Hydro Citizens

Software Design Document Version 1

February 9, 2018

Sponsor:
Dr. Benjamin L. Ruddell

Team Mentor:
Dr. Eck Doerry

Team Members:
Luis Arroyo
Logan Brewer
Ryan Ladwig
Kelli Ruddy

Table of Contents

1. Introduction

Our waterways and wetlands have an important role in our everyday lives and the more knowledge we have about them, the better. Hydrological data is water height data, groundwater modeling, and stage gauging of rivers, reservoirs, lakes and other waterways. This data is important when it comes to flood prediction/prevention, water management, and public education/knowledge. If we have more data, we can more fully understand how rainfall fluctuation affects a specific area. Flooding is one of the most destructive aspects of increased amounts of rainfall and can result in loss of homes and even death. Although we can never fully predict how severe a flood will be, we can better understand how increased rainfall will affect that area if we have the data. Another importance of hydrological data collection has to do with water management. Water management deals with measuring river flow and runoff as well as how that relates to infrastructure design. Infrastructure design refers to how roads, neighborhoods and even cities are built to handle increased amounts of rainfall. Public education/knowledge is directly connected to water management in that if a community more fully understands how increased amounts of rainfall could affect their area, this may influence how they vote for public officials i.e. they may be more likely to vote for someone who understands why this data collection and design is so important. Right now the United States Geological Survey (USGS) collects most of the hydrological data in the U.S., but the problem with this is that they only collect data from large waterways and reservoirs. This is because the gauges they set up to collect this data are extremely expensive so they usually do not collect data on smaller waterways or ephemeral rivers, which are rivers without flowing water all the time. We need to understand how rainfall fluctuation affects certain areas not gauged by USGS. Without this knowledge, certain areas could be negatively affected by increased amounts of rainfall. Having more data is also important because then we can more fully educate the public on hydrological information and its importance. Having the public understand why we need to continue collecting this type of data is important, especially when it comes to our solution.

Dr. Benjamin Ruddell is our sponsor for this project. He specializes in the water and climate sector with a focus on hydrology and data collection. He has noticed the lack of data points on smaller rivers and waterways and so with a colleague he created the idea for a citizen science web application for hydrology reporting. The idea is to get the community involved in the data collection process.

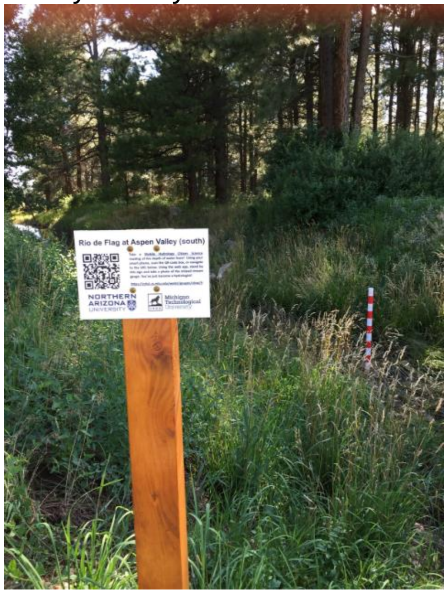

Rio de Flag at Aspen Valley (south)
NORTHERN ARIZONA UNIVERSITY
Michigan Technological University

Figure 1.1 *- Our gauging solution*

Our solution to this lack of data is to develop a mobile application that allows users to take pictures of a gauge which, unlike USGS's expensive equipment, will be a PVC pipe with red and white stripes as seen in Figure 1.1. There will also be a wooden post at each gauging location.

The wooden post will have a QR code attached that can be used to uniquely identify the site and to provide more information to passersby. Once the user has taken a photo of the PVC pipe, our image processing algorithm will add a horizontal line to where it believes the water height is in the photo. The user may adjust this line to get a more accurate water level.

Figure 1.2 is showcasing how the application will basically work. The user will submit a photo on their phone which will then be sent to a server and local database to be processed.
We will use the user's phone's geolocation in order to plot exactly where the photo has been taken and this location will be stored with the photo upon submission. From there we will store the image, location, and water height data in our local database. We will then take the water height data and send this to our local HydroServer and then it will be joined with the main national HydroServer which is where a lot of the national hydrological data is stored. Once the user has uploaded their information, we will give them feedback on their submission, which may include previous data points collected at that location or a graph showing how their contribution helps this data crisis. One key feature that separates our application from anything else will be the offline capabilities. Some of these gauges and waterways will be in areas without cell service or internet and this is where our application comes in. The user will be able to take a photo and store the information that goes along with the photo on our application and will be ready for submission when they are back in an area with coverage. The user should still be able to receive some confirmation that would have been stored on the application just for this.

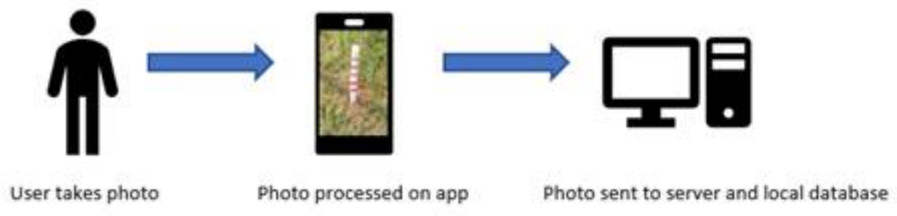


Figure 1.2 *– Data Flow Chart*

2. Implementation Overview

When looking at our solution vision, we are going to be creating a mobile application that allows users to quickly and efficiently collect water height data from already setup gauge stations. The approach we are going to take to achieve this begins with a Meteor application. From this we are going to attach a database chain to end up federating the data with the national HydroServer database. In between the national HydroServer and our application will be a database that holds the images and user notes taken for a specific reading along with the water height data.

Within the application we are going to use several packages. The first is Charts.js. This package is a data visualization tool that allows you to take data and plot it in a graphical manner that is easier to digest than plain text. This will be used to create graphs for the user to see from both collected data and data predictions from the National Water Model. The second package is the UploadFS package. This allows for data and image submissions into a mongo collection as well as storing images as a flat file. This will be used for our intermediate database that will then send the information to our local HydroServer database. Next is mdg:geolocation and dburles:google-maps. These are used in conjunction to be able to use a map graph to show the user where gauges are located as well as to give the user notifications when they are near a gauge is they aren't looking at the map at that moment. The geolocation is also used to store this information in the picture that the user takes to create more descriptive names for the images. Finally, is OpenCV. This will be our computer vision package that will allow us to plot a prediction for where the water height is on the image. This will be what we use as a baseline for the user submitted data before they adjust it if necessary.

3. Architectural Overview

Figure 3.1 represents the architecture diagram for our project, which is a visual representation of the high-level architecture of our system. Our system consists of four main components, some of which can be broken down further:

- Mobile Application
  - Notification Manager
  - Data Visualizer
  - Cache Manager
  - Image Processor
  - Phone Hardware
- Web Server
  - User Accounts
- Database
- HydroServer

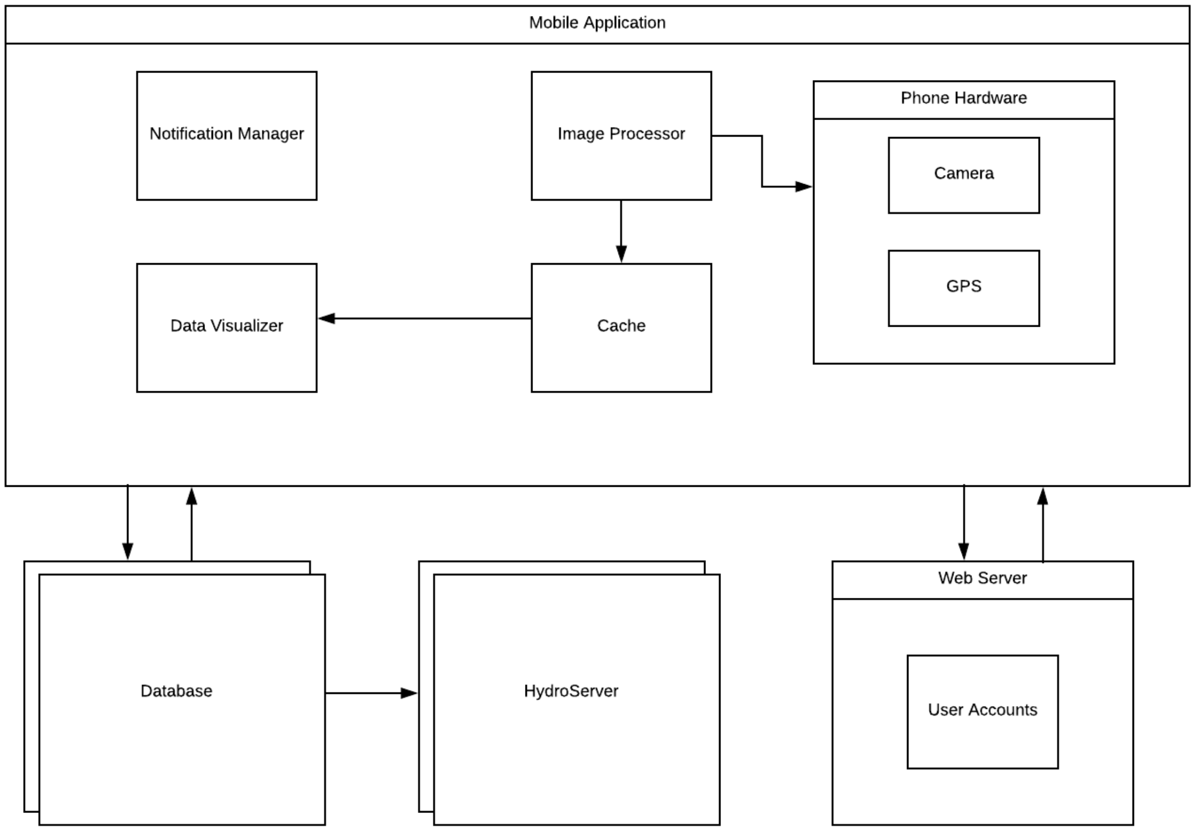


Figure 3.1 *– Architectural Diagram*

The mobile application will act as the interface through which users record measurements from water gauges, view hydro-graphs relevant to their submitted data or location, locate water gauges and personalize their notification preferences. The mobile application is also responsible for storing data to the phone's local storage, in the form of caching, to account for when the user is not connected to the internet. This data includes measurements taken from water gauges and other water data that can be displayed to the user in the form of graphs through the data visualizer. Next we will describe the key responsibilities and features of each component:

Database
The database is responsible for storing the images submitted by the user, as well as any data collected from the image. The database must also periodically submit hydrological data (water height, longitude and latitude, site ID, time of measurement) to the HydroServer.

HydroServer
The HydroServer is responsible for federating data to the CUAHSI water model, and accepts data in an XML format.

Web Server
The web server is responsible for allowing users to log into their account while using the mobile application. When taking measurements while logged into an account, those measurements will be uniquely associated with that account so that users can view past submissions.

Notification Manager
There are three types of notifications that users will receive while using the mobile application:

- Location-based
- Weather-based
- Time-based

Users will be able to adjust the frequency of each type of notification. By allowing the user to customize each notification to their personal needs, we hope to retain user interest in the mobile application and maintain a source of data gathering.

Data Visualizer
The data visualizer will be a feature within the application that users can access, which will plot user-collected data against data reported by national water models and weather services. This component must be able to communicate with the project database and the cache manager in order to retrieve user-collected data.

Cache Manager
The cache manager is responsible for managing the offline functionality of the mobile application. It must both hold data from measurements taken from water gauges before it is submitted to the database, and it must also store water data to be used by the data visualizer.

Image Processor

The image processor will act as the interface between the user and the algorithms responsible for extracting data from photos taken of water gauges. The flow of data through the image processor is as follows:

- User opens the image processor
    - Phone location is stored
- User takes a photo of a water gauge
    - Phone time is recorded
    - User verifies that the image is of reasonable quality
- Algorithms measure the water line on the pole of the water gauge
- Image Processor asks the user to confirm the location of the water line
    - User is given the option to adjust the measurements
    - Measurements are adjusted based on user input
- Data is stored to the phone’s cache:
    - Latitude and longitude
    - Phone time
    - Site ID
    - Photo of site

4. Module and Interface Descriptions

Frontend

Profile Management

This module is responsible for allowing the user to have an account if they wish and this account will allow them to view all submissions as well as control their notification settings. This fits in the architecture by connecting to the User Accounts and the database that will hold all of the settings for each user's profile.

As seen in Figure 4.1, the important functions in the profile management section are in the Settings class. The Settings class will allow access to the Notifications class and the Profile class. In the Notifications class the user will be able to turn their notifications on or off. In the Profile class the user will be able to edit what their name is.

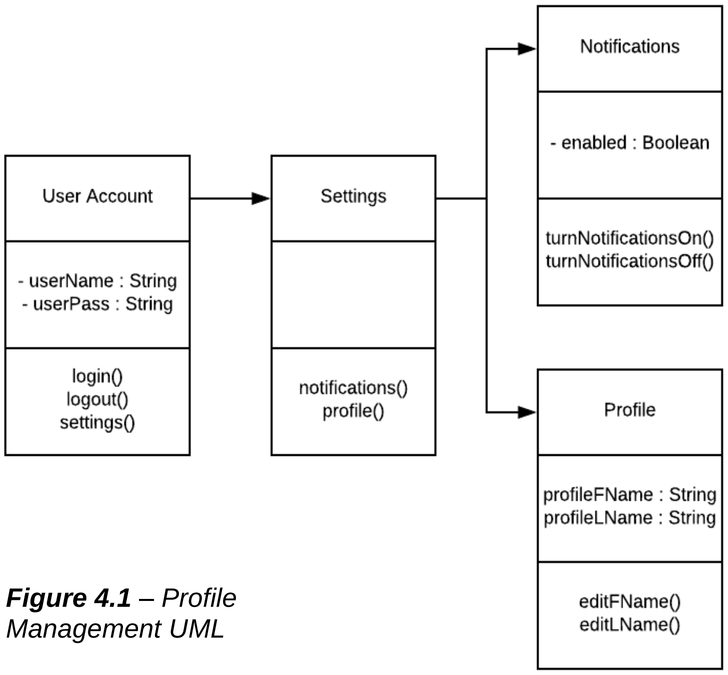
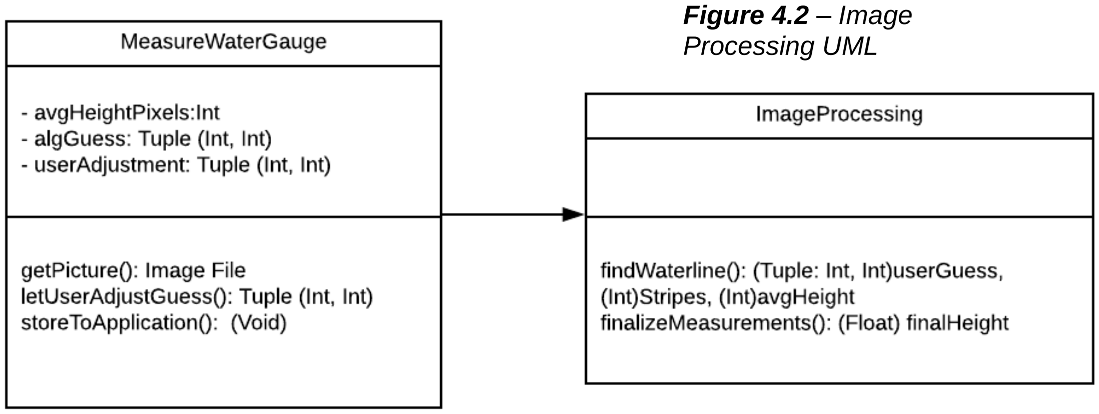


Figure 4.1 *– Profile Management UML*

Computer Vision

The Computer Vision module is responsible for allowing the user to take a picture of a water gauge, returning to the user an estimate of where the water line is on the striped pole, allowing the user to adjust the algorithms' estimate of where the water line is, and finalizing the results before storing them on the phone. Figure 4.2 outlines the UML diagram for this module.

Figure 4.2 *– Image Processing UML*



MeasureWaterGauge
- avgHeightPixels:Int
- algGuess: Tuple (Int, Int)
- userAdjustment: Tuple (Int, Int)
getPicture(): Image File
letUserAdjustGuess(): Tuple (Int, Int)
storeToApplication(): (Void)
ImageProcessing
findWaterline(): (Tuple: Int, Int)userGuess, (Int)Stripes, (Int)avgHeight
finalizeMeasurements(): (Float) finalHeight

The following classes detail the above-described process:

- measureWaterGauge()
    - getPicture()Allows the user to take a picture of the water gauge and returns the image.
    - letUserAdjustGuess()
    - Input: Tuple containing the locations of the top and bottom of the visible striped pole.
    - Process: Allows the user to use their finger to adjust lines on the screen that represent where the top and bottom of the visible pole is.
    - Output: Tuple containing the user-adjusted measurements of the top and bottom of the visible pole.
- storeToApplication()
    - Save to file:
    - Picture of the gauge
    - Site ID of the gauge
    - Water level measurement from the gauge
    - Time of the measurement according to the phone
- ImageProcessing()
    - findWaterline()
        - Input: Image of the pole from the site water gauge
        - Process: This function will analyze the image to determine where the top and bottom of the visible pole is. It will also count the stripes that are found within the image.
        - Output:Tuple containing the location of the top and bottom of the pole in pixels, an integer representing the number of stripes that were found in the image, and the average height of each stripe, measured in Pixels.
    - finalizeMeasurements()
        - Input: Two Tuples containing the location of the top and bottom of the visible area of the pole in pixels (One Tuple representing the guess from “findWaterline()” and one Tuple representing the new locations from “letUserAdjustGuess()”), and an integer representing the average height of each stripe, measured in pixels
        - Process: Uses the ratio between the the two measurements to determine the real-world height of the visible pole.
        - Output: Final distance, in feet/inches, between the top and bottom of the visible pole.

Notifications

The responsibility of the notifications is to notify users of certain event is approaching. For this project, users will be notified when they are close to a water station. In addition, users will be notified when a water station reading is necessary.

Notifications will be send to users that have an Android device. Once the user is logged into their profile, they will have the ability to receive notifications. Guest accounts will not be receiving these notifications.
Figure 4.3 demonstrates the rough layout of how the user will receive notifications on their mobile device. First the Meteor's server will ask for a project number and a server key that will call Google's Firebase Cloud Messaging (FCM). FCM is a server that is used for sending messages and notifications in an Android, iOS, or a web application. When the user is close to the water station or the user has not done a reading for a water station, Meteor will make a call from their server to the Firebase Cloud. However, the user's phone might be idling or turned off. If the phone is turned off, the Cloud message will send the notification as soon as the mobile device is turned on.

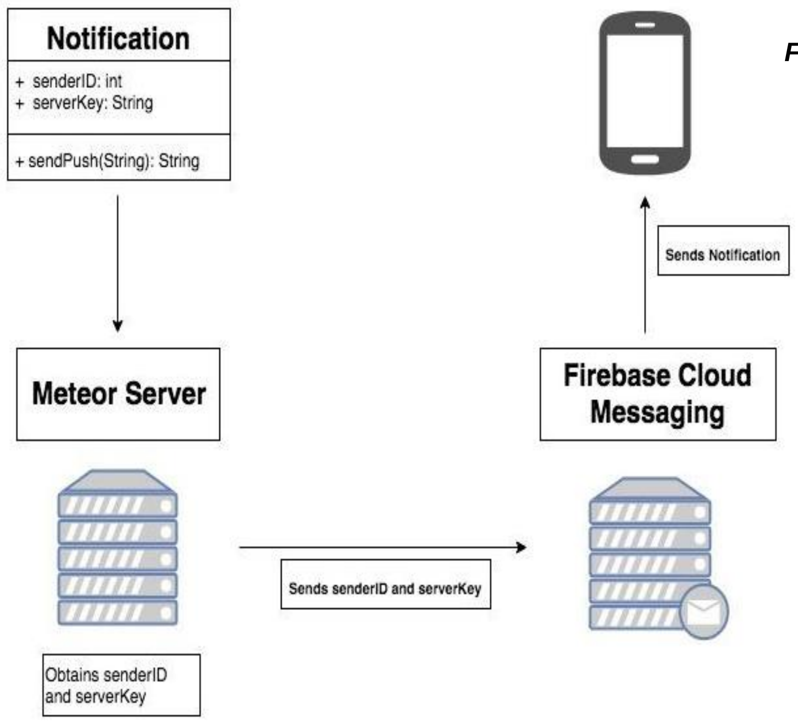


Figure 4.3 *– Notification UML*

Data Visualization

This module is responsible for creating a visual representation of the data that is being collected as well as data that is being generated by other places such as the National Water Model. This fits in the architecture by connecting to the databases that hold our water data and then taking that data and representing it in a graphical manner.

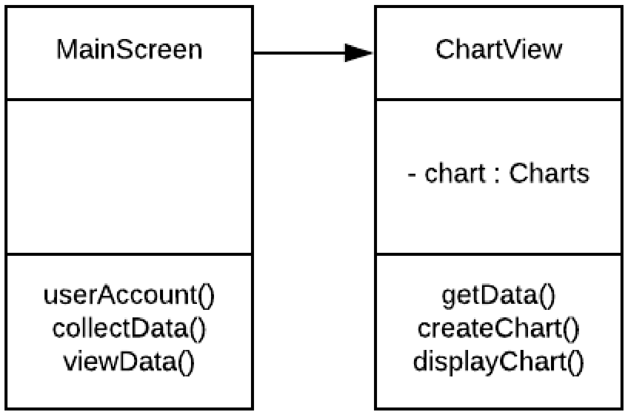


Figure 4.4 *– Data Visualization UML*

As seen in Figure 4.4, the important functions in the data visualization section are in the ChartView class. This is what will be used to generate and display the graphical representation of the data that has been collected. This will be connected to the MainScreen class which is what will have the options for the user to see their account, to collect data from a gauge, and to view data. Within the ChartView class the getData() function is used to pull data from our HydroServer or other sources. Once this data is collected, the createChart() function is used to actually create the graph. Once the graph is created, the displayChart() function will be used to put the graph on the screen.

Backend

Offline-Caching

This functional module is responsible for allowing the user to still receive some visualization and information even if they are offline. This module is important because users may want to take photos and upload data to the application in a remote area with no service or internet. This fits in the architecture by caching relevant data in the database to be shown to the user, even when offline. Once the data has been cached, if the user is offline the showData() function will be used to display data to the user. The getData() function will grab the information from the National Water Model. Figure 4.5 is a UML showing these functions and how they will interact. We will be needing a data variable which we currently have set to be some sort of list to hold the data to be displayed.

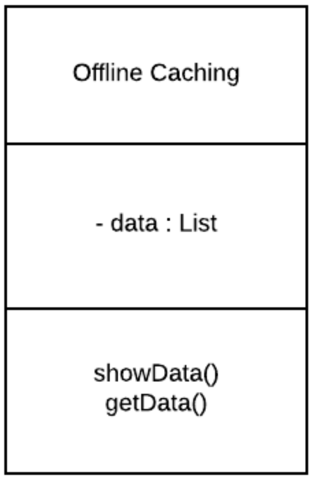


Figure 4.5 *– Offline-Caching UML*

User Accounts

This module is responsible for allowing users to log into their Mobile Hydrology account through the mobile application. This will save their user ID to each submission so that they can use the website in order to view data regarding their past submissions.

Geolocation

The responsibility of the geolocation API is to allow all users to view Google's map. In addition, it is able to be determine the user's position in the map and place a marker.

The geolocation map will let all users with the API key view the map, which is provided by Google. Once the user clicks to view the map, they will be prompted to allow finding their location. If permission is allowed, it will calculate the distance from the user's position to each of the site station. If the permission is denied, it will just show the site stations. Another feature that Google Maps has is being able to place markers. When the user presses somewhere on the map, it will send the id, the longitude, and latitude to the backend Mongo database for storage. The user is also allowed to drag a marker and remove a marker from the database.   However, only water gauge managers and system administrators are allowed to place markers.

Figure 4.6 demonstrates the rough layout of how Google Maps API will function in our mobile app. First the Meteor package will require an API key that will call the Google Maps API. Once a user opens the template, it will send the API key to the Google Maps API and it will give you the map. After that, it will prompt you with a permission to access the geolocation. If permission is allowed, it will provide the user's coordinates and place a marker of their position on the map. If permission is denied, the user will only view the station markers on the map without knowing where they are. Google Maps API also

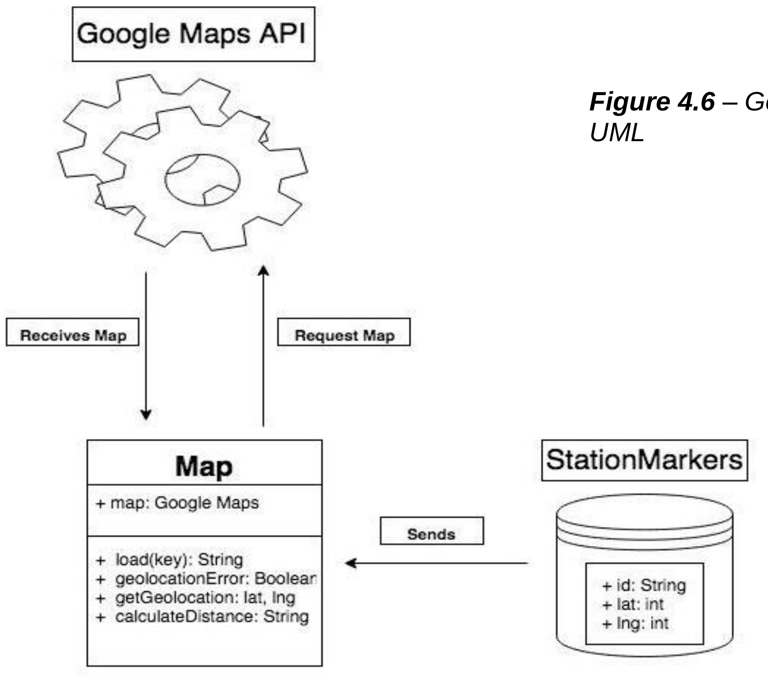


Figure 4.6 *– Geolocation UML*

involves Mongo Database backend because water gauge managers and system administrators need to place water station markers. The database will take in the id of the station, the latitude, and longitude. As markers are added or removed, the database will keep updating them in the mobile device.

Database

This functional model is responsible for storing the users' data on an uploaded image. This database will store relevant information such as water height which is taken after the image processing, gauge location which will implement geolocation and other comments a user wishes to say about the gauge. This module is important because it is storing all of our information on a specific gauge. The database must be connected to our Hydro Server so we can submit this data to a national Hydro Server. The important functions in the database section is the storing of the user submissions as well as the connection to the HydroServer. Storing will be done with an insert() function which will store the data into a database collection this will be done within submit. Connecting to the HydroServer will involve taking the data from the database and transforming this into a csv file then pushing that file to the HydroServer. This will be done with a insertInCSV(). See Figure 4.7 below for a UML class diagram of our database module.

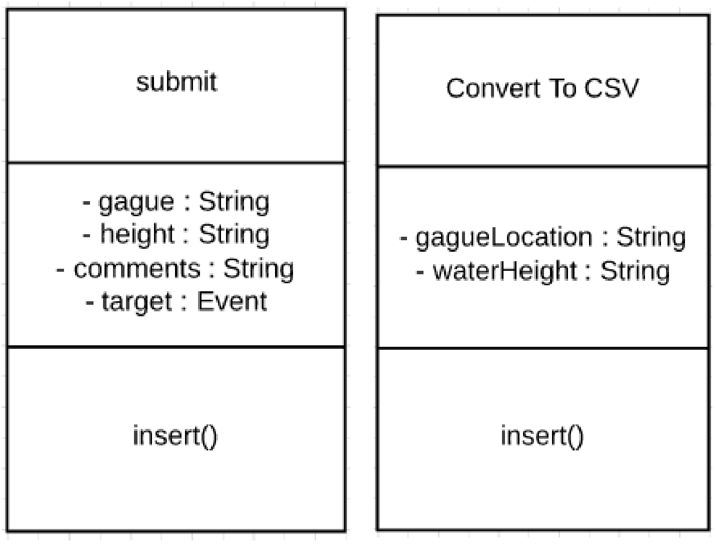


Figure 4.7 *– Database UML*

5. Implementation Plan

In this section of the document, we will describe our plan to implement the minor key requirements into the application. The Figure 5.1 demonstrates our plan to implement the milestones for our project. The main milestones that we have are:

- Geolocation
- Connect and Access the HydroServer
- Data Visualization
- Update Team Website
- Offline Capacity
- Image and User Info Stored
- User Management System
- Notification/Gamification
- Computer Vision
- App Compilation

Each team member will have specific milestones to complete and have a rough prototype with all the functionalities of our mobile application. Our rough prototype should be completed by the by mid-March.

During the month of February to mid-March, we have been making some constant progress. We have been able to demonstrate Geolocation and Image Stored on the web portal. In addition, we have started to implement App Compilation from a web portal to a mobile device.

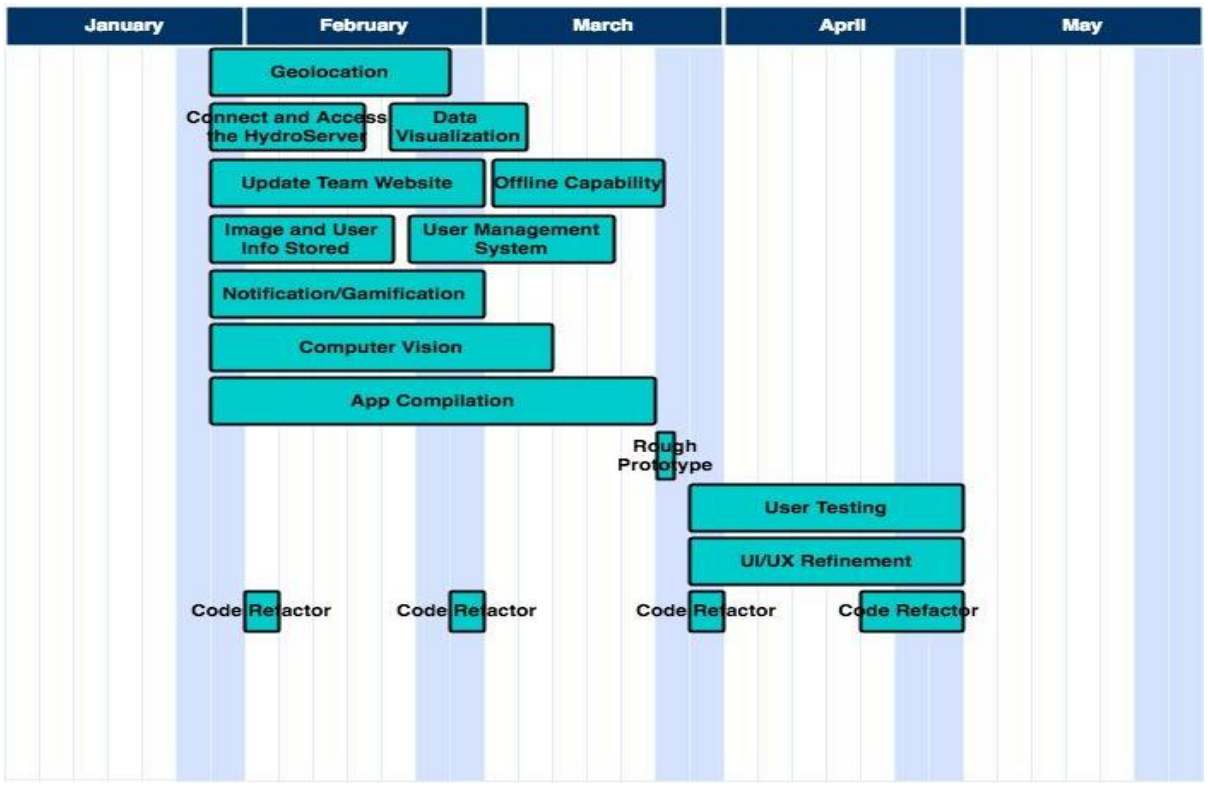


***Figure 5.1** – Schedule Gantt Chart*

As of today, we are still researching on how to connect from the HydroServer database to our mobile application in order to send and federate hydrological data. Once we successfully able connect to the HydroServer, we can then start implementing data visualization about each water station using their database. In addition, we are in the process of finding ways for users to be interactive in the application, such as sending some form of notification on their phone. We are also working on the computer vision algorithm that allows the user to adjust the water height of a water gage when the picture is taken. All of these features need to be successful when the when the user is offline and online. As we achieve these milestones, we will start compiling it into the mobile application. Once these features compile successfully, we can then demonstrate a working demo by mid-March. Once we are done with our rough prototype, we will then do user testing for a month. User testing involves gaining feedback, UI/UX refinement from the rough prototype, and bug fixing.

By the end of May, we will then be preparing to have a stable Beta version of the mobile application.

6. Conclusion

Flooding is the most destructive natural disaster that we have, costing the United States $20 billion and resulting is at least 75 deaths since 2016. As USGS loses funding for their stream gauges, we must come up with a way to continue the collection of hydrological data of the large waterways that USGS collects data on as well as smaller waterways they do not collect data on. This extreme can be prevented with the continued and increased amount of hydrologic data collection. This is where our project comes in. With the Citizens' Science Mobile Application for Hydrology Reporting, we will be able to collect more data on smaller waterways and provide the people who live in these areas with better warnings in the event of a flood. The idea is get the community more involved with this data collection through user submitted images of our gauging stations to our application. This will in turn help the public to understand the importance of this data and how it could affect them. This requirements document will help our team to decide what the most important features of our application will be. We have now outlined the key requirements and will strive to complete each and every one by the end of the year. We will use these requirements to better develop a prototype for our technical demonstration showing key aspects of the project. As a team, we are confident that we will be able to fulfill all of the requirements listed throughout this document and end the year with a fully functioning application that will be able to provide more accurate data points that could help save lives and money.