AquaWatch Mobile

by Team Blue Jelly
Ardin Kraja, Meryl Mizell, Erin Sorbella, Kenji Okura

Software Requirements Specification Document

Version: 1.1 **Date: 12/18/2023**

Table of Contents

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document for Blue Jelly's AquaWatch Mobile water quality monitoring system. This document defines scope and overall description of the project. In addition, we define here all of the requirements of the system. We write keeping in mind the organization sponsoring our project, Blue CoLab who will maintain and have new teams improve upon our system.

1.2 Scope

The follow projects fall under the scope of our system:

1. Server Side Architecture ("Blue Jelly Server") - The purpose of this server is to:
    a. make API requests to different bodies of water
    b. process, clean, aggregate data
    c. generate water quality index reports
    d. cache reports and data by month

    With this design we hope to (a) increase responsiveness (b) reduce API requests to entities hosting water quality data.
2. Shiny Web App ("Blue Jelly Shiny Web App") - The purpose of this service is to produce user friendly and informative graphics of water quality data. This should make requests to "Blue Jelly Server" directly and shall not public APIs.
3. AquaWatch Mobile App - Provide a user interface to:
    a. access the web app
    b. inform users of relevant facts water quality and animal life
    c. access AI page

1.3 Definitions, Acronyms, and Abbreviations.

Turbidity - Turbidity is the measure of the relative clarity of water. It is also defined as "the term used by Dr. Tom Schmidt and Tyler Besnoff (of the "TLX") to describe the former ChoateVisual, now, AquaWatch Mobile app".

WQI - Water quality index, the major of the water cleanliness.

Data Viz - Shorthand for data visualizations.

1.4 References

Omitted

1.5 Overview

The rest of document is organized as follows:

- Section 2: Provides an overall description of our system as a background to why the requirements exist. *(RemoveThisLine: for organizations intended to do what we did here)*
- Section 3: Provides a detailed set of requirements for the function of the system.
- Section 4: Provides an overview of the change management process.
- Section 5: Provides the list of SRS approvers,

2. The Overall Description

This app should be able to educate users on water quality metrics, such as salinity, pH, and turbidity. These metrics are used to describe the "health" of local bodies of water, such as lakes, rivers, and ponds. Understanding these metrics can be helpful in allowing a user to make informed choices about performing recreational activities in that body of water.

Additionally, a core component of the app is being able to visualize these water quality metrics. The app is able to deliver monthly water quality reports, which consists of graphs of metrics such as salinity throughout a given month, and a "grade" known as WQIs on the water quality for that month. Furthermore, the user will be able to investigate that relationship between weather and water quality, seeing how weather conditions such as rainfall can impact water quality parameters such as turbidity through graphs.

Lastly, the app will contain features that contextualize the importance of water quality data. Such features include a list of animals that can be found at a chosen body of water, and how water quality can impact their living.

To help users learn more about plant life, the app includes an AI Plant widget to identify invasive plants.

2.1 Product Perspective

Our research has shown that mobile applications that inspect and visualize water quality data of local lakes, rivers, and ponds are rather rare. Some organizations, such as USGS and HRECOS, host their own APIs for retrieving water quality data. They utilize their own interfaces for visualizations: for example, HRECOS has their own React Native app for viewing water quality data, which is accessible online.

Our goal is to create an app that is more comprehensive, maintaining a database of all bodies of water containing publicly accessible water quality APIs. This could possibly extend into partnerships with other organizations that maintain water quality records, but do not share it publicly.

Our app utilizes APIs hosted by BlueColab and USGS/HRECOS for water quality parameters, and Open-Meteo for weather data. The information collected from APIs will be used to generate monthly water quality reports. In addition, we will consider creating our own publicly accessible API so that others may use our monthly water quality reports for their own projects.

2.1.1 System Interfaces

The following is a list of external interfaces:

1. APIs that include:
   a. Blue Colab API - Service used to gather Choate Pond data. The Blue Jelly server makes a request to this API.
   b. USGS/HRECOS - Service used to gather Hudson River data. The Blue Jelly server makes a request to this API.
   c. Open-Metro - Service used to gather historic weather data. The Blue Jelly server makes a request to this API. (Future)
   d. Pl@ntNet API - Service used to process images and produce identification of plant images.
   e. GBIF REST API - Service used to get additional information on plants including photos.
2. shinyapps.io - Service used to host web app built into AquaWatch Mobile. Hosted via a web container.
3. MongoDB Atlas Database - Service used to host data (like WQIs).
4. Elastic Beanstalk - Service used to host servers (which interact with apps).

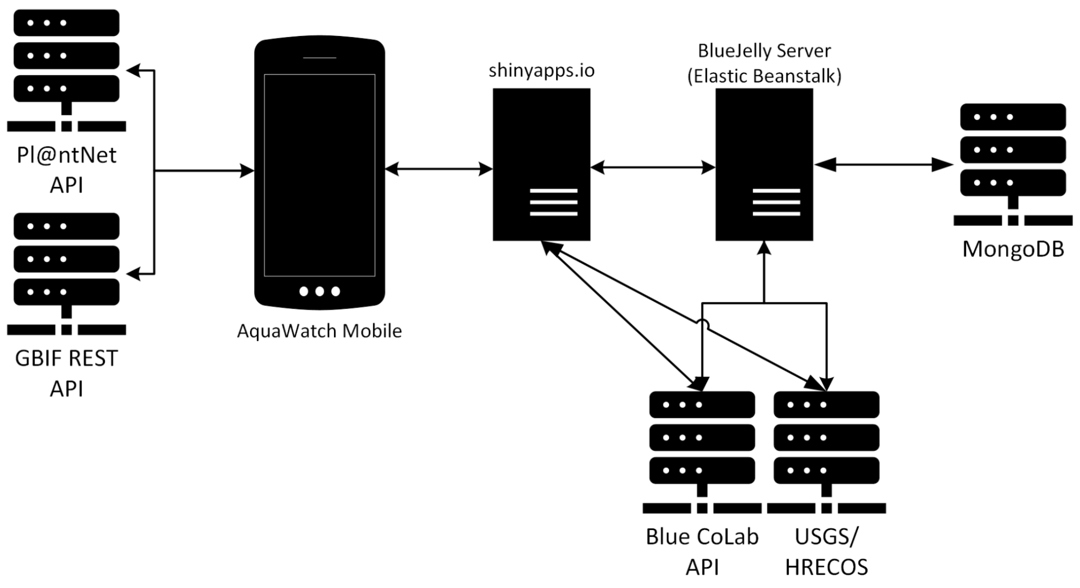

Figure 2.1: Block Diagram, showing basic interfaces.

2.1.2 Interfaces

```
AquaWatch Mobile App
└─Welcome Screen
    └─ Home Screen
        ├─ [1] Our Story
        ├─ [2] Data Hub
        │   ├─ Choate Pond Water Quality Data
        │   │   └─ [2a] Shiny Interactive UI (Water)
        │   ├─ West Point Water Quality Data
        │   │   └─ [2b] Shiny Interactive UI (Water)
        │   ├─ Poughkeepsie Water Quality Data
        │   │   └─ [2c] Shiny Interactive UI (Water)
        │   └─ Yonkers Water Quality Data
        │       └─ [2d] Shiny Interactive UI (Water)
        ├─ [3] Weather
        │   └─ [3a] Shiny Interactive UI (Weather)
        ├─ [4] Animal Life
        ├─ [5] AI Page
        │   ├─ Flip Camera
        │   ├─ Take Photo
        │   │   └─ Analyze Photo?
        │   │       ├─ Retake
        │   │       └─ Analysis Output
        │   └─ Get Photo from Library
        │       └─ Analyze Photo?
        │           ├─ Retake
        │           └─ Analysis Output
        ├─ [6] Blogs
        └─ [7] Attribution Page
```

Figure 2.2: App Interface

The primary interface used by users is the AquaWatch Mobile app. The app is organized into subsections for (1) Our Story (2) Water Quality Data for different bodies of water (3) Weather and (4) Animal Life (5) AI page (6) Blog Page (7) Attributes.

Within (2) 4 secondary interfaces show different options on which water source to look at. 3 of these options are different points in the Hudson River while one of them looks at Choate Pond on the Pace University campus. Each contains a Shiny UI. For [2a,2b,2c,2d,3a] reference the next page.

In (5) it requests camera permissions and allows you to use your camera with 3 buttons that can rotate the camera, take a picture, and look through a device's photo library. It then creates a page where it gives information on what plant the picture has in it and whether the plant is invasive.

```
Shiny Interactive UI (Water)
├── Water Parameters Dropdown
├── Location 1 Location Dropdown (required)
├── Location 1 Year Dropdown (required)
├── Location 1 Month Dropdown (required)
├── Location 2 Location Dropdown (optional)
├── Location 2 Year Dropdown (optional)
├── Location 2 Month Dropdown (optional)
└── Interactable Graph
```

Figure 2.2: Shiny App Interface for Water (2a, 2b, 2c, 2d)

The Shiny web app used to display visualizations to the user via drop-downs and interactive graphs - this is for the water.

```
Shiny Interactive UI (Weather)
├── Weather Parameters Dropdown
├── Location Year Dropdown (required)
├── Location Month Dropdown (required)
└── Interactable Graph
```

Figure 2.2: Shiny App Interface for Weather (3a)

The Shiny web app used to display visualizations to the user via drop-downs and interactive graphs - this is for weather.

Our server does not currently have a user interface. All data is updated manually. However there are plans to add automation.

2.1.3 Hardware Interfaces

The hardware used to gather water quality data shall not fall under the scope of this SRS, as we rely on the APIs to gather data on our behalf. Our system does not have any hardware devices that will interact with or control and thus the system has no hardware interface requirements.

2.1.4 Software Interfaces

The following is a list of all required software.

<u>Client-side Data Visualizations:</u>

1. R (2023.09.1, R Foundation) - R serves as a data visualization tool and also provides an user interface to access the data visualizations. More importantly we include many packages as listed here:
    a. shiny (1.8.0, postit) - Tool to create interactive web interface for accessing data visualizations. This is where the drop downs come from.
    b. jsonlite (1.8.7, Jeroen Ooms) - Tool to get and download json data.
    c. zoo (1.8-12, Achim Zeileis et. al.) - A package for working with irregular time series data.
    d. ggplot2 (3.4.4, Hadley Wickham et. al./postit) - Creating elegant data visualizations using Grammar of Graphics.
    e. shinyWidgets (0.8.0, dreamRs) - Extends Shiny with custom input widgets.
    f. dataRetrieval (2.7.14, Laura DeCicco/USGS) - Package used to download USGS water data.
    g. dplyr (1.1.4, Hadley Wickham et. al./postit) - Package to modify grammar of data.
    h. lubridate (1.9.3, Vitalie Spinu) - Helps with date and time manipulation in R.
    i. plotly (4.10.3, Carson Sievert/Chapman and Hall/CRC) - Enables interactive plots and charts.
    j. ggthemes (4.2.4, Jeffrey B. Arnold) - Provides additional themes and color scales for ggplot2.
    k. shinycssloaders (1.0.0, Andras Sali) - Adds loading spinners to Shiny applications as data loads.
    l. purrr (1.0.2, Hadley Wickham/postit) - Adds functional programming (like map) to R for data processing.

    Listed for reference by future developers. These packages are required for the product to work.

<u>AquaWatch Mobile App Front End:</u>

1. React/React Native (18.2.0, 0.72.6, Metra Open Source) - Serves as our front end alongside other packages. Various packages include:
    a. react-native-webview (13.2.2, Thibault Malbranche) - WebView component to embed website.
    b. react-native-snap-carousel (^3.9.1, Benoît Delmaire) - Create cool carousels used in wildlife pages.

2. Expo (~49.0.10, 650 Industries, Inc.) - Required as of now to display applications on phones.
    a. expo-camera (~13.4.4, Expo) - Camera access in app.
    b. expo-image-picker (~14.3.2, Expo) - Camera album access in app.
    c. expo-location (~16.1.0, Expo) - Location access in app.

3. moment (^2.29.4, JS Foundation) - Used to parse and format dates.

4. axios (^1.6.2, axios) - Used to make API requests for the AI page.

Server Side Architecture:

1. axios (^1.6.1, axios) - Makes HTTP requests to servers, used to communicate with the Blue CoLab APIs.

2. express (^4.18.2, Node.js) - Used to create a server to handle and serve HTTP requests.

3. mongodb (^6.2.0, MongoDB, Inc.) - Allows interaction with out MongoDB database. Allows connecting, querying, and manipulating data in MongoDB.

4. mongoose (^7.5.2, Mongoose.js) - It simplifies MongoDB interactions by providing a schema-based abstraction and features for data modeling.

5. node-fetch (^2.6.7, David Frank) - Also makes HTTP requests to Blue CoLab API.

2.1.5 Communications Interfaces

Accessing our own API, and the APIs of other organizations (such as USGS or Open-Meteo), can be completed through HTTP/HTTPS requests.

Interactions with the MongoDB Atlas Cluster utilize the MongoDB wire protocol, which does not need additional installations from the user.

2.1.6 Memory Constraints

Our app is intended for the modern smartphone. Since most of the heavy-duty work of generating graphs is done server side, the minimum required for this app is a smartphone running iOS 9 or Android 6.0.

Server side memory constraints are 512MB of storage, shared vCPU and RAM, as we use Atlas M0.

2.1.7 Operations

The app will be able to provide the functions specified earlier in this document; primarily data visualizations of local bodies of water.

Additionally, the app will be able to automatically generate new monthly reports for each body of water at the start of a new month. This should be able to occur without the prompting of a user or developer.

2.1.8 Site Adaptation Requirements

This app requires maintenance of Amazon EC2 instances and MongoDB clusters. In the case of the database, it will already be pre-populated with monthly reports before being released to the customer. Adjustments to this configuration must be decided upon by future team members or the customer.

The app does not require any additional software to be downloaded onto a user's phone for the corresponding React Native app.

2.2 Product Functions

Functions of the product:

- Inform the user of their water quality
- Inform the user of the local weather
- Inform the user of local wildlife
- Introduce the user to the organization, Blue Colab
- Use AI to analyze photos taken by the user to figure out what plant life is around them
- Attribute credit of APIs and other software to the proper owners
- Provide access to the Blue CoLab blogs

2.3 User Characteristics

Some of the general characteristics of the intended user are that they are local to the Westchester area and have completed at least a partial high school education. Users local to the Westchester area will find our app more useful and engaging because the contents of the app provides insight about outdoor water quality sources in Westchester. In general, users may need at least a partial high school education to fully comprehend the water quality data given and understand the implications of this data. Our app should be designed for users with minimal technical expertise as it should be easy to navigate with an intuitive design. To increase the simplicity of our design and appeal to many people, we will design a scrollable main page with widgets that give general information. If the user wants more information on a topic then they will be able to click a "see more" button. They will be able to navigate back to the main page by clicking a home icon. This intuitive design aims to oblige users that may not be accustomed to navigating many pages.

2.4 Constraints

Getting data is limited by the pace of the downstream API hosted by Blue CoLab and the USGS. We can only have as much data as they provide. We are limited to their frequency and number of parameters they collect on their sonds.

The shinyapps.io is limited to 20 hours of active hours per month globally.

The API for photos is limited to 500 photos per day globally (interface to other applications).

Both MongoDB and Elastic Beanstalk have their own constraints in max connections.

We are not limited by parallel operations, audit functions, control functions, higher-order language requirements, handshake protocols as they do not exist

We should have reliability requirements - they're not implemented just yet so therefore does not limit us.

Our system is not imminently safety critical.

The system currently does not have security considerations. There are plans in the works to add privacy policies etc., that may limit what developers can do.

2.5 Assumptions and Dependencies

The decision to embed the Shiny UI is an assumption that the shinyapps.io service is used. Shinyapps.io in its current plan is not sustainable for a large number of users. If the hosting site were to be moved to Blue CoLab hosted servers the SRS will be updated.

Much of the interface is hardcoded - any change in Sond data will not be reflected automatically. SRS needs to be updated.

2.6 Apportioning of Requirements.

Future requirements include:

1. Automation Tasks - Future updates will factor in automation which the SRS shall reflect.
2. Security/Privacy Tasks - Future updates will factor promises to meet privacy policy requirements (as we handle images now).

3. Specific Requirements

3.1 External Interfaces

The systems shall interface with…

1. Blue CoLab API
    a. Description: This is the main purpose of the app, to display water quality from the pond. It provides data such as temperature, dissolved oxygen, pH, turbidity, conductivity, and salinity.
    b. Source: Blue CoLab
    c. Output Destination: Data Viz Service
    d. Timing: Ideally quick
    e. Relationship: Source of data
    f. Data formats: json/with six parameters mentioned above
    g. Command formats: json/HTTP requests

The systems should interface with…

1. USGS/HRECOS
    a. Description: An additional data to provide context to Choate pond data.
    b. Source: USGS
    c. Output Destination: Data Viz Service
    d. Timing: Ideally quick
    e. Relationship: Source of data
    f. Data formats: R dataframe/with six parameters mentioned above
    g. Command formats: R dataframe/HTTP requests
2. Data Viz Service (R/shinyapps.io)
    a. Description: A data viz service must be used to serve the…data viz.
    b. Source: R recommended
    c. Output Destination: App
    d. Timing: Ideally quick
    e. Relationship: Source of data viz
    f. Data formats: n/a
    g. Command formats: user inputs
3. Server (Elastic Beanstalk)
    a. Description: Server to serve stuff stored on custom database.
    b. Source: MongoDB (if used)
    c. Output Destination: Data Viz Service
    d. Timing: Ideally quick, automatic
    e. Relationship: Source of data viz info
    f. Data formats: n/a
    g. Command formats: data viz service requests
4. Database (MongoDB)
    a. Description: Gotta store the WQIs and weather data
    b. Source: MongoDB recommended
    c. Output Destination: App
    d. Timing: Ideally quick
    e. Relationship: Stores data of data viz

   f. Data formats: n/a
   g. Command formats: server request

The system may…

1. Historic weather API
   a. Description: Service used to gather historic weather data. The Blue Jelly server makes a request to this API. (Future)
   b. Source: Public API
   c. Output Destination: Weather Viz
   d. Timing: Quick
   e. Relationship: Provides water data
   f. Data formats: json
   g. Command formats: json/HTTP request
2. Plant AI recognition API
   a. Description: Pl@ntNet API - Service used to process images and produce identification of plant images.
   b. Source: Public API
   c. Output Destination: App
   d. Timing: Quick
   e. Relationship: Plant idenifter
   f. Data formats: Text
   g. Command formats: Photos
3. Plant Fact API
   a. Description: GBIF REST API - Service used to get additional information on plants including photos.
   b. Source: Public API
   c. Output Destination: App
   d. Timing: Quick
   e. Relationship: Fact provider
   f. Data formats: Any
   g. Command formats: Any

3.2 Functions

3.2.1 A Useful UI

Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs.

These include:

- The system shall require 3 user actions or less to get to the data visualization page
- The system shall employ error handling that denote a specific error message:
    - When Shiny server is down

        - When the referenced has not been collected yet
        - When the months conflict
    - The system shall disregard data point outliers
    - The application shall have an artificial intelligence that allows users to identify any type of plant

3.2.2 A Solid Backend

These include:

- The system shall calculate WQIs for each month
- The system shall get water in a monthly basis
    - Current Blue CoLab api will break a monthly data if requested directly by user

3.2.2 An AI Page

These include:

- The system shall properly ask for camera permissions
- The system shall get photos
- The system shall send photos to API securely
- The system shall get data from the API
- The system shall display data from API
    - The system should handle errors, if not data is returned an error should be displayed
    - The information displayed should identify if a plant is invasive or not.
    -

3.3 Performance Requirements

This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole. Static numerical requirements include:

(a) Contain a screen that displays a WQI for Choate Pond
(b) Support 400 simultaneous users
(c) Handle a significant amount of data that has been collected every 15 minutes for the past 5 years
(d) UI meets 75% approval rating on survey results
(e) Create a graphic display of data that loads in less than 3 seconds

3.4 Logical Database Requirements

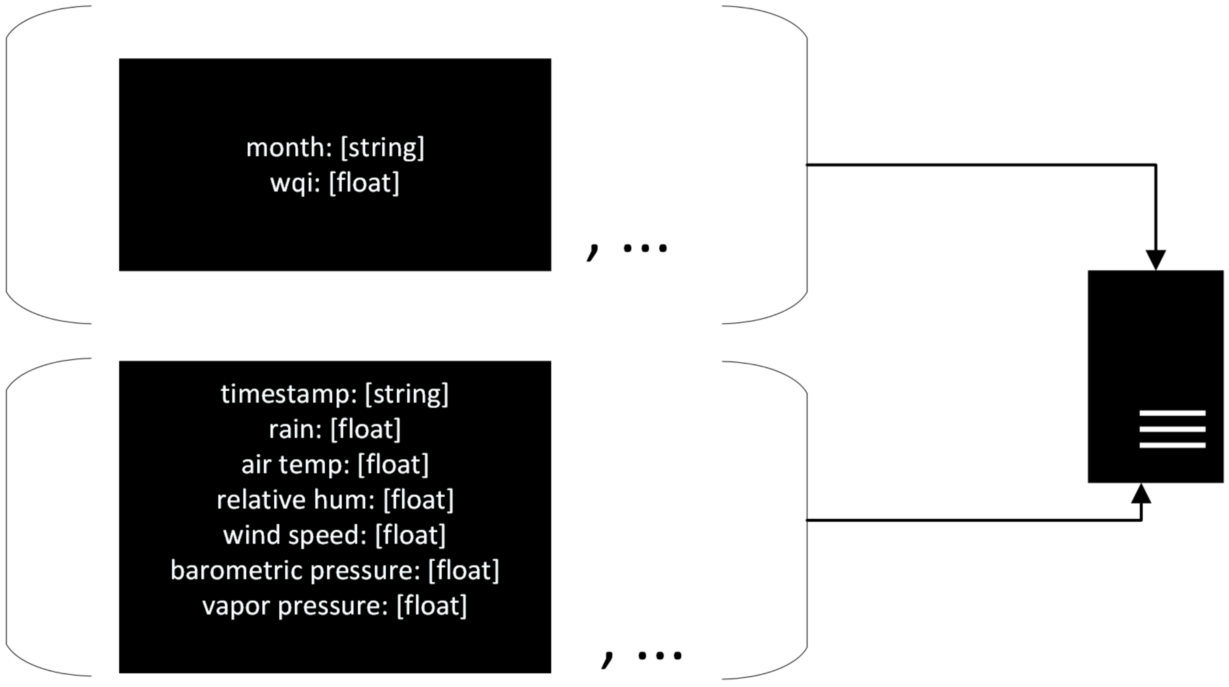

Figure 3.1: Our current server stores an index of WQIs stored by month (top). And also weather data (bottom).

The server stores WQI and weather data. They should be accessed as the user requests them. Ideally if the system were automated it would be updated monthly. All data has to be kept and accessible to valid users. The two data arrays are not related to eachother.

3.5 Design Constraints

3.5.1 Standards Compliance

AquaWatch mobile does not have any regulatory compliance rules they have to follow. (Though “Not Legal Advice™”). We are not in healthcare or the financial world. We do handle images but they are never stored locally or on our servers.

However we do have the following standards we recommend for consistency:

Data format for individual data points, with units.

timestamp:	timedate	
sensordata:	Cond	mS/cm
	DOpct	%
	Sal	ppt
	Temp	F
	Turb	NTU
	pH	N/A

Figure 3.2: Data formats and units

The following weignhts for WQI

Cond	0.08
DOpct	0.34
Temp	0.2
Turb	0.16
pH	0.22

Figure 3.3: WQI Weights

Being a nice person is also a nice standard compliance. Are you even reading this?

3.6 Software System Attributes

3.6.1 Reliability, Availability & Security

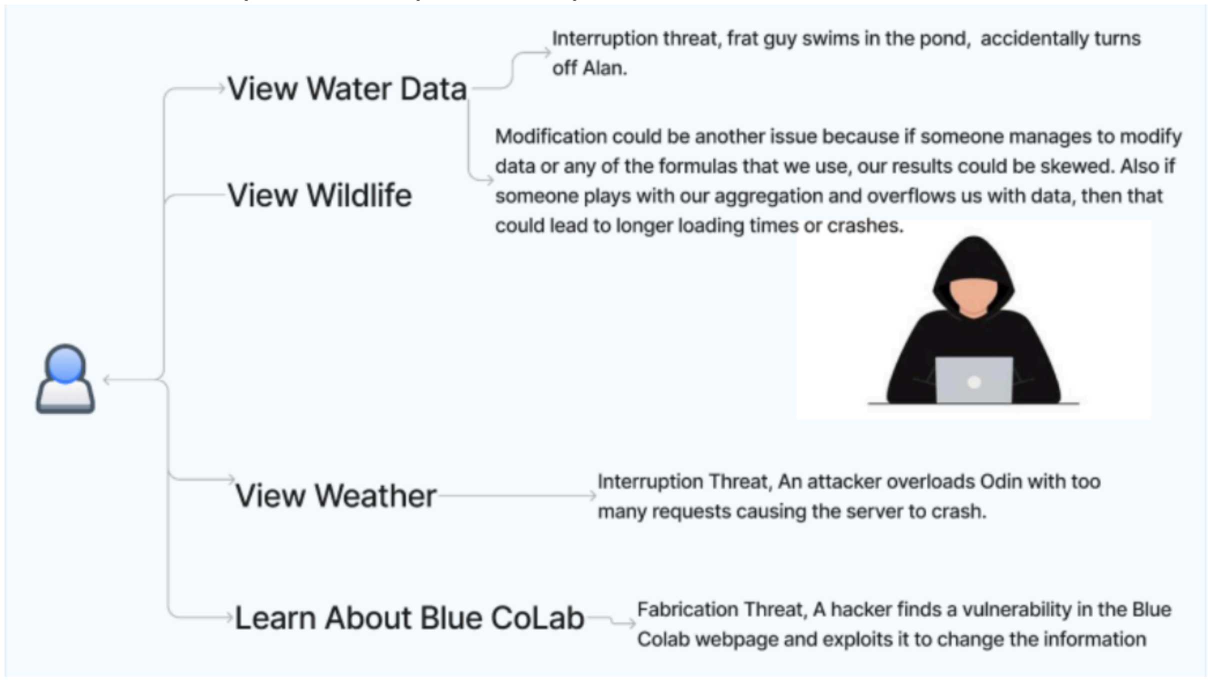
View Water Data
Interruption threat, frat guy swims in the pond, accidentally turns off Alan.
Modification could be another issue because if someone manages to modify data or any of the formulas that we use, our results could be skewed. Also if someone plays with our aggregation and overflows us with data, then that could lead to longer loading times or crashes.
View Wildlife
View Weather
Interruption Threat, An attacker overloads Odin with too many requests causing the server to crash.
Learn About Blue CoLab
Fabrication Threat, A hacker finds a vulnerability in the Blue Colab webpage and exploits it to change the information

Figure 3.4: Security

See above for what stuff should be reliable.

In terms of availability our servers have to run on-demand at the pace users access. We desire quick recovery. Currently server hosting is done by external source - we trust they will be available.

We do not have security requirements. We do not have:

- cryptographic techniques
- keeping of logs or history data sets
- checking data integrity for critical variables (for security, we check ranges, but for data viz purposes)

Future projects would need to do better in this aspect…perhaps they can implement:

- restriction in communication between areas of the program (control of API keys…)
- assign certain function to different modules (R project can be split to different files)
- keeping of logs to APIs.

We largely did not consider security in terms of privacy but keeping in mind of availability/reliability and such this section is combined.

3.6.2 Maintainability

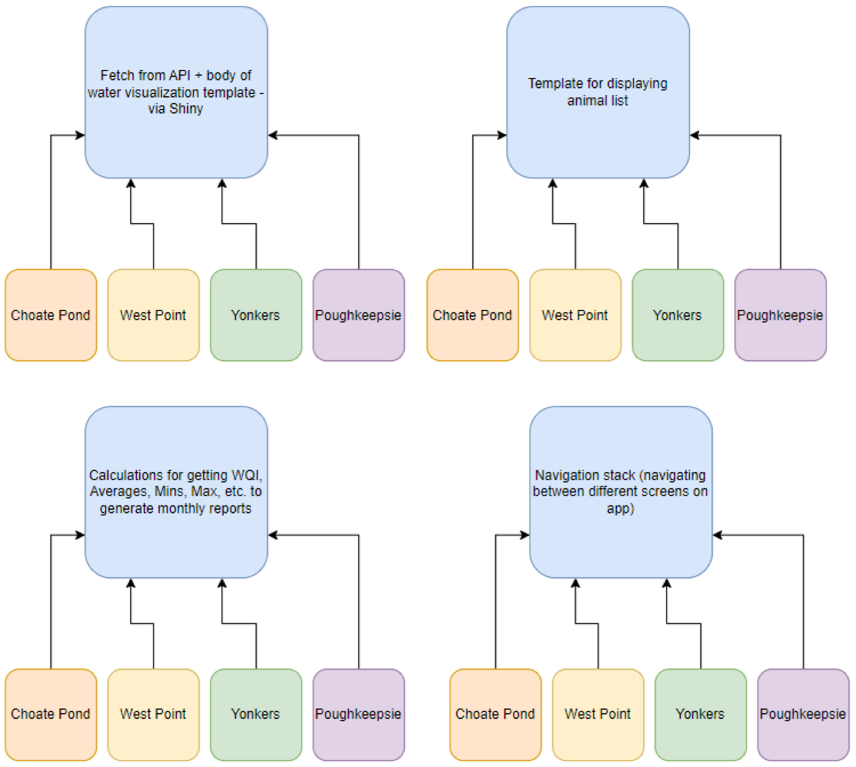
Fetch from API + body of water visualization template - via Shiny
Choate Pond
West Point
Yonkers
Poughkeepsie
Template for displaying animal list
Choate Pond
West Point
Yonkers
Poughkeepsie
Calculations for getting WQI, Averages, Mins, Max, etc. to generate monthly reports
Choate Pond
West Point
Yonkers
Poughkeepsie
Navigation stack (navigating between different screens on app)
Choate Pond
West Point
Yonkers
Poughkeepsie

Figure 3.5: Reuse! For each of the locations, we are able to do the same repeated task in the blue box.

3.6.3 Portability

ID	Characteristic	H/M/L	1	2	3	4	5	6	7	8	9	10	11	12
1	Correctness	H	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
2	Efficiency	H	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
3	Flexibility	H	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
4	Integrity/Security	L	▨	▨										
5	Interoperability	M	▨	▨	▨	▨	▨	▨						
6	Maintainability	H	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
7	Portability	L	▨	▨										
8	Reliability	M	▨	▨	▨	▨	▨	▨						
9	Reusability	M	▨	▨	▨	▨	▨	▨	▨	▨				
10	Testability	L	▨	▨	▨	▨	▨	▨						
11	Usability	H	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
12	Availability	L	▨	▨										

1. Correctness: The data shall be completely accurate, inaccurate data creates distrust. We want users to trust us.
2. Efficiency: The data shall load fast, slow loading times will cause users to lose interest.
3. Flexibility: The data shall have a level of flexibility that if given any data for any month it can display it. Saves developers work.
4. Integrity/Security: While important, some downtime is OK.
5. Interoperability: The app should interface with both the Blue CoLab API and USGS data.
6. Maintainability: The app shall have automation such that developers do not have to manually update any data.
7. Portability: The app should interface with both the Blue CoLab API and USGS data.
8. Reliability: While important, some downtime is OK.
9. Reusability: The app should interface with both the Blue CoLab API and USGS data. It should work for any month and year that has data.
10. Testability: While bugs are documented, we need to be able to test a lot to find more.
11. Usability: App gotta work to be an app?
12. Availability: While important, some downtime is OK.

4. Change Management Process

All changes will be tracked in GitHub. Document updated as we make changes. Our sponsor can request changes verbally and will be reflected in Product/Sprint Backlogs. The team should reach consensus.

5. Document Approvals

Leanne Keeley
Signature: *Leanne Keeley*
Date: 12/18/2023

6. Supporting Information

Table of Contents

Index