Bioretention Assessment App

Final Report

Bob Simmons

Associate Professor – Water Resources

The Gardening Team

CptS 423 Software Design Project II

Spring 2021

Instructor: Ananth Jillepalli

TABLE OF CONTENTS

I. Introduction

Stormwater drainage into local watersheds is one of many pressing conservation issues. The flow of water from our streets to nearby streams, ponds, rivers, lakes, and other bodies of water tends to pick up a lot of debris along the way. Much of this debris is extremely toxic to aquatic life and, with a moderate amount of difficulty, can be prevented from reaching any of these natural habitats. One such solution to this problem is the implementation of rain gardens into areas with a high amount of run-off. These rain gardens help to filter the toxic materials out of the water before it is absorbed into the ground and makes its way to a substantial body of water. WSU Extension is investigating a solution to this problem through the use of rain gardens, which help filter contaminants from runoff. The department has been collecting data about the performance of these rain gardens with the help of volunteers who must print off, fill out, and return physical forms to WSU Extension. The data is then manually entered into an Excel sheet by WSU Extension. The goal for this project was to provide a webform for the volunteers to enter their answers as well as create a database to store the collected information.

Our mentor for this project was WSU Assistant Professor Ananth Jillepalli. He is part of the Electrical Engineering & Computer Science department at WSU. He can be contacted at ananth.jillepalli@wsu.edu.

Our Client for this project was WSU Associate Professor Bob Simmons. He is affiliated with WSU Extension-Olympia. He can be contacted at simmons@wsu.edu.

II. Team Members & Bios

Daniel Hazelton is a WSU student who is studying for a bachelor's degree in computer science and a minor in mathematics. He is comfortable with many programming languages including: C, C#, C++, ML, HTML, Java, and Python. For this project he has been responsible for being the team lead and working on the frontend and backend connections.

Briana McGuire is a computer science student interested in software engineering. She has worked as a Software Engineering Intern at SEL for over a year now and has experience in ASP.NET applications and React. For this project, she mainly worked on the backend and the database design.

Bryce Turley is a computer science student interested in computer networking and digital security. He has experience working with a number of languages in a professional environment, including C/C++, Python, and Golang. For this project, Bryce worked on the initial design for the front-end UI, and built the foundation of the Golang server used in the final product.

Alex Strawn is a student at WSU pursuing a degree in software engineering. He is proficient in C, C++, C# and Python and has interests in software UI/UX and human-computer interaction. He has learned a great deal about HTML and CSS during this project. He predominantly worked on the project’s front-end UI.

III. Project Requirements Specification

This section describes the requirements stated by our client. The final product should meet all these requirements and acceptance tests have been derived from these requirements.

III.1. Project Stakeholders

Bob Simmons - As the project sponsor, Bob Simmons has a direct need for the application we have built, reflected by his request. He will be utilizing the application to better store rain garden info, make the data more accessible, and allow for easier data input for his volunteers.

WSU - As a sponsor of both Bob Simmons and this team, this application is representative of WSU. The Water Resources division and this application could reflect positively or poorly on the university.

Volunteers - This group will make up the majority of the users on the application as they will be using it to collect data at their rain garden sites. The “ease of use” and quality of the application will have the greatest effect on these individuals.

Coastal residents - By providing better tools to monitor the rain gardens through this application, residents that live on the coast benefit from having less pollution in their local water system. The application will allow volunteers to notice issues with local rain gardens and report them with greater ease. This will cause the issues to be fixed faster leading to a cleaner local water system.

Future users - This might include other Water Resources departments in other areas that need a way to better record their rain garden info.

III.2. Use Cases

Submit Form: Users are able to submit a rain garden form.

Retrieve Rain Garden Data: Users with the download link are able to access the data submitted by volunteers. This information will be made available as a CSV file.

III.3. Functional Requirements

III.3.1. Submission Database

Store Form Information: The system needs to store the information users enter into the form safely and efficiently.

Source: This requirement originated from the request for the website and it is essential for Bob Simmons and any other members of the WSU staff that will want to access the data from the forms.

Priority: Priority Level 0 - Essential and required functionality

III.3.2. Form

Fillout Form: The system needs to be able to remember a user's answers in the form until the form is submitted to the database.

Source: This will be useful to the volunteer so that they can go back to previous questions and change their answers if needed.

Priority: Priority Level 0 - Essential and required functionality

Submit Form: The system needs to be able to submit the information entered into the form to the database.

Source: This will allow the entered information to be available at a later time to Bob Simmons and other WSU staff looking to access the form data.

Priority: Priority Level 0 - Essential and required functionality

III.3.3. Data Retrieval

Retrieve Data from Database: The system needs to be able to retrieve data from the database in the format of a CSV file. The link provided to download the data should also allow filtering by group code.

Source: This requirement originated from the request by Bob Simmons to be able to view the data currently in the form database.

Priority: Priority Level 0 - Essential and required functionality

III.4. Non-Functional Requirements

[Programming Language(s)]:

The languages that were used for the construction of this project include HTML/CSS for the frontend, Golang for the backend, and mySQL for the database. Additional tests were written using Python.

[Testing Method]:

For system testing, we used the Selenium Python libraries which enabled us to write Python scripts that submit test data through the website, validate which sets of inputs succeed or fail, and ensure all interactable inputs work as intended. These tests ensure that all required fields work as intended and ensure that our program accepts valid inputs and rejects invalid inputs. For unit testing, we used Golang's built-in testing framework to ensure that the components of the Golang server work as intended. These tests are described in greater detail in Section V of this document.

[Device Compatibility]:

Since the system used to input data will route users through a website, any device that is able to use a web browser of some sort should be able to access the website. Our main focus was to get the website working on mobile devices.

[Database Storage]:

Throughout development, the database should not require more than 10GB of space. Since the amount of data stored will scale with the number of forms provided by users, the database may need to expand dramatically in the first year of use depending on popularity of the form. Based on the anticipated access rate of 350 form submissions per year, the database will likely require an initial size of 100GB of data.

[Server Uptime/Workload]:

The server will have very intermittent use and it is expected that the server will be able to handle multiple users accessing the same set of data at any given time. Users will most likely be accessing the server a handful of times per week. The server should be able to scale up based on the amount of users accessing the data.

[Server Response Time]:

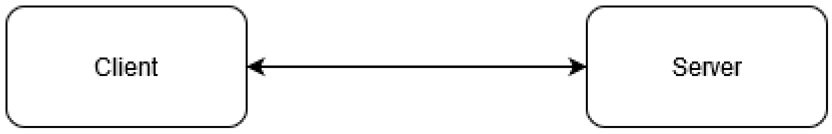
Due to the I/O nature of the data input process for this project, the server is not necessarily expected to handle requests at a rapid pace but it should still be fairly responsive to user submissions. To be more specific, it is expected that the server will be able to process a user's request to submit data within a few seconds of the user submitting the form.

IV. Software Design - From Solution Approach

IV.1. Architecture Design

IV.1.1. Overview

The software architectural pattern chosen for this project is the Client-Server model. This design pattern is best suited for this project since we have been asked to create a system which collects data from volunteers and presents it to researchers. The only two tasks that the server will ever be responsible for are inputting data into the database and outputting data from the database. The Client-Server model is a simple model which is capable of performing each of these tasks effectively.



The Client:

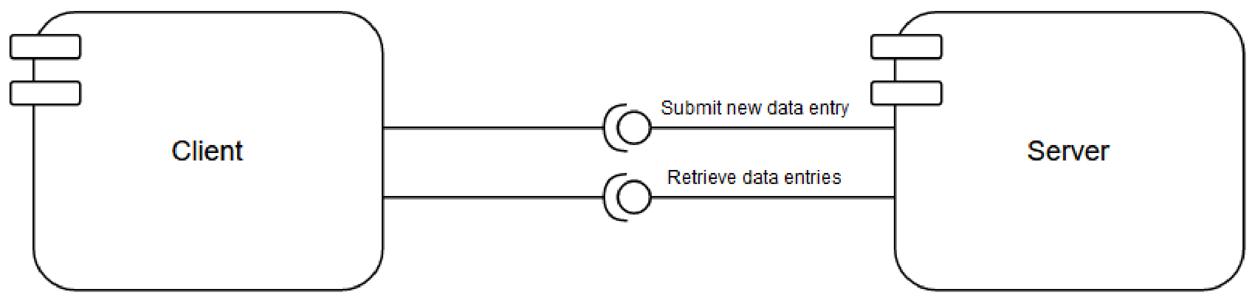
- Within the Client-Server model, the Client will be responsible for providing a way for the user to input data and submit the form.
- The Client will also provide an interface for downloading the data in the forms.

The Server:

- The Server is responsible for the creation of new entries in the database using information submitted by the Client.
- The Server is responsible for providing the database entry data as a CSV file.

Our team discussed the possibility of using the Model View Controller (MVC) architectural pattern instead, since it would allow the flexibility to add more features in the future. However, the majority of clients will only ever submit new data entries and the few clients that do access the data wish to simply download the user entries as a CSV file. For these reasons, most of the advantages of the MVC pattern would be wasted since there would be no need for a controller to separate or parse the information sent or retrieved from the database. The additional complexities involved in implementing MVC would only introduce additional problems during system design.

IV.1.2. Subsystem Decomposition



IV.1.2.1. Client

a) Description

This subsystem is responsible for providing the user with the ability to either submit new data entries to the server or retrieve the list of data entries as a CSV file.

b) Concepts and Algorithms Generated

This subsystem will be a web page that allows the user to submit new data entries through the use of a HTTP POST request and request all data entries from the Server through a HTTP GET request. The front end will also use built in HTML functionality to perform front-end validation checking for required fields and maximum lengths of text inputs.

c) Interface Description

Services Provided: None

Services Required:

Retrieve Data Entries provided by Server

Submit New Data Entries provided by Server

IV.1.2.2. Server

a) Description

This subsystem is responsible for altering and retrieving information from the database when requested.

b) Concepts and Algorithms Generated

The server adds a new entry when a user fills out the form and submits it. Also, the server will provide all of the form data for download in a csv.

c) Interface Description

Services Provided:

1. Service name: Submit New Data Entry

   Service provided to: Client

   Description: Checks that the provided information is valid before entering the provided data as a new submission entry in the database. Returns an OK HTTP response.

2. Service name: Retrieve Data Entries

   Service provided to: Client

   Description: Checks that the provided information is valid before retrieving the specified data entries from the database. Returns an OK HTTP response with the data entries in CSV format..

Services Required: None

IV.2. Data design

The diagram shown in Appendix D.1 provides a more in depth representation of the database design. The form this application is based off of can be found here: https://www.ezview.wa.gov/Portals/_1962/Documents/SAM/RainGarden-Bioretention%20Functional%20Assessment%20Form.pdf.

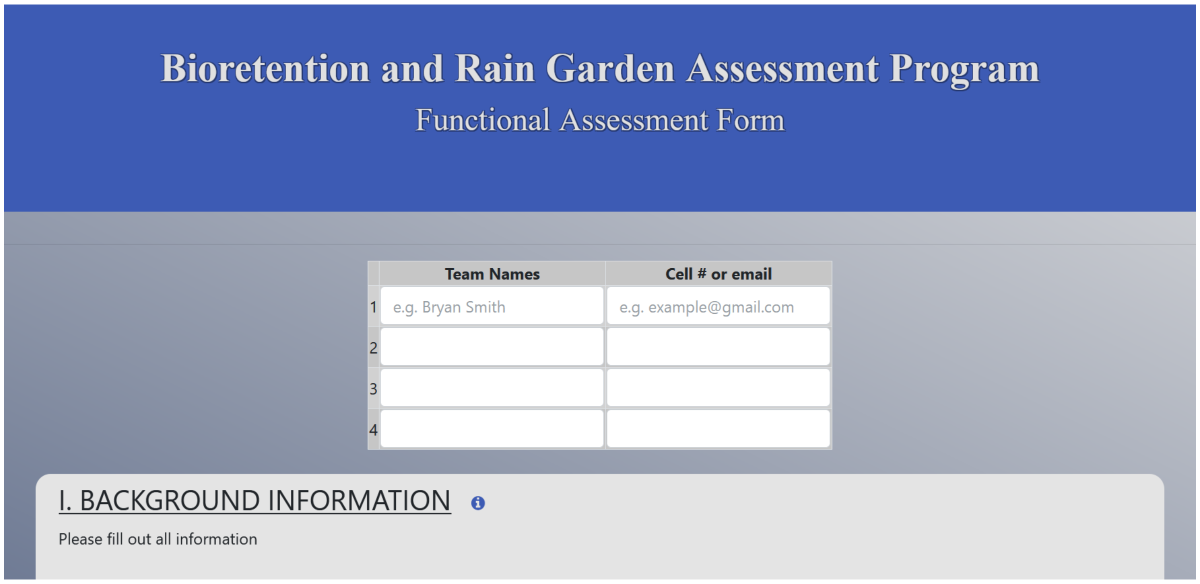
The form contains many questions that are answered with predetermined values (e.g. questions where possible answers are “Yes”, “No”, or “Unknown”). To keep the database normalized, these answers are represented by their own tables with the form having foreign key(s) to the answer tables. These answers can be represented in code by enumerated types (enums). It is important to note that the predetermined answers will need to exist in the database before any forms are entered into the database.

The main form table has foreign keys to these answer tables. Repeated sections such as sections A, B, and C or zones 1, 2, and 3 were also split out into their own tables with foreign keys back to the main table.

IV.3. User Interface Design

The product of this project will be a form hosted on a website that users can fill out and submit to a database. The database will store the information that users submit via these forms. The UI aspect of this project will be in the form of webpages. In this section we will go over the intended access, visual design, and function of these webpages.

The visual design of the forms will be fairly straightforward and minimalistic with the intention of making them as readable and understandable as possible. There are also informational buttons to help guide users through parts that they may have questions about as well as alerts triggered by entering certain values into specific inputs. Below are some examples of what these pages will look like. These examples showcase the features mentioned above.


Bioretention and Rain Garden Assessment Program

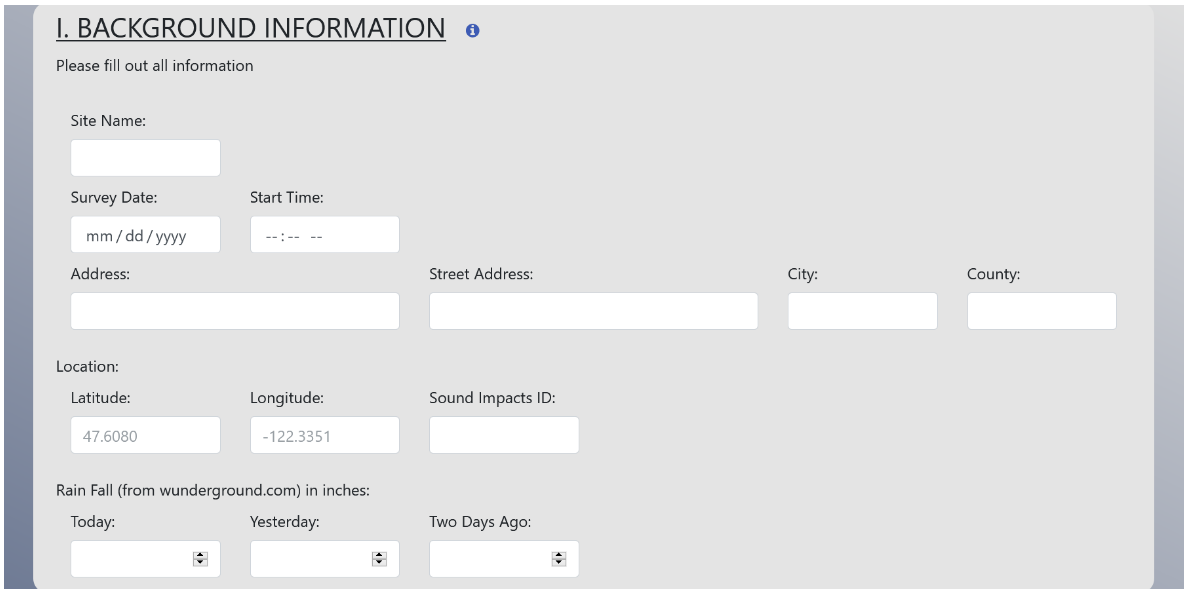
Functional Assessment Form

	Team Names	Cell # or email
1	e.g. Bryan Smith	e.g. example@gmail.com
2		
3		
4		

I. BACKGROUND INFORMATION

Please fill out all information

Figure IV.3.1


I. BACKGROUND INFORMATION

Please fill out all information

Site Name:

Survey Date: mm / dd / yyyy

Start Time: -- : -- --

Address:

Street Address:

City:

County:

Location:

Latitude: 47.6080

Longitude: -122.3351

Sound Impacts ID:

Rain Fall (from wunderground.com) in inches:

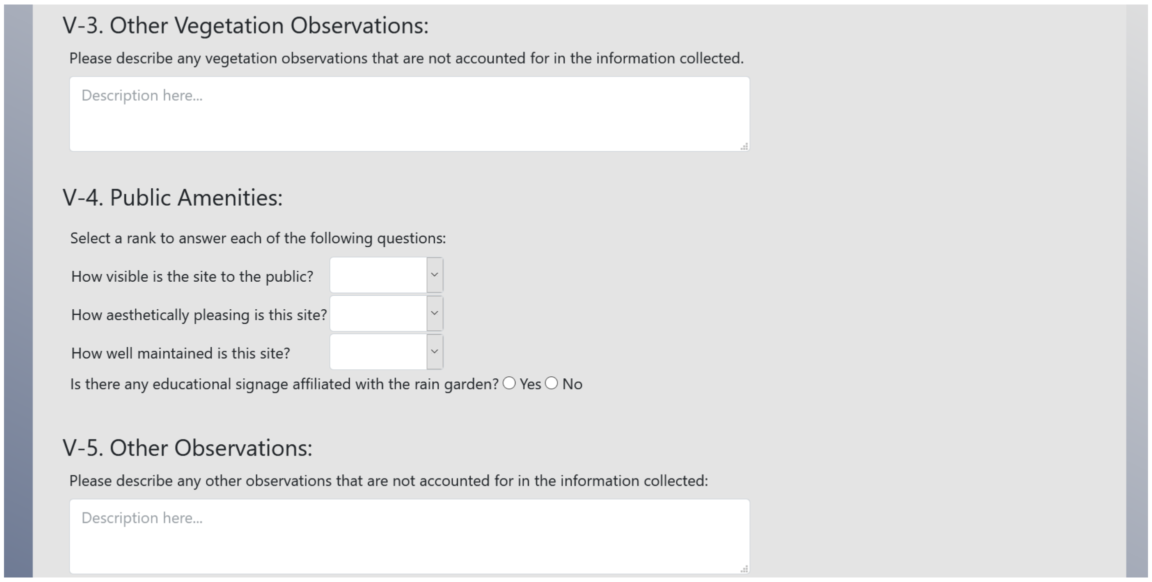
Today:

Yesterday:

Two Days Ago:

Figure IV.3.2

The above examples (Figures IV.3.1 and IV.3.2) also show the intended functionality of these pages. Input will be entered via a text box, drop down menu, radio buttons, or set of checkboxes. An example of the radio buttons and drop down menu inputs can be seen below (Figure IV.3.3). An example of the checkbox inputs can be seen below that (Figure IV.3.4).



V-3. Other Vegetation Observations:

Please describe any vegetation observations that are not accounted for in the information collected.

Description here...

V-4. Public Amenities:

Select a rank to answer each of the following questions:

How visible is the site to the public?

How aesthetically pleasing is this site?

How well maintained is this site?

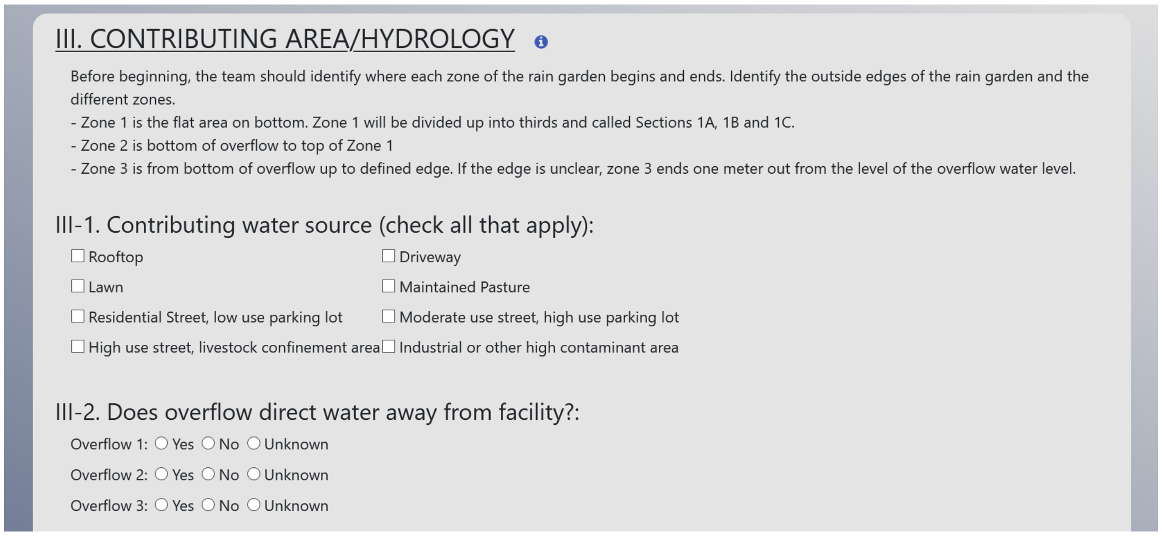
Is there any educational signage affiliated with the rain garden? ○ Yes ○ No

V-5. Other Observations:

Please describe any other observations that are not accounted for in the information collected:

Description here...

Figure IV.3.3



III. CONTRIBUTING AREA/HYDROLOGY

Before beginning, the team should identify where each zone of the rain garden begins and ends. Identify the outside edges of the rain garden and the different zones.
- Zone 1 is the flat area on bottom. Zone 1 will be divided up into thirds and called Sections 1A, 1B and 1C.
- Zone 2 is bottom of overflow to top of Zone 1
- Zone 3 is from bottom of overflow up to defined edge. If the edge is unclear, zone 3 ends one meter out from the level of the overflow water level.

III-1. Contributing water source (check all that apply):

☐ Rooftop	☐ Driveway
☐ Lawn	☐ Maintained Pasture
☐ Residential Street, low use parking lot	☐ Moderate use street, high use parking lot
☐ High use street, livestock confinement area	☐ Industrial or other high contaminant area

III-2. Does overflow direct water away from facility?:

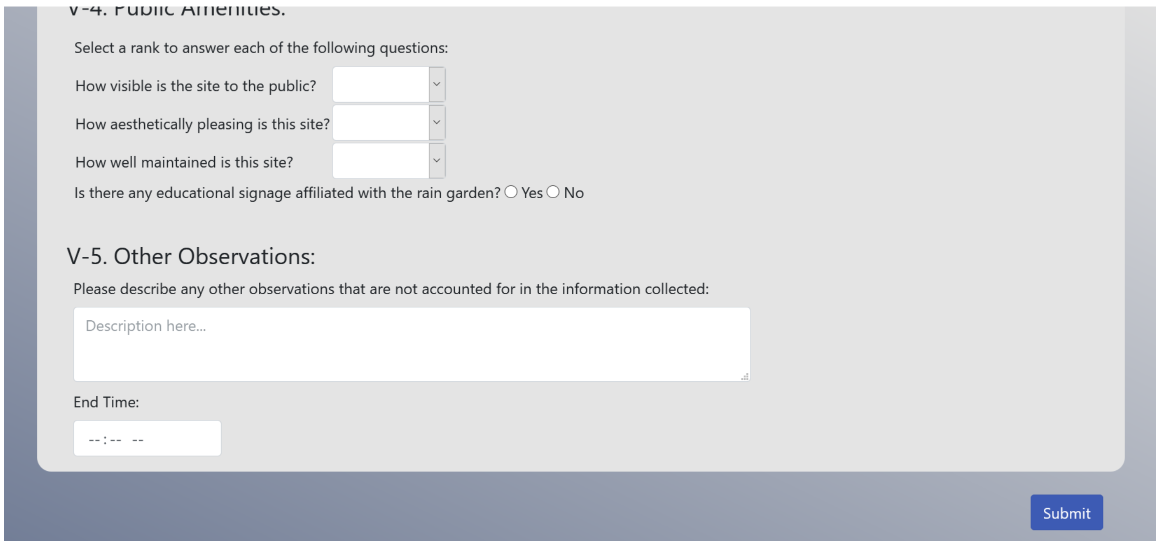
Overflow 1: ○ Yes ○ No ○ Unknown

Overflow 2: ○ Yes ○ No ○ Unknown

Overflow 3: ○ Yes ○ No ○ Unknown

Figure IV.3.4

A button will be included at the bottom of the page that will allow the user to submit the information that they have entered. This button will trigger a confirmation that asks if the user is ready to submit (Figure IV.3.6). The submit button will be made to stand out and be as visible as possible (E.g. different colors from the rest of the page elements, separated from other page elements, etc.). An example of this button can be seen below (Figure IV.3.5).


V-4. Public Amenities:

Select a rank to answer each of the following questions:

How visible is the site to the public?

How aesthetically pleasing is this site?

How well maintained is this site?

Is there any educational signage affiliated with the rain garden? ○ Yes ○ No

V-5. Other Observations:

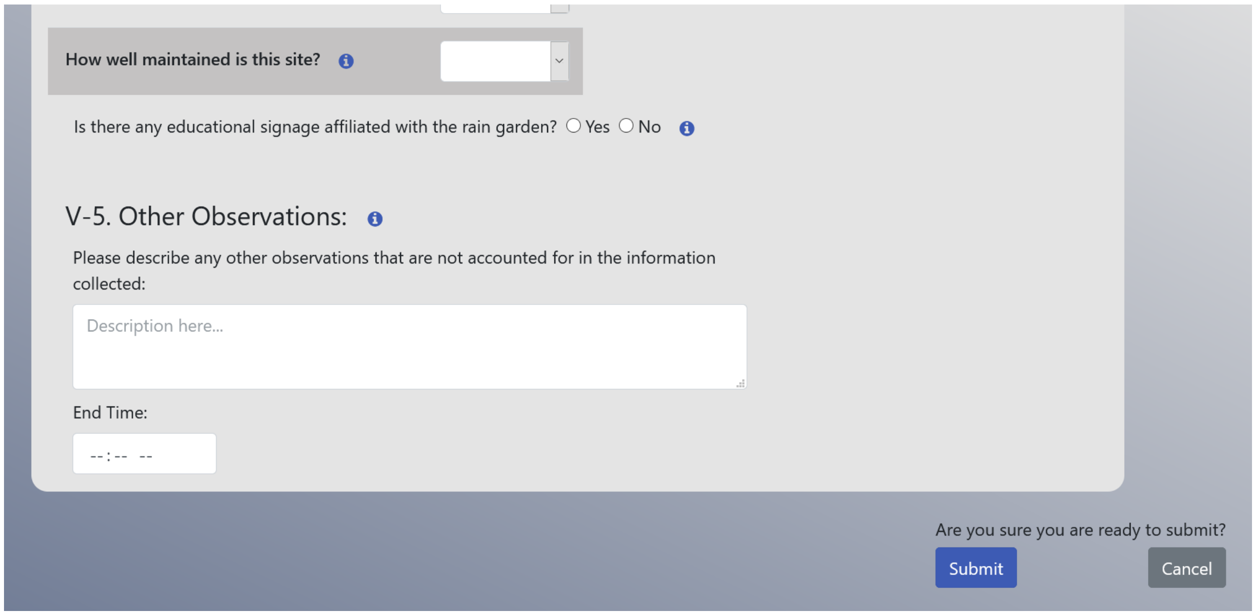
Please describe any other observations that are not accounted for in the information collected:

Description here...

End Time:

--:-- --

Figure IV.3.5


How well maintained is this site?

Is there any educational signage affiliated with the rain garden? ○ Yes ○ No

V-5. Other Observations:

Please describe any other observations that are not accounted for in the information collected:

Description here...

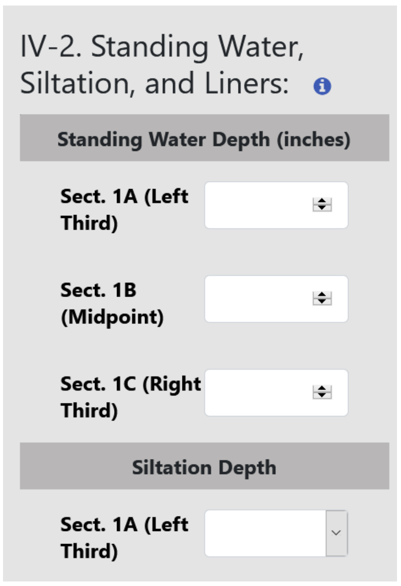
End Time:

--:-- --

Are you sure you are ready to submit?

Figure IV.3.6

These forms will also be scalable so as to be used via a smartphone or tablet's web browser. Since the rain garden projects will be outdoors it is far more likely that users will need to be able to access these forms on a smartphone or tablet as opposed to on a laptop or desktop. As such, we need to ensure that when these forms are scaled down to these smaller screens that the text is still readable, buttons are not too difficult to press, and the forms still function as expected. Our solution uses Bootstrap, which provides a set of premade CSS styles and helps with keeping styling consistent when switching from mobile to desktop. The pictures below (Figure IV.3.7, Figure IV.3.8, and Figure IV.3.9) show how the page looks when on mobile.


IV-2. Standing Water, Siltation, and Liners:

Standing Water Depth (inches)

Sect. 1A (Left Third)

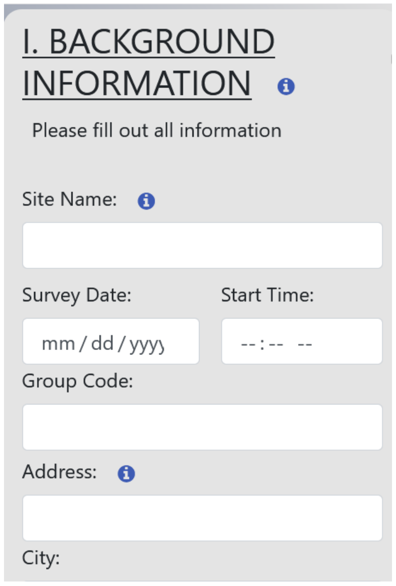
Sect. 1B (Midpoint)

Sect. 1C (Right Third)

Siltation Depth

Sect. 1A (Left Third)

Figure IV.3.7


I. BACKGROUND INFORMATION

Please fill out all information

Site Name:

Survey Date: mm / dd / yyyy

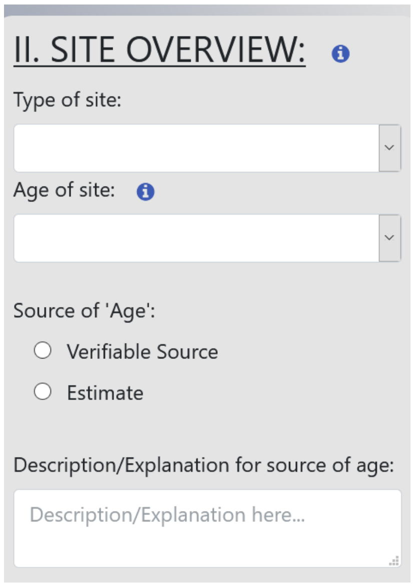
Start Time: -- : -- --

Group Code:

Address:

City:

Figure IV.3.8


II. SITE OVERVIEW:

Type of site:

Age of site:

Source of 'Age':

Verifiable Source

Estimate

Description/Explanation for source of age:

Description/Explanation here...

Figure IV.3.9

V. Test Case Specifications and Results

V.1. Testing Overview

<u>Unit Testing</u>:

Unit testing was provided through the use of Golang's built-in testing package. This testing package allows special environments to be set up between each test, and was used to allow individual functions within our Golang server to be unit tested. The unit tests currently written focus on ensuring that the data submitted by a user is available in a proper format, ensuring that environment variables are properly accessed by the program, and ensuring that the server would continually attempt to reconnect to the database. These tests played a critical role in ensuring that our system worked properly as we migrated them to Docker containers.

We did not provide unit tests for any other sections of our system as the database and front-end were not systems which could be unit tested. By the very nature of how the database and HTML front-end works, these elements would have fallen under integration testing.

<u>System Testing</u>:

System testing was provided through the use of the Python Selenium library. This library contains functions that allow python scripts to interact with a web page in the same manner as a human, but selecting elements, providing text to input boxes, and clicking on webpage elements. The system tests provided for this project are fairly comprehensive, and cover a wide variety of inputs that the user can give for the webform. These tests ensure that users cannot submit the webpage with key elements missing, and ensure that the page successfully submits when valid information is given. These system tests further provide a way of automatically submitting data to the database so that our team could properly test downloading data in the ‘csv’ file format.

One downside of this library is that if changes are made to the UI that change the hierarchy of the ‘div’ elements on the HTML webpage, then the tests will need to be restructured so that they can access the elements of the webpage in their new locations. Altering these CSS selectors to allow elements to be found and interacted with is trivial, but time consuming. If further changes are required, one method of circumnavigating this issue is to provide unique ID tags on all elements in the webpage. Doing so allows the elements to be found simply using the ID rather than relying on the exact placement of the element on the webpage.

V.2. Environment Requirements

In order to run the Unit Tests, the only requirement is the installation of Golang. These tests do not require a functional database, and the code itself is contained within a Golang module that handles all installation of extra libraries when run.

In order to run the System Tests, the following components are required:

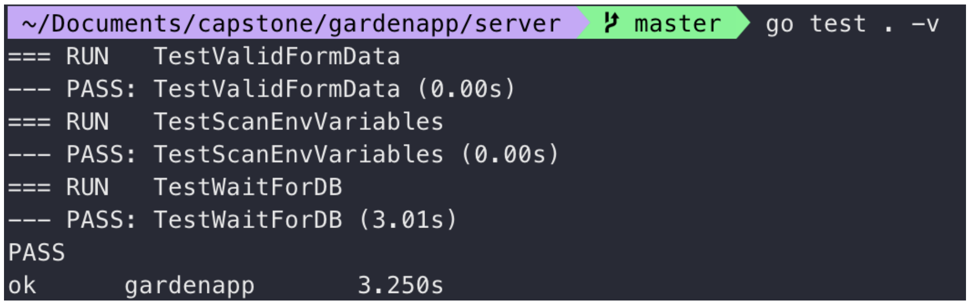
- Python 3
- Selenium Python Library
- Firefox Web-Browser

The details of how to install these elements are detailed at the top of the ‘RunAllTests.py’ file which contains the tests themselves. As described in the project documentation, these tests input the test input data from the ‘test_inputs’ folder into a physical web browser and attempt to submit the information automatically.

V.3. Test Results

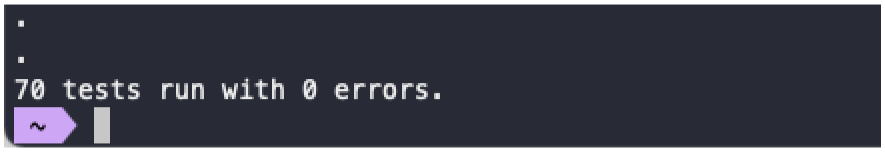
In the project’s current status (Tagged as v1.1 on GitLab.), all tests pass.

The results from the Unit tests are as follows:



```
 ~/Documents/capstone/gardenapp/server  master  go test . -v
=== RUN   TestValidFormData
--- PASS: TestValidFormData (0.00s)
=== RUN   TestScanEnvVariables
--- PASS: TestScanEnvVariables (0.00s)
=== RUN   TestWaitForDB
--- PASS: TestWaitForDB (3.01s)
PASS
ok      gardenapp       3.250s
```

The results from the System Tests are as follows:



```
.
.
70 tests run with 0 errors.
 ~ 
```

(Note: a '.' is printed for every successful system test, so only the tail of the tests are shown above.)

VI. Projects and Tools used

Bootstrap	Generating layout and visual rendering on web interface.
MariaDB	A more secure version of a MySQL.
Selenium (Python)	Automatically filling out the webform for system testing.
Docker	Simplified deployment by containerizing product components.
Docker-Compose	Allowed automatic deployment of multiple containers.
Golang MySQL Driver	Let the Golang server make SQL queries to the database.

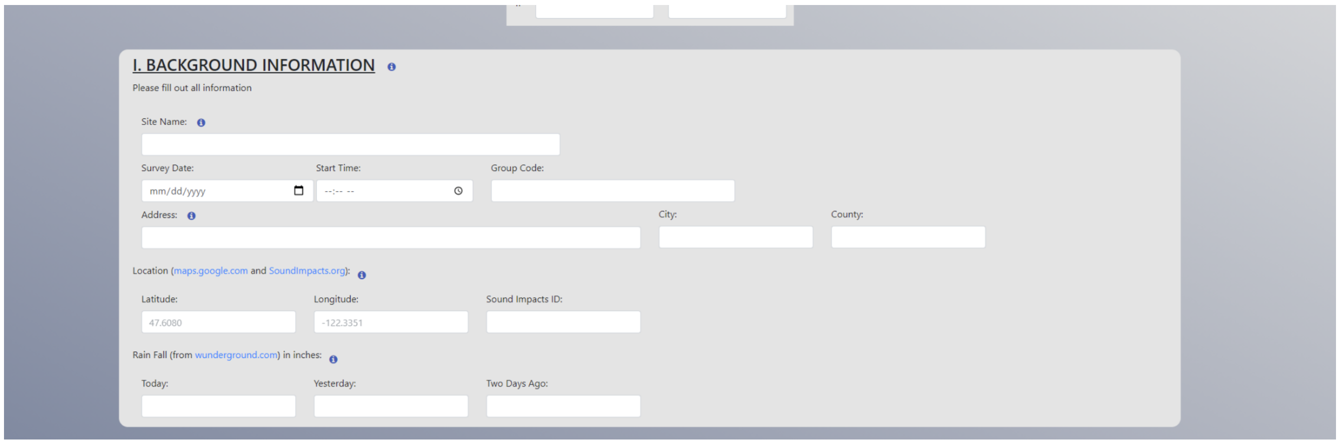
Languages Used in Project			
Golang	Python	HTML	CSS
SQL	Markdown		

VII. Description of Final Prototype

The final prototype that we have provided is a Bioretention and Rain Garden Functional Assessment Form by Bob Simmons that has been implemented into a webform with a focus on mobile use, that can be submitted to a database. We have also provided a way for our client to search the database based on a keyword and to download entered webform data as a CSV file.

Screenshots of the webform:

Desktop view:



I. BACKGROUND INFORMATION

Please fill out all information

Site Name:

Survey Date: mm/dd/yyyy

Start Time: --:-- --

Group Code:

Address:

City:

County:

Location (maps.google.com and SoundImpacts.org):

Latitude: 47.6080

Longitude: -122.3351

Sound Impacts ID:

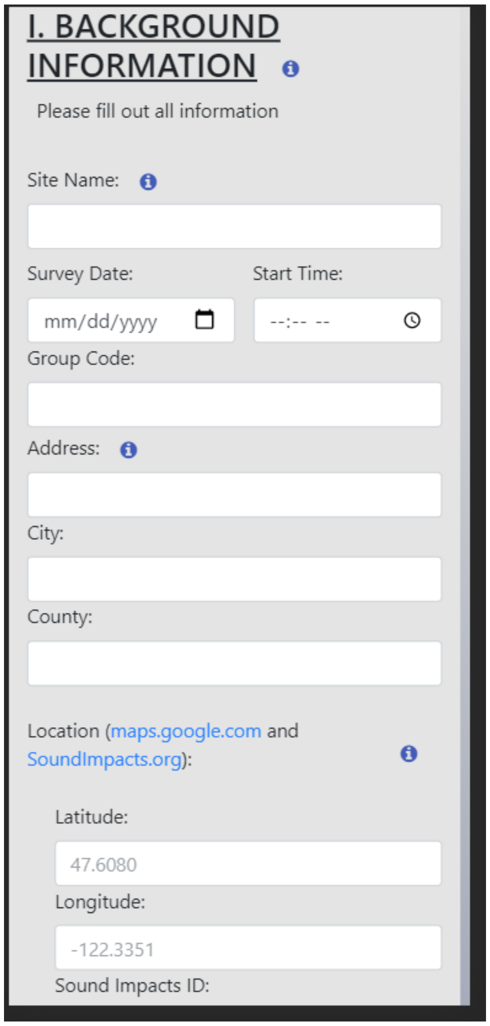
Rain Fall (from wunderground.com) in inches:

Today:

Yesterday:

Two Days Ago:

Mobile view:



I. BACKGROUND INFORMATION

Please fill out all information

Site Name:

Survey Date: mm/dd/yyyy

Start Time: --:-- --

Group Code:

Address:

City:

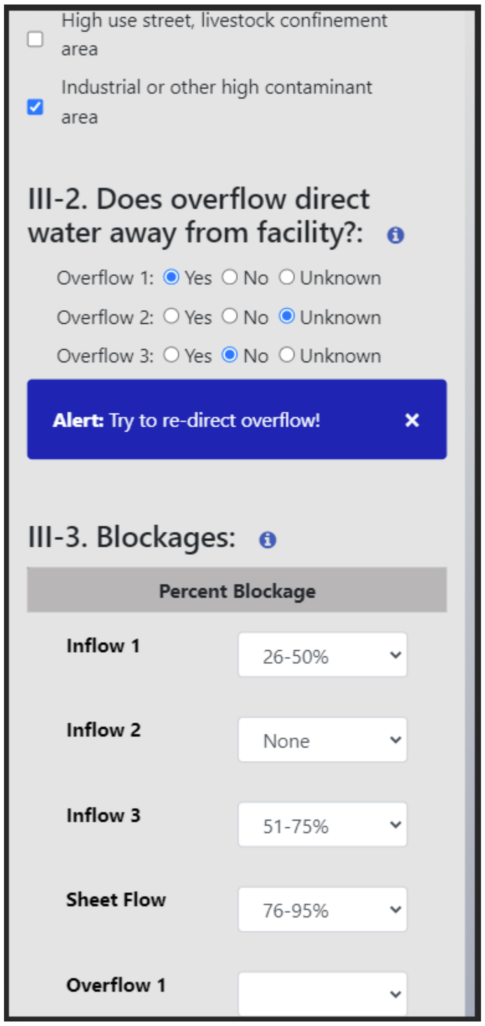
County:

Location (maps.google.com and SoundImpacts.org):

Latitude: 47.6080

Longitude: -122.3351

Sound Impacts ID:



- [ ] High use street, livestock confinement area
- [x] Industrial or other high contaminant area

III-2. Does overflow direct water away from facility?:

Overflow 1: [✓ Yes] No Unknown

Overflow 2: Yes No [✓ Unknown]

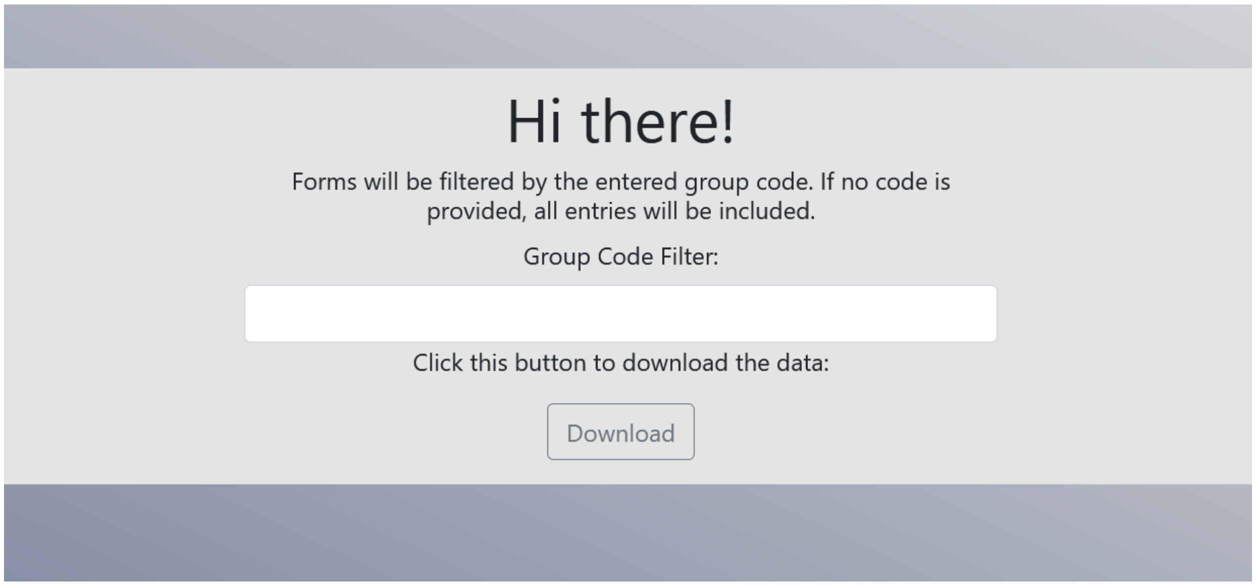
Overflow 3: Yes [✓ No] Unknown

Alert: Try to re-direct overflow! ×

III-3. Blockages:

	Percent Blockage
Inflow 1	26-50%
Inflow 2	None
Inflow 3	51-75%
Sheet Flow	76-95%
Overflow 1	

Our CSV download page:



Hi there!

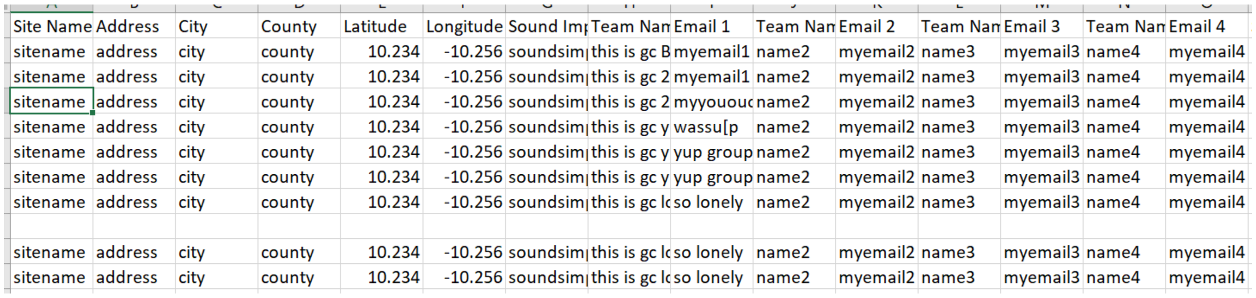
Forms will be filtered by the entered group code. If no code is provided, all entries will be included.

Group Code Filter:

Click this button to download the data:

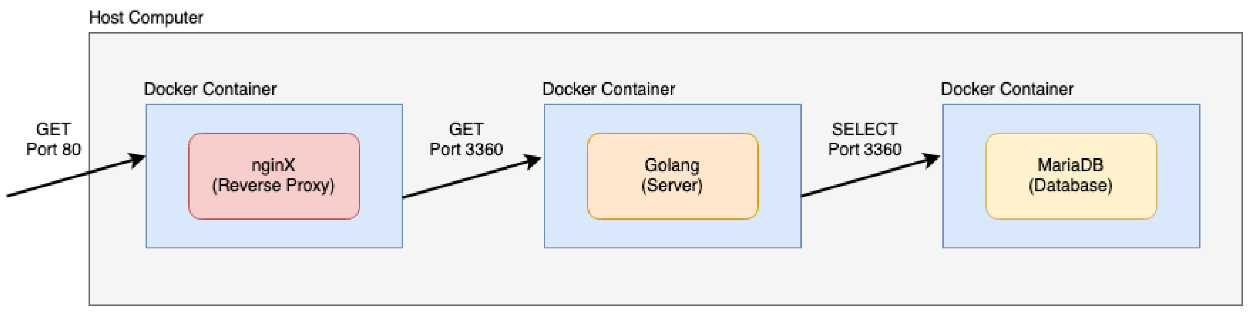
Download

The first couple columns of the CSV page:



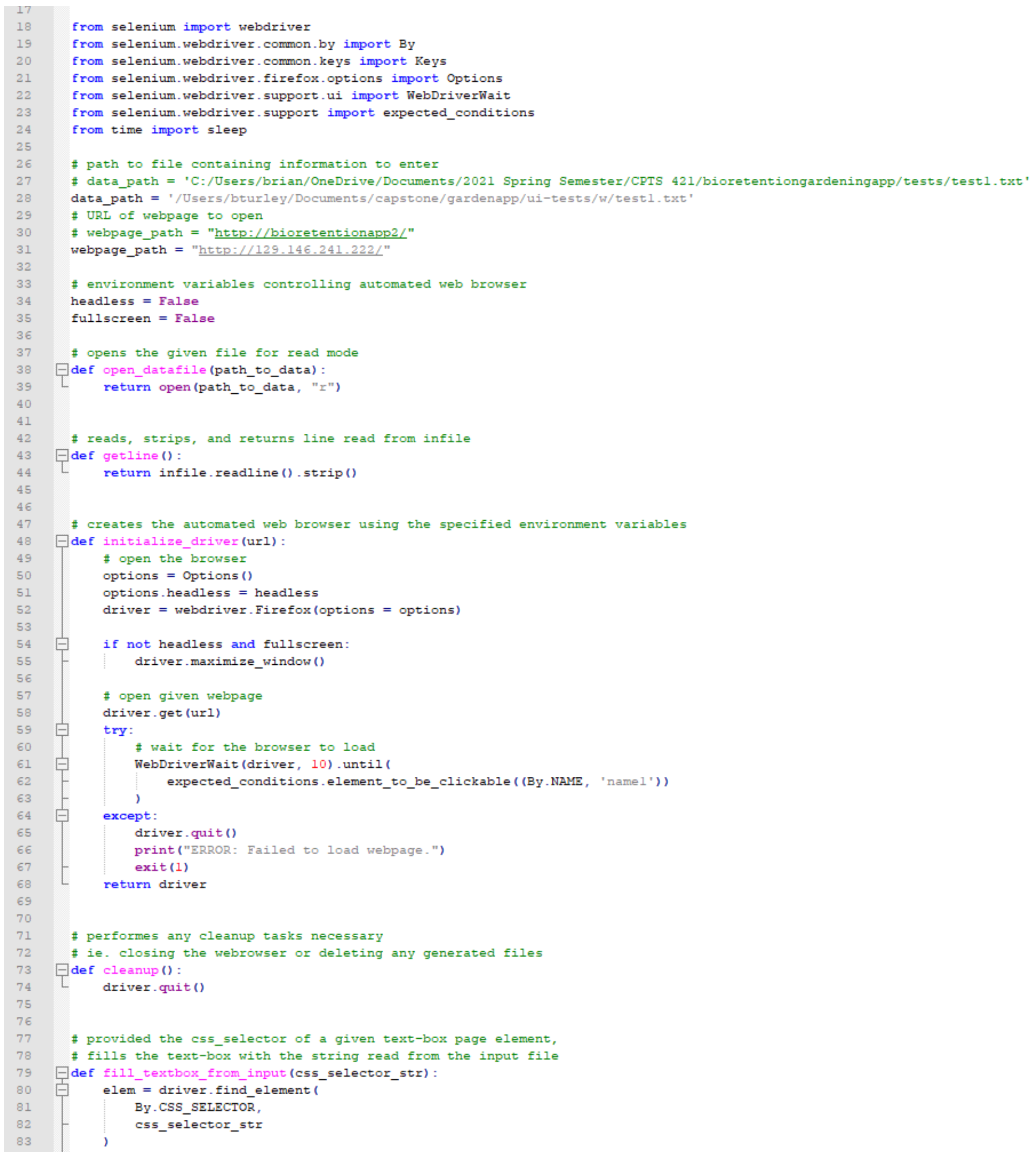
Site Name	Address	City	County	Latitude	Longitude	Sound Imp	Team Nam	Email 1	Team Nam	Email 2	Team Nam	Email 3	Team Nam	Email 4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc B	myemail1	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc 2	myemail1	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc 2	myyouou	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc y	wassu[p	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc y	yup group	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc y	yup group	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc l	so lonely	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc l	so lonely	name2	myemail2	name3	myemail3	name4	myemail4
sitename	address	city	county	10.234	-10.256	soundsim	this is gc l	so lonely	name2	myemail2	name3	myemail3	name4	myemail4

Our prototype is set up using HTML, Golang, and MariaDB, with the Golang server and MariaDB database each running inside of separate Docker containers. The Docker-Compose tool is used to configure each of these containers, and run them together. The use of Docker containers allows our code to be easily modified, more extensible, and allow it to run on any operating system. For example, VCEA IT wanted to use nginx as a reverse proxy to help secure the application on their servers, and they were able to simply modify the docker-compose file with an nginx container and it was easily added to our system. The below diagram demonstrates the complete system and each of their respective containers. The entire system can be started, stopped, and monitored with the use of Docker's built-in commands.



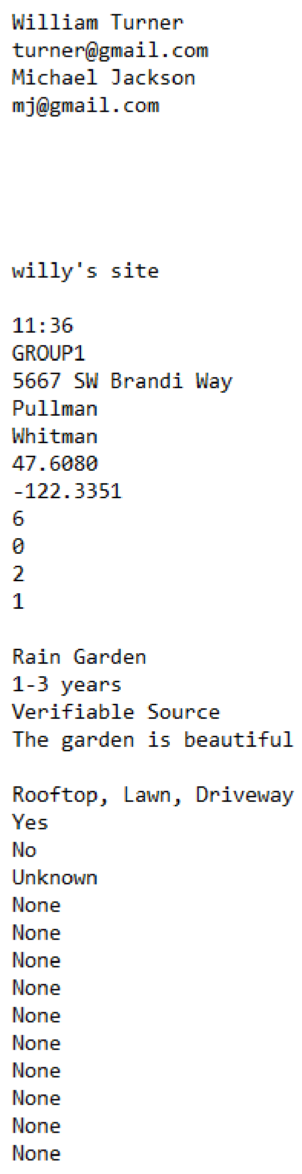
To provide testing for the webform we used Selenium to write python code that will auto fill the webform with specified data from written input files.

An look at the python testing code - “TestFunc.py”:



```python

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
from time import sleep

# path to file containing information to enter
# data_path = 'C:/Users/brian/OneDrive/Documents/2021 Spring Semester/CPTS 421/bioretentiongardeningapp/tests/test1.txt'
data_path = '/Users/bturley/Documents/capstone/gardenapp/ui-tests/w/test1.txt'
# URL of webpage to open
# webpage_path = "http://bioretentionapp2/"
webpage_path = "http://129.146.241.222/"

# environment variables controlling automated web browser
headless = False
fullscreen = False

# opens the given file for read mode
def open_datafile(path_to_data):
    return open(path_to_data, "r")


# reads, strips, and returns line read from infile
def getline():
    return infile.readline().strip()


# creates the automated web browser using the specified environment variables
def initialize_driver(url):
    # open the browser
    options = Options()
    options.headless = headless
    driver = webdriver.Firefox(options = options)

    if not headless and fullscreen:
        driver.maximize_window()

    # open given webpage
    driver.get(url)
    try:
        # wait for the browser to load
        WebDriverWait(driver, 10).until(
            expected_conditions.element_to_be_clickable((By.NAME, 'name1'))
        )
    except:
        driver.quit()
        print("ERROR: Failed to load webpage.")
        exit(1)
    return driver


# performes any cleanup tasks necessary
# ie. closing the webrowser or deleting any generated files
def cleanup():
    driver.quit()


# provided the css_selector of a given text-box page element,
# fills the text-box with the string read from the input file
def fill_textbox_from_input(css_selector_str):
    elem = driver.find_element(
        By.CSS_SELECTOR,
        css_selector_str
    )
```

A partial screenshot of a test input file:


```
William Turner
turner@gmail.com
Michael Jackson
mj@gmail.com





willy's site

11:36
GROUP1
5667 SW Brandi Way
Pullman
Whitman
47.6080
-122.3351
6
0
2
1

Rain Garden
1-3 years
Verifiable Source
The garden is beautiful

Rooftop, Lawn, Driveway
Yes
No
Unknown
None
None
None
None
None
None
None
None
None
None
```

VIII. Product Delivery Status

The product is currently live at *bioretention.eecs.wsu.edu*, and was demoed to our project sponsor and his team on Tuesday, April 26th 2021. Since VCEA IT wanted to make some last minute changes to add nginX into the system, our team will continue to monitor the status of the webpage until the end of finals week to ensure that the product works as intended.

All project materials, documentation, and source code can be found at the GitLab repository for our project (Project ID: 3755). In order to gain access to the source code and make changes to the project, a support ticket will need to be submitted to *support.vcea.wsu.edu*. The VCEA IT team is available to help with issues surrounding the deployment, hosting, and scalability of the project, but is not available to work directly on project maintenance and implementation of new features.

The documentation for the project is primarily found in the GitLab repository's README, and each file containing code has a comment block at the top of it describing that file's intended purpose, requirements, and interactions. The documentation provided is intended to be as comprehensive as possible, and covers topics ranging from a file breakdown to a guide on installing the system dependencies to an example application setup and execution on a new operating system.

IX. Conclusions and Future Work

IX.1. Limitations and Recommendations

While working on our current prototype, due to time constraints, we were not able to implement all the possible features that the website could benefit from. These features include:

Auto fill of location using Google Maps: Our team did not have experience in implementing a feature like this. We were able to implement a way to search for your location using Google Maps and instructions on how to obtain the necessary information from it but the webform needs to be filled in manually.

Uploading photos to the database: A feature we wanted to add was the ability for a user to upload images of the rain garden, from camera roll or taken live, and send the photos to the database with the webform. While we were focusing on this feature and had begun to research the topic we realized that the amount of time it would take to implement would be too great for the amount of time we had to work on more important features. We also realized that the addition of photos to our webform would greatly increase our needed storage capacity for the database. As we were already in the midst of hosting on VCEA servers and had sent our database requirements to VCEA IT, we were concerned that making changes to this request would cause issues and falter our plans of releasing the final prototype on time.

Using a login system to keep track of users: Our original idea when working on this project was to implement a login system where a user could login to an account or continue as a guest. We instead create a system that allows for our client to search for forms using codes that can be given to users. If the login system was to be implemented, we believe it could be done using Google/Facebook without much addition to the database requirements.

Having the webform be multiple pages: While this was never a requested requirement, we believe the webform would be more user friendly if it was split into multiple pages. However this was not high up on our TODO list and was not implemented in time.

IX.2. Future Work

Future work/additional features:

- Having the longitude and latitude fill itself out based on current location on button press.
- Allowing users to provide photos of the site and upload them to the database.
- Create some sort of incentive to encourage more people to use the app.
- Provide a way to show nearby rain garden locations.
- A login system using Google/Facebook logins to interact with the website and to track users.
- The webform is currently only one page, it may be beneficial to have the form split into multiple pages that save the filled out progress.

X. Acknowledgements

We would like to personally thank:

Professor, Ananth Jillepall, for organizing this opportunity for us.

Erik Ostrom from VCEA IT for greatly helping us on the technical side of the project and allowing us to easily host our website on VCEA servers.

Bob Simmons, our project sponsor, for not only being very cooperative and helpful but for providing this project for us to create. We are very grateful to have worked on this.

XI. Glossary

Rain Garden: A simple shallow depression that uses a special mix of sand and compost that allows water to soak in rapidly and supports healthy plant growth. Can be landscaped with a variety of plants to fit the surroundings. This type of garden is usually used to filter stormwater so that pollutants being carried by run-off do not make their way to larger bodies of water.

XII. References

Rain Gardens. 2020. Assessing And Monitoring Rain Gardens | Rain Gardens | Washington State University. [online] Available at: https://extension.wsu.edu/raingarden/monitoring-rain-gardens/ [Accessed 26 October 2020].

Selenium with Python. 2018. Selenium | Baiju Muthukadan. [online] Available at: https://selenium-python.readthedocs.io/ [Accessed 9 November 2020].

Web Accessibility. 2020. Web Communication | Washington State University. [online] Available at: https://web.wsu.edu/accessibility/ [Accessed 9 November 2020].

XIII. Appendix A – Team Information

Team Members: Bryce Turley, Briana McGuire, Alex Strawn, Daniel Hazelton

Team Photo:

XIV. Appendix B - Example Testing Strategy Reporting

For a detailed description of our testing strategy, see Section V.

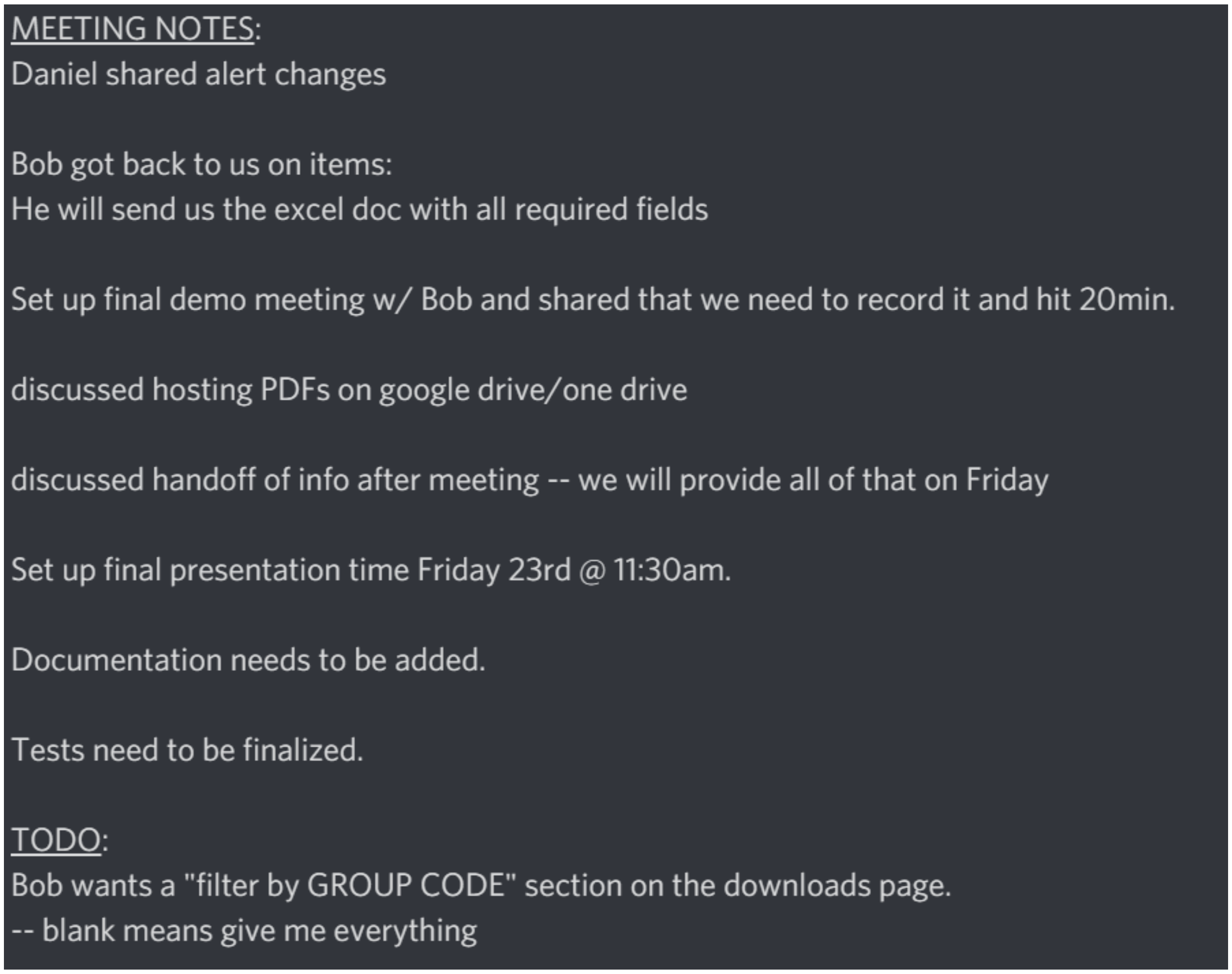
For user testing, we frequently gave our client, Bob Simmons, access to a running version of our website. After using the website, he would give us feedback at our bi-weekly meetings. We would then address the feedback and give Bob access to the changes again before our next meeting. Bob's feedback included making alerts more specific, changing placeholder values to be more accurate to the inputs, and removing unnecessary inputs.

XV. Appendix C - Project Management

Our team had bi-weekly meetings with our client to discuss our progress and his feedback along with any questions we had about the requirements. After these meetings, we would meet as a team and discuss Bob's feedback and what needed to be done before the next meeting. Sometimes our team would also meet in between client meetings to check on the progress that had been made and to make sure that we were on track to be ready for the next client meeting. We also had meetings during the week before a writing assignment was due in order to work on those.

We found the meeting after client meeting to be most useful for our team since we were able to get a collective understanding of the feedback provided by Bob and make sure everyone knew what they were doing for the next two weeks. These meetings also provided a time for anyone to get help with one of their tasks that they were having trouble with or ask for the team's opinion on a feature.

Typically, during our client meeting, one of the team members would take notes on the meeting and then share them with the group using Discord. An example of these notes is shown below in Figure XV.1.:



MEETING NOTES:
Daniel shared alert changes

Bob got back to us on items:
He will send us the excel doc with all required fields

Set up final demo meeting w/ Bob and shared that we need to record it and hit 20min.

discussed hosting PDFs on google drive/one drive

discussed handoff of info after meeting -- we will provide all of that on Friday

Set up final presentation time Friday 23rd @ 11:30am.

Documentation needs to be added.

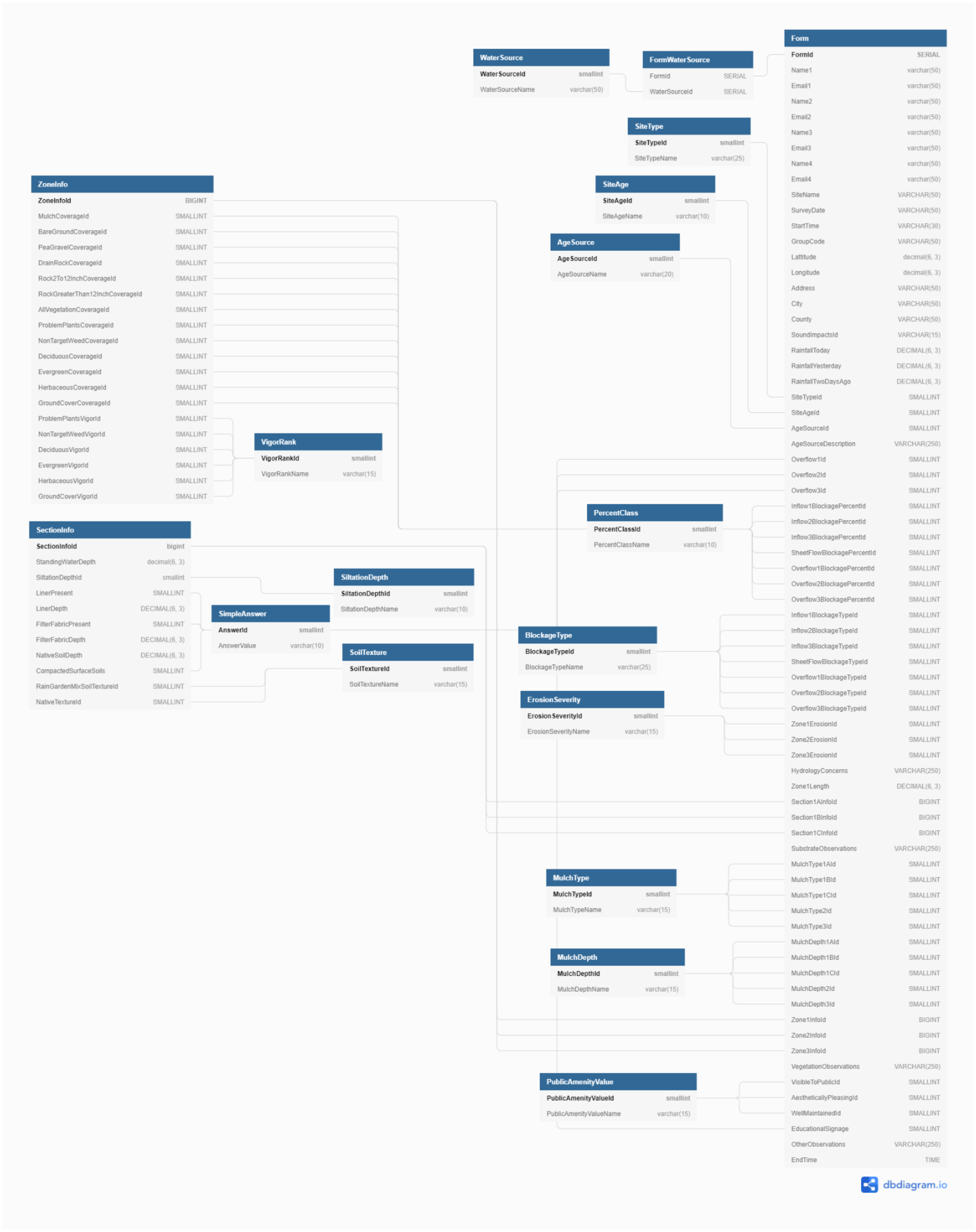
Tests need to be finalized.

TODO:
Bob wants a "filter by GROUP CODE" section on the downloads page.
-- blank means give me everything

Figure XV.1

XVI. Appendix D - Other

Appendix D.1


WaterSource
WaterSourceId smallint
WaterSourceName varchar(50)
FormWaterSource
FormId SERIAL
WaterSourceId SERIAL
Form
FormId SERIAL
Name1 varchar(50)
Email1 varchar(50)
Name2 varchar(50)
Email2 varchar(50)
Name3 varchar(50)
Email3 varchar(50)
Name4 varchar(50)
Email4 varchar(50)
SiteName VARCHAR(50)
SurveyDate VARCHAR(50)
StartTime VARCHAR(30)
GroupCode VARCHAR(50)
Lattitude decimal(6, 3)
Longitude decimal(6, 3)
Address VARCHAR(50)
City VARCHAR(50)
County VARCHAR(50)
SoundImpactsId VARCHAR(15)
RainfallToday DECIMAL(6, 3)
RainfallYesterday DECIMAL(6, 3)
RainfallTwoDaysAgo DECIMAL(6, 3)
SiteTypeId SMALLINT
SiteAgeId SMALLINT
AgeSourceId SMALLINT
AgeSourceDescription VARCHAR(250)
Overflow1Id SMALLINT
Overflow2Id SMALLINT
Overflow3Id SMALLINT
Inflow1BlockagePercentId SMALLINT
Inflow2BlockagePercentId SMALLINT
Inflow3BlockagePercentId SMALLINT
SheetFlowBlockagePercentId SMALLINT
Overflow1BlockagePercentId SMALLINT
Overflow2BlockagePercentId SMALLINT
Overflow3BlockagePercentId SMALLINT
Inflow1BlockageTypeId SMALLINT
Inflow2BlockageTypeId SMALLINT
Inflow3BlockageTypeId SMALLINT
SheetFlowBlockageTypeId SMALLINT
Overflow1BlockageTypeId SMALLINT
Overflow2BlockageTypeId SMALLINT
Overflow3BlockageTypeId SMALLINT
Zone1ErosionId SMALLINT
Zone2ErosionId SMALLINT
Zone3ErosionId SMALLINT
HydrologyConcerns VARCHAR(250)
Zone1Length DECIMAL(6, 3)
Section1AInfoId BIGINT
Section1BInfoId BIGINT
Section1CInfoId BIGINT
SubstrateObservations VARCHAR(250)
MulchType1AId SMALLINT
MulchType1BId SMALLINT
MulchType1CId SMALLINT
MulchType2Id SMALLINT
MulchType3Id SMALLINT
MulchDepth1AId SMALLINT
MulchDepth1BId SMALLINT
MulchDepth1CId SMALLINT
MulchDepth2Id SMALLINT
MulchDepth3Id SMALLINT
Zone1InfoId BIGINT
Zone2InfoId BIGINT
Zone3InfoId BIGINT
VegetationObservations VARCHAR(250)
VisibleToPublicId SMALLINT
AestheticallyPleasingId SMALLINT
WellMaintainedId SMALLINT
EducationalSignage SMALLINT
OtherObservations VARCHAR(250)
EndTime TIME
SiteType
SiteTypeId smallint
SiteTypeName varchar(25)
SiteAge
SiteAgeId smallint
SiteAgeName varchar(10)
AgeSource
AgeSourceId smallint
AgeSourceName varchar(20)
ZoneInfo
ZoneInfoId BIGINT
MulchCoverageId SMALLINT
BareGroundCoverageId SMALLINT
PeaGravelCoverageId SMALLINT
DrainRockCoverageId SMALLINT
Rock2To12InchCoverageId SMALLINT
RockGreaterThan12InchCoverageId SMALLINT
AllVegetationCoverageId SMALLINT
ProblemPlantsCoverageId SMALLINT
NonTargetWeedCoverageId SMALLINT
DeciduousCoverageId SMALLINT
EvergreenCoverageId SMALLINT
HerbaceousCoverageId SMALLINT
GroundCoverCoverageId SMALLINT
ProblemPlantsVigorId SMALLINT
NonTargetWeedVigorId SMALLINT
DeciduousVigorId SMALLINT
EvergreenVigorId SMALLINT
HerbaceousVigorId SMALLINT
GroundCoverVigorId SMALLINT
VigorRank
VigorRankId smallint
VigorRankName varchar(15)
PercentClass
PercentClassId smallint
PercentClassName varchar(10)
SectionInfo
SectionInfoId bigint
StandingWaterDepth decimal(6, 3)
SiltationDepthId smallint
LinerPresent SMALLINT
LinerDepth DECIMAL(6, 3)
FilterFabricPresent SMALLINT
FilterFabricDepth DECIMAL(6, 3)
NativeSoilDepth DECIMAL(6, 3)
CompactedSurfaceSoils SMALLINT
RainGardenMixSoilTextureId SMALLINT
NativeTextureId SMALLINT
SiltationDepth
SiltationDepthId smallint
SiltationDepthName varchar(10)
SimpleAnswer
AnswerId smallint
AnswerValue varchar(10)
SoilTexture
SoilTextureId smallint
SoilTextureName varchar(15)
BlockageType
BlockageTypeId smallint
BlockageTypeName varchar(25)
ErosionSeverity
ErosionSeverityId smallint
ErosionSeverityName varchar(15)
MulchType
MulchTypeId smallint
MulchTypeName varchar(15)
MulchDepth
MulchDepthId smallint
MulchDepthName varchar(15)
PublicAmenityValue
PublicAmenityValueId smallint
PublicAmenityValueName varchar(15)
dbdiagram.io