data from the database and displaying or interacting with it on the website, the main potential for errors is data in an incorrect format. Our website is not prepared to read or display data in nonspecific formats so if there were an implementation error on the mobile application side, and thus improperly formatted information, our website will error. This is more of a user (developer) error than integration one and is circumvented by ensuring our team has strict and predefined standards on the format of data in our database and the way it is created and interacted with on each of the website and mobile application. So, we have discussed and documented all the naming conventions for variables and collections in our database to ensure data is stored and interacted with properly.

# 4. Usability Testing

The purpose of usability testing is to check how the end users will interact with our product, both on the website and the application. Usability testing is highly important for our software, since difficult-to-use interfaces will rend our products nearly obsolete. A key requirement for the software we are developing is that it is very user-friendly, so we must be thorough in testing usability.

Our usability testing will be two-pronged, as the rest of our testing. We must test the mobile application and web application individually, since they have different intended audiences and criteria for user-friendliness.

## 4.1 Mobile Application

Our mobile application is intended to be used by essentially all of NAU's community and our testing needs to reflect that. Our focus will be on the student population, though we intend to make sure that employees and faculty are able to use it as well.

Since we expect the main users of the application to be NAU students, we want the average student to be able to use our application intuitively. They should be able to figure out where they need to navigate to based on the user interface and workflow, and there should be no instructions handed to the end users when using the product for the first time. We do not expect a student's background (gender, major, ethnicity, etc.) to have any bearing on how usable our application is for them, though we do distinguish between students and faculty/employees given the difference between the typical age of a student and the typical age of a faculty member. Therefore, testing our application with students can be as random as we like. We expect our application to be relatively intuitive for non-students as well, though we are implementing a "help" page on the application for those that would like or require some guidance.

When testing usability, we have to be mindful of our performance requirements. The number of pages on the application has been limited to increase simplicity. We require most users to be able to make a quick report in under one minute, a full report in under two minutes, and a full report with a photo attached in less than three minutes. So, these are the criteria we must be testing for. When handing our application to a user for testing we will need to time how long it takes them to make reports after becoming familiar with the premise of the application and its general interface. We will also want to note how most people navigate it for the first time, where they tend to click first, and what assumptions they make based on what they see. This data will help us improve the interface to be more intuitive and easier to use.

## 4.2 Web Application

The usability testing for our web application does not need to be as expansive as for the application. The audience for our web application is a specifically small group of people, NAU's Moving & Recycling department. We need to make sure that our client, Brock Brothers, can use our website relatively easily, as well as some of his team members. We are comfortable with our web application having a slight learning curve or new users requiring some guidance, since the web application is comparatively geared towards a very small audience.

We have already met with Mr. Brothers and given him detailed instructions on how to use our website, so our focus is making sure that he is able to show others how to use it or that they will be able to figure it out on their own. So, we have planned a meeting with Mr. Brothers and a few of his team members to see how they fare with it. We will be taking notes on how they intuitively navigate the website and adjust our interface from there. We have prepared a sort of manual for using the website and will be keeping it updated as our interface changes and the functionalities are tweaked.

With our demographic and end-user considerations in mind, we have decided what our interfaces need to look like and how we need to test our software. Bad design can lead to confusion for the end users and makes it necessary to test our product with the general

public and those who are not necessarily tech savvy. The distinctions between user base of our application and website allow us to define different approaches for testing each.

## 5. Conclusion

The current way to report issues on campus is rarely used and unmaintainable. Our software will act as a tool for getting students and faculty more involved with keeping their campus clean and taking responsibility for the places that are like home to them and replacing NAU's current and obsolete solution. Our team and clients envision this application being a template that other campuses can use to motivate their communities to get more involved. We would also like to create more awareness amongst students and generate a sense of responsibility towards sustainability efforts.

To ensure our software can be used as intended and fulfill its purpose, we have developed this plan for testing our software. This document overviews our plans utilizing unit testing. We will test our software and modularity, integration for the interaction between modules and their components, and usability testing. For each of the three types of testing we will be doing, unit testing, integration testing, and usability testing, we distinguish a plan for each the mobile application and web portal. For unit testing our application, our plan is making two unit tests for taking a picture and the report is sent with valid content. For unit testing our website, our plan is making one unit test for displaying tasks based on the date specified. For integration testing both the application and the website, our plan is to make menu items work properly and any information passed from the application to the database that is not proper will break. For usability testing our application, our plan is to observe a variety of different people to test out the application and take notes on any difficulties they are running into so we can further improve our product. For usability testing our website, our plan is to observe Brock Brothers and his team members test out the website and take notes on any difficulties they are running into so we can further improve our product. This testing will ensure errors in our product are minimized and easy to use for our end users.