MoGreen

Final Report

Version 1

May 10, 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 that are outside the scope of what any community member can handle.

This becomes more problematic as the size of the NAU community grows, as has been identified by Ellen Vaughan and Brock Brothers who lead the efforts for sustainability on the NAU campus.

Ms. Vaughan was the manager of the Office of Sustainability here at NAU. Mr. Brothers is a supervisor and coordinator for NAU's Moving and Recycling. Everyday there are issues Mr. Brothers and his team deal with, but generally he and his team will not know if there is an issue unless they come across it in person and visually see the blockage, or someone reports it using the existing mechanisms, which they rarely do.

Currently, students can report minor issues by calling NAU Facilities or filling out a form using their website. Nearly everything done to keep the campus clean and maintained is done manually. There is not tools or systems in place to accurately track when areas around NAU have been cleaned, preventing the NAU community from getting involved if they desire to. They are limited to calling or reporting via a work request made online, a feature which most are unaware of. Either way, the current reporting method is outdated, unintuitive, and most students will not make the time to report minor issues.

The proposed Capstone project we have implemented is the "Clean My Campus" mobile application. This application has been developed by our four-person team and will make reporting minor issues easier for students and the general community. 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.

Based on our assessment of our sponsor's needs, we've also implemented an administrative web portal, where campus administrators can manage reports made on the application and edit how the campus is viewed and interacted with on the mobile

application. The complete software we've created is the "Clean My Campus" mobile application and web portal.

2. Process Overview

2.1 Team Roles and Standards

At the beginning of the year, our team assigned roles to each team member preliminarily, which have evolved and reassigned as our individual strengths have become clear. Initially, the roles assigned were:

- **Team Leader:** the team leader will run meetings, follow-up on team deadlines and productivity, schedule meetings with the client, and ensure each team member is represented and able to provide opinions.
- **Customer Communicator:** the customer communicator will be the primary point of contact for the clients and be available to the client as appropriate.
- **Proofreader:** the proofreader will review all team documents and deliverables for cohesion, formatting, and professionalism before submission. They will also be responsible for ensuring documents are printed and submitted.
- **Release Manager:** the release manager will coordinate version control and maintain the GitHub repository. This includes ensuring readability and cohesion of documentation.
- **Webmaster:** the webmaster will manage and update our team website. This includes uploading any required pages and ensuring their readability. They will also assign tasks regarding creating and updating the pages.
- **Calendar Manager:** the calendar manager will ensure the team calendar is updated appropriately as needed, specifically documenting team and client meetings and formal and informal due dates.
- **Deliverable Manager:** the deliverable manager will make outlines of group deliverables and ensure the team can access them. They will also organize Google Drives and keep copies of all team documents. As our team needs to make presentations, the deliverable manager will coordinate the creation of presentation materials and the presenters.
- **Recorder:** the recorder will take thorough notes at each team, mentor, and client meetings. They will record main ideas and decisions made and make legible documentation for team reference.

As we began development, we also fell into specific development roles.
Web Portal:

- Version controller & developer
- Developer

Mobile application:

- Version controller & developer
- Developer

Justin and Cassie worked on the administrative web portal component of the project, and Jennie and Chase worked on the mobile application component. The two pairs had to work together for combining modules and working on common components, like managing the database, which both the mobile application and web portal utilized. The dynamic and roles within the pairs fluctuated throughout the semester as well based on an individual's strengths, other team responsibilities, or individual responsibilities at that time.

2.2 Development Processes

Of each development pair for the mobile application and web portal, one person was in charge of version control. For each component, as someone would update and create files, they would upload them to their own dedicated branch in the repository, after which the version controller for that pair would be responsible for accurately merging them. Merging files and versions was often complicated, leading to the need for a designated version controller in each pair. At times, it was possible we could each be working on different components of the same file or perhaps formatting a file while another was working on a functionality of it, complicating merges.

Software components were not developed in a specific order, except for what was intuitive for us. Each week we would assign tasks for the next, based on what each pair thought was appropriate for their component. Some modules needed to be developed linearly, i.e., we had to have a map being displayed before we could draw over it. So, we typically started from the bottom and worked up, more so for the mobile application. For the web portal, it was a little easier to work concurrently since each webpage has its own functionality. Occasionally, one of us would get stuck on implementing a specific feature and extend it to the next week or switch tasks with their partner. Our weekly meetings were integral to assessing and deciding together what was most important and needed to get done next, and prioritizing tasks as development problems came up.

3. Requirements

The requirements were how we and our client decided what our plan for implementation was and the specific functionalities we promised to implement, as well as our highest priority 'extra' features. We have outlined them below into three components which are functional, performance, and environmental requirements.

3.1 Functional Requirements

Functional requirements outline each specific functionality we implemented, detailing ways that different types of users would be able to interact with the application or web portal.

3.1.1 Mobile Application

View Zones

One of the main features of our application was to allow the users to view zones on a map. These zones would act as markers to show the users specific locations that may need cleaning or attention. A timer would function for each zone to degrade the color of the zone over time. If a zone was marked fully clean and received no reports within a specific time range, the zone's color would gradually change from a lighter green to a darker red, to indicate it probably needs attention. Users would know which zones need attention by viewing the color shade the zone indicates from the main interface, rather than needing to tap the zone for more information.

User: Reporters and administrators.
Description: View a section of a map that contains colored smaller areas that designate areas as clean or in need of attention.
User Error: Possibility for user to select a wrong area and send a misleading report.
State of the System Before: Map will display area normally, zones will display a color based on the status of their zone.
Flow: User pans in and out of the map area. When they discover their location on the map, they can select a zone.
State of the System After: After tapping, a popup displays the types of reports a user can submit.

Quick Reports

Once a reporter selected a zone, the application would bring up a small window allowing the reporter to create a quick report or full report. If the user selected Quick Report, they would be presented with default options such as: litter, full trash cans, or the ability to indicate that the zone did not need attention at that time. Depending on the report given, it would either add time to the zone maintaining or improve the zone condition (indicated by color) or decrease the timer on the zone. The quick reports were the main action that determined a zone's status, making this functionality almost purely based on crowdsourcing, thus requiring less effort from an administrator or campus worker.

User: Reporters.
Description: The ability to send a predefined report after selecting a zone. The report will only be used to report smaller issues such as litter in an area, an overflowing trash can, or an area is clean.
User Error: User may send in a quick report on accident.
State of the System Before: Zone is displayed in its current status.
Flow: A user sends in a report. This report is sent to our database and is recorded for that zone.
State of the System After: Zone color shifts slightly to be redder or greener based on if it was reported as dirty or having been cleaned.

Full Reports

After selecting a zone, the user also had the option to make a Full Report. This allowed users to send in a detailed report about vandalism, a maintenance issue or misplacement of trash. We did not let the full report modify the status (color) of the zone as these issues are generally out of any reporter's control and would require staff intervention. Zone colors were based solely on crowd-sourced reports of needing to be cleaned or having been cleaned.

User: Reporters.
Description: A custom report that a reporter can submit from a zone. This report will allow for more information on an issue to be received. This report will not modify the status of the zone.
User Error: Reporters may accidently report from the wrong zone or create a misleading report.
State of the System Before: Zone is displayed.
Flow: Reporter taps a zone and selects the full report. They fill out relevant details in a text box and attach a photo if desired.

State of the System After: Zone is displayed normally, report is added to the database and displayed on the website to administrators.

3.1.2 Web Portal

We created a web portal for administrators that allowed them to review reports, change the location for the map, and add or modify zones. In order for the map to be used on the application and through the web portal, the administrator must enter an API key. This key is responsible for linking any features used to their accounts and applying any charges if they appear. Included on this page were simple instructions for acquiring information such as finding specific coordinates for zones, or easily entering the information needed to show their campus on the map. They are also be able to create new administrator accounts, and view traffic data.

View Reports

The web portal can display any report made, which administrators would be able to view. Reports could be queried from a date range to simplify the amount of information displayed. They would also be able to view any relevant information about the reports such as the contact information of the sender, the zone the report was sent from, and the content of the report created by the reporter.

User: Administrator.
Description: Full reports can be viewed and sorted based on date or location of the report.
User Error: Administrators choose an invalid date or sorting type.
State of System Before: User will be presented with a short form allowing them to enter a date range and a location.
Flow: Once the form is completed all reports that match the query will be returned.
State of System After: A list of reports and all related information will be included on the page.

Heatmap

A heatmap was the minimum way that traffic data is displayed to administrators, specifically displaying the frequency of reports. Information regarding how ‘popular’ a zone is, meaning how much traffic that zone has seen during a specified time period, is displayed.

User: Administrator.
Description: Data is displayed in a 'heatmap' format, showing traffic patterns on a campus over a specified time period, regarding reports made.
User Error: Administrators choose an invalid date or sorting type.
State of the System Before: A map displaying the last month (30 days) of traffic data.
Flow: The administrator will enter a date range if desired. After choosing the time period, the map will update to reflect traffic in that time period.
State of the System After: A map displays traffic data.

User Registration

Any individual who wished to use this application was required to sign up for an account, requiring some basic information. Information such as the user's email address was needed to ensure that reporters could be contacted if needed, though this was unlikely.

User: Reporters.
Description: A user can register an account and log into the application.
User Error: Users enter a password with incorrect case and have trouble logging in.
State of the System Before: A user is at the login page for the application without the ability to sign in.
Flow: The registration button is selected when the application has finished loading. The reporter can then fill out the form and create an account.
State of the System After: Reporter is able to access the application. These summarize the main functionalities we have implemented the minimal viable product for this project.

3.2 Performance Requirements

Performance requirements specified expectations for how our project was expected to perform. These requirements are measurable and provable and worked as guidelines for us to create and test our application.

Simplicity

The application has a simple interface so the users of the application could complete tasks in a short amount of time. We designated three main page interfaces that the average user would be using on the application, being: the main map interface, the quick report menu, and the extended application menu. Submitting reports on the application was simpler than NAU's current reporting mechanism, with more functionality.

Usability

Users of the application should be able to access any feature quickly and easily. An average user can be defined as any student or faculty member on campus. Below are specific measurements we used to test the usability of the application with many average users.

Quick Report Efficiency

Quick reports were able to be created in under a minute. These short reports could be created by selecting a zone and choosing quick report. After this option was selected, the reporter would be able to select predefined report options and confirm their submission.

Full Report Efficiency

Full reports were able to be created in less than two minutes, or three minutes if a picture is included. The user would need to manually enter the details of the report.

Reliability

Our application needed to handle multiple requests and store a variety of information, whether it be text or image. Our application was expected to handle a minimum 30,000 requests at once, defined by the current NAU student, faculty, and employee populations, plus some room for visitors. These requirements ensured our application was both desirable and efficient to use and was an appropriate replacement for current reporting mechanisms.

3.3 Environmental Requirements

The environmental requirements described some more technical constraints we had regarding our project implementation and development environments.

Cross Compatibility

Initially we planned to release our application as Android only but were planning to use an environment that would allow for the application to be cross-compatible, as we would like any user to be able to use our application regardless of the software on their phone. However, the overhead required to create a fully cross-platform application or separate versions of the application for each Android and iOS couldn't be overlooked. Therefore, we committed solely to an Android release with cross-compatibility moved out of our minimal viable product.

Gamification

Future expansions of our application have heavy focus on gamification. To be mindful of our future additions to our minimal viable product, we allocated database space for creating user teams and assigning point values to actions, so that different functionalities may be implemented. More information about extended expansions of our application is below, in the "future work" section of this document.

Google Maps API

The Google Maps API outclasses all of its competitors with its built-in features and compatibility with multiple development environments. It should be noted that if we encountered some sort of problem with the Google Maps API, there were hardly any comparable replacements for it. This posed a risk, though it was insignificant.

Hosting Service

For our hosting service we needed an environment that supports JavaScript and MySQL. This service was the basis for our application's administrative features, reports view, and statistics.

Geolocation

Allowing users to submit reports without the need to select a zone assisted with our two-minute average goal for submitting full reports. As an addition to the side menu on our application, users were able to select 'full report' and submit a report based on their actual location rather than a selected zone. Using this information, we could somewhat confirm the accuracy of reports as we could compare geolocation with the reported zone. The user could also use this information if they were not sure of their current location and needed assistance selecting a zone.

Extensive Map Icons

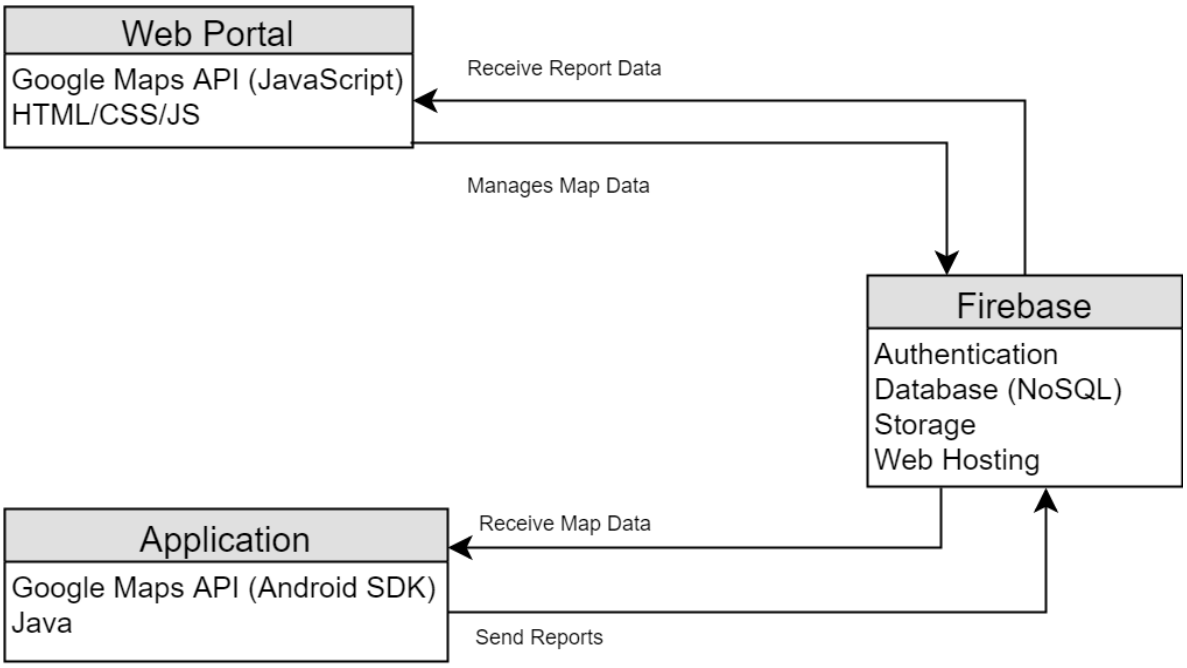
We wanted to see people use this application as a tool to easily locate different features around their campus, making it that much more useful. If a student was looking to find a station for recycling, we wanted to make this search easier by including a filter on the map that marked these locations. These locations could be used for multiple different resources, such as: recycling booths for glass or other materials, trash bins, or assistance for students to locate green or related organizations on campus. These were the main functionalities we implemented after our minimal viable product was fully implemented.

4. Architecture and Implementation

Our team has developed the “Clean My Campus” application in pairs. We have both an administrative website and an Android application we have developed in parallel to get the MVP completed. The administrative website is where the application administrators can manage tasks sent for them to complete, view the NAU campus based on client defined zones, edit those zones, manage admin accounts, and view statistics. The Android application is where most users will be interacting with. They will be able to view the zones on a map of the NAU campus and make various reports of issues they come across.

As it turns out, developing in parallel of both the administrative website and Android application became natural as either part required nearly the same steps in development. We started by implementing authentication and login features using Google Auth. This feature was convenient, secure, and compatible with the website, application, and database. The verification and authorization have been implemented, making modifications and data-adding to our database secured and only possible by authorized users. Authorized users were those who have signed up with our service. Once again, both application and website were displaying the NAU campus map with overlay of zones. This map display feature was implemented nearly the same in both parts but adjusted for their own environments. The application compensated for smaller screen size and user experience while the administrative display was directable to admin-specific functions. After implementing the map views and functionality, the development tasks for the mobile application and web portal started to diverge. The application would be creating tasks and uploading to the database with its various information while the website would display those tasks for the admins to complete.

Dataflow



The main function of the application and web portal is to be able to transport required information to each other to effectively display that data to the user. The application creates reports which regardless of type send the coordinates of the area, information related to the report, a report type, and the user's name. With this information it also includes select variables that will allow the website to translate this information and sort as needed. The website will take these reports and generate statistics and display this information. The website is also responsible for creating zone data and marker data. This data includes the marker type, the zone coordinates, and the zone color. Like the reports that it receives, it also includes variables that help with the storage and displaying of the zones. The correct data from each source is essential for the functionality of our software. Both sections will allow the user to register and will allow the user to log into the Mobile Application.

4.1 Mobile Application

The Mobile application is the face of our software that the majority of the user base will see. The front page of the application will allow the user to register and log in. Any user including the administrators will be able to log into the application. After logging into the application, the user will be greeted by the Main Map View.

Main Map View

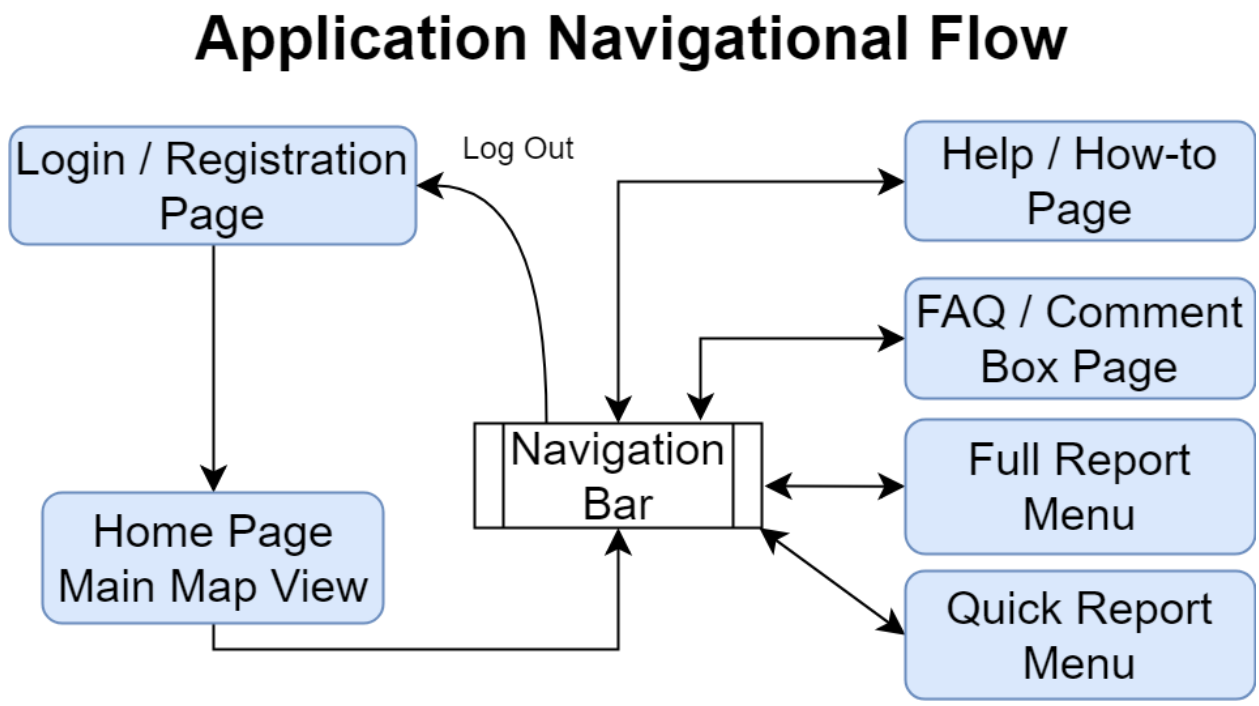
The main map view is responsible for informing the user of the current zone statuses as well as helping the user determine their location. The map on this page will display the zones as well as their statuses. In the Main Menu on this page, the user will be able to determine what type of report they would like to make, select which zones they would like to see and view a Help page that will direct them to the About page on the web portal for more assistance.

Full Report

The full report will allow the user to submit reports anywhere on their campus. The contents of this report include a report type, report content and optionally a picture to satisfy the report. The full report will also record the user sending the report and the location the user selected.

Quick Report/Clean Report

The quick report is a simple version of the Full Report that contains less information. The quick report will only allow the user to select a report type but will still require the user to select a location on the map. If the user wishes to send a clean report, a report that claims a cleaning action to a zone, they can toggle the Mark as Cleaned button in the quick report pop-up. This report will only allow the user to claim litter as the report type.



4.2 Web Portal

The web portal is an administrative access point which allows its users to view reports and various statistics. This page will only be accessible by users with administrative access, any other users will be redirected to the about page and will not successfully log into the web page. After login, the user will be redirected to The Home Page where they can view an overview of the created zones and markers that were created and a navigation bar where other features can be accessed.

Heatmap and Statistics

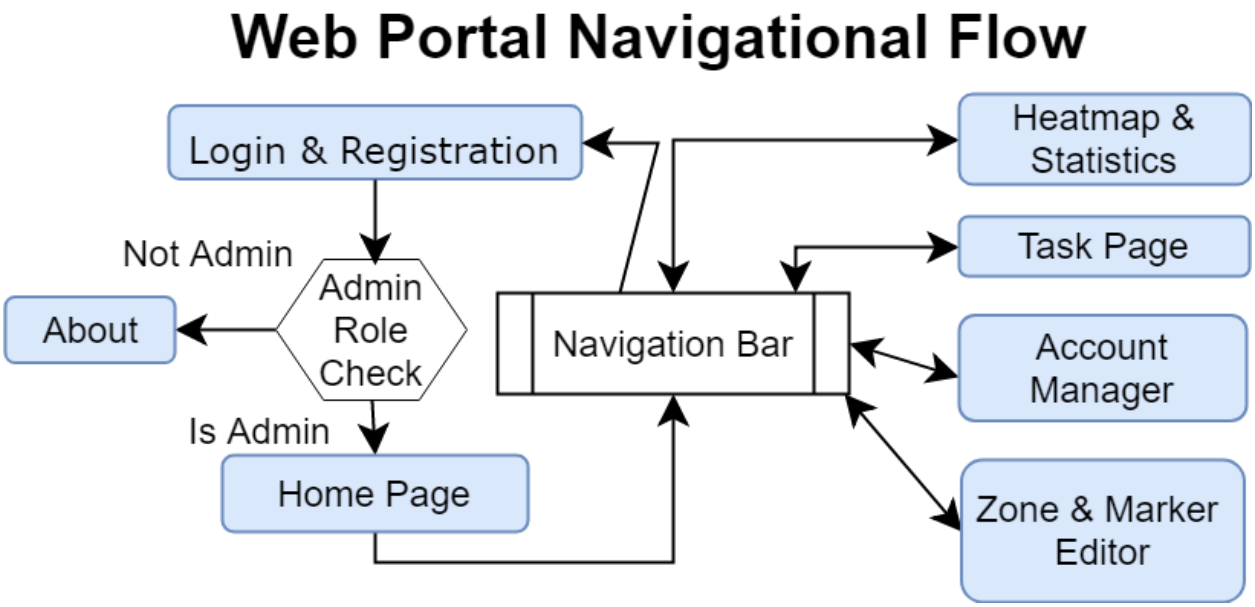
The first page to display data from reports is the Heatmap and Statistics page. This displays the basic information included in the home page including markers and zones while also providing a heatmap of reports that were created. The purpose of the heatmap is to display the frequency of reports created in a specific area. The user will also be able to toggle report markers which will display what types of reports are included in an area.

Task Page

The next page displaying data is the Task Page which includes a general breakdown of reports received. This page is divided into two sections, one that displays active reports and completed reports. The active reports contain all information submitted about the report including the area, the submitters email, the report type, a description of the report, and optionally an image of the situation. If the report is marked as completed, it will only be displayed in the Completed Reports. Both types of reports can be marked as completed or can be reverted to the active state. All reports on this page can be sorted based on the report type or the date the report was submitted.

Zone and Marker Editor

This page allows the user to create zones and place markers that will display on the application. A map will be displayed with tools that help create zones and markers simply. The marker editor allows the user to select a marker type and an area on the map. The zone creator is a tool that allows the user to draw out a zone. While the zone is being created other attributes can be added such as the time for expiration. Along with these creation features, we also have tools that can be used to remove markers or zones.



5. Testing

Our team completed 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 is how we ensured that all of our key requirements are being met. Integration testing allowed to make sure that all of the modules and components of our software are interacting correctly, and that data is being stored and retrieved correctly. Usability testing determined what features we need to focus on and what will make the user's experience better. 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.

5.1 Unit Testing

Our unit testing ensured that the data created is compatible with each section of the software. The main points for unit testing are ensuring that a map can be viewed with zones that display its status, reports can be created and displayed including all information in the report, and the ability to create markers and display them on both the mobile application and the web portal. For these tests, we wanted to make sure that the data being received is usable and is able to be displayed. However, since both the mobile application and web portal rely heavily on existing APIs, we had very few units to actually test.

The unit tests performed for the mobile application are detailed as follows.

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.

The unit tests performed for the mobile web portal are detailed as follows.

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.

5.2 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 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.

Testing the application navigation is handled nearly automatically by Android Studio. We tested 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.

Integration testing for our administrative website was 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 needed 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.

5.3 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 developed was that it is very user-friendly, so we had to be thorough in testing usability. We had to test the mobile application and web application individually, since they have different intended audiences and criteria for user-friendliness.

When testing usability, we had 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 tested for. When handing our application to a user for testing we needed to time how long it takes them to make reports after becoming familiar with the premise of the application and its general interface. We also noted 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.

The usability testing for our web application did 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 needed 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. When testing usability of our web portal, we met with Brock and his team multiple times to make sure he could use it, show others how to use it, and provided him with detailed instructions along with our user manual. We are confident our web portal can be used with instructions, if not intuitively.

6. Project Timeline

~**September 15, 2018:** We began by determining our team name and creating the team website. This website would become our team repository containing capstone reports and other required documents for the entirety of this project.

~**October 1, 2018:** We started research on the client's project description looking into its feasibility. At this point we were determining the client's needs and the solutions we could offer them to implement their ideas.

~**September 1, 2018:** Once our feasibility research was completed, our team determined functional, non-functional, and performance requirements for the project. We created a requirements document that the clients had to acknowledge and sign. These are the requirements that our team needed to implement for our final product to be considered as completed.

~**November 1, 2018:** Work began on our prototype to be demonstrated at the end of the Fall 2018 semester.

~**November 20, 2018:** Our team presented the first Design Review for the project.

~**November 30, 2018:** We found that the technologies we had selected for creating the project were not the best tools for the job. We discovered during the creation of the prototype that the tools were insufficient for the project requirements.

~**December 15, 2018:** End of Fall 2018 semester. We completed our prototype and learned the limitations of the technologies we used at this time.

~**January 15, 2019:** Start of Spring 2019 Semester. With the knowledge gained from the prototype, our team changed technologies to Android Studio and Firebase. Development on both the Android application and Administrative website begins.

~**February 25, 2019:** Our second Design Review was given at this time.

~**March 15, 2019:** The project's requirements have been met at this point which means our team has completed the minimum viable product (MVP) for the project. We have also given a demonstration of our product in its current state to both our team mentor and clients.

~**March 25, 2019:** Started on the final build of the software for the project. Application and website testing begin.

~**April 11, 2019:** Our team presented the third Design Review at this time.

~**April 26, 2019:** Our team gave our Capstone presentation at UGRADS. We also participated in Capstone Poster presentation at the Union building on the NAU campus. Team MoGreen's website has been completed at this point. Application and website UI have been updated for final demonstration to mentor and clients.

~**May 2, 2019:** Our team demonstrated the final build of the project to our team mentor and clients. Clients acknowledged and signed off on the final build checklist. The clients received source code and documentation of the website and application.

~**May 9, 2019:** The final report has been completed and turned in to our team mentor. This concludes our teams Capstone experience.

7. Future Work

7.1 Current Build

This section will cover future work on features that have been implemented in the final build of our project.

7.1.1 Application

General

User Interface Updates
The applications' overall UI should be updated as needed. This is only the first iteration of the applications UI and improvements can always be made. Icons were custom created by our team, but they should probably be considered placeholders for higher quality and more relevant icons.

Reports

Report Type Quality of Life
Users should not be able to choose 'Report Type' after dropdown list was been selected.

Image Size
Currently, images are sent as small thumbnails to Firestore. Changes to image resolution should be made to ensure tasks have detailed images accompanying them.

Quick Report Type Quality of Life
Report types for quick reports should be reassessed for the usefulness of a quick report of that type. Example being that a 'General Issues' report with no description or image has very little use for tracking and assessing the urgency of the matter. Quick reports of 'Trash Can', 'Shopping Cart', and 'Litter' do not require photos or a description since they are nearly self-descriptive. Full reports of the excessive litter or overflowing trash cans can still be made. From a client's business needs perspective, some quick reports are inherently useless.

Main Map View

Center on Location
This should be updated so that when the center on location button is pressed, the marker placed should be located on the application users' current location. Currently, when pressed, the marker will be placed at the center of the screen wherever the map is at.

Icons Quality of Life
All resource icons should be togglable on the application. As more resource icons are added, the application should reflect the ability to toggle them on or off. Additional icons such as quick report and full report icons should be updated. This can be address under UI updates.

Help/F.A.Q.

Define Report Types
Users should be given a description or examples of what falls under each category. This should help new users correctly choose what type of a report should be made under which circumstances. This will help reduce mislabeled task types for tracking.

About Page Resolution
Any link to external web pages should adjust resolution according to mobile device screen sizes.

7.1.2 Website

General

User Interface Updates
The websites overall UI should be overhauled. As of this writing, it was created to be more functional than aesthetically pleasing. Icons were custom created by our team, but they should probably be considered placeholders for higher quality icons.

Administrative Access
Anyone who signs up can currently access the administrative website. The original intention was to have administrators assign other accounts as administrators using the Account Manager webpage, however, during development, there were complications with using Firebase Authentication. The website will need be secured for administrative use only. There was an alternative our team found but we ran out of time to fully test and implement it. Firebase Authentication does allow specific access rules to any Firestore

database collection. These rules are fully customizable, and we were able to limit user access to a collection based on Firebase Authentication UID.

Miscellaneous

Information Update
The Privacy Statement and Terms of Service web pages need to be updated to reflect their proper content. Currently they have placeholder text so they users are not directed to 404 – webpage not found.

Filter Quality of Life
The filters on the Heatmap and Task List web page feel a little clunky to use. Quality of life fixes should be made such as more response when applying filters, reset button for time range, and overall make it more intuitive to use. The date ranges also have the ability to choose non-existent dates such as April 31st. This does not affect functionality since the Date Object built into JavaScript is robust enough to properly identify this and not fail when applied.

Display Resolution Issues
Different display resolutions can adversely affect how the webpages are displayed and we found that the menu bar can change based on browser. Revisions and additions need to be made to account for different display resolutions and browsers.

About Web Page

Information Update
This should be updated to reflect more information about the “Clean My Campus” application and website.

Heatmap Web Page

Color Correction
During the creation of the website, our Google Maps displayed darker than the default look of Google Maps. This may affect the colors we have chosen in a negative way. Updating the color schemes based of the lighter default Google Map should be accessed in the future.

Task List Web Page

Limit Displayed Tasks
Currently, the amount of task in the database are the same number of tasks displayed for the task list. This has obvious performance issues with larger sets of tasks. This should be address by limiting the number of available tasks displayed at a time. Set to a range such as one to ten, one to 25, and so on.

Task Readability
With the many task currently showing, it can be hard to follow a particular task horizontally. This can be fixed by adding alternating colors when the list is created.

Task Duplicates
This can be refined to catch and group the reports even more accurately. The duplicate 'mark' should persist even if the task has been completed. This may require an additional field in the database collection to track and maintain the 'mark'.

Task Completed By
When tasks are completed, they should be able to show who marked the task as completed and when. This will require two additional fields for the database collection to track.

Map Editor Web Page

Creation Tools Quality of Life
All the tools are functional but there seems to be several issues after creation of zones, markers, editing current zones, deleting zones or markers, and resetting zone colors. The biggest issue is having to refresh the webpage in order to interact with newly placed markers and zones. When a zone is created, the user cannot immediately delete that zone until after they refresh the page. This is the same issue with markers. Resetting zone color and deleting zone or markers will not show them disappear or change zone color on click. Once again, the user will need to refresh the webpage to show the changes made. Updates will need to be made to correct these issues.

Tasks Affect Zone Timers
The framework for implementing this feature is ready but we ran out of time to fully implement. This feature should reduce a zone timer based on the tasks within them by a set amount such as if the task was "Litter", then we decrement six hours off the timer for that zone. Tasks such as "Shopping Cart" should not have any impact on the timer but this can change based on client preference.

Cleaned Report Affect Zone Timers
The framework for implementing this feature is ready but we ran out of time to fully implement. This feature should increase a zone timer based on a cleaned report.

Account Management Web Page

General Overhaul
With the complications we came across using Firebase Authentication, this webpage has become obsolete until another method can be implanted and use it.

7.2 New Features

This section will detail features that were in our extended features list or entirely new features that would be nice to have in later version of the software.

General Features

Add Clean Teams
Application users should be able to join teams where a leader can volunteer to clean zones. Messaging may be a bit much for this application but a notification board for the leader should be implemented for team communication at the minimum. Users should get be able to know which zones have been volunteered for cleaning and when the team should be there to start. There should be checks to ensure that multiple teams can or cannot, depending on how this is implemented, volunteer to clean the same zone(s).

Add Calendar
Each application user should have a calendar accessible, so they know if their group has scheduled a zone clean up. They will also be able to see special events hosted by the campus, but these types of events will be discussed under the Gamification and Incentives section.

Gamification and Incentives

One of the biggest features we would like to see added in the future would be incentives for helping. This can be done by allowing for points to be added to a profile for reports made, zones cleaned, leaderboards, and additional ways our team has not even thought of. These points could then be exchanged for various stuff on a rewards page. There are many ways to implement such a feature and would be a huge undertaking just by itself.

Website administrators can be given the ability to mark certain zones as 'hot spots' worth bonus points. Zones of this nature can be created after large social events shown as a special event clean up event on the calendar. Administrators can also assign days throughout the month as bonus point days and other little mini events to incentivize volunteers.

Points can be given to users that create tasks for issues. There are many questions that need to be answered in giving points per task report generated. Are certain tasks worth more points than others? What happens with multiple reports made by not only the same profile but multiple different accounts? These types of questions must be answered thoroughly and will be quite difficult to address.

We believe different tasks should be worth different points. These points can be distributed based on the number of reports made on that specific task. This will require the task duplicate algorithm to be more advanced and manual verification may be required as well to catch outliers. There may be a need to implement a system where user specific issues such as their sink in housing is broken may need to be worth a different set of points as opposed to a public issue like a broken sprinkler head outside. The difference being that the internal issue is only reportable by one person while the public issue can be seen by anyone allowing for multiple reports.

Zone clean up, either by solo or groups, should award points as well. This will need to have a verification system setup to avoid clean report abuse. Points can be determined by size of the zone and the length of time it has been since it was last cleaned. A yellow zone will be worth less that a red zone, a large yellow zone will be worth more than a small yellow zone, and so on.

Leaderboards should be implemented for individual and group efforts. This would be nice to have so teams or individuals can compete and/or have their efforts on display. There can also be end of semester rewards for the highest individual and teams. Recognition by the campus for their help and bonus points are all options for encouraging volunteers.

Achievements would also be a nice feature to implement as well.

Add Verification
With the ability to gain points, there will need to be accountability to ensure no one is abusing the system and are doing the work. For zones, a viable solution would require a facility student worker to go to a zone that a volunteer or group leader has sent a cleaned report from and check their work. Once verified, then the facility worker can approve that cleaned report. Task verification should be done as well since tasks

Profile Updates
The profile system should be modified to accommodate for the additional features that are implemented.

Update Account Management
Administrators should be able to disable abusive accounts or accounts that ignore the software's Terms of Service.

7.2.1 Application

General

Application Administrative View
Administrators should be able to view the task list, view a map of where the tasks are located, and be able to mark task as completed. This view will be required for the new features added so verification can be done by administrators on their phone. This will help campus caretakers respond to generated reports quickly.

7.2.2 Website

Heatmap Web Page

Display Task Information on Icons
The heatmap display contains task icons. These icons should be clickable and when clicked, display task information for that task. The image should be available for view as well.

8. Conclusion

We have developed a mobile application that will allow community members to:

- View colorized campus zones and markers
- Make quick reports about problems they notice on campus
- Make full reports about problems that require more detail

We have developed an administrative web portal to allow campus employees to:

- Draw and design new zones and markers, viewed on the application
- View application reports as checkable tasks
- View report traffic data in as a heatmap

Our mobile application is a new, intuitive solution for NAU to replace the existing website. Our administrative web portal gives NAU all new functionalities and lets campus administrators take control of how the campus is viewed and interacted with on the application. They work in conjunction to provide NAU with the tools to help create a culture of sustainability and foster a clean and resourceful community.

Each component of our software was developed by a pair from our team with us holding meetings each week to decide what modules to develop next, allowing us to finish our key requirements by mid-March. Different roles were assigned to team members initially, and adjusted as appropriate throughout the Capstone year, helping us to work more efficiently as a team.

We had monthly meetings with our clients where we initially got familiar with their businesses and workflow to help orient our solution to solve their problems. From these meetings, we compiled and agreed on key functional, performance, and environmental requirements for our software.

Our software system has 3 main components: a mobile application, a web portal, and a database to moderate data flow between them. Each of the mobile application and web portal has its own individual modules, which also interact with each other. The main components for the mobile application are the map view, quick report menu, full report menu, and main menu for the application. The main components for the web portal are individual HTML pages, and are the zone and marker drawer page, the task list page, and the heatmap viewer page.

For testing our software, we used unit testing, integration testing, and usability testing to ensure our software was robust. Since our project is meant to replace an outdated and unusable website it was very important that our application be usable and intuitive, so we placed a higher emphasis on usability testing. Much of our project makes use of existing APIs, so there wasn't too much to do in the way of unit testing. Integration testing was important, but errors were typically the result of communication errors among our team and were resolved internally by us.

Our team spent the Fall semester meeting with our clients, deciding requirements, and deciding and testing different software's. We spent the first half of the Spring semester developing our product, and the second half doing testing, bug fixing, and tying up loose ends.

This project has numerous useful improvements and extensions. The key extensions we would like to see a future group implement for this project are mobile application oriented, and are cross-compatibility, user profiles and teams, and the ability to earn points and achievements.

The current way to report issues on campus is rarely used and unmaintainable. This application 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. 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.

9. Glossary

Android Studio: An application used to create Android applications.

API: Application Programming Interface, these are a set of functions for use by developers.

APK: Android Package, is a file format used by the Android operating system.

CLI: Command-Line Interface.

Environmental Requirements: These requirements describe technical constraints.

Functional Requirements: These requirements describe what the application and website do within the system. They should be clearly defined with inputs and outputs.

Gamification: Applying typical game playing techniques to a product or service. This should encourage users to use a product more.

Heatmap: A colorized gradient based on a dataset. The data points in the set are reflected in magnitude based on proximity to other data points.

IDE: Integrated Development Environment, usually consists of a source code editor, build automation tools, and a debugger.

Integration Testing: These tests are for testing how modules within a software interact with each other to ensure capability.

NAU: Northern Arizona University.

Non-Functional Requirements: These requirements describe how the application and website work. They should be clearly defined with inputs and outputs.

Performance Requirements: These requirements are measurable and provable which specify our expectations on how the software will perform.

SDK: Software Development Kit, is a group of software tools used for development.

UGRADS: The Undergraduate Research Symposium.

Unit Testing: These tests are for testing the components of a software given an input and expected output.

Usability Testing: These tests are for testing how well users can learn and use the software in question.

10. Appendix A

10.1 Hardware

Application

The application was developed and tested using Android Studio. Required hardware for the application development are:

- A PC or laptop with Windows installed. Windows 10 was used during development.
- Android Studio installed. The Android Emulator requires additional specifications:
    - 3 GB RAM minimum, 8 GB RAM recommend; plus 1 GB for the Android Emulator.
    - 2 GB available disk space minimum, 4 GB recommended.
        - 500 MB for IDE
        - 1.5 GB for Android SDK and emulator system image.
    - 1280 x 800 minimum screen resolution.
- (Optional) An Android Smartphone with an operating system version of 4.0 or higher.

Website

The website was developed mostly on Notepad++ using Google Chrome to test the website locally and see immediate changes. Required hardware for the website development are:

- A PC or laptop with Windows installed. Windows 10 was used during development.
- A text editor to edit HTML5, CSS, and JavaScript files. Notepad++ and Atom were used during development.
- An internet browser to help test and develop the website. Google Chrome was used during development.

10.2 Toolchain

Application

Google Maps API (Android SDK): Used for integrating and enabling Google maps on the application.
Firebase: This is a Google owned platform that has many features that we use such as Firestore, our database, Storage, for storing images, Hosting, to host our website, and Authentication.
Android Studio: This is an IDE that helps with developing and testing Android applications. This IDE has built in version control integrated with GitHub and can build Android application APK's.
Java: This is an object-oriented programming language; it is also the language Android Studio uses.

Website

HTML5: Hypertext Markup Language revision 5, is the standard languages used for development on the World Wide Web.
CSS: Cascading Style Sheets, is used to format HTML5 elements.
JavaScript: This is an object-oriented programming language used to make web pages more interactive.
Firebase: This is a Google owned platform that has many features that we use such as Firestore, our database, Storage, for storing images, Hosting, to host our website, and Authentication.
Firebase CLI: This is for managing the Firebase file on a local machine. This allows a developer to deploy the website to live servers.
Google Maps API (JavaScript SDK): Used for integrating and enabling Google maps on the website.
Notepad++: This is a text editor for editing source code files of HTML5, CSS, and JavaScript.
Atom: This is a text editor for editing source code files of HTML5, CSS, and JavaScript.

10.3 Setup

Application

1. Download and Install the latest Java SE Development Kit.
2. Download and install Git.
    a. Optionally install the Bash command line included MinGW.
3. Download and Install Android Studio including the SDK tools.
4. Open up Android Studio and click Project from Version Control.
    a. Select Git.
    b. Enter https://github.com/cbm97/MoGreenApp in the URL field.
    c. Choose a Directory to install the project in.
5. After this, you have successfully loaded the application! We will now need to get some keys to ensure that the application will operate correctly.
6. Navigate to the Firebase Console and choose the MoGreen project.
    a. Click on the Hamburger Button, then click on the Cog icon, then User Settings.
    b. In the “Your Apps” section, click on the button labeled “google-services.json”
    c. After downloading this file, Move the file to the ‘app’ folder.
7. Go to the Google API Console and select the MoGreen project.
    a. In the Hamburger Menu, choose APIs & Services then Credentials.
    b. Copy the latest Android Key (click on the square button next to or below the key to copy it to your keyboard).
    c. In the loaded project on Android Studio, locate the AndroidManifest.xml document.
        i. App > manifests > AndroidManifest
    d. Locate the meta-data named "com.google.android.geo.API_KEY"
    e. Paste the key in the section below named android:value=
        i. EX. android:value=”<API_KEY>”
8. In Android Studio, go to Build > Make Project.
9. When completed, you can load the application in two ways.
    a. Running the application on a plugged-in device.
        i. Plug in an android device to the computer running Android Studio.
        ii. Go to Run > Run ‘app’.
        iii. Select your phone from the device list.
    b. Building an APK then loading on device.
        i. In Android Studio, go to Build > Build Bundle(s) / APK(s) > Build APK(s).
        ii. A popup will appear near the bottom of the Android Studio window, Click Locate to pull up the APK.

iii. Add this file to your Android Device.
iv. From your Android Device, locate the APK from your File Explorer and run the APK to install the application.

Website

At the time of this writing, the administrative website was developed using Windows 10. Use of the Command Prompt is needed. To install and host the "Clean My Campus" administrative website, you must have Firebase and npm installed. You must also have a Firebase project already started and a Google Maps API key.

1. To start a Firebase project, go to https://console.firebase.google.com
    a. Click "Go to console".
        i. Sign into the account you can associate with developing.
    b. Click "Add project".
        i. Name the project and set the other options as needed.
        ii. Click "Create project".
2. Install npm from https://www.npmjs.com/get-npm, the instructions are located on the site.
3. To get a Google Maps API key, follow the instructions at https://developers.google.com/maps/documentation/embed/get-api-key
4. Create a new folder, this folder will contain the website files eventually and will be your working directory. Example: "C:\Users\User1\Desktop\FirebaseWorkFolder"
5. Open command prompt.
    a. Install Firebase tools by entering "npm install -g firebase-tools".
        i. Enter "firebase login"
        ii. You will be asked to sign in to an account. This account should be the one that created the Firebase project or an account that has been added to the project.
    b. Navigate to the newly created folder.
    c. Enter "firebase init" and follow the instructions.
        i. Are you ready to proceed? Y
        ii. You will be asked to select which features you will be using. Follow the instructions given and select:
            1. Firestore.
            2. Hosting.
            3. Storage.
        iii. Select the Firebase project that was created by following Step 1.
        iv. Use all the basic rules by pressing Enter.

v. You can name your public directory as anything you would like but anyone else who wishes to add to the project must name their directory exactly the same thing. “public” is default.
vi. Configuring as a single-page app does not matter as we will be copying our source files over.
vii. There is one more basic rule to enter then you are completed with initializing firebase for use in that folder.

6. In file explorer, open your “C:\Users\User1\Desktop\FirebaseWorkFolder” and you will see that this folder now contains some files and a folder, by default, named “public”.
    a. Navigate into the “public” file and delete any files inside.
7. Go to https://github.com/LugwellDR/cmcSrcFiles and download as zip, the project.
    a. Unzip the files
8. There are several files that need to be modified for a given Google Maps API key and configuration setup from the Firebase project. Open any editor to edit four HTML files and one JavaScript file.
    a. Your specific Google Maps API key needs to be copied into the script tag that looks like this.
        i. <script async defer src="https://maps.googleapis.com/maps/api/js?key=[GOOGLEAPIKEY]&libraries=drawing,visualization"></script>
    b. The four files that contain this script are:
        i. Home.html
        ii. cmcMapEditor.html
        iii. cmcHeatmap.html
        iv. cmcAbout.html
    c. Next, edit the domainConfig.js file.
        i. Navigate to the Firebase project console and click “Add app”.
        ii. Choose the web option and a code snippet will be shown. This information can be copied and pasted to replace the variables within the domainConfig.js file.
9. Copy all of files and folders into the “public” folder.
10. Return to the command prompt and navigate to the “public” folder. Example: “C:\Users\User1\Desktop\FirebaseWorkFolder\public”.
    a. Enter “firebase deploy”. This will upload the files to the Firebase project and will be readily available to the general public!

10.4 Production Cycle

Application

To edit a page layout, locate the XML file in the Android Studio Project tab by going to app > res > layouts

To edit the java files, locate the Java filed in the Android Studio Project tab in app > java > educapstoneprojectscs2019mogreen_s19

After making edits, the code changes will save after clicking Run. Ensure that an android device is plugged in or choose an emulator after clicking run. The application will load on the device of your choice.

To submit your changes to the Git, click Commit and confirm your changes. After this, go to VCS > git > Push to send in the commit.

Website

To edit a HTML5, CSS, or JavaScript source code file, you should navigate to the directory containing the file and open the desired file in your preferred text editor.

After making your edits, you should open the command prompt and navigate to the directory where the source files are stored. Firebase should be already initialized in this directory path.

In command prompt, enter “firebase deploy”, this will deploy the website to live servers.