

**SANTA CLARA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Date: June 14, 2023

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

**Marcus Chavez**  
**Sean Leininger**  
**Joseph Pham Nguyen**

ENTITLED

**FlowView: A Web Application for Hydrologic Predictions**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING



David Anastasiu (Jun 16, 2023 17:05 PDT)

---

Thesis Advisor



Silvia Figueira (Jun 17, 2023 17:07 PDT)

---

Department Chair

# **FlowView: A Web Application for Hydrologic Predictions**

by

Marcus Chavez  
Sean Leininger  
Joseph Pham Nguyen

Submitted in partial fulfillment of the requirements  
for the degree of  
Bachelor of Science in Computer Science and Engineering  
School of Engineering  
Santa Clara University

Santa Clara, California  
June 14, 2023

# FlowView: A Web Application for Hydrologic Predictions

Marcus Chavez  
Sean Leininger  
Joseph Pham Nguyen

Department of Computer Science and Engineering  
Santa Clara University  
June 14, 2023

## ABSTRACT

The Santa Clara Valley Water District, or Valley Water, provides water management resources to the two million residents in Santa Clara County. A few of their services include supplying safe water, ownership of streams, and flood protection. However, given the evolving conditions of the climate throughout the state of California, Valley Water seeks to include machine learning analytics as part of its water management services. Additionally, the increased access to the Internet has expanded the use of web applications, which provide a flexible platform to integrate the predictions of machine learning models. In this paper, we design and implement a production-grade web application that incorporates machine learning to provide useful data predictions for reservoir levels and stream flows. The web application consists of a full-stack architecture with a backend implemented using the Flask Python framework and the frontend served using the Jinja HTML templating library. Because machine learning jobs can be time-consuming, we use RabbitMQ and Celery to asynchronously delegate tasks to GPUs so Flask can continue to handle other client requests. Lastly, Tailwind CSS and Alpine.js are used to style the Flask web application and provide interactivity as needed. We deploy our application onto an AWS EC2 cloud instance leveraging technologies like Docker and Nginx as a reverse web proxy secured with SSL certificates. By bundling the analytical power of machine learning with the accessibility of web applications, we provide an easy-to-use service that can save time and resources in a critical public sector.

## ACKNOWLEDGMENTS

We would like express our deepest thanks to Dr. Anastasiu for providing unwavering guidance and extensive knowledge during the implementation of our project. Dr. Anastasiu's insightful suggestions during the planning of this project were invaluable and contributed significantly towards our project development. We have also had the pleasure of working with Santa Clara Valley Water Authority as they have provided us the opportunity to create an exciting solution with a positive impact. We would also like to give our deepest thanks to Valley Water for sponsoring our development with an Amazon cloud instance, which was invaluable towards the deployment and testing of our project. Without Dr. Anastasiu and Santa Clara Valley Water Authority's collaboration, the conception of this project would not have been possible.

A special thanks to the Computer Science and Engineering department for the learning and knowledge extended during our time at SCU. We are also extremely grateful to the School of Engineering for the support provided in funding our project. Finally, we would like to express our gratitude toward our families for the steadfast support they have provided throughout our academic careers.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement . . . . .	1
1.2	Project Proposal . . . . .	1
<b>2</b>	<b>Project Requirements</b>	<b>3</b>
2.1	Design Constraints . . . . .	3
2.2	Functional . . . . .	3
2.3	Non-Functional . . . . .	4
<b>3</b>	<b>Literature Survey</b>	<b>6</b>
3.1	Web Development . . . . .	6
3.1.1	Accessibility . . . . .	6
3.1.2	Cloud Technology . . . . .	6
3.2	Security . . . . .	7
3.2.1	Security Patterns . . . . .	7
3.2.2	Security Technologies . . . . .	8
3.2.3	Authentication . . . . .	9
3.2.4	Cross-Site Request Forgery . . . . .	9
3.3	Database Technologies . . . . .	10
3.3.1	Relational and Non-Relational Databases . . . . .	10
3.3.2	Database Technologies . . . . .	10
3.4	Messaging Queues . . . . .	11
3.4.1	Message Queuing Technologies . . . . .	11
3.4.2	Comparison of Message Queuing Technologies . . . . .	12
3.5	Applied Use Cases . . . . .	13
3.5.1	Web Application Use Cases . . . . .	13
3.5.2	Machine Learning Integration . . . . .	14
<b>4</b>	<b>Design Methods</b>	<b>15</b>
4.1	Architecture . . . . .	15
4.1.1	Flask . . . . .	16
4.1.2	MariaDB . . . . .	16
4.1.3	RabbitMQ . . . . .	16
4.1.4	Celery . . . . .	16
4.1.5	TailwindCSS . . . . .	17
4.1.6	Alpine.js . . . . .	17
4.1.7	Valley Water API . . . . .	17
4.2	Developer Operations . . . . .	17
4.2.1	Docker and Docker Compose . . . . .	17
4.2.2	AWS EC2 Instance . . . . .	18
4.2.3	CertBot . . . . .	18
4.2.4	Nginx . . . . .	19
4.3	Architectural Flow . . . . .	20

4.4	Use Case Diagrams . . . . .	22
4.5	Activity Diagrams . . . . .	25
4.6	Development Timeline . . . . .	28
4.7	Risk Analysis . . . . .	31
<b>5</b>	<b>Evaluation</b>	<b>32</b>
5.1	Results . . . . .	32
5.1.1	Map Interface . . . . .	32
5.1.2	Sensor Dropdown . . . . .	33
5.1.3	Adding A Sensor . . . . .	33
5.1.4	Data Chart . . . . .	35
5.1.5	Manage Sensors . . . . .	37
5.1.6	Manage Users . . . . .	38
5.1.7	Manage Cron Jobs . . . . .	39
5.1.8	Change Profile . . . . .	40
5.1.9	Add Model . . . . .	41
5.1.10	View Model . . . . .	43
5.1.11	User Alerts . . . . .	45
5.1.12	Email Notifications . . . . .	46
5.2	Challenges . . . . .	47
5.2.1	Deploying FlowView with Docker . . . . .	47
5.2.2	Model Training . . . . .	47
5.2.3	Data Storage . . . . .	48
<b>6</b>	<b>Societal Issues</b>	<b>49</b>
6.1	Ethical . . . . .	49
6.2	Environmental Impact . . . . .	50
6.3	Usability . . . . .	50
6.4	Lifelong Learning and Compassion . . . . .	51
6.5	Engineering Character . . . . .	51
<b>7</b>	<b>Conclusion and Future Work</b>	<b>52</b>
7.1	Future Work . . . . .	52
7.2	What We Have Learned . . . . .	52
7.3	Why it is Important . . . . .	53

# List of Figures

4.1	Architecture for FlowView. . . . .	15
4.2	Securing the Flask application with Nginx and CertBot Using SSL certificates. . . . .	19
4.3	The steps taken in a user's request through FlowView's architecture. . . . .	20
4.4	Celery worker sends an email to the user via the Flask-Mail extension and Google SMTP server. . . . .	21
4.5	Use case for a standard user. . . . .	22
4.6	Use case for a manager. . . . .	23
4.7	Use case for an administrator. . . . .	24
4.8	Activity diagram for a standard user. . . . .	25
4.9	Activity diagram for a manager. . . . .	26
4.10	Activity diagram for an administrator. . . . .	27
4.11	Fall quarter development timeline. . . . .	28
4.12	Winter quarter development timeline. . . . .	29
4.13	Spring quarter development timeline. . . . .	30
5.1	The map interface displayed within the main application dashboard. . . . .	33
5.2	The user has the ability to view the data over the lifetime of a sensor. . . . .	35
5.3	The user may select sensor data over an interval to view in closer detail. . . . .	36
5.4	The user has the ability to edit sensor metadata within FlowView. . . . .	37
5.5	Administrator users have the ability to manage user permissions. . . . .	38
5.6	Resource Managers and Administrators can view and manage the Cron Jobs. . . . .	39
5.7	Users may change their account settings. . . . .	40
5.8	Resource Managers and Administrators can train new machine learning models. . . . .	41
5.9	Resource Managers and Administrators can copy an existing model's configurations. . . . .	42
5.10	Resource Managers and Administrators can view and manage existing models. . . . .	43
5.11	Resource Managers and Administrators can view an existing model's configurations. . . . .	44
5.12	Success alert notifies the user after performing a significant action. . . . .	45
5.13	Error alert notifies the user after performing a significant action. . . . .	45
5.14	Email notifications are sent to the user upon completion of asynchronous requests. . . . .	46

# Chapter 1

## Introduction

### 1.1 Problem Statement

The Santa Clara Valley Water Authority (Valley Water) provides safe and clean water for Santa Clara County while helping to manage environmental risks like floods or droughts. Managing and predicting water reservoir levels is of the utmost importance, especially given the droughts California has been facing the past several years. Currently, a machine learning model has been developed by a doctoral student working with Dr. Anastasiu to make model inferences on streamflow levels using previously collected environmental data. However, this model has limited access via executing a script on a command line interface and provides no visualization of predictions to help understand the data. As a result, Valley Water has expressed a need for an application graphical user interface to help manage the training of this machine learning model as well as to visualize the predictions outputted by the machine learning model. Furthermore, they wish for the application to allow functionality for training new machine learning models on additional locations with newer sensor data. Due to the specificity of Valley Water's needs, there are currently no existing implementations that can satisfy their requirements for solving this problem. Attempting to utilize any existing data visualization tool would require a large amount of customization and development in order to interface with Valley Water's existing API's and the machine learning model. As a result, this problem is most reasonably solved by building a web application solution from scratch. This type of solution will be tailored directly to the needs of Valley Water and will also allow for greater control over the implementation of the solution.

### 1.2 Project Proposal

Working with Valley Water, this problem will be solved through the development of a full-stack web application. This application will connect to Valley Water's API's for pulling sensor data, automate the retraining of the machine learning models, and provide a clear visualization of the latest and most accurate model predictions of streamflow in Santa Clara County. This tool will allow Valley Water to make informed decisions about the future of water resources in the form of a simple and easy to use application that requires minimal work from the part of Valley Water in keeping



the application updated with accurate predictions. By creating an easily accessible, user-friendly web application, this project will satisfy the two primary needs of Valley Water: visualizing streamflow predictions for existing locations, and adding new locations to train new streamflow predictions. This important tool will allow simple upkeep and additions to the existing system without the need for increasingly complex knowledge of machine learning models and web interfaces. This will also allow for the application to be used across virtually any device with access to the Internet. Keeping thorough documentation of our work and development will allow for Valley Water and other stakeholders to maintain and extend the application beyond our initial work, should additional features be desired.

## Chapter 2

# Project Requirements

### 2.1 Design Constraints

Design constraints are composed of constraints that affect the planning and development process of our web application. The primary design constraint for our web application is that it must be as lightweight as possible. This constraint will ensure that our web application is easy to maintain while still providing powerful functionality such as storing large amounts of sensor data and utilizing complex machine learning models. Another design constraint for our web application is that its user interface must be modern, easy to use, and visually engaging. Our web application must be professional given that it is being developed for a client with strict needs and requirements. Most importantly, this professional and modern user interface must include a dashboard to easily view historical sensor data as well as streamflow predictions from the machine learning algorithm integrated into the application. Accordingly, another design constraint of our web application is that it must securely store all user information and sensor data. Valley Water may desire to store private information such as user data and application metadata, which emphasizes the need for ensuring data integrity and confidentiality. Another design constraint for our web application is that it must provide asynchronous real-time data processing and task operations to handle potentially time-consuming machine learning jobs. Our web application must be able to identify when hydrologic sensors have gathered new data, in which case an asynchronous job is triggered to re-run the machine learning model to yield new hydrologic predictions. A final design constraint of our web application is that it must be implemented under budgeting and time constraints. Our budget constraints are due to the fact that Valley Water is publicly funded by the state with limited budget. Our time constraints are due to the fact that Valley Water has provided a rapid development timeline to meet their needs in serving Santa Clara County.

### 2.2 Functional

Functional requirements are defined functionalities that the product application must perform and how it plans to perform its various features and functions. To understand end-users' needs, we needed to be able to identify critical

workflows based on the type of end-user and then construct a list of functional requirements. A user accessing our web application will belong to one of the following three categories: regular user, water resources manager, administrator. A regular user will have the sole ability to interact with the application dashboard to view historical sensor data as well as streamflow predictions outputted from the trained machine learning model integrated into the application.

A water resources manager will retain the existing functionality of a regular user, but will also have the following abilities. A manager in FlowView will be able to add and delete reservoir, stream, and precipitation sensors. When new sensors are added to the application, the manager will be able to train new machine learning models using the added sensors with limited hyperparameters. As a manager is fetching new sensors to add into FlowView and training machine learning models, they will also have the ability to manage the task queues for these asynchronous jobs. Finally, a manager will have the ability to manage Cron Jobs for refetching sensor data and retraining machine learning models inside FlowView.

An administrator will retain the existing functionality of a water resources manager but will also have the following abilities. An administrator in FlowView will be able to train new machine learning models with the full suite of hyperparameters supported for a given model type. An administrator will also have the ability to approve users registering for FlowView as well as to delete users. This functionality is reserved for administrators so that Valley Water can ensure only authorized users access FlowView for their internal use. Furthermore, administrators in FlowView will have the ability to manage user permissions such as promoting and demoting individual users to different permission levels.

## **2.3 Non-Functional**

Non-functional requirements can be seen as an extension of the functional requirements, but provide more flexibility and qualification as opposed to the hard-set requirements. Non-functional requirements tend to describe the general properties of the system, such as security, performance, reliability, user interface, and user experience. Here, we describe these mentioned properties as well as additional properties regarding the application dashboard and distributed messaging brokers and task queues.

The first non-functional requirement is application security, which will require the application to ensure that transactions involving user data, sensor data, and application metadata, are all properly secured.

The next non-functional requirement is ease of deployment, which will allow our web application to be portable. We may facilitate portability of our application by writing clean and maintainable code. In addition, packaging our application's services into containers would facilitate ease of deployment. We would like to make our application portable for Silicon Valley Water Authority to easily deploy our application to their personal servers for hosting.

Another non-functional requirement is the application dashboard. In implementing the dashboard, we aim to

provide an interactive map that allows users to select the sensor they wish to view. Furthermore, this dashboard will allow users to view historical sensor data and streamflow prediction data using an interactive chart. When complete, this intuitive and modern interface will make interactions with the application easier and more visually appealing.

Implementing a secure messaging broker with task queues is another non-functional requirement of our web application as this will facilitate availability. Our messaging broker must be secure in order to avoid attacks by malicious actors such as forgery of fake messages. Furthermore, this security requirement for the messaging broker is to ensure that no attacker is able to submit their own malicious messages and exploit the workers used to process the asynchronous jobs. A messaging broker will allow us to provide status updates to the user by tracking the progress of time-consuming tasks. Using asynchronous task queuing will allow us to hide the latency of time-consuming jobs by running these jobs in the background while the user executes other actions in the web application.

The last non-functional requirements are performance and reliability. Ensuring the performance of our web application means we will create a responsive user interface with minimal latency, which is ideal for providing essential information quickly. Implementing reliability will enable the application to handle moderate-to-high workloads, allowing for a greater number of users to access the web application at the same time.

# Chapter 3

## Literature Survey

### 3.1 Web Development

In this section, we review development styles and technologies for building web applications, specifically web accessibility patterns and cloud technologies.

#### 3.1.1 Accessibility

Jeschke et al. [1] describe accessibility as the “unlimited use of Web applications for all people - a participation in the Web independent of individual impairments.” They emphasize the importance of starting web application development with the design of the user interface (UI) to follow a user-oriented approach that meets accessibility standards. They note the most adequate and common accessibility guidelines as those given by the World Wide Web Consortium (W3C) such as Web Content Accessibility Guidelines, Authoring Tool Accessibility Guidelines, and User Agent Accessibility Guidelines. They also describe examples of Web Accessibility Patterns regarding overview and navigation such as Access Keys, which are shortcuts activated by hotkeys to “simplify repetitive activities.” Bread crumbs are another accessibility pattern they describe which provide indication of the current path of users within a hierarchical document tree. Bread crumbs may be implemented in the style of forms, tables, or lists. Ultimately, these approaches facilitate the development of accessible Web applications to accommodate different types of users.

#### 3.1.2 Cloud Technology

Phatak and Kamalesh [2] define the cloud as a “large pool of easily usable and accessible virtualized resources (such as hardware, development platforms, and services)” They note the recent transition of enterprises to cloud computing due to the benefits it provides such as lower costs as compared to on-site hosting and scalability. However, cloud computing also presents concerns such as reliability and lack of inter-portability. More specifically, cloud resources are prone to frequent outages, and integration of services from different providers becomes difficult when an entity is subscribed to a single cloud provider. These drawbacks stem from the use of a two-tiered cloud architecture in which

the cloud computing stack is broken down into two layers: the cloud provider and the clients. Phatak and Kamalesh [2] propose a three-tiered cloud architecture composed of three layers: the cloud provider, the service provider, and the clients. Reliability and inter-portability is improved from separating the cloud provider into a service layer by adding redundant data centers in the service layer and allowing clients to subscribe to multiple cloud providers through the service layer.

## **3.2 Security**

In this section, we review technologies used for implementing secure web applications such as identifying vulnerabilities and authentication techniques.

### **3.2.1 Security Patterns**

As Disawal et al. [3] state, web application vulnerabilities are a massive concern in web application development which can lead to loss of time, money, and effort in industry. Web application vulnerabilities tend to be grouped according to severity levels i.e. critical, high, medium, and low. Classifying security issues by severity level helps developers gain a larger understanding of the web development process and create a comprehensive solution. Common vulnerabilities include cross-site forgery requests, brute-force attacks, denial of service, SQL injections, and content spoofing. Lastly, the impacts of these attacks are grouped into different categories, such as confidentiality impacts, access complexity, and source of attack. Some causes that Disawal et al. [3] describe that lead to security vulnerabilities include lousy development practices and/or only using standard configurations instead of customized policies. In the deployment phase, some issues like insufficient session expiration, application misconfiguration, and etc. allow for exploits like session fixation. The most important distinction made by Disawal et al. [3] is that from start to finish, developers should always keep in mind security and ensure that during the implementation stage, security policies are properly customized and configured based on the end user's needs to avoid exposing vulnerabilities and confidential information.

Kunda et al. [4] report their empirical observations from using a variety of open source testing tools for web application security tests. Because of the relative ease and accessibility of web applications, web applications are increasingly popular, which has not only evolved how applications are made, but also led to a variety of attacks to compromise them as well. As a result, Kunda et al. [4] report a simplified workflow for creating a test and development cycle, such as identifying testing goals and scope, searching for relevant tools, performing cyclical security and vulnerability testing, and etc. Some open source tools identified by Kunda et al. [4] are Burp Suite, Bega, SoapUI, Metasploit, and OWASP ZAP, which are well-known tools for overall security testing. These tools make it possible to address vulnerabilities from attacks like Cross-Site Scripting, SQL injection, cookie thievery, and more. A recommended practice of development was to use test-driven development to avoid early security failures. After

experimenting with several tools, some common vulnerabilities highlighted were CSRF attacks, missing X-Content-Type-Options headers, clickjacking, removing debugging from applications in production, Cookie No HttpOnly flags, and HTTP method fuzzing.

### **3.2.2 Security Technologies**

Nakhuva et al. [5] remark on how RESTful web services are often used in applications due to their scalability, lightweight, and ease of development. However, security is always a point of contention for securing the RESTful web services. While there are a variety of existing security implementations that are used, such as OAuth, token-based authentication, OpenID, and SSL/TLS for instance, each method does have their own vulnerabilities and strengths that need to be taken into consideration when securing RESTful web services. In particular, Nakhuva and Champaneria discuss SSL/TLS in greater detail and compare it against other potential security provisions. SSL/TLS are protocols that encrypt segments at the transport layer and utilize symmetric cryptography when transmitting data from a client application to the server. By using MAC addresses, message reliability is also ensured, which gives a secure end-to-end communication between the client and server. SSL/TLS can be thought of as a four-phase process, with a greeting or “hello world” phase, certificate transfer phase, change cipher phase, and then communication phase.

Nakhuva and Champaneria emphasize that while SSL/TLS is very secure due to the handshaking process, it can also consume a large amount of network bandwidth which can limit accessibility to mobile devices that may not have as much processing power. As such, a light-weight alternative algorithm is suggested, which uses TLS to secure the communication between client and server, and by using hashed password authentication and token-based authentication are used in combination to authenticate a client against the server. Once a client is authenticated, when using the RESTful services, lightweight hashing/signature algorithms like Blowfish, MD5, and HMAC-based SHA are used to ease the burden of processing on mobile devices. The biggest benefit was alleviating messaging overhead and processing time since speedy hashing algorithms were used once the client was authenticated against the server.

Lee et al. [6] report on their evaluation of the latest version of the Transport Layer Security(TLS), looking at the protocol from a number of different perspectives to demonstrate how it contributes to the internet. Using security parameters, handshake messages, and platform information, they were able to look at the adoption, security, performance, and implementation of TLS 1.3. While being adopted significantly faster than previous iterations, TLS 1.3 also is more secure due to its use of forward-secret and AEAD cipher suites, though unstable days, largely due to short periods of downgraded server use due to various causes, can be an issue. However, TLS 1.3 does include a downgrade sentinel that will prevent connections from TLS 1.2. In terms of performance, an increase is seen across every major geographical region. However, one lacking piece of TLS 1.3 lies in the common implementation, where half of the major libraries available properly implemented both signed certificate timestamps and OCSP sampling, meaning that some implementations of TLS 1.3 will not fully take advantage of the security benefits.

### **3.2.3 Authentication**

Akanksha and Chaturvedi [7] detail the implementation and security provided by JSON web tokens (JWT). Particularly they discuss how JWTs are versatile and platform independent, being composed of a header, payload, and signature, which help to authenticate a client to a server in many cases. They also have the flexibility of securing the payload and header data through numerous cryptographic methods, with either a secret key, or public/private key pair being able to decrypt the data, securing it to help ensure the user data contained in the JWT is not accessed by other sources. Another aspect of JWTs that contributes to their security is the ability to set token lifespans or to blacklist certain tokens altogether, and while the blacklist method requires more storage for all of the invalid JWTs, can still help to provide authentication successfully. In their research, Akanksha and Chaturvedi [7] clearly claim that JWTs provide a better method of authentication than session authentication because of the reduced load on the database due to it being stateless, and that it can provide a shorter average response time.

### **3.2.4 Cross-Site Request Forgery**

Likaj et al. [8] detail the popularity and severity of Cross-site Request Forgery (CSRF), finding it to be a particularly understudied area. As they describe, these attacks involve tricking a web browser into sending an authenticated HTML request to a state-change without the user knowing. In seeking to understand the available defenses and their effectiveness, they evaluate a wide variety of available defenses, their security, and the development challenges these defenses present. They found that Python was the most exposed language by looking at the risks presented by the surveyed frameworks, while C# proved to be the least exposed. It should be noted that Flask in particular only faced security threats from implementation mistakes, meaning its proper implementation should be secure, according to the collected data. As a final note, Likaj et al. [8] criticize the lack of documentation for most major frameworks, which increase the risk of implementation errors that could lead to vulnerabilities.

Rankothge et al. [9] describe their solution for mitigating CSRF attacks by way of a secret token pattern technique. This CSRF token is primarily embedded as a hidden parameter in web page elements such as forms, and this token is generated by the server and given to the client upon successful authentication. By ensuring that this CSRF token is embedded within every legitimate request, the server can successfully authenticate that only a legitimate user is trying to access its services, and if it is empty or does not contain the correct value, the server rejects the malicious attacker. This CSRF token is generated with a random number generator, expires after a short time, and is generally safer to verify compared to other alternative token-based authentication solutions. A new CSRF token is generated for each user session, which ensures that if an attacker attempts session-hijacking, at worst the previous CSRF token will be duplicated or empty, which is rejected by the server. Rankothge et al. remarks that the CSRF token is secure and ensures that authenticated users can perform their web application activities with no concerns.



### **3.3 Database Technologies**

In this section, we review two types of database technologies: relational and non-relational databases. We also examine specific platforms using these two types of database technologies and their respective advantages and disadvantages.

#### **3.3.1 Relational and Non-Relational Databases**

Sahatqija et al. [10] compare the performance and functionality of relational and NoSQL databases and explore their respective advantages and disadvantages. They note that relational databases are traditionally used in applications that require consistency and utilize transactional data. On the other hand, they note NoSQL databases have grown out of a need to handle flexible schemas with increasing amounts of heterogeneous data sources. NoSQL databases can be categorized into four-types: key-value, column-oriented, document-based, and graph databases. Sahatqija et al. [10] explain the differences between relational and NoSQL databases in terms of scalability, flexibility, security, and data management.

Relational databases use vertical scalability by expanding the computing power in response to increased volume of data. Non-relational databases use horizontal scalability by adding more nodes for data storage rather than increasing the performance of a single node. Relational databases also fulfill the properties of ACID (Atomicity, Consistency, Isolation, Durability) while non-relational databases fulfill the properties of BASE (Basically Available, Soft state, Eventually consistent). ACID properties prioritize consistency and reliability while BASE properties prioritize flexibility. This prioritization follows the CAP Theorem, in which only two out of the three properties of consistency, availability, and partition tolerance may be fulfilled for distributed systems.

Relational databases must provide a predefined database schema that remains static while non-relational databases have dynamic schema that can handle changes in the type of data being stored. The structured data schema of relational databases allows easy management of security while the unstructured data of non-relational databases requires greater consideration of key security factors such as access control and data encryption. Data redundancy is avoided in relational databases due to the normalization of data while data in non-relational databases may include redundancy due to the lack of relationships between data.

They note that software applications that prioritize integrity and consistency of data should choose relational databases. Software applications that do not prioritize data integrity in favor of availability, performance, and scalability should choose non-relational databases.

#### **3.3.2 Database Technologies**

As Tongkaw and Tongkaw [11] describe, the world has entered an era of big data, and open source database projects to handle these have become very popular, with MySQL and MariaDB standing at the forefront. They then compared the performance of these two databases utilizing a variety of online transaction processing (OLTP) measurement tasks to

evaluate the transactions per second of each database under various test settings to get a full view of their performance. Through their testing, they concluded that while the performance of the two was similar in situations with fewer threads and workers, MySQL proved to perform much better beyond those as the thread count reached 1000, all while using roughly the same amount of resources. It should be noted that their results did show certain situations where MariaDB would outperform, but those were the cases of fewer threads or workers for the various data sets utilized in the testing.

As the internet has evolved, applications that used to only have thousands of users have now increased to millions of users at all times around the world. As Györödi et al. [12] describes, there are relational databases that can handle limited data, but nowadays the availability of data has expanded massively, which has led to the rise of NoSQL. NoSQL is a methodology that is centered around handling immense amounts of unstructured data and storing them using some sort of identification keys scheme. Popular and competing examples include using JSON documents, key-value pairs with distributed dictionaries, columnar/parquet format, and more. The key trait is that the data has no schema, which is the opposite of SQL databases like MySQL, Postgres, and etc. While both approaches have their strengths and weaknesses, Györödi et al. [12] performed a series of experiments to compare the performance of MongoDB and MySQL, which are popular databases from both of the NoSQL and SQL paradigms. Different actions like insert, select, update, and delete were tested. In conclusion, MongoDB proved to perform better in all four actions than MySQL.

## **3.4 Messaging Queues**

In this section, we review messaging queue technologies and how they may be used for developing asynchronous messaging services in web applications.

### **3.4.1 Message Queuing Technologies**

Due to how resource-intensive and time-consuming data science and machine learning tasks can be, Chaowvasin et al. [13] state that message queuing and containerization of applications should be used in order to maintain ease of development and deployment while reducing tolerance for time-consuming tasks. Some common modern technologies are Kubernetes and Docker, which allow for containerization of running applications and scaling said containers up or down based on user/workload demands. Different application architectures like SOAP, MVC, and REST were discussed, and in experimentation Docker, Kubernetes, RabbitMQ, and a REST API were developed. During experimentation, different numbers of machine learning nodes were tested, with the REST API funneling requests through RabbitMQ to simulate workloads to analyze time performance. They found that scaling the number of nodes up worked well to more quickly process ML tasks and that RabbitMQ was able to route requests from the REST API without message loss.

Basavaraju et al. [14] outline the basic fundamentals of the Advanced Message Queuing Protocol (AMQP) like

the use of brokers, publishers and consumers, exchanges, communication patterns, and etc., and describe two popular messaging queue frameworks that implement AMQP: RabbitMQ and ActiveMQ. Some differences between RabbitMQ and ActiveMQ are further described, such as RabbitMQ's implementation of AMQP 0-9-1 versus ActiveMQ's use of the 1-0 AMQP protocol. They perform a series of experiments between RabbitMQ and ActiveMQ to empirically analyze their respective performance. Basavaraju et al. [14] list some parameters for analysis, such as latency, publish rate, data-rate, and latency change as message payloads increase. The conclusion found was that ActiveMQ was more performant than RabbitMQ, although it should be noted that the ActiveMQ brokers were allocated more RAM on their Amazon AWS instances. However, an important distinction that was noted was that ActiveMQ does not utilize exchanges due to the AMQP 1-0 protocol not using them as well.

Pathak et al. [15] provides analysis and description of the basic queueing mechanisms and communication models used in modern applications that involve heterogeneous devices. The primary models like request-response, publish-subscribe, exclusive pair, and push-pull are discussed in conjunction with popular frameworks like Message Queuing Telemetry Transport Protocol (MQTT) and Advanced Message Queuing Protocol (AMQP). Pathak et al. [15] also highlights key metrics like availability, scalability, interoperability, and security that are used to measure the performance of message queueing systems. While Pathak et al. [15] compliments the increased scalability and efficiency of the AMQP protocol that RabbitMQ provides, he also exposes various weaknesses in its implementation. For instance, the queues in RabbitMQ are of limited size, and under intense workloads, consumer-to-exchange acknowledgement may fail, leading to packet duplication; the exchange itself is also blind to queue status, which can congest the queue, consume extra disk space, and increase latency while decreasing throughput. Pathak et al. [15] suggest an architecture in which they create subqueues that can detect failures in acknowledgments and effectively queue dropped messages to avoid congesting the primary queues and maintain throughput while minimizing latencies.

### **3.4.2 Comparison of Message Queuing Technologies**

Fu et al. [16] emphasize how modern technology produces information at such a large scale that older technologies can no longer keep up with important processes like data ingestion and real-time data analysis. Thus, message queueing systems were developed in order to handle large streams of data via brokers, producers, and consumers. Messaging queues allow for a variety of flexible implementations such as asynchronous processing, decoupling of producer and consumer, and more. Several production features and design choices of messaging queueing systems are described such as development language, consumption mode, compatibility, system architecture, message queueing model, and more. Additional metrics for measuring performance are delivery guarantee, ordering guarantee, reliability, latencies, scalability, and more. The following messaging queueing systems are also compared against each other: Kafka, RabbitMQ, RocketMQ, ActiveMQ, and Pulsar. Kafka is a logging service based on the TCP protocol that uses topics to classify different groups of messages and stores partitions of topics in brokers to distribute messages in parallel to

different consumers. Kafka leverages a peer-to-peer architecture, and the topics work primarily in a push-pull communication pattern. However, Kafka also requires a Zookeeper to perform third-party cluster management between brokers. RabbitMQ implements the AMQP protocol and does not rely on a cluster manager like Kafka and supports normal cluster mode and mirror cluster mode. RabbitMQ can achieve scalability and availability through queue mirroring and an efficient producer acknowledgment system. RocketMQ uses a master-slave architecture and load balancing for reliability and availability, and uses a centralized store to store messages into a physical file. One worthy weakness noted is that RocketMQ's community is less active in comparison to Kafka and RabbitMQ. ActiveMQ's unique features provide priority queuing, batch transmission, and transactional messages, but lack community support. Lastly, Pulsar utilizes a brokers-bookies separated architecture, and while this increases scalability, needing to use a Zookeeper and Bookkeeper adds network overhead and decreases performance.

As Dobbelaere and Esmaili [17] describe, Apache Kafka and RabbitMQ are two large, open-source, and commercially used publish/subscribe systems with their own advantages and disadvantages, which they break down and evaluate for a holistic view of both systems. RabbitMQ operates by having producers publish messages that eventually route to a queue that consumers are able to push or pull from to get messages, while Kafka has publishers publish to a disk based append log, which consumers can pull messages from. Through their research, they discovered that RabbitMQ is more consistent in lower latency, with almost no factors causing significant jumps in latency compared to Kafka, and in terms of a higher throughput, both were able to scale effectively with RabbitMQ being held back by routing complexity. They ultimately conclude with each having use cases more tailored to their features and strength, like RabbitMQ being better equipped to the requirements of a layer of a web application, while some cases would most benefit from their combined use, since some features of both are required for certain implementations.

## **3.5 Applied Use Cases**

In this section, we review applied use cases of developing web applications integrated with machine learning models and data visualization features.

### **3.5.1 Web Application Use Cases**

Verma et al. [18] discuss approaches to building a full-stack web application with machine learning models. They note the importance of Cloud Computing to discover large amounts of data to make informed decisions. They also emphasize the importance of choosing the correct frameworks, libraries, and technologies for building a web application for a particular set of users. They also utilize Cloud Deployment technologies to take advantage of rapid deployment and to eliminate the need to buy a physical server to host a web application.

Baseri et al. [19] define information visualization as “the process of representing data in a visual and meaningful way in order to make a better understanding.” They note the importance of providing sufficient context and choosing

appropriate visualization techniques of data to effectively communicate the meaning of these visualizations. They describe an approach to building a web application for data visualization using agile web development techniques and the Python Flask framework. Their agile methodology includes requirements analysis, planning, design, implementation, and deployment.

### **3.5.2 Machine Learning Integration**

Singh et al. [20] discuss how they utilize machine learning algorithms to predict flowers and encapsulate those predictions into a singular web application using the Flask web framework. Because Python is a popular language for machine learning and the ease of using the Flask framework, it is often a chosen solution for a machine learning web application. Additionally, by using a Python web framework like Flask, it allows for a modular approach that can incorporate other important resources like an ORM (Object Relational Manager).

Because of the rising use of machine learning to perform predictive data analytics also needs a convenient and user-friendly way to display and utilize those models. Thus, Lakshmanarao et al. [21] used Flask to display the desired results of the machine learning algorithms after they perform predictions on whether or not a URL is malicious. By using tools like pickle to store the machine learning models for predictions, Flask could then call the models given the user's URL input.

As Trianti et al. [22] describe, Flask and Python can be used for integrations with web applications and machine learning because of their ease of use and popularity in machine learning algorithms. Flask is a relatively secure solution that can solve both front-end and back-end integrations, which leaves more attention for focusing on the machine learning functionality while displaying user-friendly data. There are several important conclusions that Trianti et al. [22] highlight. First, GPUs should be used for machine learning tasks to reduce wait times and reduce CPU usage. Flask is a useful Python framework that can integrate Python code into useful and simple HTML content that can be displayed in an accessible web browser.

# Chapter 4

## Design Methods

### 4.1 Architecture

Figure 4.1 below showcases the architecture for FlowView. For our architecture, we chose to use lightweight and easy-to-learn technologies and frameworks for implementing FlowView. Valley Water has specified a strict set of needs and requirements as well as a rapid development timeline. As a result, we have chosen technologies that can facilitate an iterative development process with adaptability to changing requirements. Additionally, we have prioritized technologies that are open-source due to strict budget requirements.

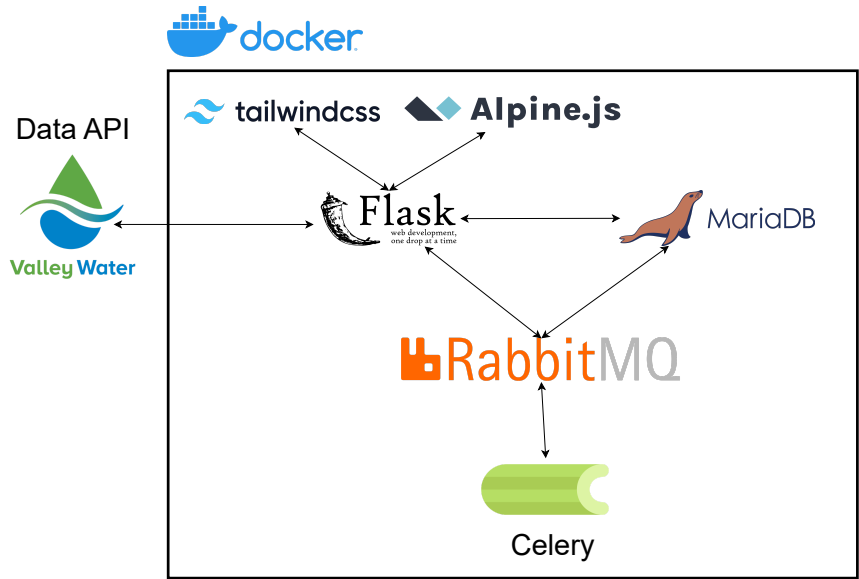


Figure 4.1: Architecture for FlowView.

### **4.1.1 Flask**

Flask is a lightweight Python micro web framework that provides extensive tools for developing web applications. These tools include application component blueprints, HTTP routing, and session management. We use Flask as our web server to handle the primary application logic while serving a thin client to the user. Flask uses the Jinja templating engine to render HTML pages to the user. We make use of Flask extensions such as Flask-Login to create production-grade server logic. Flask also has extensive public documentation available, which makes it an ideal web framework to use should Valley Water desire to develop FlowView beyond our deliverables.

### **4.1.2 MariaDB**

MariaDB is an open-source relational database management system. We use MariaDB to store and query user login information and sensor data. We chose to use MariaDB as our relational database management system due to its status as free and open-source software. MariaDB's status as open-source software helps meet our budgeting constraints for FlowView. Furthermore, we chose to use a relational database over a non-relational database to prioritize features such as consistency and reliability in data transactions as noted in Section 3.3.1. This is especially emphasized by the need to fetch and store large amounts of sensor data from Valley Water.

### **4.1.3 RabbitMQ**

RabbitMQ is an open-source messaging broker used for distributed messaging. RabbitMQ is lightweight and supports a variety of messaging protocols like AMQP, MQTT, and XMPP. FlowView utilizes the AMQP messaging protocol with RabbitMQ. Furthermore, RabbitMQ is relatively easy to deploy and is efficient and scalable for general use. It is implemented in Erlang and handles messages in DRAM. We use RabbitMQ for handling asynchronous requests initiated by the user, which are routed to Celery as our task queue.

### **4.1.4 Celery**

Celery is an open source asynchronous distributed task queue that can process immense amounts of data in real-time. Celery is cross-platform which makes deploying Celery on an operating system easy to do. Celery provides support for multiple types of messaging brokers and integrates well with popular web frameworks like Django and Flask. Celery takes messages in a queue filled by some message broker such as RabbitMQ and distributes these messages to appropriate consumers. We use Celery to handle and process asynchronous jobs such as fetching sensor data and training machine learning models.

### **4.1.5 TailwindCSS**

TailwindCSS provides a lightweight framework for easily styling HTML pages. We use TailwindCSS to create a modern-looking website with clear visuals. TailwindCSS is fast as it has zero-runtime and is flexible due to the way in which it generates styles from class-names and writes these styles to a static CSS file. TailwindCSS also provides extensive documentation on their website for how to install and use its library, which allows for minimal setup and an intuitive development process.

### **4.1.6 Alpine.js**

Alpine.js is a lightweight Javascript framework that allows developers to compose Javascript behavior in HTML pages. We use Alpine.js to add interactivity to our website, which is necessary for behavior regarding event handling and displaying dynamic alerts to users.

### **4.1.7 Valley Water API**

Valley Water has provided a public API that FlowView calls to fetch sensor data. This sensor data is then stored in FlowView's MariaDB database. The Valley Water API is an architectural component external to FlowView, but it is important for gathering the sensor data that the machine learning models in FlowView will be trained upon.

## **4.2 Developer Operations**

In this section, we detail the technologies used to facilitate the deployment process of FlowView. In Section 4.2.1, we discuss the use of Docker to build our web application into isolated containers. The use of Docker allows us to easily deploy our application on an Amazon AWS EC2 instance generously provided by Valley Water described in Section 4.2.2. In Section 4.2.3, we elaborate on the methods used to obtain SSL certificates for our web application to enhance security. Then, in Section 4.2.4, we discuss the final deployment step of establishing an Nginx reverse proxy to allow communication between clients and the Flask server.

### **4.2.1 Docker and Docker Compose**

Docker is a platform that allows developers to develop, test, and deploy their applications in isolated environments known as containers. By isolating the running applications in containers, this provides security from resource collisions between multiple containers, and they are lightweight and run only on the host machine. Additionally, through Docker's platform, it is easy to ship applications via images so that developers are able to run the application regardless of machine, operating system, and etc.

Docker Compose is another tool that is used for running multi-container applications. Unlike Docker, which is limited to single container applications, multiple services are defined in a YAML (Yet-Another-Markup-Language)



configuration file, and these services are then run in different containers that communicate with each other. Docker Compose allows developers to specify different runtime policies, manage the status of the containers, and manage the overall lifecycle of the multi-container application.

FlowView uses Docker via Dockerfiles to instantiate the initial build of the Flask application, while Docker-Compose is used to install other dependencies and spin up all of the services in different containers and run the entire web application as a whole.

### **4.2.2 AWS EC2 Instance**

Our sponsor Valley Water has kindly provided us with an Amazon EC2 instance for us to deploy our production-grade web application on. The specific instance provided was a g4dn.xlarge, and some defining characteristics are its NVIDIA T4 Tensor Core GPU, 125 gigabyte storage, and 4 virtual CPUs. These specifications allow us to more efficiently perform operations such as training the machine learning models, fetching sensor data, and maintaining security and reliability throughout FlowView. We use Docker and Docker-Compose to deploy our application onto this instance, and the computational capabilities of the EC2 instance provide greater ease of use and reduction in latency.

### **4.2.3 CertBot**

Certbot is an open source tool that uses Let's Encrypt certificates to provide HTTPS (Hypertext Transfer Protocol Secure) to websites via SSL certificates. It was developed by the EFF (Electronic Frontier Foundation), which emphasizes digital privacy and free speech. Certbot allows developers to secure their web servers and also automatically renew their certificates to ensure the security of the website. By enabling HTTPS, Certbot helps ensure that the data transmitted between the client and the web servers are securely encrypted and cannot be easily intercepted by malicious attackers. FlowView runs Certbot as a separate Docker service to request the SSL certificates and properly configure the Nginx reverse proxy to enable HTTPS.

## 4.2.4 Nginx

Nginx is an open-source and free HTTP server and reverse proxy, and is a common solution used when deploying web applications. Nginx utilizes an event-driven, asynchronous, and scalable architecture. Additionally, it boasts high performance with a relatively small memory footprint. Nginx is a widely used solution among companies like GitHub, SoundCloud, Heroku, and many other well-known companies. Nginx is used as a reverse proxy by FlowView in order to add an additional layer of security by parsing requests before sending them to be handled by the Flask application. Figure 4.2 below displays how Nginx and Certbot are used to secure the connection between the client and the Flask application.

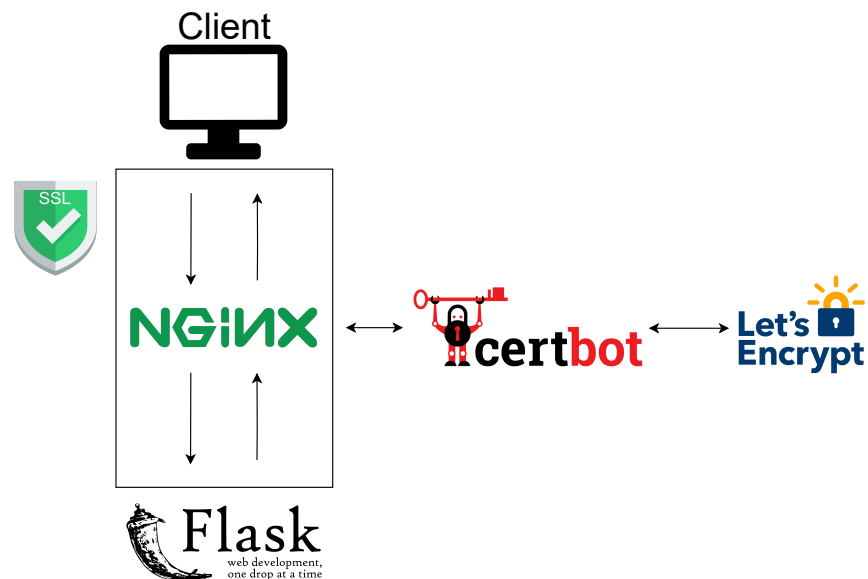


Figure 4.2: Securing the Flask application with Nginx and CertBot Using SSL certificates.

### 4.3 Architectural Flow

Having explained each architectural component of FlowView, we may now walk through the architectural flow that occurs as a user interacts with FlowView. Figure 4.3 below details the steps taken for a user’s request to travel between the different services in FlowView’s architecture.

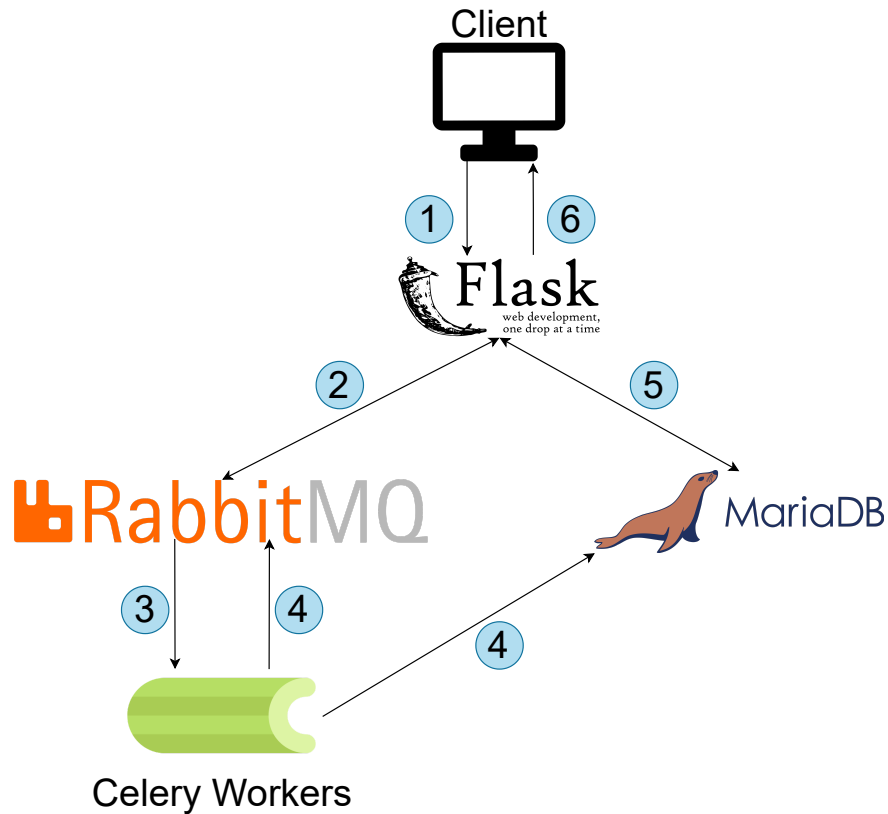


Figure 4.3: The steps taken in a user’s request through FlowView’s architecture.

The first step in the architectural flow occurs when a user visits the FlowView application from their web browser. When a user submits a request for an asynchronous job such as fetching sensor data or training a machine learning model, the Flask application will process the request and send a message to the RabbitMQ messaging broker. RabbitMQ will place the message on its queue until an available Celery worker takes the message off of the queue. The Celery worker will process the job asynchronously, which allows the user to interact with other features of FlowView as they wait for their asynchronous request to complete. Once the Celery worker has finished processing the asynchronous job, it will notify RabbitMQ of the completion of the job and whether the job was successful or failed.

All asynchronous jobs implemented in FlowView send an email to the user to notify them of the completion of their asynchronous request. We use the Flask-Mail extension to facilitate this process, which provides an interface for sending emails via the Google SMTP server. This process of sending emails to the user is visualized in Figure 4.4 below.

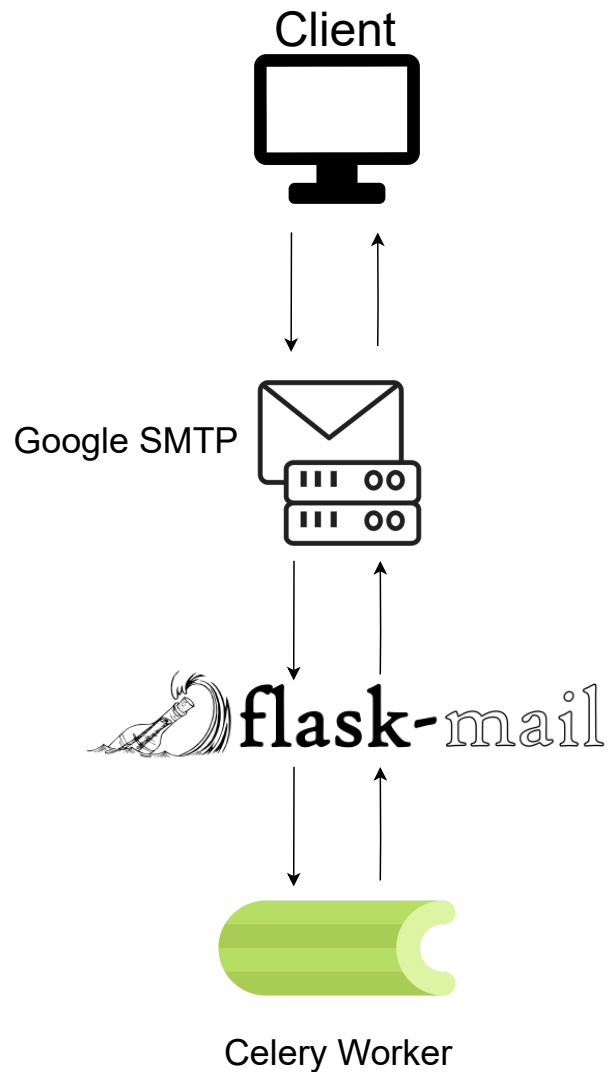


Figure 4.4: Celery worker sends an email to the user via the Flask-Mail extension and Google SMTP server.

In addition to sending emails to the user upon completion of asynchronous jobs, the Celery worker will place the results of the asynchronous job in MariaDB. When a user revisits FlowView to check on the status of their job, Flask will fetch the results of the asynchronous job from MariaDB and display the final results on the user's web browser.

## 4.4 Use Case Diagrams

Figure 4.5 details how a standard user would use FlowView. A standard user is only able to interact with the homepage where they can select sensors to view the streamflow predictions.

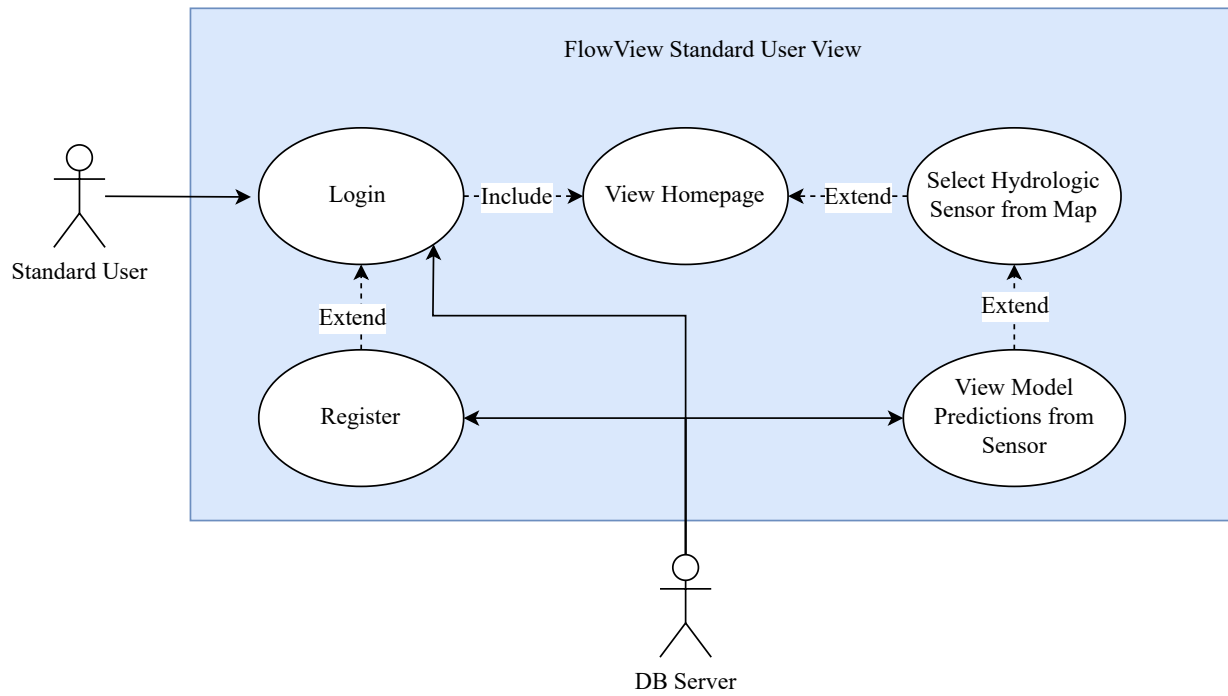


Figure 4.5: Use case for a standard user.

Figure 4.6 details how a manager would use FlowView. A manager would retain the same use cases as a standard user but would also have the ability to add and delete sensors. In addition, a manager would be able to train machine learning models within the application.

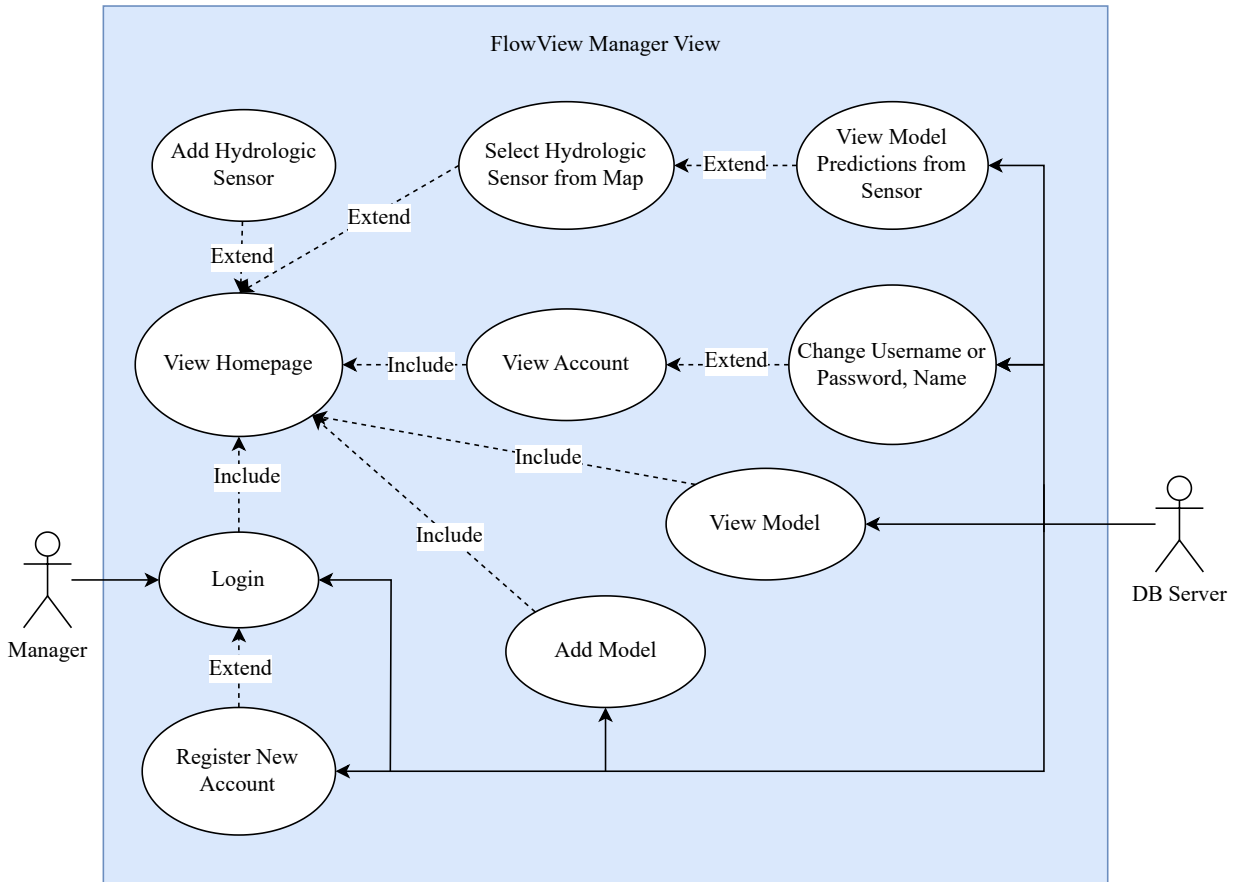


Figure 4.6: Use case for a manager.

Figure 4.7 details how an administrator would use FlowView. An administrator retains the same use cases as a manager but would also be able to manage users and their permissions for the application.

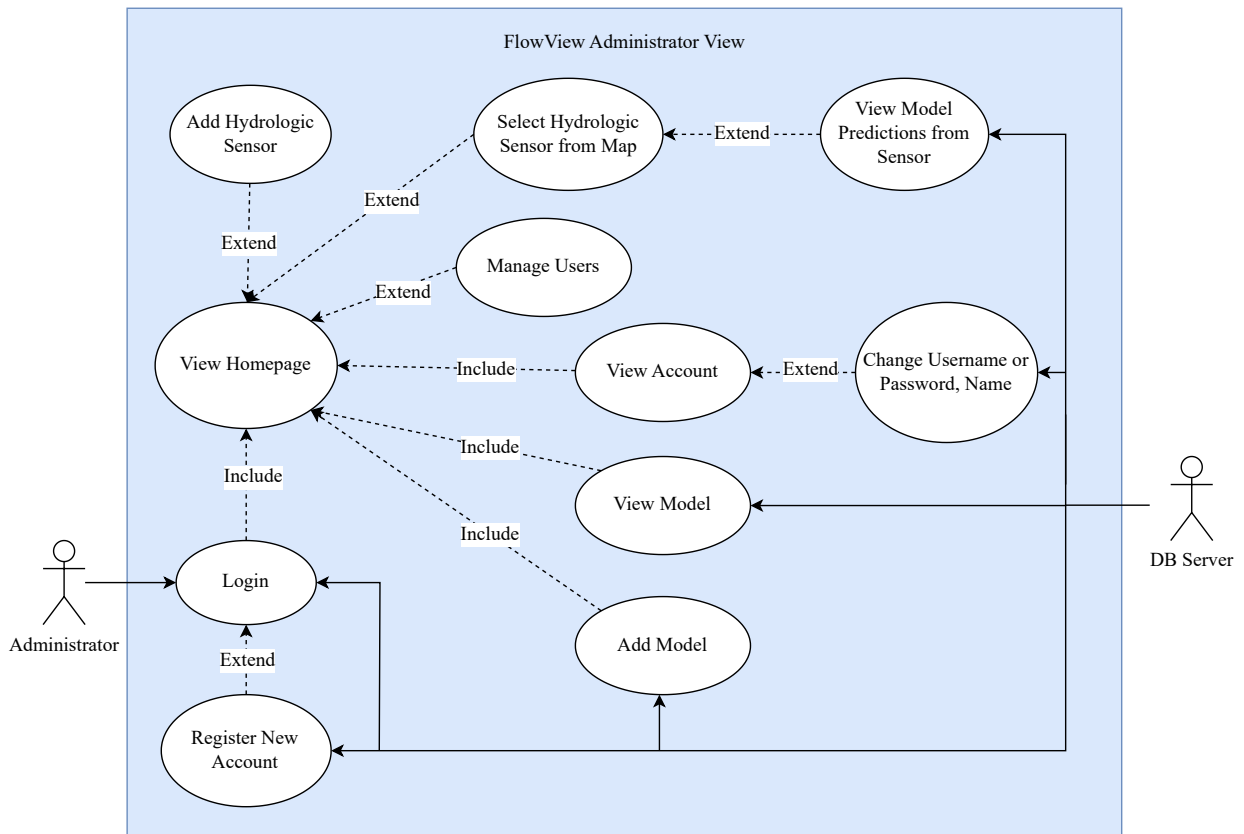


Figure 4.7: Use case for an administrator.

## 4.5 Activity Diagrams

Figure 4.8 details the flow for interacting with FlowView for a standard user. After logging into FlowView, a standard user would initially be greeted with the homepage where they can view streamflow predictions from existing machine learning models that have been trained in FlowView.

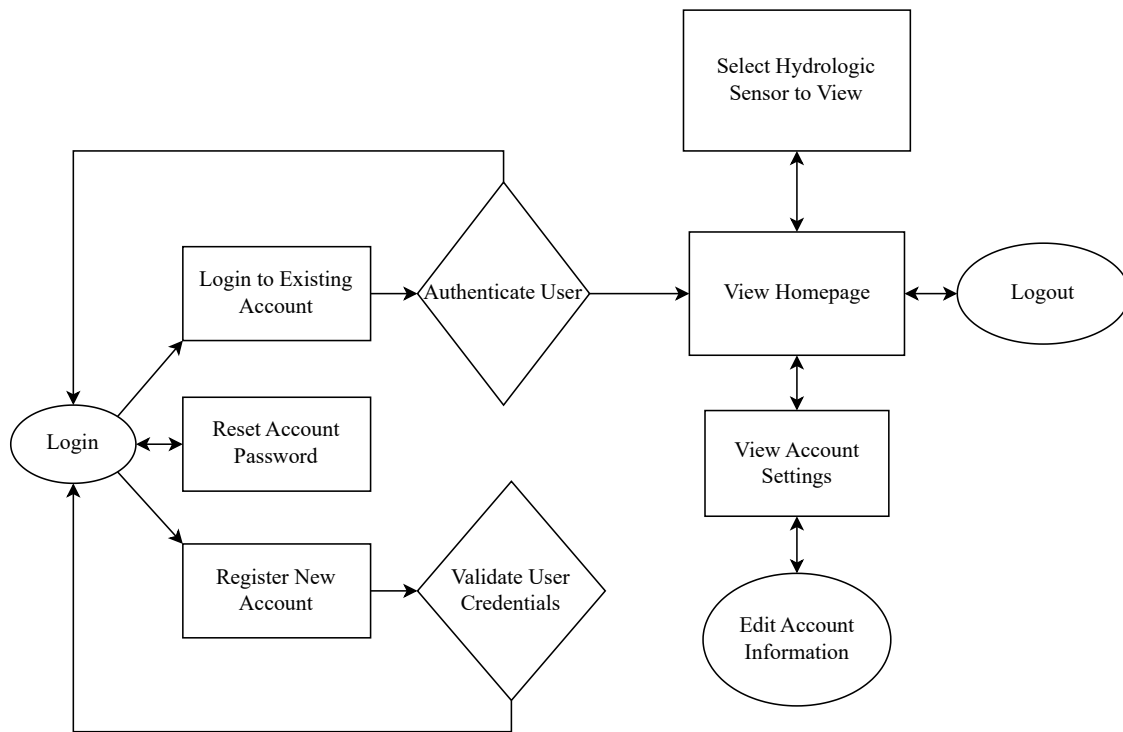


Figure 4.8: Activity diagram for a standard user.



Figure 4.9 details the flow for interacting with FlowView for a manager. A manager is initially greeted with the homepage similar to a standard user. However, a manager gains additional functionality, including the ability to add and delete sensors in FlowView. In addition, a manager may choose to train new machine learning models with a limited set of hyperparameters.

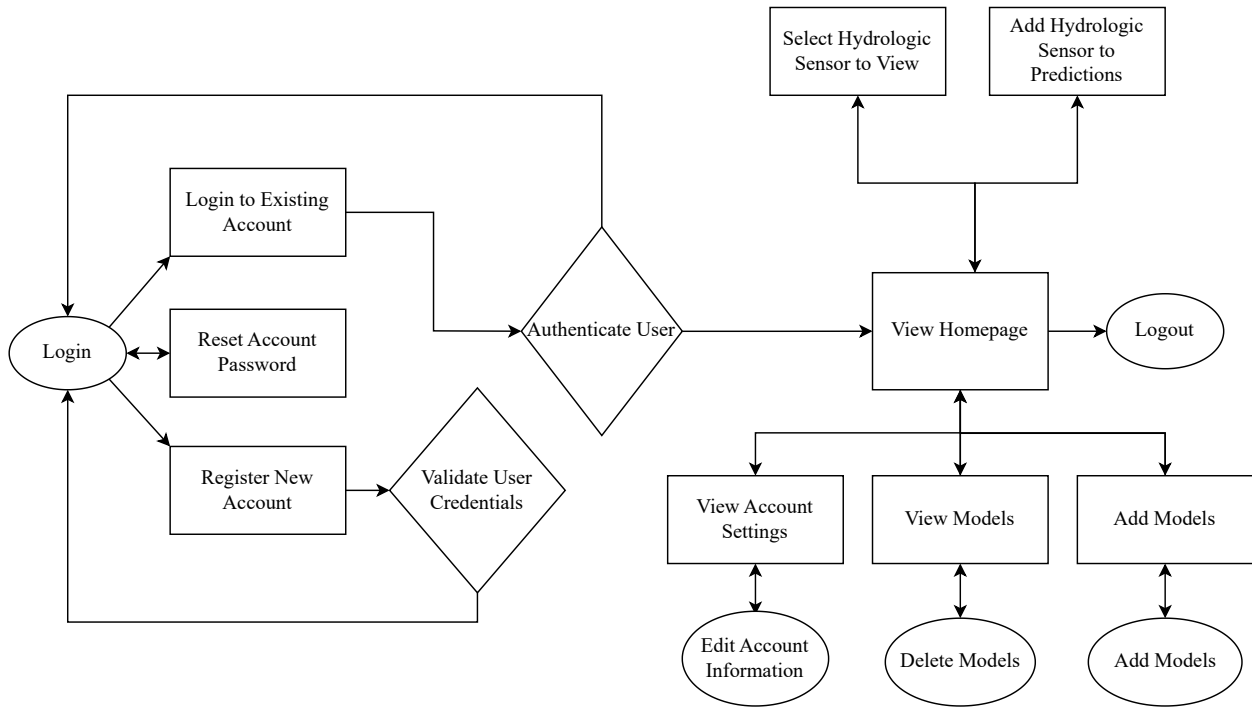


Figure 4.9: Activity diagram for a manager.

Figure 4.10 details the flow for interacting with FlowView for an administrator. An administrator navigates through the same flow as a manager but may also modify user permissions on the application.

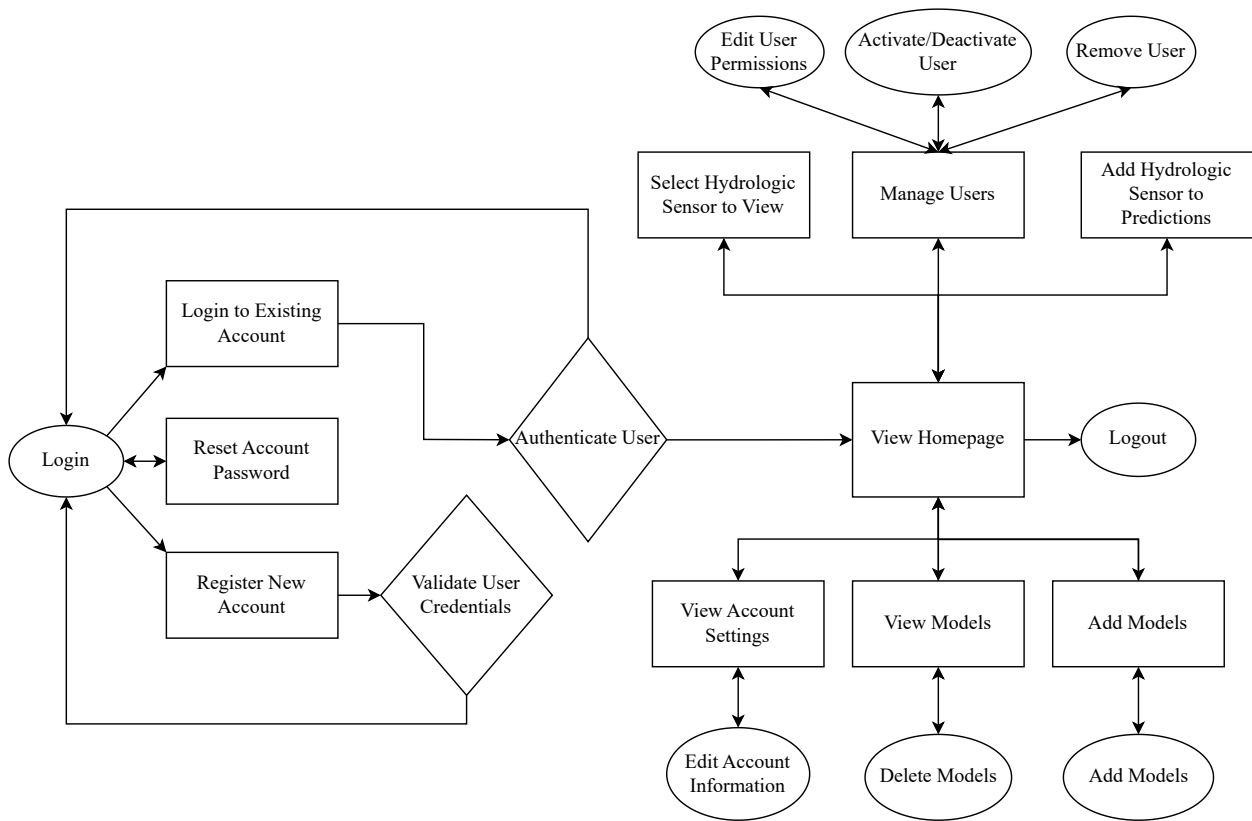


Figure 4.10: Activity diagram for an administrator.

## 4.6 Development Timeline

Figure 4.11 shows our proposed timeline for project implementation in the fall quarter of the 2022 to 2023 academic year. Our timeline for the fall quarter includes project research, planning, and initial implementation to achieve a working prototype of FlowView. It is most important to complete the initial implementation of our product during the fall quarter to allow sufficient time during the remainder of the academic year to gather and iterate on feedback from Santa Valley Water Authority.

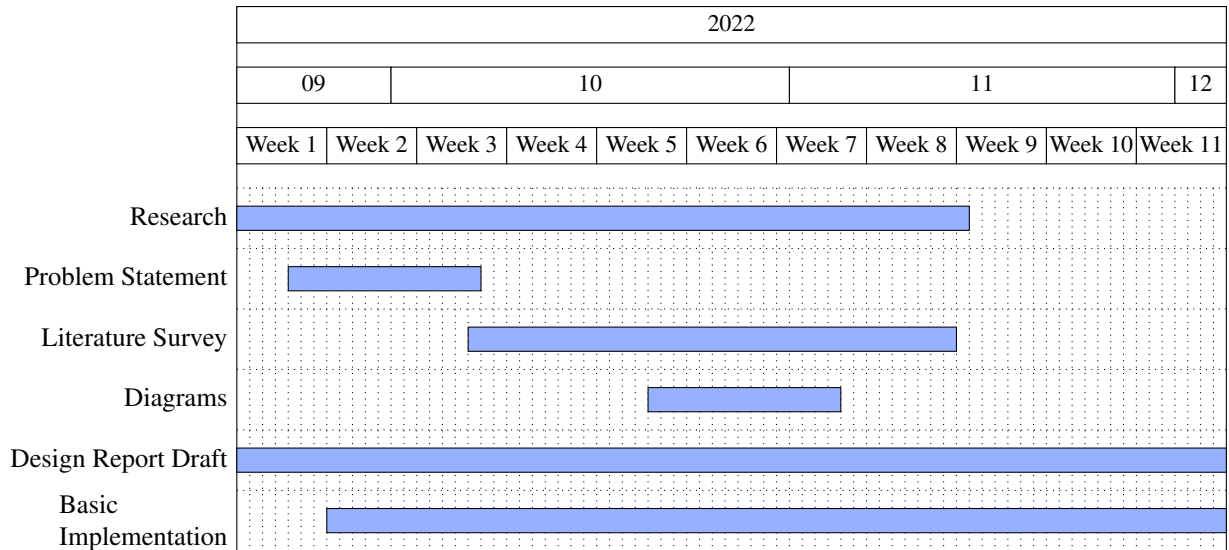


Figure 4.11: Fall quarter development timeline.

Figure 4.12 shows our proposed timeline for implementing our project in the winter quarter of the 2022 to 2023 academic year. Our timeline for the winter quarter includes the design presentations, interactions with the Santa Clara Valley Water Authority, and testing our web application on a cloud deployment. Feedback from the Santa Clara Valley Water Authority during the winter quarter is most important as the success of our web application primarily depends on whether it meets their requirements and expectations.

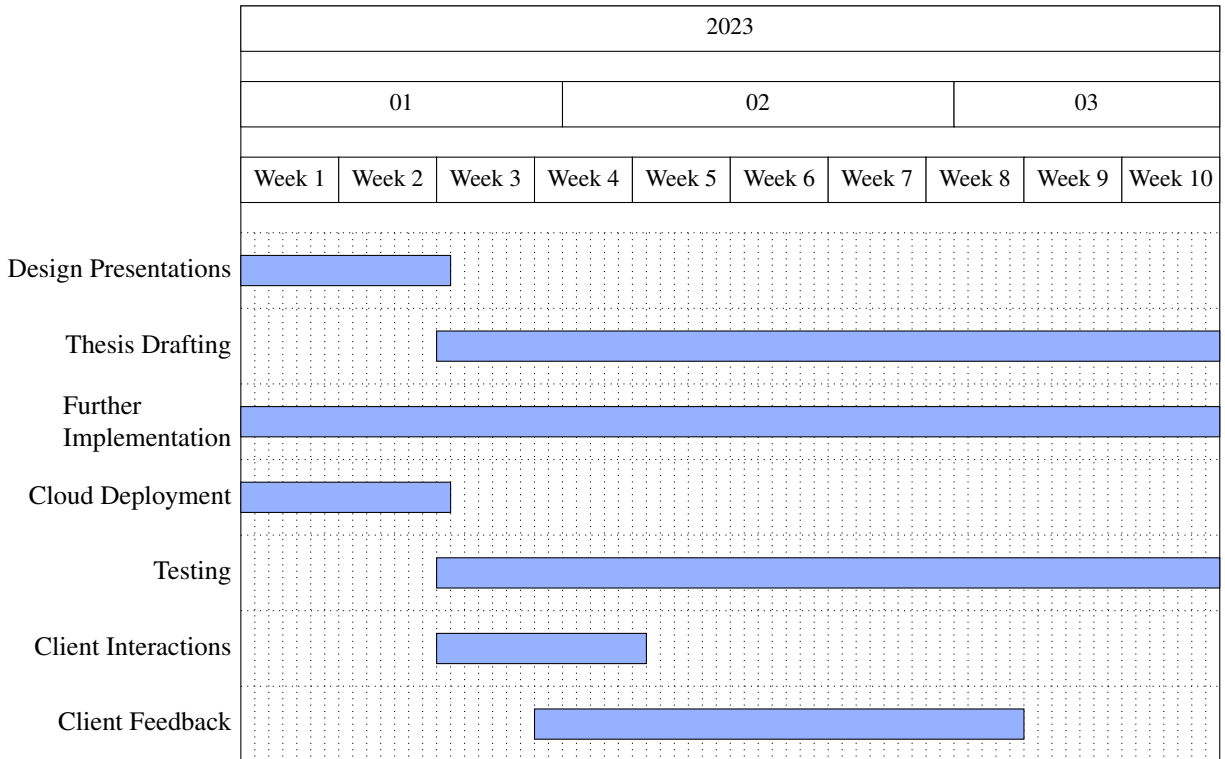


Figure 4.12: Winter quarter development timeline.

Figure 4.13 shows our proposed timeline for implementing our project in the spring quarter of the 2022 to 2023 academic year. Our timeline for the spring quarter includes final testing and implementation, preparation for the Senior Design Conference, and final drafting of our thesis for submission. It is most important that we meet the deadline for the submission of our thesis during this quarter.

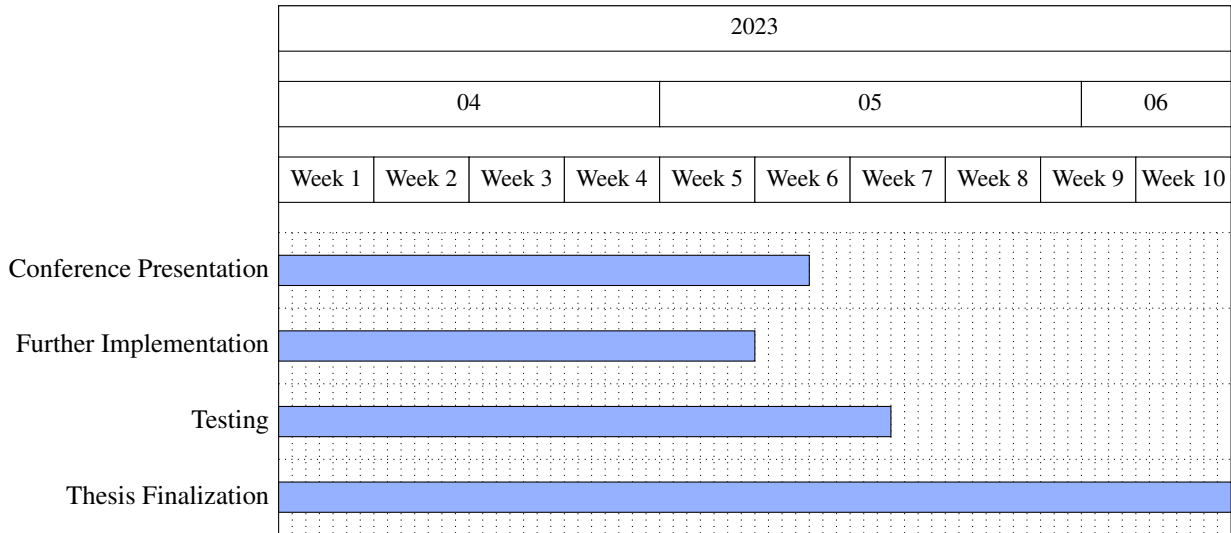


Figure 4.13: Spring quarter development timeline.

## 4.7 Risk Analysis

Table 4.1 below details our risk analysis for our project’s development. The severity of each risk is ranked on a scale of 1 to 10 with 1 being the lowest and 10 being the highest. The probability of each risk ranges from 0 to 1, with 0 as highly improbable and 1 as most probable. We also detail a mitigation strategy for each risk to create plans of actions should these risks occur.

Table 4.1: Risk Evaluation Outline

<b>Risk</b>	<b>Description</b>	<b>Severity</b>	<b>Probability</b>	<b>Mitigation</b>
Funding	Lack of funds restricts our ability to test our product on cloud-based GPU instances	4	0.3	Professor Anastasiu has some resources that could allow for limited GPU testing. We have also obtained some funding from the SCU School of Engineering for initial setup and testing on a Digital Ocean virtual machine. Lastly, Valley Water has kindly sponsored us by providing an Amazon EC2 instance by which to deploy our application on.
Time Constraints	Lack of time could prevent us from fully implementing the planned features of the final product	8	0.5	Planning ahead and carving out dedicated work time on the project should help to minimize risk of not completing the project
Development Risks	Having to reorganize our code base or re-establish the dependency list as a result of coding could lead to more work and time requirements	7	0.25	Thoroughly researching techniques to improve our code base and actively planning out the code should minimize this risk. We are also using Docker and Docker-Compose to streamline our application deployment and lifecycle.

# Chapter 5

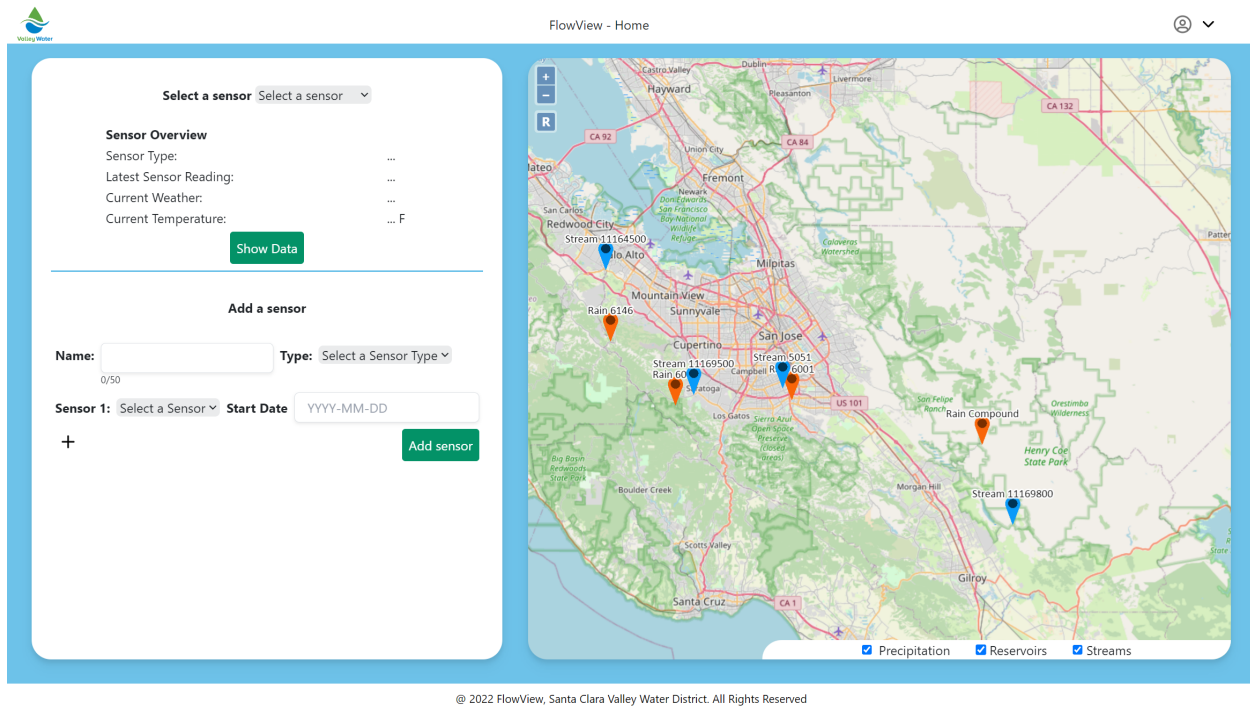
## Evaluation

### 5.1 Results

In the following sections, we will discuss in detail the results of our development process, giving a detailed look into the inner workings of the various pages and functions of our final web application.

#### 5.1.1 Map Interface

As seen in Figure 5.1 below, the dashboard utilizes an interactive map in order to help the user navigate between sensors and select them to show data. This map was built utilizing the OpenLayers Javascript library, which uses the Open Street Maps API in order to load map data into our application. This library was chosen for two primary reasons: ease of integration and cost. OpenLayers was relatively easy to integrate into our system, thanks to extensive documentation that allowed us to design the map to best help our users. In terms of cost, because OpenLayers supports the use of the Open Street Maps API, we can avoid the potential costs associated with other APIs like Google Maps API, and thus provide a cheaper, easier to maintain application to Valley Water. As a result, the map interface was very successful, as it is integrated within our dashboard, provides the user another tool of selecting between added sensors, and does not slow down the application.



@ 2022 FlowView, Santa Clara Valley Water District. All Rights Reserved

Figure 5.1: The map interface displayed within the main application dashboard.

## 5.1.2 Sensor Dropdown

The left side of the homepage seen in Figure 5.1 above shows a quick actions section. The dropdown in the top of this quick actions section is the primary method for showing quick information about any given sensor. Selecting from the dropdown not only loads data to the small table directly below the dropdown menu, but also adjusts the map to center it on the selected sensor and loads data in the background for the full chart of historical and predicted data. Additionally, interacting with sensors on the map will result in an update to the dropdown, fully integrating the various portions of the dashboard together. This dropdown successfully implements necessary features for a responsive and easy to use interface for FlowView’s primary application dashboard.

## 5.1.3 Adding A Sensor

The quick actions section on the left side of the home page seen in Figure 5.1 above also displays functionality for adding a new sensor to FlowView. This functionality has been placed on the home dashboard since a new user of FlowView will most likely desire to add a sensor as their first action in the pipeline for training a machine learning model. To add a new sensor in FlowView, the user must first enter a name they would like to give to the sensor they are adding as well as the type of the sensor. Once the type of the sensor is selected, such as a stream sensor, the user must select the Valley Water sensor they would like to add from the “Sensor 1” dropdown. The user may then specify the start date for the Valley Water sensor they are adding if they know the sensor starts at a particular date. However, the



user may also leave the start date field empty, and FlowView will automatically infer the start date of the sensor using metadata acquired from Valley Water's API. A plus button is present below the "Sensor 1" dropdown to allow the user to add an additional Valley Water sensor. Once the user adds an additional Valley Water sensor, the sensor being added to FlowView now becomes a composite sensor in which the data from multiple Valley Water sensors is being fed into the composite sensor created in FlowView. This functionality is important as it allows the user to combine the data of multiple sensors from Valley Water. For example, if a sensor from Valley Water were to be shut down and replaced with a new sensor with a new identification number, the user may want to be able to associate the data of the old and new sensor together. This functionality in FlowView for adding multiple sensors from Valley Water into one composite sensor inside FlowView addresses this need.

Once the user is satisfied with the fields they have filled for adding a new sensor in FlowView, they may click the "Add Sensor" button. This user action sends a request to the Flask backend server and sends an asynchronous message via RabbitMQ to initiate an asynchronous job for fetching sensor data from Valley Water's API via an available Celery worker. The request to add a new sensor is handled asynchronously by a Celery worker, and the user is free to interact with other features of FlowView without being blocked by the completion of their request to add a new sensor. Furthermore, once the asynchronous job to add the new sensor has been completed by a Celery worker, an email notification will be sent to the email address the user registered with to notify the user that their asynchronous request has been completed. This data management pipeline for adding a new sensor abstracts the backend logic so that the user only needs to be concerned with the parameters for adding a new sensor while the Flask backend handles the complexity of communicating with Valley Water's API.

## 5.1.4 Data Chart

The data chart, shown in Figure 5.2 below, allows the user to view the entire range of historical data for a selected sensor. The chart also loads data in accordance with the range of dates shown on the chart, that way wherever data exists within the bounds of the chart, that data will be loaded. As the user zooms out, an increasingly large set of data would naturally slow the application down. To prevent this slowdown, the backend utilizes data aggregation at certain predetermined intervals, meaning that when a data set over a certain size is requested, the data will be averaged to hourly, daily, or even monthly data, since some of Valley Water’s sensors have data dating back to the 1940’s.

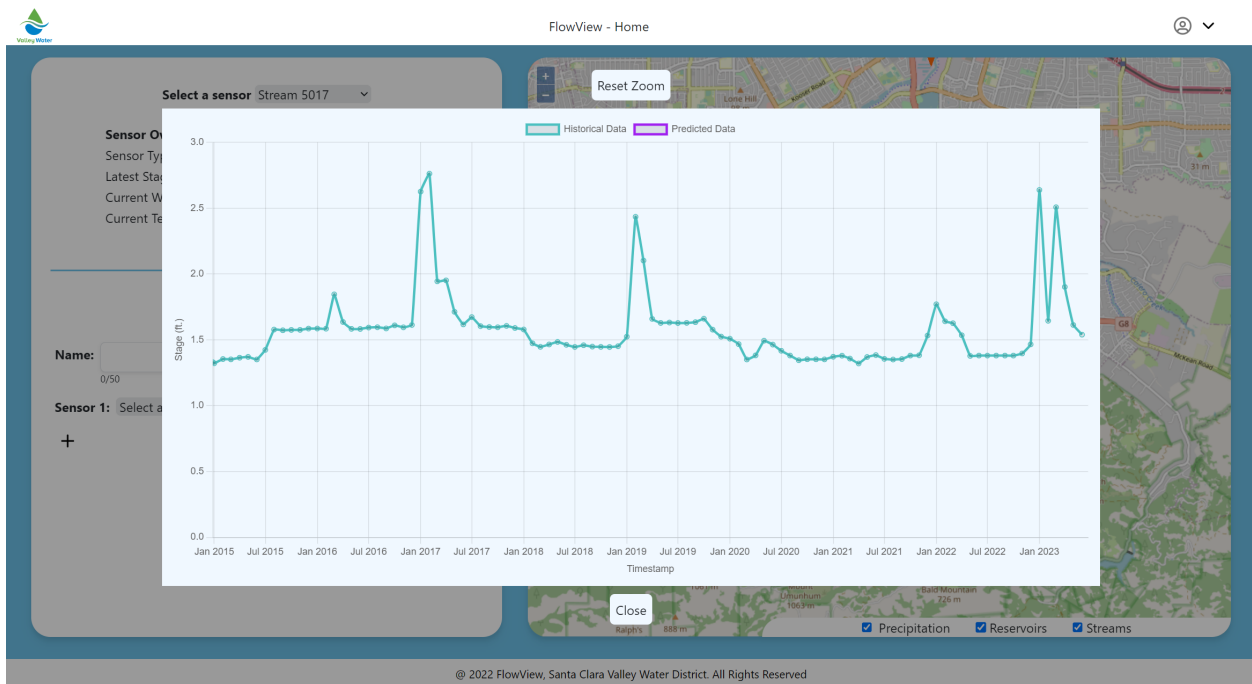


Figure 5.2: The user has the ability to view the data over the lifetime of a sensor.

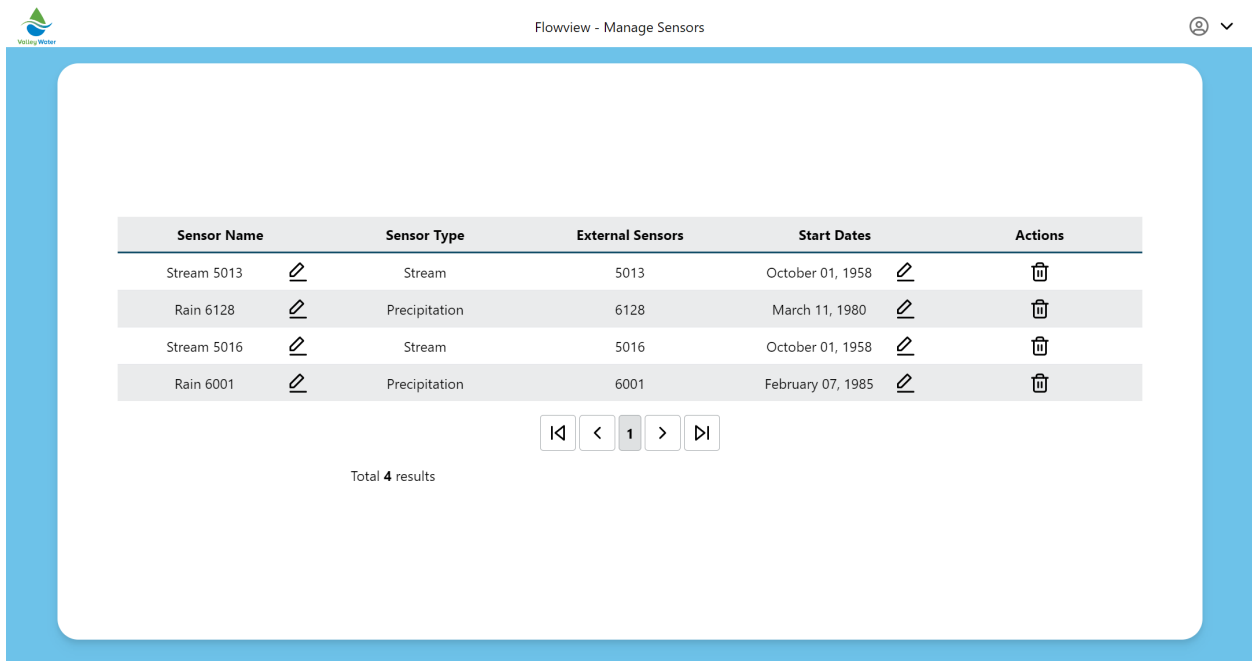
The data chart, shown in Figure 5.3 below, allows the user to view not only the historical data, but the collected prediction data as well, allowing for comparisons between the two. By overlaying the previous predictions with the historical data, the user can also have some insight into the reliability of the predictions, giving the predictions more weight. As a result of these features, the data chart is able to appropriately meet the needs of the project and the user, providing vital information to those utilizing it at Valley Water.



Figure 5.3: The user may select sensor data over an interval to view in closer detail.

### 5.1.5 Manage Sensors

Figure 5.4 below shows the user interface for managing sensors inside of FlowView. Information about previously added sensors or sensors that have data currently being fetched are displayed in a table to view the relevant metadata about each sensor. The table utilizes pagination to address overflow if the added sensors in FlowView do not fit in one page. Pagination buttons are present at the bottom of the table to allow the user to visit different pages and see all sensors in FlowView. Edit buttons are provided to allow the user to edit the name of the sensor or the start dates of the sensor after they have already been added. If the user edits the start date of a sensor, this change will be reflected in the backend logic for fetching and displaying sensor data in the data chart shown in Figure 5.2 above. Ultimately, the interface in Figure 5.4 below provides the user a simple method for viewing and editing details about all sensors that have been added to FlowView.

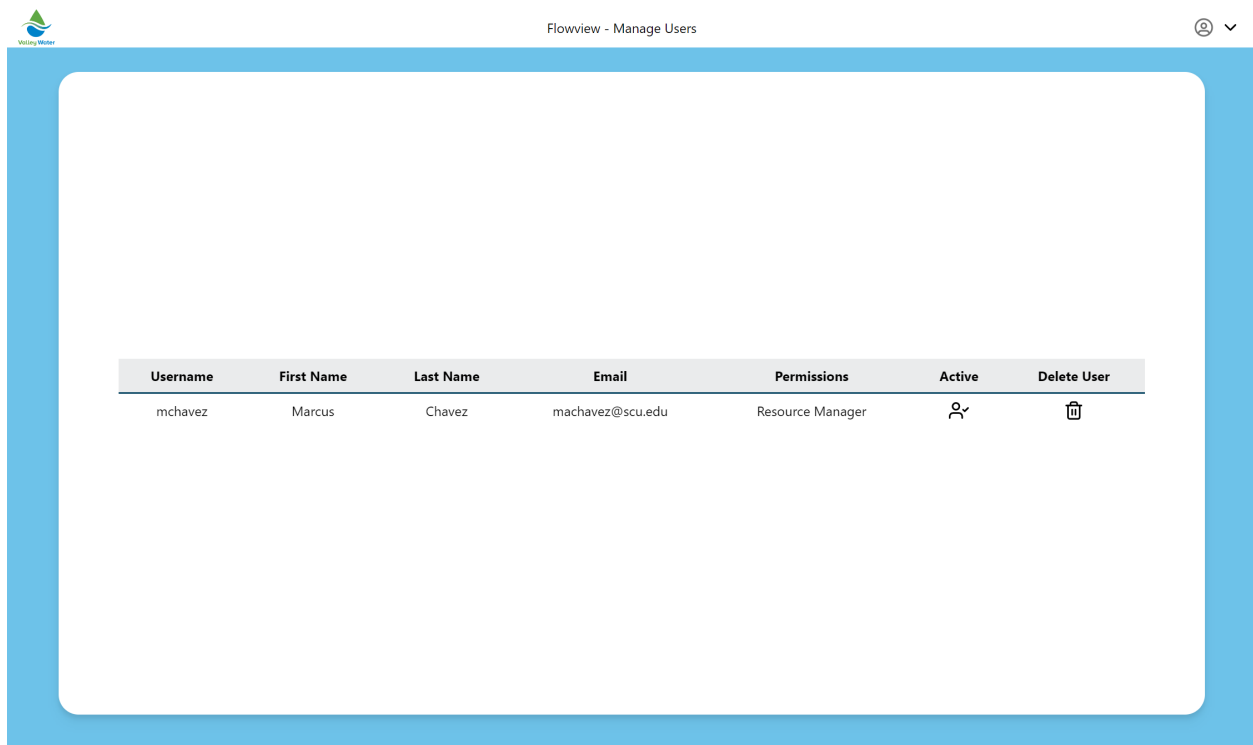


© 2022 FlowView, Santa Clara Valley Water District. All Rights Reserved

Figure 5.4: The user has the ability to edit sensor metadata within FlowView.

## 5.1.6 Manage Users

Figure 5.5 below shows the user interface for managing the users in FlowView. Once a user has registered, they will be populated into the table but will be set as inactive users, where an Administrator can then view the details of that new user. The administrator can activate users by selecting the corresponding dropdown and confirming their selection, and this will trigger a Flask application action to approve the user, which will allow them access to the application. Of course, the reverse is true in which an administrator can revoke their active status. All users that register and are approved will begin at the basic user permissions, but the administrator can easily adjust their permissions simply by clicking on the corresponding dropdown and selecting the possible permissions by which to set the user to. Last but not least, an administrator may remove users from the application by using the convenient user interface. All of the actions discussed above will inform the user by sending them an email detailing the action the administrator has just taken for their account upon a successful action execution. The user interface allows the administrators to easily make changes to the users as needed.

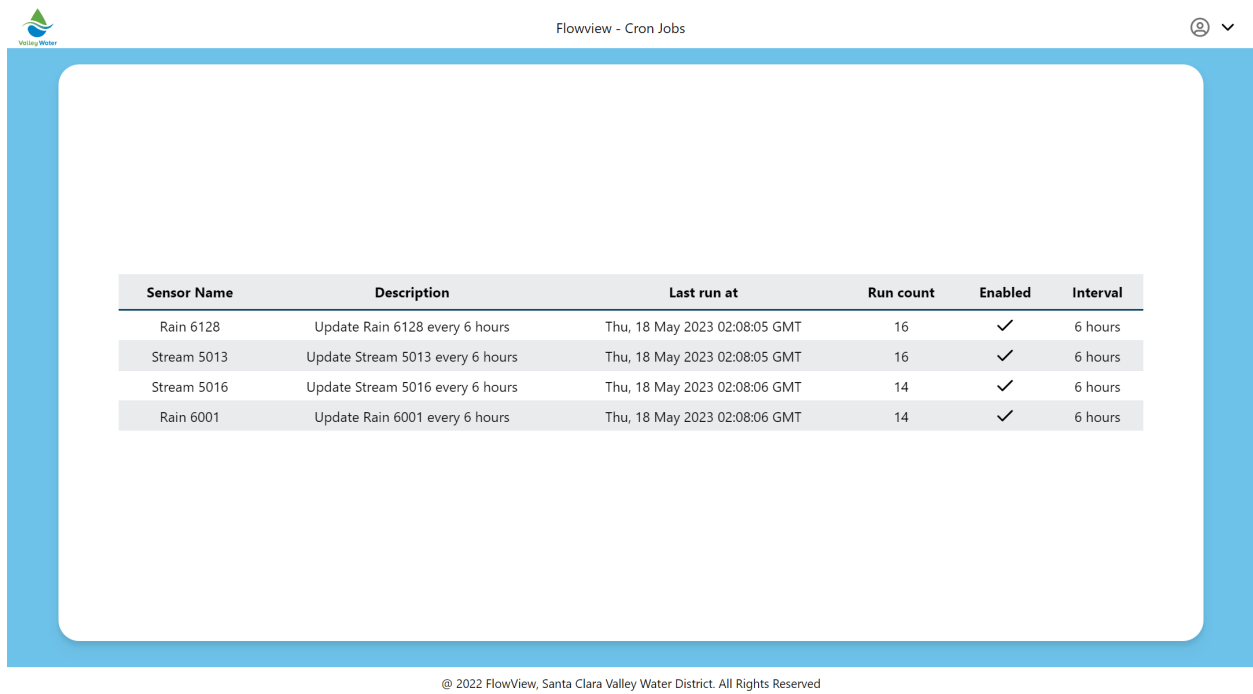


© 2022 FlowView, Santa Clara Valley Water District. All Rights Reserved

Figure 5.5: Administrator users have the ability to manage user permissions.

## 5.1.7 Manage Cron Jobs

Figure 5.6 below shows the user interface for managing Cron Jobs. Once a sensor is added to FlowView by fetching its historical data, a Cron Job is created in FlowView to automatically update the sensor's data as Valley Water's API is updated with the newest sensor data. FlowView provides a simple interface using a table for allowing the user to view and manage the functionality of Cron Jobs. The user can enable or disable any Cron Jobs inside of FlowView if they wish to begin running or stop running a Cron Job respectively. In addition, the user can modify the interval at which the Cron Job runs by choosing from a list of default interval settings of 6, 12, or 24 hours. Furthermore, the user can view metadata about each Cron Job such as its description and when it last ran. Ultimately, the user interface shown in Figure 5.6 allows the user to make detailed edits to the settings of Cron Jobs inside FlowView with a simple and easy-to-use interface.



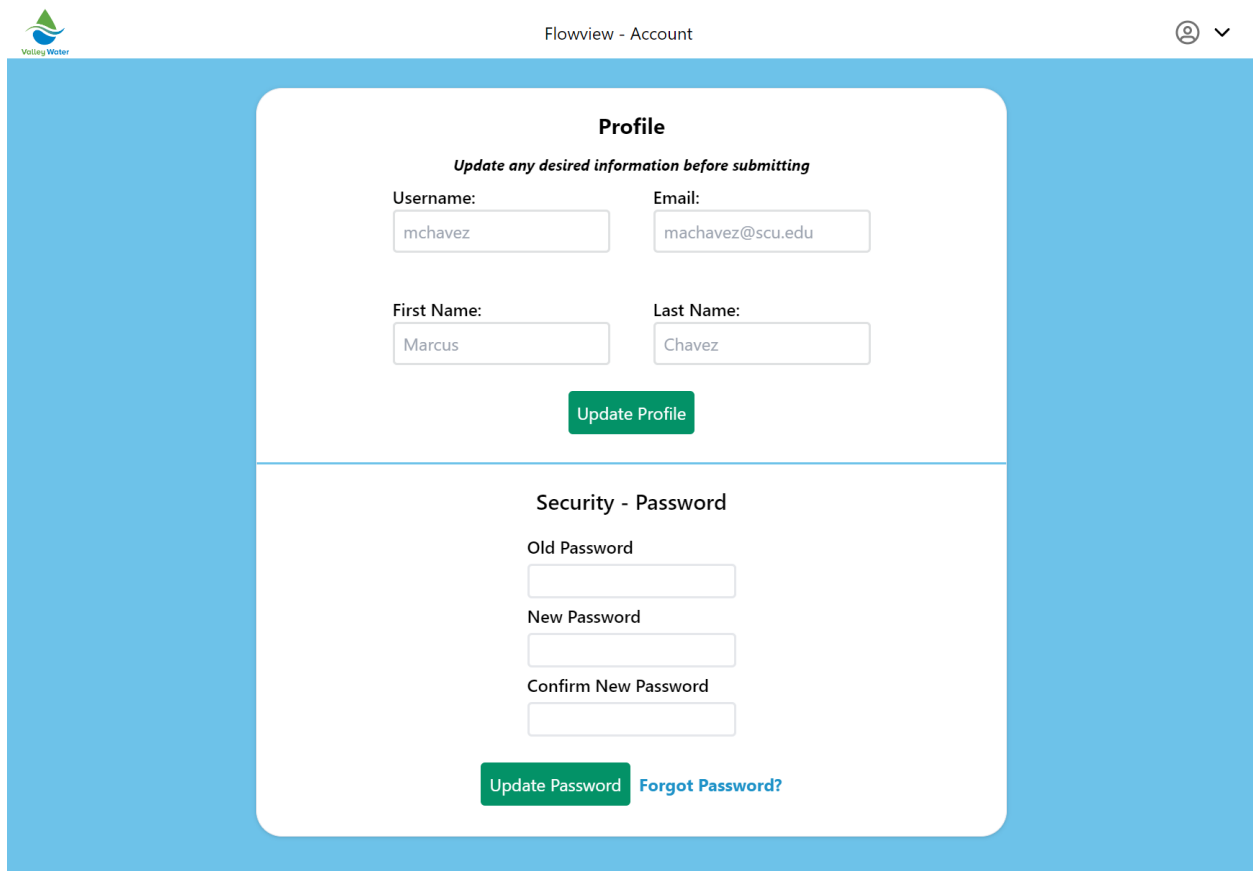
Sensor Name	Description	Last run at	Run count	Enabled	Interval
Rain 6128	Update Rain 6128 every 6 hours	Thu, 18 May 2023 02:08:05 GMT	16	✓	6 hours
Stream 5013	Update Stream 5013 every 6 hours	Thu, 18 May 2023 02:08:05 GMT	16	✓	6 hours
Stream 5016	Update Stream 5016 every 6 hours	Thu, 18 May 2023 02:08:06 GMT	14	✓	6 hours
Rain 6001	Update Rain 6001 every 6 hours	Thu, 18 May 2023 02:08:06 GMT	14	✓	6 hours

© 2022 FlowView, Santa Clara Valley Water District. All Rights Reserved

Figure 5.6: Resource Managers and Administrators can view and manage the Cron Jobs.

## 5.1.8 Change Profile

Figure 5.7 below shows the user interface for managing one's own user profile. In the top section of the form, a user may want to edit some information like their username or email address. The change profile form has conveniently been prepopulated with the current information so that the user may make informed decisions. They also do not need to enter values in all of the fields and may simply change only the information that they would like to change. Upon submitting the form and successfully changing their desired profile information, a user alert will be displayed showing a successful change, and the prepopulated data will reflect those changes as well. Lastly, there is a change password section, where the user may enter their original password, a new password, and a confirmation of that new password before submitting. Again, upon a successful password change, a user alert will be displayed to notify the user. If the user has forgotten their password, they may click on the forgot password hyperlink which will take them to the forgot password web page. There, they may enter their email address and receive an email secured with JWT (Json Web Token) and follow the instructions to change their password.



The screenshot shows the 'Flowview - Account' page. At the top left is the 'Valley Water' logo. At the top right is the text 'Flowview - Account' and a user profile icon with a dropdown arrow. The main content area is a white card with a blue border. The card is divided into two sections. The top section is titled 'Profile' and contains the instruction 'Update any desired information before submitting'. It has four input fields: 'Username' (pre-filled with 'mchavez'), 'Email' (pre-filled with 'machavez@scu.edu'), 'First Name' (pre-filled with 'Marcus'), and 'Last Name' (pre-filled with 'Chavez'). Below these fields is a green 'Update Profile' button. The bottom section is titled 'Security - Password' and contains three input fields: 'Old Password', 'New Password', and 'Confirm New Password'. Below these fields are two buttons: a green 'Update Password' button and a blue 'Forgot Password?' link.

© 2022 FlowView, Santa Clara Valley Water District. All Rights Reserved

Figure 5.7: Users may change their account settings.

## 5.1.9 Add Model

Figure 5.8 below shows the user interface for creating a new machine learning model. Any machine learning model will have its hyperparameters, which can be configured and changed in order to try to achieve the best predictions possible. Thus, the user interface offers a simple form on the right hand side that's prepopulated with the default hyperparameter values for users to fill in before submitting a new model. Note that this form does not require all fields to be filled, because a user may want to keep some of the default values or not even insert any new hyperparameters. If so, the user can submit, and this will generate the model and configuration ORM (Object Relational Mapping) objects which will be stored in MariaDB. It should be noted that the table holding the models merely contains basic information on the model such as its ID and time created. The table storing the models uses a foreign key to reference the configurations in another table that specifically holds the configurations. This usage of a foreign key is used to ensure that whenever a new model is created, there is always a configuration object present with it. Thus, that is why two ORM objects are created when instantiating a model. A user alert will be displayed upon successful model creation to let the user know that the model has been created, and that new model ID will be dynamically updated in the menu on the left hand side.

The screenshot shows the 'Add DANM Model' interface. On the left, a 'Current Models' box lists existing models. The main form contains the following fields:

Field	Value
STREAM SENSOR	Select a stream sensor
RAIN SENSOR	Select a stream sensor first
BASIN	Select a stream sensor first
BATCH SIZE	48
EPOCHS	50
LEARNING RATE	0.001
LEARNING RATE ADJUSTMENT POLICY	type4
TRAIN VOLUME	3000
START POINT	2015-01-01
TRAIN POINT	2023-05-27
TEST START	2021-06-08
TEST END	2022-01-20
TRAIN SEED	1010
VAL SEED	2007
HIDDEN DIM	384
CNN DIM	256
LAYER	1
INPUT DIMENSIONS	1
OUTPUT DIMENSIONS	1
INPUT LENGTH	1440
OUTPUT LENGTH	288
T. EPOCHS	8
EVENT FOCUS LEVEL	18
VAL. SIZE	120
OVERSAMPLING	80
R. SHIFT	0
WATERSHED	1

Buttons: Add Model (green), Clear Hyperparameters (red)

@ 2022 FlowView, Santa Clara Valley Water District. All Rights Reserved

Figure 5.8: Resource Managers and Administrators can train new machine learning models.



Another convenient feature is that if the user wants to copy an existing model's configurations, they may simply select a current model's ID, and a new popup will show that model's configurations, which they can view and potentially copy over into the form as seen in Figure 5.9 below. This feature is especially useful for saving a user's time if they do not want to re-input a series of hyperparameters and if they only desire to change a few hyperparameters.

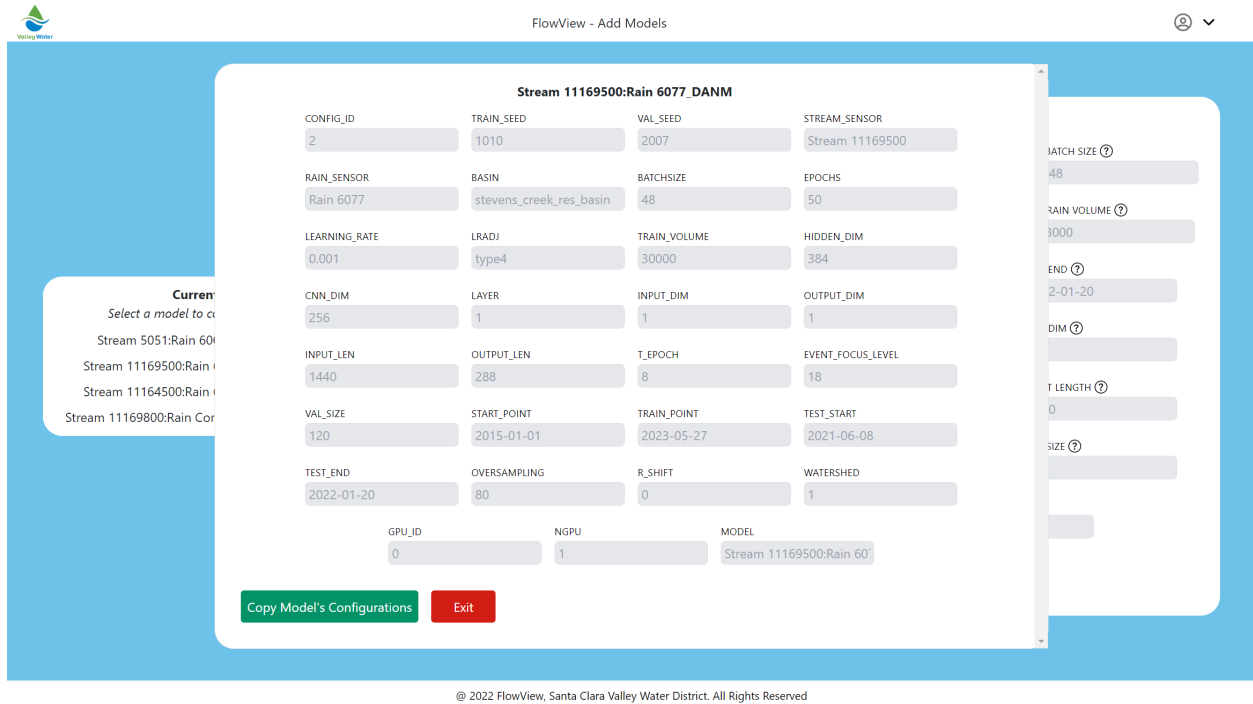


Figure 5.9: Resource Managers and Administrators can copy an existing model's configurations.

This is all that the user will see in the application, but on the backend, a series of actions are triggered. First, as mentioned, the database ORM objects for the model and configurations are created and stored into MariaDB. Then, a task is passed to the Celery workers via RabbitMQ in order to begin training the new model on the dataset. Notably, the dataset is always evolving as there are automated background jobs that are constantly pulling new sensor data every several hours. Once the model has finished training, it will automatically run its first inference job, and once its predictions have been made, the user who created the model is notified by email so that they may go and check the results. After the first inference job, Flask will automatically generate a cron job that periodically runs inference jobs and updates the predicted data displayed in the dashboard. All of these services are automatically handled by Flask, RabbitMQ, and Celery in order to offload work from Flask to the asynchronous Celery workers, which helps contribute towards the ease of use and abstraction for the users.

### 5.1.10 View Model

Figure 5.10 below shows the user interface for managing the machine learning models in FlowView. A user first selects a sensor for which they want to view trained models. All of the current models are populated into a table where information like the model type, training date, and configurations can be viewed. In addition, the root mean squared error value for the model's validation set is displayed to allow the user to easily compare between different models trained on the same sensor. The user can utilize the comparison between the RMSE values to make a decision on which model they would like to use for future inference jobs to make predictions.

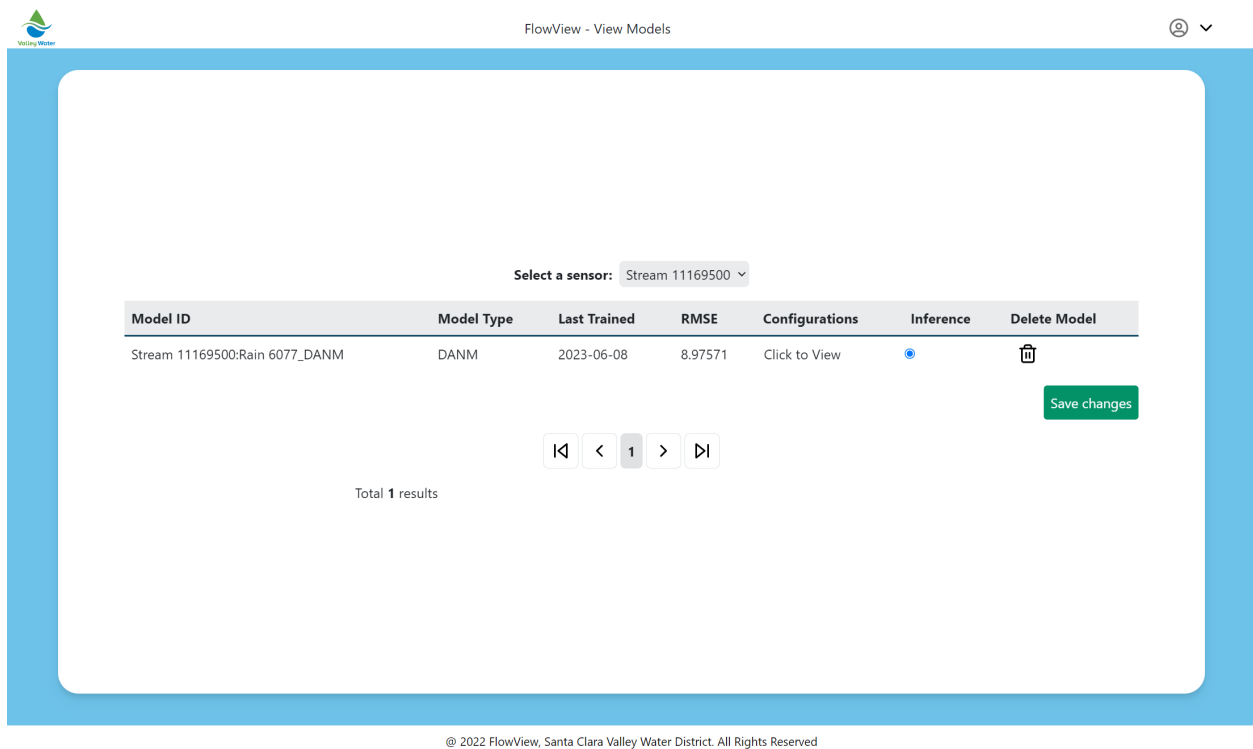
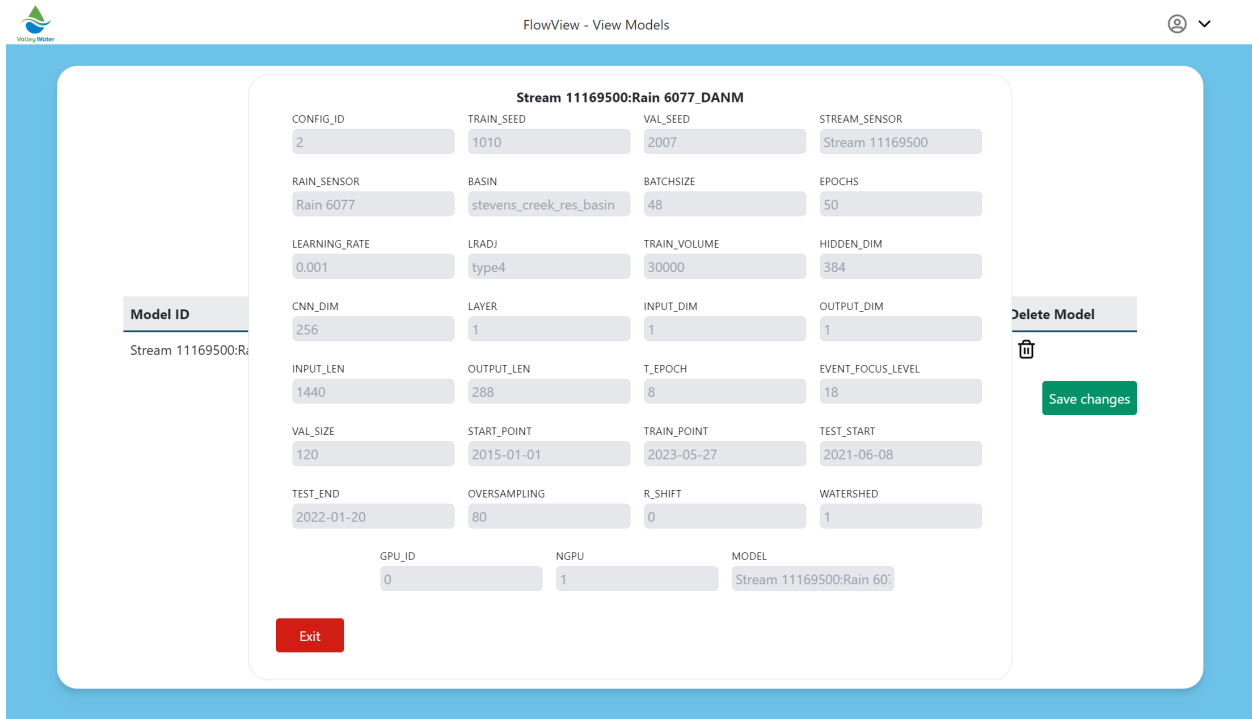


Figure 5.10: Resource Managers and Administrators can view and manage existing models.

To view the configurations for a model, the user will click on the “Click to View” button in the corresponding row, and the popup showing the configurations will be displayed as shown in Figure 5.11 below. Lastly, models can be removed from the application simply by clicking on the trash can icon in the corresponding row, and once this action is performed, the user will get an alert letting them know the model has been removed. Removing the model in the backend involves deleting the configuration ORM object first and then referencing the table storing the models in order to delete the model ORM object due to the foreign key constraint mentioned in the “Add Models” functionality. Overall, this functionality allows users to keep track of the models currently in use while abstracting away unnecessary details.



© 2022 FlowView, Santa Clara Valley Water District. All Rights Reserved

Figure 5.11: Resource Managers and Administrators can view an existing model’s configurations.

### 5.1.11 User Alerts

When a user submits an important action to FlowView such as adding a sensor, training a machine learning model, or changing a user's permissions, they will be notified with an alert on whether their action was successful. These alerts will be displayed at the bottom left of the user's browser window. Figure 5.12 below shows an example of a success alert when an administrator changes the permission of a user.

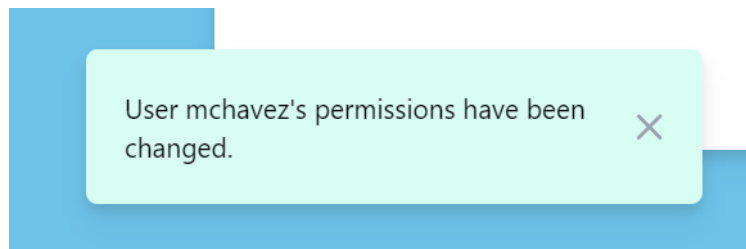


Figure 5.12: Success alert notifies the user after performing a significant action.

If a user submits a request that fails, it is important to properly notify the user of the failure of their action. Figure 5.13 below shows an example of an error alert when a user attempts to change the name of a sensor added in FlowView.

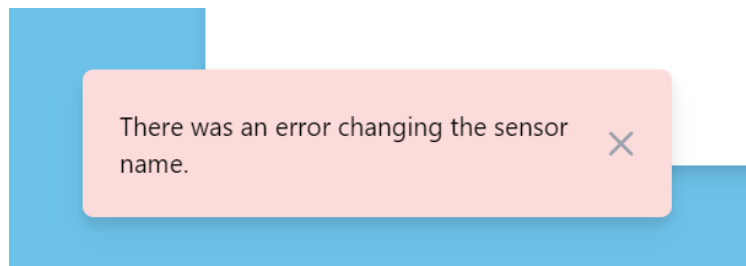


Figure 5.13: Error alert notifies the user after performing a significant action.

### 5.1.12 Email Notifications

As described in the architectural flow in Section 4.3, a user will receive an email upon the completion of their asynchronous request by a Celery worker. Email notifications are sent from the FlowView application Gmail account and contain relevant details about the user's completed request. Figure 5.14 below shows an example of an email notification for a user's request to add a stream sensor to FlowView. This email notification in particular reminds the user that a Cron Job has been automatically created to periodically refetch the data for the sensor that they added to FlowView.

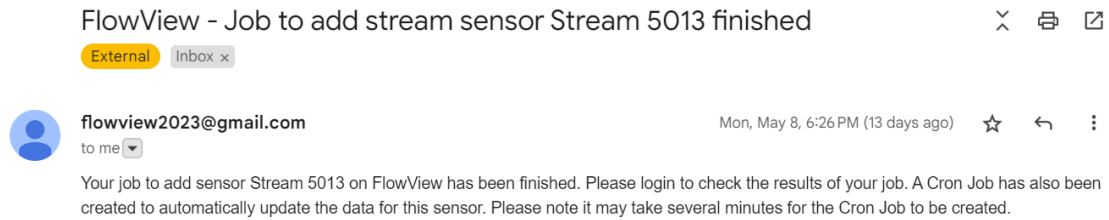


Figure 5.14: Email notifications are sent to the user upon completion of asynchronous requests.

## 5.2 Challenges

The following section provides a look into the difficulties of developing FlowView, and where we particularly struggled and persevered in order to complete the project. We discuss the issues we encountered when deploying with Docker in Section 5.2.1, the various problems with integrating the machine learning model in Section 5.2.2, and the data storage issues with our database in Section 5.2.3.

### 5.2.1 Deploying FlowView with Docker

When running FlowView as a multi-container application, the database (MariaDB) was its own service, an issue that initially occurred was that the MariaDB service would not persist its data once the application was torn down. This was an issue because for services like fetching sensor data, potentially millions of data points would be fetched, and if that data was not persisted, then that data would have to be re-fetched from Valley Water's APIs (application programming interface). The solution to this was to use Docker volumes, which is an independent file system that exists independently of the running containers and persists data. Docker volumes is an ideal solution because of its ease of management and the fact that it does not increase the size of the containers themselves.

Additionally, since each of the primary dependencies (i.e Flask, MariaDB, Tailwind, etc.) ran in its own container, Docker Compose relies on its own internal networking to communicate between the different services. However, this led us to reconfigure our entire Flask application in order to ensure that the different containers could properly network with each other and run the necessary services. This difficulty was notably seen when configuring the nginx web service with the SSL certificates, which had to be obtained with CertBot, which was also running as its own service.

### 5.2.2 Model Training

Running and integrating the machine learning model provided to us by Dr. Anastasiu's doctoral student proved to be a much more difficult task than we initially thought. Due to the model still being an in-development prototype, there was little documentation available to us. This required us to dig through the code in order to better understand it and integrate it into our system. Additionally, it required strong lines of communication with the student developing the model in order to fully understand the steps needed to run the model and get predicted data successfully. Beyond the minimum documentation, data processing was also required, as the model had previously been working with much cleaner and static datasets. This meant passing in real world sensor data by itself was not possible. The real-time sensor data being fetched from Valley Water's API's had to be pre-processed in a format that the machine learning model could take as input. Specifically, the machine learning model expects sensor data in even 15-minute intervals and does not work well with gaps in sensor data. Although most of the sensor data from Valley Water's API are structured in even 15-minute intervals, the data periodically changes frequency into 5-minute intervals or into uneven

15-minute intervals. Additionally, the sensor data from Valley Water's API periodically has gaps, which has to be filled in with empty values before being sent to the machine learning model.

Another issue with model training was getting the training working in Docker, which took a few additional steps of ensuring that our GPU would be visible from within the Docker image, even though the GPU existed outside of the containers. This model challenge further increased our understanding of machine learning models and pushed us to improve our communication skills as well in order to understand what would be required of us in order to run the machine learning model.

### **5.2.3 Data Storage**

Implementing a dynamic form of data storage was another challenge faced in the development of FlowView. In the early stages of development, the sensor data from Valley Water's API was stored in FlowView's database in an isolated manner where each sensor's data was separate from each other. However, as design discussions progressed with Valley Water, they expressed a need to be able to combine the data of multiple sensors. This feature can be seen in Figure 5.4 above where a user can add multiple external Valley Water sensors to an internal sensor being created in FlowView. In addition, they wanted to be able to freely move the start dates of sensors after they had already been added to the application. This feature is described in Section 5.1.3 above where the start dates of individual sensors can be modified. These dynamic needs required us to design a database schema that could effectively map the external sensors from Valley Water's API to the internal sensors stored in FlowView. More importantly, the database schema for FlowView had to be continuously updated to best fit Valley Water's changing needs.

# Chapter 6

## Societal Issues

In this chapter, we discuss the societal issues of concern for the development and maintenance of FlowView. In Section 6.1, we discuss the ethical concerns surrounding the security of FlowView as a web application. Specifically, this includes user security and privacy as well as defending from common web attacks such as cross-scripting attacks. Section 6.2 considers how FlowView can aid Valley Water in their mission to provide Silicon Valley with safe and clean drinking water in the face of droughts. Section 6.3 highlights how FlowView addresses concerns of web usability and anticipates how FlowView can provide assistive technologies to increase web accessibility for users with disabilities. Section 6.4 describes how we have changed as engineers to appreciate the process of learning in order to help our surrounding communities while developing FlowView. Lastly, in Section 6.5, we detail about how we sought to exhibit proper engineering character throughout the process of building FlowView.

### 6.1 Ethical

The primary ethical concerns for FlowView involve user security and privacy, specifically with regards to storing user sensitive information such as passwords. The MariaDB database utilized by FlowView uses strong passwords for both the root user and the default mysql user with lower privileges. In addition, user passwords stored in the database for FlowView are hashed to enforce the security of our application. If a malicious attacker were to login to the FlowView database, they would not be able to read any user passwords due to them being hashed rather than stored as plain text.

It is important that we implement these security measures as user passwords are the most sensitive information stored in FlowView's database. If an attacker were to read a user's password in FlowView, they could potentially use that password along with the user's username or email to access other websites where the user has re-used the same login information. This scenario would threaten the privacy of the user as highly sensitive information could be read by the attacker such as banking or medical information.

Another security concern that FlowView addresses are cross-scripting attacks, with Cross-site Request Forgery (CSRF) tokens as the primary measure to address this attack. Cross-scripting attacks involve an attacker injecting a



malicious script into a user's browser. The user's browser may then execute this script if there is no method for the browser to verify the authenticity of the script. The attacker could then read any sensitive information from the user's browser session such as cookies and session tokens via the malicious script. CSRF tokens address this attack and help verify that any user requests submitted to the server are authentic and can be trusted. It is important that we utilize CSRF tokens to protect against cross-scripting attacks as sensitive user information could be exposed otherwise.

## **6.2 Environmental Impact**

FlowView will aid Valley Water in their mission to “provide Silicon Valley safe, clean water for a healthy life, environment, and economy.” In that simple statement, it is clear that the environment is an important part of their mission, particularly in their stewardship of streams throughout Santa Clara county. The information that FlowView provides Valley Water enables their work with protecting and restoring habitats and endangered species. Having information about future predictions that they would not otherwise have access to allows Valley Water to plan out their environmental work. Ultimately, this helps to ensure that the ecosystems and habitats that Valley Water maintains will continue to thrive and support endangered species.

While FlowView will allow Valley Water to potentially have a very positive impact on the environment, it should be noted that the power drawn by FlowView, particularly by the accelerated GPU used to enable machine learning model training, could have a negative impact on the environment. As with any computer software that requires an intense load on the GPU, the power draw will potentially have a negative impact, though potentially negligible, on the environment, and as such should not be entirely ignored.

## **6.3 Usability**

An important issue to address is web usability for FlowView. It is important that FlowView presents information to the user in a concise and consistent manner so that there is no ambiguity or confusion when a user is reading important information such as sensor data or account information. FlowView currently implements these factors for web usability by using consistent table displays as well as consistent styling to ensure that the browser experience provided to the user is coherent.

Web accessibility is another important issue for FlowView to address, specifically with regards to ensuring that FlowView is accessible to individuals with disabilities. This can be accomplished by providing assistive technology such as text equivalents for images that can be accessed by screen readers. Currently, FlowView does not utilize any major images or graphics as the information is provided in a text format. However, if images or graphics are implemented into the browser experience in the future, then providing assistive technology towards the goal of web accessibility becomes a more pressing matter.

## **6.4 Lifelong Learning and Compassion**

Through the development of this project, our team further developed our senses of lifelong learning and compassion. Due to the full stack nature of FlowView, all members of the group were constantly learning and teaching themselves new technologies in order to assure that the project could be completed to the best of our ability. In completing the project, our team fostered lifelong learning and bettered our ability to do so, as we continually strove to understand and help on all aspects of the project, from front end development to integrating the machine learning model into our system. By overcoming the challenges before us through learning and teaching ourselves new skills, lifelong learning has become an integral part of our process when tackling new projects. Along the way, compassion not only helped our members be understanding of each other as they were learning new technologies, but aided us in communicating with Valley Water in order to create a final product that would better fit their needs and wants.

## **6.5 Engineering Character**

Through the development and creation of FlowView, our team sought to demonstrate proper engineering character. Proper engineering character means that we pursued the project in a way that would firstly do no harm, the most basic tenant of most engineering ethical obligations. Starting from that basic point, we continued to pursue the project focusing on building it for the reasons of protecting human rights, which in this case is mainly focused on the right of access to clean drinking water. Additionally, we took the extra steps to avoid the ethical pitfalls of our project, ensuring that our app was secure to avoid potentially harming users by failing to properly secure things like user data. As a result, we were able to demonstrate proper engineering character.

## Chapter 7

# Conclusion and Future Work

### 7.1 Future Work

While the initial prototype of FlowView was able to be completed within the defined constraints, there is still room for improvement. First, FlowView's UI (user interface) does not support mobile devices and also does not fully implement responsive design. Further work with this would include implementing responsive breakpoints and viewport scaling in the web pages to ensure that the webpage elements flexibly adjust based on the size of the device and window upon which the user is viewing the application.

While implementing mobile-friendly breakpoints for FlowView as a web application would allow greater accessibility for users accessing the application via their mobile web browsers, converting FlowView from a web application to a fully portable mobile application may provide greater flexibility for mobile users. This conversion would require much work due to having to design a new mobile architecture for FlowView. However, this work would open up exciting opportunities for Valley Water to access FlowView from a wider variety of devices.

Additionally, FlowView currently supports only one type of machine learning model, and that is still a prototype. As development with FlowView progresses, eventually there will be support for multiple types of models, which will require changes in the frontend UI and backend services. For this scenario, we have designed our codebase to be as modular and well-documented as possible so that additional models can be integrated. However, even with the current codebase, some work is still needed in generalizing some of the backend services to ensure smoother integration of future machine learning models.

### 7.2 What We Have Learned

Throughout this project, we have learned how to build an entire web application from scratch and take it all the way to a production environment. Each of us has learned new skills and applied them to fulfill the requirements set by the client while taking ownership of the features that we have designed and implemented. All of us were able to expand upon our creative skills when creating the frontend interface, as well as more efficiently design, write, and automate

various backend services. Lastly, we gained more valuable experience in integrating other collaborator's codebases in order to cohesively tie together different services to create an end-to-end product.

### **7.3 Why it is Important**

As California continues to suffer from continuous droughts, it is more important than ever to have predictive analysis that can analyze historical data to give a glimpse of what is to come in the future regarding the state's water resources. In addition, FlowView is a unique application in that many current solutions and utility softwares don't leverage machine learning, and this can give a further edge in predicting upcoming severe weather such as drought or flooding, which can save time, energy, and money. As a result, California's history of drought necessitates an underlying demand for knowledge and preparation, which FlowView can help satisfy.

# Bibliography

- [1] S. Jeschke, O. Pfeiffer, and H. Vieritz, “Using web accessibility patterns for web application development,” in *Proceedings of the 2009 ACM Symposium on Applied Computing, SAC '09*, (New York, NY, USA), p. 129–135, Association for Computing Machinery, 2009.
- [2] M. Phatak and V. Kamalesh, “On cloud computing deployment architecture,” in *2010 International Conference on Advances in ICT for Emerging Regions (ICTer)*, pp. 11–14, 2010.
- [3] S. Disawal and U. Suman, “An analysis and classification of vulnerabilities in web-based application development,” in *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 782–785, 2021.
- [4] M. A. Kunda and I. Alsmadi, “Practical web security testing: Evolution of web application modules and open source testing tools,” in *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, pp. 152–155, 2022.
- [5] B. N. Nakhuva and T. A. Champaneria, “Security provisioning for restful web services in internet of things,” in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 1–6, 2017.
- [6] H. Lee, D. Kim, and Y. Kwon, “Tls 1.3 in practice:how tls 1.3 contributes to the internet,” in *Proceedings of the Web Conference 2021, WWW '21*, (New York, NY, USA), p. 70–79, Association for Computing Machinery, 2021.
- [7] Akanksha and A. Chaturvedi, “Comparison of different authentication techniques and steps to implement robust jwt authentication,” in *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, pp. 772–779, 2022.
- [8] X. Likaj, S. Khodayari, and G. Pellegrino, “Where we stand (or fall): An analysis of csrf defenses in web frameworks,” in *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses, RAID '21*, (New York, NY, USA), p. 370–385, Association for Computing Machinery, 2021.
- [9] W. Rankothge and S. M. Randeniya, “Identification and mitigation tool for cross-site request forgery (csrf),” in *2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC)*, pp. 1–5, 2020.
- [10] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, “Comparison between relational and nosql databases,” in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 0216–0221, 2018.
- [11] S. Tongkaw and A. Tongkaw, “A comparison of database performance of mariadb and mysql with oltp workload,” in *2016 IEEE Conference on Open Systems (ICOS)*, pp. 117–119, 2016.
- [12] C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, “A comparative study: MongoDB vs. mysql,” in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 1–6, 2015.
- [13] K. Chaowvasin, P. Sutanchaiyanonta, N. Kanungsukkasem, and T. Leelanupab, “A scalable service architecture with request queuing for resource-intensive tasks,” in *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 67–70, 2020.

- [14] N. Basavaraju, N. Alexander, and J. Seitz, "Performance evaluation of advanced message queuing protocol (amqp): An empirical analysis of amqp online message brokers," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–8, 2021.
- [15] A. Pathak and C. Kalaiarasan, "Rabbitmq queuing mechanism of publish subscribe model for better throughput and response," in *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–7, 2021.
- [16] G. Fu, Y. Zhang, and G. Yu, "A fair comparison of message queuing systems," *IEEE Access*, vol. 9, pp. 421–432, 2021.
- [17] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems, DEBS '17*, (New York, NY, USA), p. 227–238, Association for Computing Machinery, 2017.
- [18] A. Verma, C. Kapoor, A. Sharma, and B. Mishra, "Web application implementation with machine learning," in *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*, pp. 423–428, 2021.
- [19] N. B. Ahmat Baseri, J. A. Bakar, A. Ahmad, H. Jafferi, and M. F. Zamri, "Smvs: A web-based application for graphical visualization of malay text corpus," in *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 30–35, 2020.
- [20] A. Singh, R. Akash, and G. R. V, "Flower classifier web app using ml & flask web framework," in *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pp. 974–977, 2022.
- [21] A. Lakshmanarao, M. R. Babu, and M. M. Bala Krishna, "Malicious url detection using nlp, machine learning and flask," in *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, pp. 1–4, 2021.
- [22] C. A. Trianti, B. Kristianto, and Hendry, "Integration of flask and python on the face recognition based attendance system," in *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, pp. 164–168, 2021.