

Technological Feasibility

October 26, 2017

**Sponsor:**

Dr. Benjamin L. Ruddell

**Team Mentor:**

Dr. Eck Doerry

**Team Members:**

Luis Arroyo

Logan Brewer

Ryan Ladwig

Kelli Ruddy

[**Introduction**](#_uqrg741bagze)2

[**Technological Challenges**](#_rlz63827v4o3)3

[**Technical Analysis**](#_i1yb1h2x8r4t)3

[App Development Framework](#_3saxvosnhqbj) 3

[Database Management and Design](#_3mccmymc363d) 6

[Computer Vision Framework](#_xpqh5relgmp8) 8

[Data Visualization](#_ubxz3hbgyj8c) 10

**Technology Integration** 12

[**Conclusion**](#_wz8agkrc0y0d)13

# Introduction

We are the Hydro Citizens and our team members are: Logan Brewer, Kelli Ruddy, Luis Arroyo, and Ryan Ladwig. Our mentors are Dr. Eck Doerry and Dr. Benjamin Ruddell, who is also our sponsor. Our capstone project focuses on the collection of hydrological data. A lot of the hydrological data collected in the United States is from large bodies of water and not a lot of data is included for smaller waterways or waterways that do not flow year-round. The more data we have the better we can predict flood and drought severity, water availability, and water quality. Our solution to this lack of data is to develop a mobile application that allows users to take pictures of a wooden post and red-/white-striped PVC pipe and then upload the water height and the picture to a database. We will use their phone’s geolocation in order to plot exactly where the photo has been taken. The post will have a QR code attached that can be used to uniquely identify this site and to provide more information to passersby. We will be using image processing to draw a line on the photo that the user takes, and they will be able to adjust the algorithm’s calculation to get a more accurate water level. 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 from there it will be joined with the main national HydroServer. Once the user has uploaded their information, we will give them feedback on their submission, which may include the last time it rained or previous data points collected at that location.

We will begin with the Technological Challenges section by outlining all of the major technological challenges and needs that we will be facing while developing our project. In the Technology Analysis section, we will analyze methods for resolving these challenges, decide on a solution to each of our challenges, and explain our rationale for choosing our solutions. Finally, in our Technology Integration section, we will describe how all of our technologies work together and how they will resolve our technological challenges and needs.

# Technological Challenges

The purpose of this section is to list and describe the technological challenges that we expect to encounter during this project.

We came up with four major challenges for this project.

* We will need a platform that allows us to create this application, allowing a smooth connection to the database.
* We will need a way to store large quantities of data and be able to access this data quickly.
* We will need to be able to use an image processing tool that connects with our application and allows us to plot a line indicating the water level.
* We will need to be able to use a data visualization tool in order to present our collected data as well as other data in a more visually pleasing manner than simple text data.

# Technical Analysis

The purpose of this section is to fully describe and analyze each technological challenge for our project. We will then present alternatives and determine a solution that will work best in the context of our project.

## App Development Framework

**Introduction**

Finding the best app development framework to suit our needs is pivotal to our success when developing this project. The framework that we choose for this issue will need to support a stable connection to the database, efficient image processing, and data visualization. The application should allow the user take a picture of a water gage and adjust a line of the water level. The user will be given the option, and encouraged, to write about their observations at each site such as the weather condition or extreme water levels. Once they have completed their submission, the application will send the image and the information they wrote to a local database. The picture and any special observations will stay in the local database, but the water level data will be transferred to the Hydroserver. If the user does not have internet connection, the information the user collected should be stored on the device until they gain access to a stable internet connection. The user is also able to see where they are located on the map and if there are any gages close by.

**Alternatives:**

The possible mobile application development framework options that will help us implement a database, image processing, and data visualization are Android Studio, Meteor, and Visual Studio. When analyzing each alternative, we are looking at which applications could help us implement these features easily, which ones would allow for the potential of cross platform development, and what languages each alternative uses. We will compare each of the following alternatives and determine which one we will use for our project based on their capabilities and known compatibility.

* Android Studio - An IDE platform that provides tools for building apps on any type of Android device. The mobile applications are developed in Java and XML programming languages. Android Studio also allows the inclusion of APIs which allows us to implement the features needed for this capstone. With these APIs, we are able to connect to a database, integrate image processing using OpenCV, and data visualization. However, the problem with these APIs are that they only target a certain Android version, which can lead to conflicts using the APIs. Another problem is that if we want to work with iOS, then this would require rewriting the code using the Swift programming language. The last issue is that Android Studio mostly supports the local databases. If multiple users want to access the same database, it can lead to conflicts such as the database being corrupted.
* Meteor is a JavaScript web framework that is used to work with web and cross-platform mobile applications. Meteor integrates APIs from Angular, React Native, MongoDB, npm, and Cordova. With the APIs that it provides, we can manage to integrate most of features we are required to implement. With these APIs, we can connect to a database using MongoDB, integrate image processing using OpenCV, and data visualization using D3.js. However, the disadvantages of this framework is that if the local database continues to expand, the application will start to become slow when writing and accessing a table. The phone itself will also have a high CPU usage, high memory usage, and a slow performance.
* Visual Studio is a cross-platform IDE that is used to develop web, mobile, and desktop applications. The mobile applications are developed in C# and F# languages. Visual studio integrates APIs from Xamarin and React Native. With these APIs, we can connect to the database using SQLite and SQL Server database management systems. We can also integrate image processing and data visualization from Xamarin. However, the disadvantages of using Visual Studio is that we are very limited in how much coding we are allowed outside of this framework. When developing a new application, it would take 2GB of space without implementing anything on the app. If we use the API’s, they would take more space into the application. Visual Studio is considered a cross-platform, however some of the code would be required to be rewritten from the front-end and back-end of the application, leading to time consuming.

**Chosen Approach:**

Based on the research and looking at the challenges that we will be facing with each app framework, the best option is to use Meteor. Meteor is the best option because it will allow us to implement most of the features required for this capstone project in an easy and efficient way. The table shown below covers Meteor being the best option for app development framework, and comparing the other frameworks.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Android Studio | Meteor | Visual Studio |
| Image Processing | 4 | 4 | 2 |
| Database | 4 | 3 | 3 |
| Data Visualization | 5 | 4 | 3 |
| Cross-Platform | 1 | 4 | 4 |

**Proving Feasibility:**

Meteor offers tutorials on YouTube on how to implement some of the features that would be necessary for this capstone project. If we want to know more about a method and how it is used, Meteor also offers an API Documentation Guide that states about how to use a package and what functions it includes. We can use APIs that the platform offers in order to implement the requirements. For our demonstration we will be installing the Meteor Development Framework and implement basic features that are required for this capstone. We plan on improving the application next semester by implementing a better user interface app and improve the existing required features.

## Database Management and Design

**Introduction:**

This is important for our application because we need to be able to store a large amount of images and data points that are collected by our user. We may also need to store information from the National Weather Service to provide to the users as feedback. Our database needs to be scalable to account for user submissions of information and photos. We also need connection reliability in order to be able to access a user's submitted data whenever requested. Our database needs to access this submitted data in a timely and efficient manner.

**Alternatives**

Options to solve our database problem are MySQL, MongoDB and Apache Cassandra. When looking at each option, we made sure to look at the scalability, connection reliability and how quickly it accesses data. Each of these topics we looked into are the basis for how our application will store data. We chose these three options after looking into many other database options that did not fit specific criteria. These options were the closest to what was needed for this project and in this document we will more fully investigate which is the best option.

* MySQL - MySQL is an open source relational database management system so data is not just stored in one big table. It is also advertised as fast, reliable and scalable, which tackle each of our main concerns with a database. MySQL has a PHPMyAdmin built in support API which would be helpful for our project, it works well with geographical information systems and has great data security. It is also a good database if creating a web application. Some cons of using MySQL are you need multiple add one, it is not good with storing qualitative data such as images which is a key part of our project and frequent updates are required for large amounts of data. One of the biggest drawbacks to MySQL for this specific project is the storing of images. MySQL does not handle the storing of images very well.
* MongoDB - MongoDB is an open source and NoSQL database. MongoDB uses collections and documents rather than rows and tables. It is a fairly new solution to the database problem emerging in the early to mid 2000’s. It utilizes something called BSON which is just a binary representation of JSON documents. MongoDB offers dynamic schemas, is accessible from many devices, works very well with large amounts of data and has in memory speed. To store images, MongoDB utilizes GridFS which will allow for the storage of user submitted images. Since this is a newer product, there is less information and support. It also scales horizontally which is something our team had first discussed wanting vertical scaling.

* Apache Cassandra - Apache Cassandra is an open source NoSQL database and best with unstructured data. Similar to MongoDB, it has dynamic schemas and can handle large amounts of data.Given the large amounts of data, Apache Cassandra is very reliable. It also offers something called CQL which is similar to SQL. It can have some unpredictable performance.

**Chosen Approach**

After some discussion we realized we decided a NoSQL option would be the best because of how NoSQL supports more dynamic data. Our final decision was between MongoDB and Apache Cassandra. Based off of the research done and looking at the challenges we will need to face with this database for the application, we have chosen to work with MongoDB. Outlined below is a table showing that MongoDB covers most of the desired capabilities of a database. Storing images is a huge part of our application and the fact that MongoDB offers GridFS to do this was a big plus. Both MongoDB and Apache Cassandra were good for storing large amounts of data. MongoDB outshined Apache Cassandra in that MongoDB is great for real-time analytics and image storing.

|  |  |  |  |
| --- | --- | --- | --- |
|  | MySQL | MongoDB | Apache Cassandra |
| Scalability | 5 | 5 | 5 |
| Connection Reliability | 3 | 5 | 5 |
| Data Lookup Efficiency | 4 | 4 | 4 |
| Device Accessibility | 5 | 5 | 5 |
| Handles large data amount | 3 | 5 | 5 |

**Proving Feasibility**

Based on the pros and cons of each of the researched database frameworks, MongoDB is the best solution. Our main concerns with a database are that it can store images, that it is scalable, and that it has connection reliability. We will further develop this by testing many images and data points to analyze how it works under these conditions. We will also test by continuing to add and delete data.

## Computer Vision Framework

**Introduction**

For our project we will need to be able recognize the gage height of a waterway from a picture of a red and white pole that is planted in a stream bed. For this process we will need to be able to run a computer vision framework natively on both iOS and Android devices, and not require internet access. The algorithm that we will need for this project must be able to both recognize the waterline on the pole, and be able to determine the height of the waterline using the known dimensions of the pole. The image processing must run within a reasonable amount of time and not require a large amount of resources from the user’s device.

**Alternatives**

Possible alternatives for our Computer Vision Framework include Tracking JS, which is a relatively new system which runs with JavaScript and and is primarily used for facial tracking and recognition, OpenCV, which is a complete computer vision framework that is written in C/C++ and has wrappers for Python and Java, and finally JS Feat which is another CV framework written in Javascript.

* Tracking JS - Tracking JS is a Javascript package for computer vision that would allow us to run our algorithm(s) in Javascript, which would aid in the cross-platform compatibility of our project. There are a few tutorials on how to use Tracking JS, but we would need to spend a reasonable amount of time determining how we can learn Tracking JS from existing projects and examples. Tutorials for Tracking JS include color recognition and feature recognition.
* OpenCV - OpenCV is a very popular computer vision framework with many examples and projects, that can be used as a reference, readily available on the internet. This would allow us to jump into programming with OpenCV without already having extensive knowledge about Computer Vision. There are a few projects, that use OpenCV, on the web that bear strong similarities to the computer vision framework requirements that we have to fulfill (that we will need for our project).
* JSFeat - Was recommended by several resources that dealt with image processing and computer vision. Was unable to find an extensive collection of examples or tutorials for JSFeat, meaning that we would likely need to have a deep understanding of computer vision if we were to use this framework. Because there are not very many tutorials for using JSFeat, we would need to extensively research the ins and outs of computer vision to understand the capabilities of JSFeat.

**Chosen Approach**

For this project, we will be moving forward with using OpenCV as our computer vision framework. While all of the aforementioned frameworks would be suitable for our project, the flexibility of OpenCV (JS, Java, C++/C, Python), along with its widespread-use and available examples/tutorials, is what drove us to choose it as our computer vision framework. Once we gained access to the client’s existing project, we realized that it also uses OpenCV as its Computer Vision Framework, so we feel comfortable with our choice.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Tracking JS | OpenCV | Feat JS |
| Offline | ✔ | ✔ | ✔ |
| JS | ✔ | ✔  (With reduced functionality) | ✔ |
| Python |  | ✔ |  |
| Android | ✔ | ✔ | ✔ |
| iOS | ✔ | ✔ | ✔ |
| Full Computer Vision Support (1-5) | 4 | 5 | 5 |
| Support / Tutorials  (1-5) | 3 | 5 | 1 |

**Proving Feasibility**

Stanford university offers a course on Computer Vision Frameworks / Image Processing and has posted all of their tutorials and projects online. The assignments in the course use OpenCV and one of the chapters deals with using OpenCV with Android. The projects that caught my attention deal with shape and color recognition, but we can also use these tutorials when we try to run OpenCV on Android, as the course website includes instructions on how to eliminate the need to have the device connect to a server. For our demonstration, we will be installing OpenCV on an Android device and implementing the existing algorithms on that device. We plan on improving the algorithm in the coming semester, but we will be using the existing algorithm for our demonstrations this semester.

## Data Visualization

**Introduction**

Data visualization is important because we need to present the data in a way that is visually pleasing and easier to digest than simple text data. We need this to be scalable for massive quantities of data, it needs to be highly customizable in order to create the charts exactly the way we want, and it needs to be able to be implemented with our app development tool.

**Technical Analysis**

We are going to be taking hydrologic data and displaying it in different ways that are easier for the user to see and understand, such as interactive scatter plots. Because of this reason, data visualization is very necessary for our final product. This is going to be used to display the information in a way that isn’t simple text. We need to take all of the data we will be using and collecting and display it in a more visually pleasing manner. Some possible alternatives are D3.js, Google Charts, and Chart.js.

* D3.js - Some pros are that it is highly customizable and is able to handle massive quantities of data. Another is that it is rendered as an SVG which allows interaction with every element of the chart. For cons, there is one main one and it is that D3 is very difficult to learn and is very complex to code. Every element and part of the charts needs to be coded, resulting in 100+ lines of code for a simple bar chart.
* Google Charts - Some pros are that it is easy to learn, it is interactive, it works on all modern browsers, it includes maps through GeoChart, and it can read data from several different data sources. Some disadvantages of Google Charts are that it isn’t as customizable as something like D3, it doesn’t give complex statistical processing, and it requires an internet connection in order to use.
* Chart.js - The biggest advantage is that it is very simple to use, learn and implement, and it comes with a lot of detailed documentation. The cons are that it is canvas based instead of SVG so the charts cannot be interacted with, and although it is very simple, this comes along with it only allowing simple charts to be used.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **D3.js** | **Google Charts** | **Chart.js** |
| **Charts rendered in** | SVG | HTML5 using SVG and VML | canvas |
| **Input data format** | JSON and XML | JavaScript API | JavaScript API |
| **Browsers supported** | All modern web and mobile browsers | All modern web and mobile browsers | All modern web and mobile browsers |
| **Chart and map types** | No pre built charts | 13 2D charts, maps available | 6 chart types |
| **Licensing** | BSD-3 | Free for all usage | Free under MIT license |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **D3.js** | **Google Charts** | **Chart.js** |
| **Customizability** | 5 | 3 | 1 |
| **Scalability** | 5 | 3 | 1 |
| **Ease of use** | 1 | 4 | 5 |

**Chosen Approach**

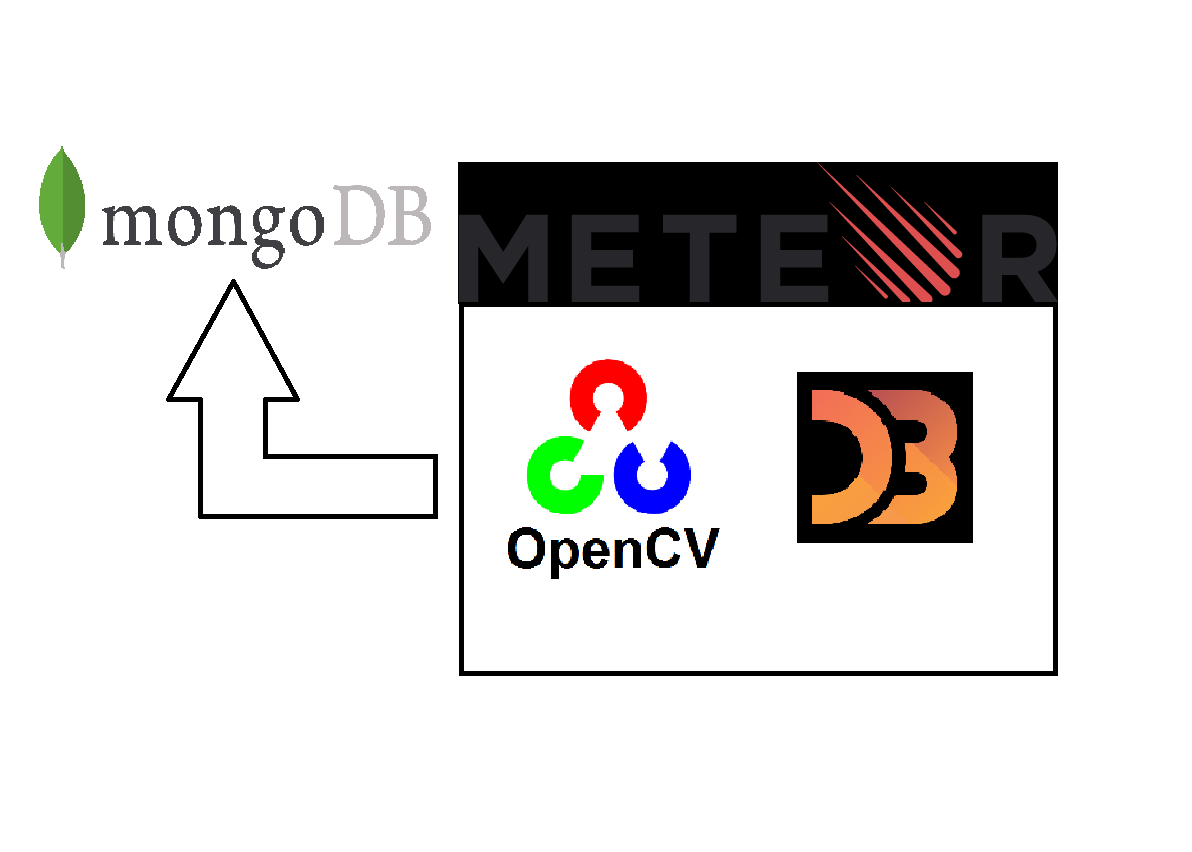
When looking at all of the pros and cons, we will be using D3.js. This decision is based on the customizability and interactivity that is available in D3. We don’t want to be limited in our possibilities by choosing a tool that gives less flexibility and options. Google charts was a close second but the requirement to be online in order to get charts, as well as the limited amount of charts available to use, were a heavy deterrent and lead to the choice to use D3. We also do not believe that the learning curve for D3 will be too much of an issue and the customizability and scalability outweigh this one con.

**Proving Feasibility**

For proving feasibility, we will be taking different chunks of data and creating a wide range of charts that will show our ability to use D3 anyway that we want to. We will also be directly testing charting example data that would be gathered from our app in order to show possible direct applications. Finally we will be implementing this in our app development tool and see how that interaction works within the app we are creating for our demo.

# Technology Integration

Now that we have seen all of our solutions, we will be doing our best to combine them. The Meteor development tool will be our base for our app. In connection with this, we will be attaching a MongoDB database to the app in order to solve our storage issue. We will be using D3.js in order to display the data we store in our database in a more meaningful manner than plain text. Finally we will be using OpenCV in connection to Meteor for our computer vision framework. Part of our testing for each individual part will be to see how they will come together and interact. Below you can see our system diagram, showing D3 and OpenCV running within our Meteor application, and MongoDB connecting to our Meteor application from the outside.



# 

# Conclusion

The collection of hydrological data is important when it comes to flood prevention, water quality and public education and knowledge. In order to have more accurate data, we must collect more. Through our application we hope to get the public more involved with this data collection. Through this document, we hope to have fully discussed every technological issue and analyze in order to decide on which technologies will be best suited for our problems.

|  |  |
| --- | --- |
| **Meteor** | **Confidence Level** |
| Image Processing | 4 |
| Database Connection | 3 |
| Data Visualization | 4 |
| Cross Platform | 4 |

|  |  |
| --- | --- |
| **MongoDB** | **Confidence Level** |
| Scalability | 5 |
| Connection Reliability | 3 |
| Image Storage | 5 |

|  |  |
| --- | --- |
| **D3.js** | **Confidence Level** |
| Customizability | 5 |
| Scalability | 5 |
| Ease of use | 1 |

|  |  |
| --- | --- |
| **OpenCV** | **Confidence Level** |
| Image Recognition | 5 |
| Image Processing | 4 |
| Easy to learn | 5 |

We plan on testing these more fully as noted in the proving feasibility section of each technological issue analysis. After much discussion and analysis, as a team we have decided these four technologies will best support our problems we will face with our project this year.