Water Utility Sensing Device

CSE 123B Design Document

Tim Kraemer, Mitchell Tansey, Kyven Wang, Gary Mejia-Martinez, Tirath Shah, Darren Yu, Gustavo Fonseca

Computer Science and Engineering
University of California, Santa Cruz
Baskin School of Engineering

Contents

1 Introductional Overview **3**
1.1 Executive Summary . . . 3
1.2 Ethics Statement . . . 3
1.3 Sustainability Statement . . . 4
1.4 Personas . . . 4
1.5 Problem Definition . . . 5
1.5.1 Need & Goal Statement . . . 5
1.5.2 Design Objective . . . 5

2 Detail of Design **5**
2.1 Overview of Entire System . . . 5
2.2 Sensing Device + Home Hub Design . . . 5
2.2.1 General Overview . . . 5
2.2.2 Diagrams . . . 6
2.2.3 Technologies Used and Software Design . . . 7
2.2.4 Hardware Design . . . 8
2.3 Remote Web Server Design . . . 8
2.3.1 General Overview . . . 8
2.3.2 Diagrams . . . 9
2.3.3 Technologies Used & Software Design . . . 9
2.3.4 Security and Encryption . . . 11
2.4 Mobile App Design . . . 13
2.4.1 General Overview and Application Architecture . . . 13
2.4.2 Diagrams . . . 13
2.4.3 Technology and Language Choices . . . 13
2.4.4 Log-in View . . . 14
2.4.5 Sign-Up View . . . 14
2.4.6 Home View . . . 14
2.4.7 Mobile Device to Server Communication . . . 15
2.5 Setup and Usage Details . . . 15
2.5.1 Base Station Setup . . . 15
2.5.2 Sensor Setup . . . 16
2.5.3 Sensor Usage and Maintenance . . . 16
2.6 Water-Proofing & Life-Proofing . . . 16
2.6.1 Water-Proofing . . . 16
2.6.2 Life-Proofing . . . 16
2.7 Simulation & Testing . . . 17
2.7.1 Hardware . . . 17
2.7.2 Mobile App . . . 17
2.7.3 Web Server . . . 17
2.8 Future Additions and Upgrades . . . 17
2.8.1 Integration with Additional Sensors . . . 17

3 Evaluation **18**
3.1 Resulting Functional Prototypes . . . 18
3.1.1 Overview . . . 18
3.1.2 Results . . . 19
3.1.3 Prototype Renders . . . 20
3.1.4 Prototype Images Version 1 . . . 22
3.1.5 Prototype Images Version 2 . . . 23
3.1.6 Final Prototype Drawing . . . 24
3.1.7 Prototype Hardware Images . . . 25
3.1.8 Prototype Web Server Screenshots . . . 26
3.1.9 Prototype App Screenshots . . . 27

3.1.10 Prototype Test Environment Images . . . 28

4 Appendix 1 - Problem Formulation **29**
4.1 Conceptualizations . . . 29
4.2 Brainstorming . . . 29
4.3 Decision Tables . . . 29
4.4 Morphological Charts . . . 30

5 Appendix 2 - Planning **31**
5.1 Basic Planning . . . 31
5.2 Gantt Chart . . . 31
5.3 Critical Path Analysis . . . 31
5.4 Division of Labor . . . 31
5.5 Collaboration . . . 32
5.5.1 Sensor, Home Hub Hardware and Software . . . 32
5.5.2 Web Server and Database System . . . 32
5.5.3 IOS App . . . 32
5.5.4 3D Modeling and Case Printing . . . 32

6 Appendix 3 - Test Plan & Results **33**
6.1 Introduction . . . 33
6.2 Test Objectives . . . 33
6.2.1 Introduction . . . 33
6.3 Test Scope . . . 33
6.3.1 Test Parameters . . . 33
6.4 Test Environment . . . 33
6.5 Test Schedule . . . 34
6.6 Test Cases & Results . . . 34
6.6.1 Water Recording Accuracy Test . . . 34
6.6.2 Server/App Update Speed Test . . . 34
6.6.3 Leakage and Waterproofing Test . . . 34
6.6.4 Sensor Connectivity and Uptime Test . . . 35
6.6.5 Power Consumption and Lifespan Test . . . 35
6.7 Analysis . . . 36

7 Appendix 4 - Review **37**
7.1 Tim . . . 37
7.2 Mitchell . . . 37
7.3 Gary . . . 37
7.4 Kyven . . . 37
7.5 Darren . . . 38
7.6 Tirath . . . 38
7.7 Gustavo . . . 38

1 Introductional Overview

1.1 Executive Summary

Problem Statement: In large households, it's easy to overlook individual water consumption, leading to inefficiencies and increased utility costs. Current methods of water consumption tracking are insufficient, with only one water meter per house, water bills simply providing total usage, and no ability to keep track of individual water sources. There's a clear need for localized water tracking at the faucet level to promote water conservation and utility cost savings.

Proposed Solution: We aim to address this need by introducing a device that attaches to faucets, tracking and storing various water consumption statistics. Through this innovation, users can potentially save money by reducing utility costs and minimizing their environmental impact. The device aims to provide users with easy and accurate tracking of current and average water usage from different sources within their home, along with leak and over-usage notifications.

Industry and Market: The market for water consumption tracking devices predominantly caters to industrial and commercial sectors, where large-scale water management systems are essential for operational efficiency and regulatory compliance. However, there is a significant gap in the market when it comes to products specifically designed for home use. This absence of products catering to localized water tracking in residential settings presents a significant market opportunity. With growing concerns about water scarcity, rising utility costs, and environmental sustainability, there is increasing consumer interest in tools and technologies that enable efficient water management at the household level. A user-friendly, cost-effective faucet attachment capable of tracking and analyzing water consumption stands to fill this gap and meet the evolving needs of environmentally-conscious consumers.

Benefits:

Utility Cost Savings: Users can take proactive steps to reduce their utility bills by accurately tracking water consumption and identifying inefficiencies.

Environmental Impact Reduction: Through effective water conservation measures, users can minimize their environmental footprint, contributing to sustainability efforts.

Convenience and Accuracy: The device provides convenient access to detailed water usage data, allowing users to make informed decisions about their consumption habits.

Leak Detection: The device offers leak detection capabilities, alerting users to potential issues and enabling prompt repairs to prevent water wastage.

Conclusion: Our water consumption tracking faucet attachment offers a practical solution to address the challenges associated with household water management. By empowering users with accurate data and actionable insights, we aim to promote water conservation, utility cost savings, and environmental stewardship.

1.2 Ethics Statement

Our team is committed to upholding the highest ethical standards in creating a safe and reliable water meter faucet attachment. We believe that ethical considerations are vital in all engineering endeavors, especially those that will impact the consumer's health and safety. We will be honest and transparent about the capabilities and limitations of our device while providing sufficient instructions on how it works and how data is collected. This device will be accessible to all users regardless of their physical abilities or technological experience. Our environmental footprint is our top priority, and we will not cut any corners when it comes to manufacturing. We will minimize our energy consumption, carbon footprint, and waste production in order to promote sustainability. Individually, we hold ourselves to high standards with the utmost honesty, integrity, and professionalism at all times. This also means

that we invite all forms of feedback and constantly seek to improve our devices. Following these principles, we plan to create a product that not only adheres to the technical requirements of our capstone project but also serves the best interests of the consumers.

1.3 Sustainability Statement

We are committed to engineering a system that addresses our customers' needs and aim to be sustainable for both customers and planet. Our product will be made of recycled materials to reuse what we as a society have already use. Servers should aim to use enough power to not contribute to our planet's green house emissions.

1.4 Personas

John Drab, 40: Common Homeowner

- Husband and Father of 2
- Lives in the Bay Area, California: High utility costs and drought prone
- **Goal**: Wants to be catch unnecessary water usage at home, track water usage in a large family.
- **Key Concerns**: Intuition, easy to use, wants quick/real-time updates

Bob Kelsey, 52: Landlord/Property Manager

- **Goal**: Motivated to provide high-quality living experience while maximizing sustainability
- **Concerns**: Upfront costs, user-friendliness for tenants and himself

Steve Peterson, 35: Plumber

- Most common issue called for: Unknown water usage
- **Goal**: Provide reliable plumbing solutions, educate clients on water conservation
- **Challenges**: Limited time to address all clients concerns, space constraints for specific tools
- **Key Concerns**: Desires a simple, and detailed "hub" to track water usage effectively throughout the house

1.5 Problem Definition

1.5.1 Need & Goal Statement

It's easy to forget how much water you consume in a large household, and current methods of water consumption tracking are two general (only one water meter per house). There is a need for localized (faucet/singular appliance) water tracking to conserve water and save money on utilities.

We propose a device that attaches to faucets which tracks and stores several statistics. The goal is for users to potentially save money through reduced utility costs and also reduce their environmental impact. This would be done through the ability to more easily and accurately track current and average water usage from different sources within their home as well as leak/over-usage notifications.

1.5.2 Design Objective

Design an inexpensive faucet attachment that can use flow rate and temperature to measure water utility usage, localized with each individual faucet/appliance in a household, and present findings in a simple and intuitive manner to consumers through a web-app interface. In addition design methods to notify and provide tips for water consumption based on recent data.

2 Detail of Design

2.1 Overview of Entire System

The product consists of 3 major hardware and software components: The sensor + home hub, the web-server, and the mobile app. The sensor consists of a standard hall effect mechanism that measures water flow based on the speed a fluid turbine spins in the path of the flow. This data is collected by a micro controller on a custom form-fitted PCB in the device, which pre-processes data and sends packets to a home hub system using a secure local connection. Data is then sent to a remote web server to be polled by the user's app.

2.2 Sensing Device + Home Hub Design

2.2.1 General Overview

The hardware component constitutes 2 separate sections, the IoT sensing device, and the central home hub. The sensing device is attached to the underside of a sink unit, using standard threading and attachment points (US standard is 1/2" MtM or FtF). Data is collected and prepackaged and sent to the hub through a secure local short range communication method. The package is further processed and sent directly to the remote hosted web server for storage.

2.2.2 Diagrams

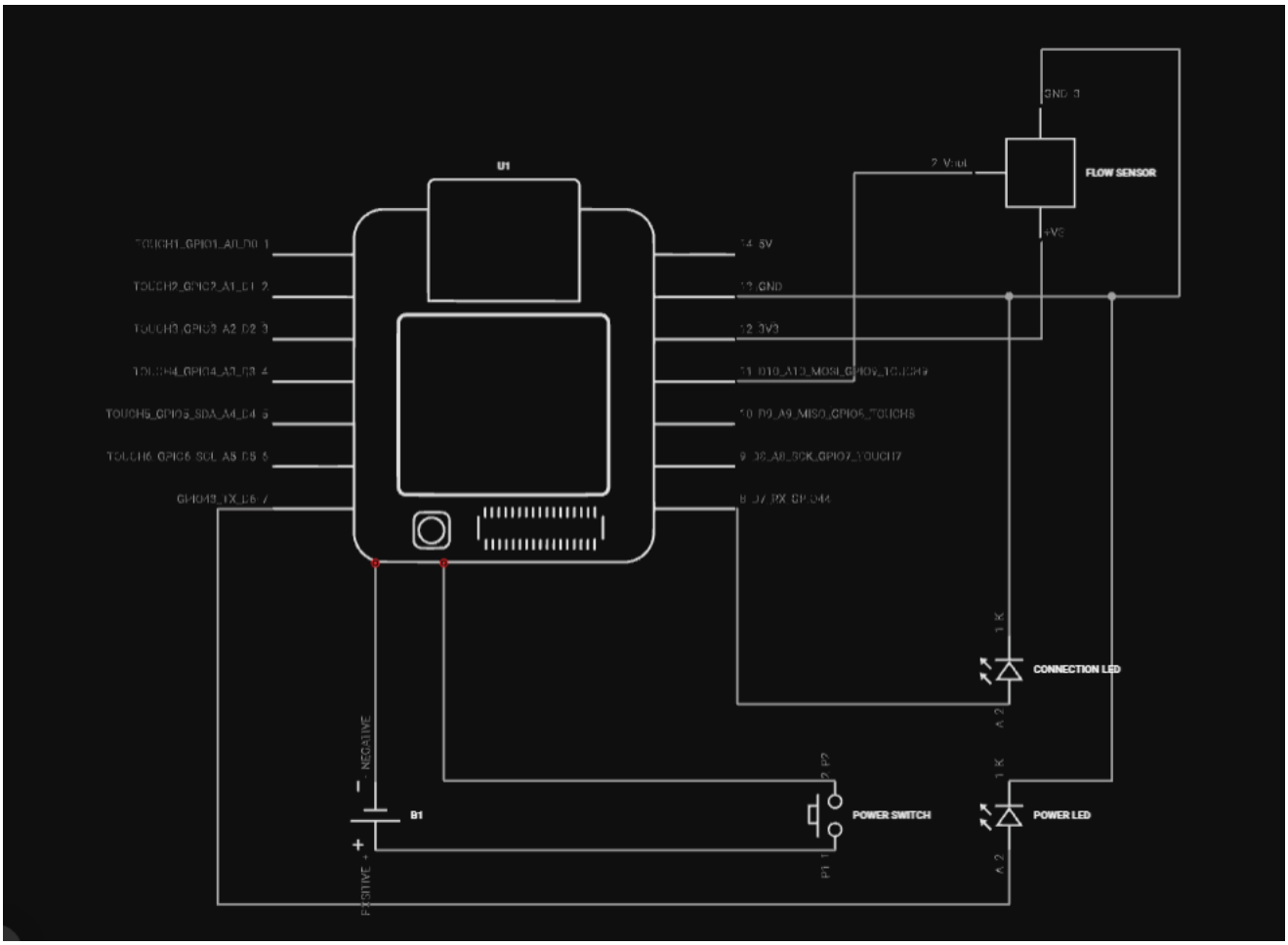


Figure 1: Sensing Device Wiring Diagram

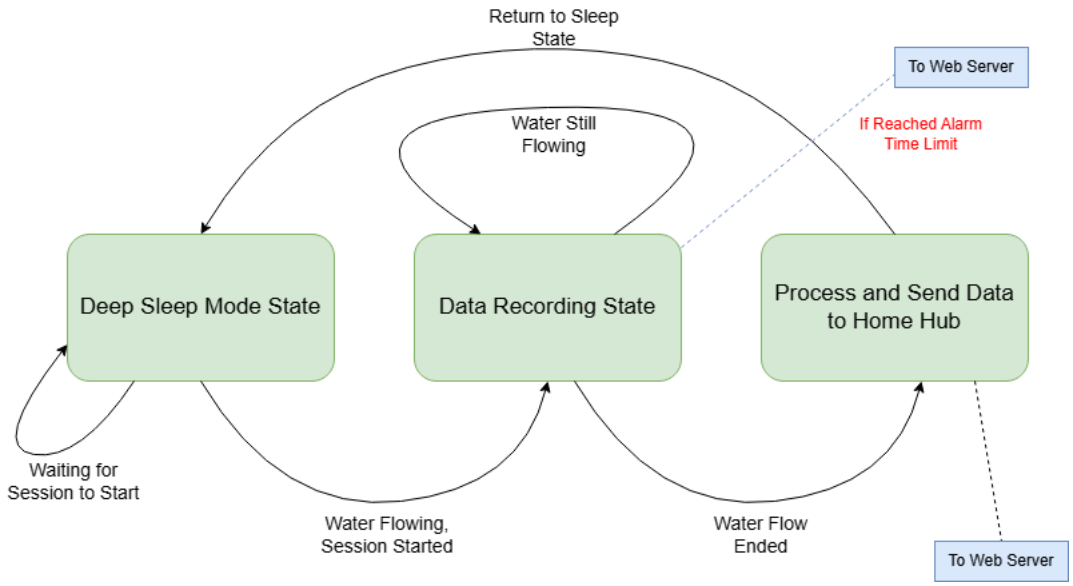


Figure 2: Sensing Node State Transition Diagram

2.2.3 Technologies Used and Software Design

The software side of the hardware design is integral to leveraging the full potential of the microcontroller and ensuring seamless interaction with the water flow sensor. This section details the development environment, programming languages, libraries, and technologies employed in crafting the microcontroller firmware. The code for the microcontroller was made completely with C++ through the Arduino IDE. This makes it easy to flash code and upload any bug fixes/additions that may be required. Upon start-up of the microcontroller, the power LED turns on, and it begins to ping the home wifi, attempting to connect to it. All wifi credentials are sent from the Home Hub using a Wi-Fi Provisioning ESP-IDF library. This library uses the BLE protocol to transfer the wifi credentials in JSON format. The microcontroller then deserializes the data and runs a wifi connection function. Once connected, the microcontroller will enter the next stage of connecting to the MQTT broker on the Home Hub device. Once connected, the connection LED turns on, and the microcontroller is ready to process any incoming data from the flow sensor. The microcontroller reads data every 1 second, waiting to sense the first reading above right; a session starts when the flow rate exceeds 0. These sessions are how we collect data on the entire usage period from a water supply. It begins when the water starts to flow and ends when the water supply is no longer being used. When a session starts, each data reading is divided by 60 and added to a cumulative summation that tracks all data points during that session. It also tracks the start and end times to calculate the session's duration. Once the session has ended, the microcontroller publishes the duration and total water usage to the MQTT broker.

The Home Hub maintains and runs the MQTT broker, which is all written in Python using the MQTT "paho" library. The Hub's main purpose is to receive data, package it, and send it to the server. On the side, the Hub also connects and helps maintain the microcontroller's connection to the home internet. Our hub uses a MQTT IoT messaging protocol to send and receive data. The MQTT broker serves as a one-way middleman that allows for data to be published from the microcontroller, then processed and sent to the main server. Upon start-up of the hub, a bash script runs that starts the broker and puts it into a constant loop that continuously listens for publishes. When a session is over, the MQTT broker will receive the packaged data from the microcontroller and parses everything respectively. It then uses the RPi "DateTime" library and the received duration to determine when the session started and ended. After that, it packages the start time, end time, used water amount, API key, session ID (which increments every session), date, and sink ID. That packet is then sent to the main server using a simple POST request.

2.2.4 Hardware Design

The core of our hardware design is the microcontroller, a critical component responsible for controlling and processing data from the water flow sensor. This section details the selection, integration, and functionalities of the microcontroller in the project. Our final design consists of a microcontroller with only a few requirements: Wifi/Bluetooth connectivity, 2 GPIO pins, 1 ADC pin, low power consumption, and external power supply capabilities. Wifi is used to connect to the hub, GPIO pins are used to power the LEDs, the ADC pin is used to read the flow sensor, low power consumption is used to extend battery life, and external battery capability is required for complete wireless design. The main role of the microcontroller is to interface with the flow sensor. The sensor outputs a pulse signal proportional to the water flow rate, which is fed directly into the ADC input. The microcontroller then converts the data into the correct units and calibrates to account for the sensor's error. Finally, it sends the data to the home hub. The circuit, as seen in Figure 1, is quite simple. The microcontroller operates at around 3.3V, which we satisfy using 3 AAA batteries. The main power for the microcontroller is controlled using a simple slide switch connected to the positive terminal of the microcontroller and the ground of the batteries. The LED's are connected to GPIO pins and the microcontroller's ground. The water flow sensor is connected to the 3v3 power pin, the ground pin, and an ADC pin.
Outside of the microcontroller, the home hub is also a vital component of the hardware design. The home hub is simply a Raspberry Pi 4 that runs the MQTT broker as mentioned in the previous section. The RPi4 requires a wall outlet to run and an ethernet cable to provide internet. We also allow for wifi connection via the app and scanning of the QR code, eliminating the option of an ethernet cable if the consumer wishes. Since the hub doesn't need to be maintained at all, it can remain plugged into the cables and run continuously.

2.3 Remote Web Server Design

2.3.1 General Overview

The remote webserver is a secure destination for incoming sensor data. Data is is passed through registered API endpoints and stored securely in a connected database, ready to be accessed by the user through a web-app. The webserver includes endpoints for registering new users, logging in, registering and removing sensing devices, requesting graphical and sensor data, etc.

2.3.2 Diagrams

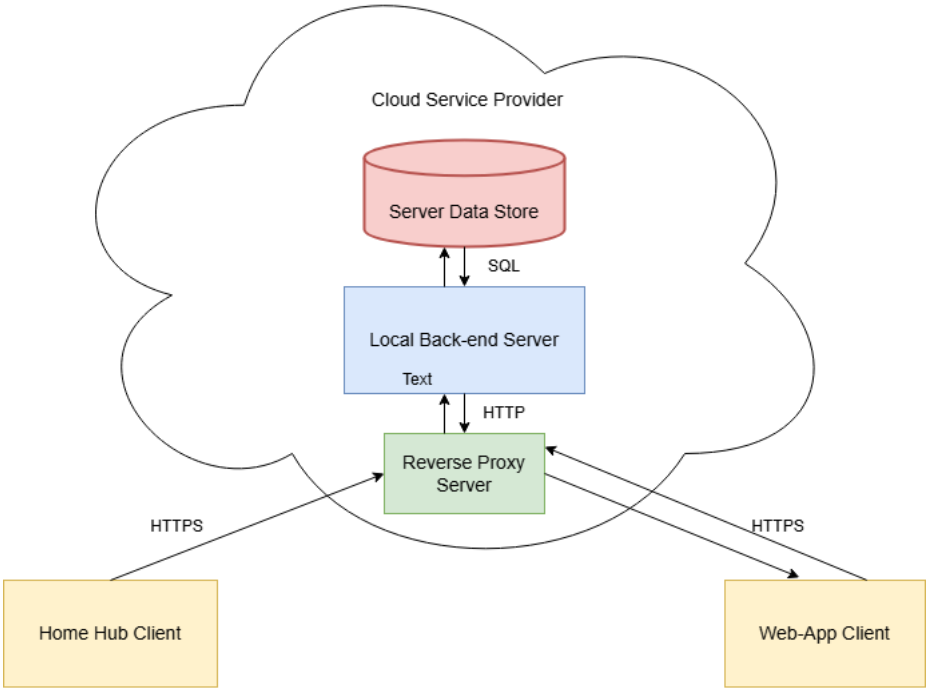


Figure 3: Cloud-based Webserver Overview

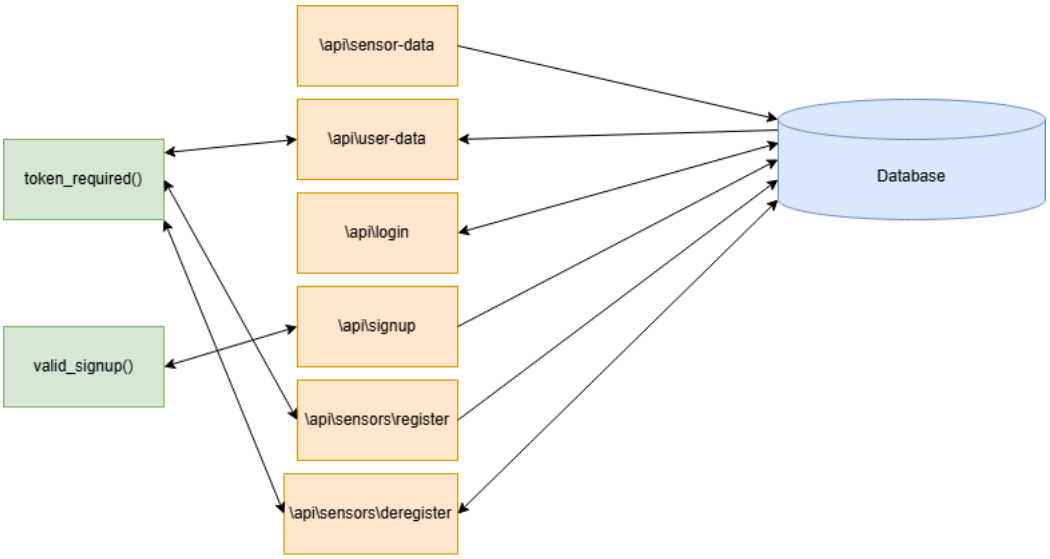


Figure 4: Back-End Server Routing Server Design

2.3.3 Technologies Used & Software Design

The webserver is composed of 4 important blocks: the cloud hosting platform, the reverse server proxy, the backend routing layer, and the database connection.

- **Cloud Hosting Platform**: The system will utilize a remote cloud hosting platform to host the server infrastructure. This platform will be assigned a dedicated IPv4 address, enabling

web domain pointing and DNS configuration. Access to the server infrastructure will be secured using SSH (Secure Shell) protocol, restricted to authorized personnel. Authentication and access control mechanisms will ensure that only registered engineers can establish secure connections to manage and maintain the server. In the case of our prototype, AWS LightSail made a perfect platform for prototyping and testing, with a domain provided by SquareSpace.

- **Reverse Server Proxy**: Built ontop of the Cloud Platform, a reverse proxy system acts as an intermediary for requests between clients and the backend infrastructure of the webserver. It forwards client requests to the appropriate resource and returns the server's response to clients (in this case the web-app and home hub). It handles the decryption of HTTPS requests using an SSL certificate, verifying via handshake, and forwards the simplified HTTP request internally to the backend. In the case for our prototype, we use NginX as a safe and reliable reverse proxy, accompanied by an SSL certificate from Let's Encrypt.

- **Backend Routing Layer**: The Routing Layer and backend software is the core of of the remote webserver. It includes the subroutes that dictate how to pass data, register users, log them in, etc. Below is a simple yet intuitive description of each required route. All routes follow RESTful API conventions:

endpoint Name	**Description**	**Request Type**	**Return Codes**
/api/sensor-data	API endpoint for sensor data from home hub. home hub provides registered API key for verification. if key expired, reissue new key.	GET	403 (bad key), 201 (success)
/api/user-data	API endpoint to request user data from sensors. Request uses URL arguments for upper and lower date ranges, as well as the requested sensor location.	GET	401 (invalid), 200 (success)
/api/user-data/graph	API endpoint to request data for the graphical purposes, returning total water usage per day. URL arguments are used for upper and lower date ranges.	GET	401 (invalid), 200 (success)
/api/login	API endpoint for user login. Accepts POST requests with email and password, which are validated against the database, including password hashing and comparison. Creates a jwt token with user info embedded and sends it back.	POST	400 (could not verify), 201 (success)
/api/signup	API endpoint for new user registration. Accepts POST requests with username, email, and password. Checks for existing accounts and stores new user info with a UUID and hashed password.	POST	202 (user already exists), 201 (success)
/api/sensors/register	API endpoint used to register a new sensor. Data collected from QR code includes sensor id number, which is registered to user id.	POST	409 (already registered), 201 (success)
/api/sensors/deregister	API endpoint used to de-register a new sensor. Same concept and construct of register route.	DELETE	409 (sensor doesnt exists), 201 (success)

/api/sinks	API endpoint that returns a list of currently active sinks to a user. Used by the app to	GET	401 (invalid), 201 (success)

In addition to these routes, 2 wrapper functions were written to assist common functions:

Function	**Description**	**Routes Required**
token_required	Function that must be used whenever user data must be verified, such as during data request and sensor registering.	user-data, user-data/graph, register, de-register
valid_login	Function that is run when a new user registers. It is simply a server side sanity check for password validity (good length, correct characters). Uses simple regex expressions to check both email and password for valid format.	signup

For the general final design, these are the only endpoints that are needed in the webserver to provide basic functionality to the user.

In the case of our prototype, we used a simple python routing layer called Flask, which provides a relatively accessible and fast way to develop a backend with straightforward syntax.

- **Database System**: The database system is the backbone data warehouse that associates data to users, stores encrypted sensitive information, and connects registered devices to accounts. A simple database management system such as PostgreSQL (used in prototype), MongoDB, MySQL, etc can be utilized in conjunction with the backend routing service as long as it is fast and reliable (no data loss).

Table Name	**Description**	**Attributes**
Sessions	Table which stores session information including amount of water, date and time, as well as sensor and sink ID for identification.	session_id, sensor_id, sink_id, water_amount, duration, start_time, end_time, date
Graph	Table which stores cumulative data for graphical purposes. Includes cumulative water amounts for each user on each day for the past 30 days.	user_id, cumulative_sum, date
Sensors	Table which stores data on which sensors are registered to which user. Includes sensor id and the accompanying sink id.	user_id, sensor_id, sink_id
Users	Table which stores data on users registered with the service, including name, email and password hash.	public_id, name, email, password_hash
Keys	Table that stores registered API keys linked to in use home hubs. Includes the key sequence and the issue date.	key, issue_date

Please see diagram in 2.3.2 for detailed look at attribute dependencies.

2.3.4 Security and Encryption

Many parts of the web server handle sensitive information pertaining to the users, sensor information, and API key information. In this section we look in-depth at the security layer that is found on the web server which aims at protecting sensitive information from falling into the hands of malicious actors.

- **SSH**: SSH on the development side is crucial to restricting access to the back-end and routing layer such that only registered engineers can access them. Engineers provide a public key generated for their work station, which is then added on the cloud platform as an authorized user. SSH'ing is as simple as follows:

```
$ ssh ubuntu@cse123-flowsensor-server.com
```

- **Nginx & SSL Certificates**: Any reputable web server or website employs SSL certificates and mandates that requests are sent via HTTPS *(Hypertext Transfer Protocol Secure)*. SSL certificates, issued by a Certificate Authority, verify the identity of the web server to incoming traffic, ensuring clients that they are communicating with the authentic server. Although we manage all requests internally for our own security, maintaining this feature is crucial. We utilize Nginx as a proxy to handle all necessary handshakes with the request source and manage all HTTPS requests.

- **Password Hashing**: User passwords are sensitive information. Together with their email, bad actors can access a users account and view their sensitive information. Whenever a new user registers an account with the web server, the web server will take the plaintext password and convert it into a "random" string of characters, adding salt (a random string) to the plaintext password so no two hashes of the same password are the same. Adding salt to hashed passwords protects users against exploits such as pre-computed rainbow tables. When a user logs in, the web server fetches the user's hashed password and salt, and compute the hash of the login password with the given salt and compare the two hashes. In our prototype, we use a python framework named **Werkseug** which provides us with the necessary tools for password storage and verification.

- **Home Hub API Keys & Reissuing**: Data sending endpoints are vulnerable entry points to our system, which means that only verified and registered Home Hubs can send data to our web server. When a hub receives user information (from the mobile app via Bluetooth), it generates a public/private key pair. The private key is stored internally and encrypted for extra protection. The public key is sent to a hub registration endpoint alongside its hub ID, which the web server cross checks with a database of authorized ID numbers. The web server stores the information and generates an API key which it stores in its local database and sends to the hub to store. Any subsequent communication with the web server must include this API key. Periodically, the web server will also reissue a new API key for the hub to use.

- **JWTs (*JSON Web Tokens*)**: JWTs are compact, URL-safe means of representing claims between two parties. Primarily, they are used to validate a user accessing restricted material and endpoints on an application. When logging into our web app, the server returns a custom JWT to the user, which includes information such as the user ID. The general breakdown of a JWT is as follows:

```
HEADER
{
    "alg": "HS256",
    "typ": "JWT"
}
PAYLOAD
{
    "user-id": "123456789",
    "username": "John-Doe"
}
```

  The JWT contains a header, then a payload, then a signature which is created using the server's private JWT key to verify server authenticity by the user. All of this is encoded and sent as one long character string. The JWT is checked and cross-referenced when any subsequent request by the user to an API endpoint is made which requires JWT authorization.

- **Database Security**: The web server's local database contains sensitive information that we don't want just anybody to access, even if some information is encoded. Most database management systems have options to set up usernames and passwords, which PostgreSQL has. Simply a strong username and password must be created, which is then stored securely in a .env file on the system, possibly encrypted. Individuals must also be granted read and write access to the .env file to be able to access it.

2.4 Mobile App Design

2.4.1 General Overview and Application Architecture

Our product requires users to receive immediate alerts away from home to be effective. Smartphones allow for these alerts to be received and interface with the system. Our mobile application consists of three main views or pages: the log-in view, sign-up view, and home view. Below is a diagram of the application frontend architecture and will be explained in their own respective subsection.

2.4.2 Diagrams

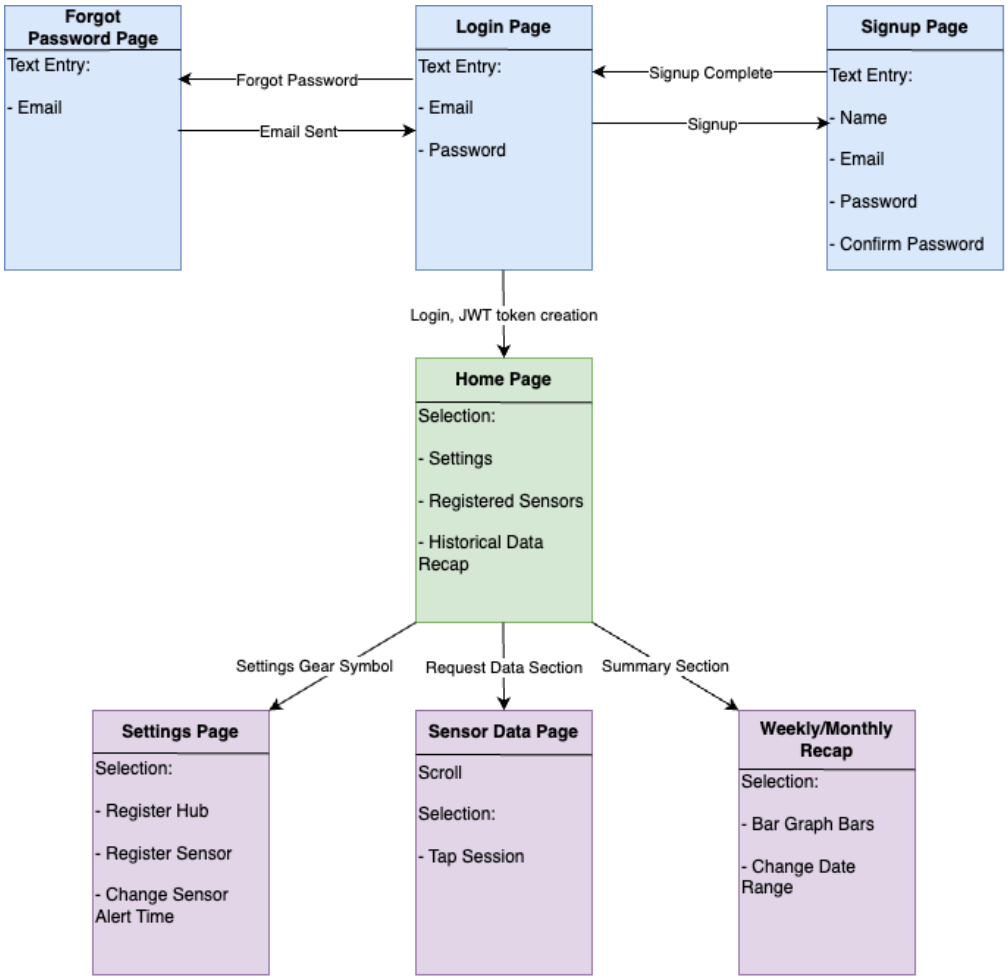


Figure 5: General Front-End Layout

2.4.3 Technology and Language Choices

Due to us choosing to deploy on iOS we chose to use Swift as our language for both the mobile applications frontend and backend. With first party software we have access to Apple's SwiftUI and other libraries for easy frontend design and create a backend library for us to interface with our server.

We chose Xcode as our integrated development environment to have access to the iPhone simulator and Content Previews for debugging and prototype deployment before building onto an actual phone.

2.4.4 Log-in View

The Log-in view is the first view the user will see, in figure 3 it is in blue. As the name suggests is handles user logging into their account to access the home view but has additional user management. To create a new user the log-in view has a "sign-up" button which takes the user to a sign up page. Additionally if a user forgets their password, a "forgot password" button will take the user to a subview to fill in their email and their security questions to input a new password.

2.4.5 Sign-Up View

The Sign-Up view handles user creation, it is red in figure 3. Users input their name, a username, email address, passwords twice, and a set of security questions. While the usernames are stored directly in the database, all other information is hashed as a security measure. Upon successful user creation, the view is switched to the log-in view and the user is presented with a message of "User Created!".

2.4.6 Home View

The home view is the main view of the application, it is yellow on figure 3. This is where users will be able to request data per sensor, register/delete sensors, see a cumulative water usage report for the previous weeks, and change sensor settings. All individual functionality is separated by tiles and are ordered as follows:

- **Tile 1: Basic Data Query**
  With a date selector, the user picks a single date and enter the name of a sensor to view all the sessions that day. Our tile displays the following information from top to bottom: time slice of the entire session, sensor identification number, and water amount in liters used per session. All sessions are sorted based on their start time for easy back tracking of water usage for that sensor.

- **Tile 2: Sensor Registering and Deletion**
  This tile handles both sensor registering and deletion. Using a tab selector, the tile switches between register and delete mode. In delete mode we require the user to give both a sensor's identification number and name to delete it. Additional confirmation is done by answering one of their security questions to prevent accidental deletion by a user.

  Registering a new sensor requires the use of a QR scanner from SwiftUI and the QR code from the sensor's packaging. Upon scanning this code users are prompted with entering a name for the sensor. Once the name is inputted, the sensor is added into the user's database entry as a device they own. With a Bluetooth connection on the phone, we can then click the "Pair" button to search for a Base Station nearby to pair to the sensor. This pairing will pass onto Wifi credentials of the Base Station to the sensor and allow for the sensor to communicate with the server.

- **Graph of Previous Month's Total Water Consumption**
  To visualize data, our third tile graphs the total water consumption of the previous week. Using SwiftUI's Charts feature we query the entire database for each day of the previous seven to add up total water usage among all sensors. Once charted, the user has access to buttons corresponding to each day to see a graph of total water consumption per sensor for that day. This way the user can spot discrepancies and find out where water is wasted. Additionally it helps regulate their water usage by show a clear trend in their own water habits.

- **Settings**
  The settings tile will have different options for each individual user to tailor the system to their needs. Aside from mobile app customization, all functional parameters are held here. Users are able to change how long the sensor can measure a session before considering it a timeout and sending alerts. By default this is two minutes but sensors may be used in different environments, for example shower heads may run longer before sending a timeout and thus a push notification.

2.4.7 Mobile Device to Server Communication

As mentioned before the backend of the mobile application is also written in Swift. Our HTTP library interacts with all the API endpoints. All follow a similar structure:

```
    func name(jwt: String, /*args*/..., completion: @escaping (<Type>) -> Void) {
    let endpoint = /*Specfied Endpoint*/
    let url = URL(string: endpoint)!
    var request = URLRequest(url: url)
    request.httpMethod = /*GET or POST*/
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    request.addValue("\(jwt)", forHTTPHeaderField: "x-access-token")
    var completion_return: <Type> = /*Want we want from Server*/
    let task = URLSession.shared.dataTask(with: request) { (data, response, error) in
        guard let data = data else {
            print("Request Failed")
            completion(competion_return)
            return
        }
        /*Process Task*/
        completion_return = /* Data from Server */
        completion(completion_return) // return what we want
    }
    //complete the http request
    task.resume()
}
```

All functions in this library besides the ones to handle user log-in and sign-up require JWT authentication for security. Depending on the task at hand, other arguments are used such as **http_query_session** requires a date and "sinkid" both as strings. The **completion: @escaping (<Type>)** argument is the completion handler. Since HTTP requests require time to be processed one can not simply just return say a string. We need to wait until the request is done, the completion handler allows the function to store something for it to be used later. The first couple of lines in this function skeleton is initializing the endpoint and request to process. To begin the request the line "**let task = URLSession.shared.dataTask(with: request)**" begins the block of code to process the request; within this block the first declared variables data,response,error are created where data is the data from the endpoint, response is the response code, and error is a error object to help with debugging. Once the code enters this block the server runs the specified task at the endpoint and sends the data to the phone's backend. The final line of importance is "**task.resume()**" which actually completes the request.

2.5 Setup and Usage Details

Setup requires no additional items to be purchased outside of our product. Our setup process is designed to simple to understand.

2.5.1 Base Station Setup

Half of the local setup is centered around the base station. Due to our broker starting upon boot of the station, we either need to connect the base station via Ethernet or provide a guide on how to connect to WiFi manually. For a manual WiFi connection the user will do the following:

1. **Connect Hub and Phone via Bluetooth** If the hub is not connected to WiFi it will be accepting any Bluetooth pairing.

2. **Transfer Network Login Credentials** Once the Bluetooth connection is established the hub will take the phone's WiFi credentials and use it to establish its own WiFi connection

2.5.2 Sensor Setup

The sensor setup has more setups than the base station setup and requires some hands on work.

1. **Register Sensor** - Scan the QR attached to the sensor's packaging or enclosure with the mobile app to register to an account
2. **Connect Sensor to Hub** - Pair the sensor with the phone over Bluetooth, and then pair the phone with the hub over Bluetooth as well. Find the hub under the devices tab in the homepage of the app and press the button to connect to WiFi. In a similar process to connect the hub to WiFi, the sensor will use the phone's network credentials to establish its WiFi connection.
3. **Install Sensor to Pipe** - Unscrew the inlet pip, attach our provided female to female adapter for the specific pipe threading, attach the sensor, reattach the inlet pipe, and switch on the sensor.

2.5.3 Sensor Usage and Maintenance

After installation, users only interface with the mobile app until maintenance. Push notifications are the default method to notify the user of a session timing out. Regular sessions do not push notifications but users have the option to be notified to check the app every seven days for weekly usage.

Maintenance of the sensor is designed to be easy and rare. Due to the simple installation process, removal of the sensor follow the steps in reverse. Replacing of batteries should be done once a year, the sensor may be removed for easier access to the battery compartment. In the case of a non responsive sensor, each sensor may be rebooted to reconnect to the network by using the power switch.

2.6 Water-Proofing & Life-Proofing

For a device that aims at longevity and durability, it is important to design our product to handle everyday wear and tear during installation and handling. In addition, for a device that spends all its time around water inlets and under a sink, it is crucial that the water proofing of the sensing device is fool proof and causes no leaks.

2.6.1 Water-Proofing

This device is meant to survive many years attached to a client's water outlet, such as sinks, shower heads, and even washing machine inlets. As such, water proofing is a main design challenge to guarantee the lifespan of the electronics inside the device, and minimize damage caused by leaks or any other design flaw.

The main cause for leaks at merge points is an incomplete threaded seal. The threaded portion of the sensor must be long enough to fit snuggly with the attachment point to the faucet, and if this is not possible, then a suitable rubber o-ring fitting between the sensor and the attachment point is added to minimize leakage. Following standard plumbing practice, threaded tape can be wrapped around the sensor threads to also guarantee a snug and water tight fit.

Any hole that is intentionally designed and added as a part of the casing has a potential in being an entry point for moisture and water from a potential leak. To minimize this, rubber sealing is used to surround any seam between the opening edge and device (LED, charging port, antenna wire). In addition, a rubber gasket lines the edge of the case, creating a perfect seal between the edges of the two halves of the case.

2.6.2 Life-Proofing

As for life-proofing, the device is not intended to be thrown around daily like a mobile phone, meaning that intense life-proofing is not necessary. However, it is still important to secure the hardware inside the device in the event of an inevitable fall. To achieve this, the perfboard, sensor, and battery pack will effectively be connected to the inside of the casing walls through attachment points to prevent any movement on the inside.

2.7 Simulation & Testing

2.7.1 Hardware

The hardware component testing focused on ensuring the accuracy and reliability of the water flow sensor. After fully designing and manufacturing the sensor, we conducted several preliminary tests to confirm its basic functionality. In multiple sessions, we passed set amounts of water through the sensor to verify its measurement accuracy. We used precision instruments to measure the actual water flow and compared these measurements with the sensor's readings. We also simulated the transmission of water flow data from the sensor through the ESP (a microcontroller with built-in Wi-Fi) to the home hub using the MQTT (Message Queuing Telemetry Transport) protocol, ensuring data was accurately and reliably communicated without loss or corruption. During the testing, we encountered issues with Wi-Fi provisioning, which is the process of connecting the sensor to the home network. These issues involved intermittent connectivity and failed communication attempts. Troubleshooting these problems highlighted areas for further network configuration and firmware optimization improvement.

2.7.2 Mobile App

Simulation and testing for the mobile app is relatively straightforward.

Testing the mobile app involved several steps to ensure its functionality, user interface, and reliability. Initially, we tested the basic functionality of the mobile app, ensuring that all user interface elements were responsive and that navigation within the app was smooth. The mobile app communicated with a web server to retrieve and send data, which initially posed challenges. However, using Postman, we could simulate HTTPS GET/POST requests to the server's API gateways, validating the communication endpoints and ensuring data was correctly received and sent. We also monitored the database to check for changes corresponding to app actions, ensuring that data was accurately updated and reflected in the app. Finally, we conducted user testing sessions where potential users interacted with the app, and feedback from these sessions was used to make refinements and improve the overall user experience.

2.7.3 Web Server

Testing the web server was crucial to ensure it handled requests correctly and maintained data integrity. We tested each API endpoint to confirm that requests were being correctly processed, sending various types of requests (e.g., GET, POST) with different data payloads to observe the server's responses. The server's interaction with the database was thoroughly tested to ensure queries were accurate and efficient, involving checks that data was correctly inserted, updated, and retrieved from the database. We simulated erroneous data inputs to observe how the server handled such cases, implementing proper error handling mechanisms to either discard or correct incorrect data inputs, ensuring the server's robustness and reliability. Additionally, we conducted load testing to evaluate the server's performance under various conditions, ensuring it could handle high traffic and large amounts of data without significant degradation in performance. Each of these components—hardware, mobile app, and web server—was tested rigorously to ensure the overall system's reliability and efficiency. The insights gained from these tests were used to refine and improve the design, leading to a more robust and user-friendly final product.

2.8 Future Additions and Upgrades

The design detailed in this document is by no means a complete and sound plan for a final product. Below, we detail additional ideas and add-ons that can be integrated with the product for future iterations:

2.8.1 Integration with Additional Sensors

Other than the primary flow sensor, additional sensing components could be included to provide a more detailed analysis of a users water consumption metrics. This includes additional temperature

sensors, salinity sensors, and purity sensors. Some Hall Effect Flow Sensors have included thermometers already, such as the Taidacent YF-B7 High Precision Water Flow Sensor:

Figure 6: Temp Integrated Flow Sensor

Figure 7: Water Quality Sensor

3 Evaluation

3.1 Resulting Functional Prototypes

3.1.1 Overview

The prototype that resulted from this design contains many of the important components and functionality that you would find in the final design. The sensing device is able to screw onto a 1/2 female water inlet for a sink with a pipe to the water inlet then screwing into the bottom of the device. It can measure water sessions as they are happening. It then packages and sends this data to the home hub.

The prototype encasing of the sensor itself does its job well in packaging everything up snugly. It is a 3D printed, hollowed out box with holes at two opposite faces that allow the flow sensor to fit tightly. There are also holes for a power LED, connection LED, power switch, and antenna. A green LED signifies connection to the home hub, and a yellow LED indicates device power. The box is made up of two halves that can easily come apart and easily hold together. They are held together by simple pegs and holes. The design ended up being larger than we would have liked, mainly due to the constraint of having to fit three AAA batteries inside.

The home hub itself is a Raspberry Pi, and acted as the MQTT broker. It receives water usage data from sensing devices, which it then packets and sends to the web server over via an HTTP POST request.

The web server is hosted on AWS Lightsail. Nginx is used as a reverse proxy to handle handshakes and HTTPS. We obtained an SSL certificate from Let's Encrypt CA to certify our server's identity to any clients. The server issues API keys to any new home hubs for them to use in any subsequent POSTs so that our server doesn't accept random POST requests from unauthorized devices. New keys are issued every week for added security. JWTs are used to verify the credentials of any user accessing the server through the mobile app. PostgreSQL is used for the database which holds user data, session data, registered sensors, etc. User passwords are encrypted properly using salt.

The mobile application is an iOS app coded on Swift. It has most of the basic functionality that

we originally envisioned. Users can register/log in, register new sensors through QR codes, view water usage summary over a specific date range, and view historic usage data from specific sensors during specific dates.

3.1.2 Results

The prototype is successful and can record water usage sessions accurately and quickly. It is accurate within 0.1 liters, which is still a solid result for a prototype, while higher than the 5% accuracy we were aiming for.

Whenever the faucet is turned on, a session is immediately started. Once the faucet is shut off, the data is formatted and posted to our server within 2 seconds. This means that the MQTT connection from the sensor to the hub is established, used, and disconnected within a second of the water being shut off.

The web server never crashes or falters, and is able to store and retrieve data effectively with no delay, meaning a session is posted and retrieved to the app within a couple seconds.

3.1.3 Prototype Renders

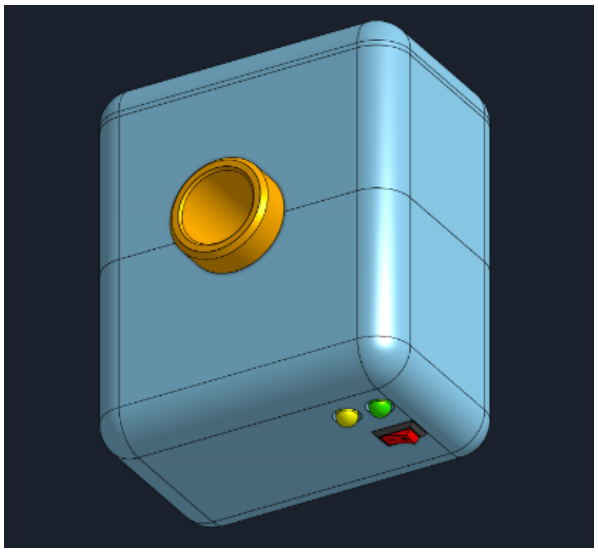


Figure 8: Render of Closed Sensing Device

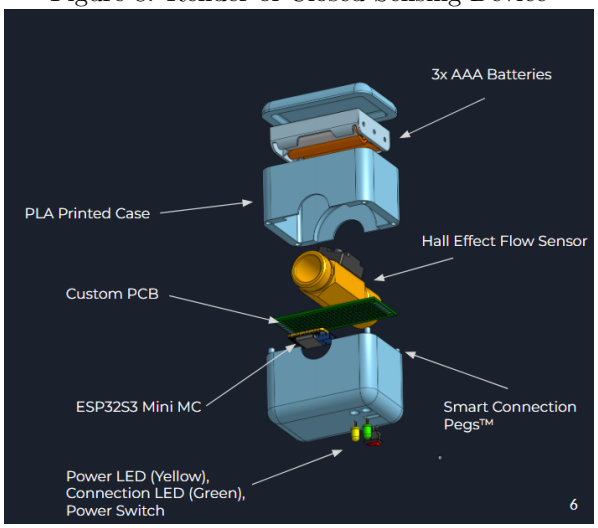


Figure 9: Diagram of Open Sensing Device

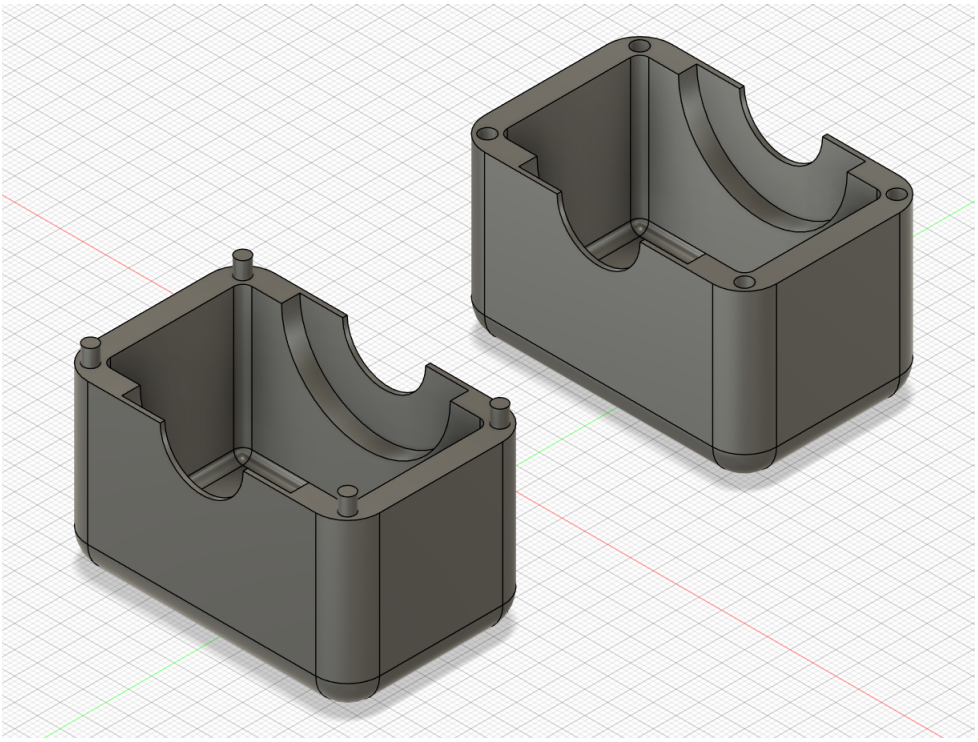
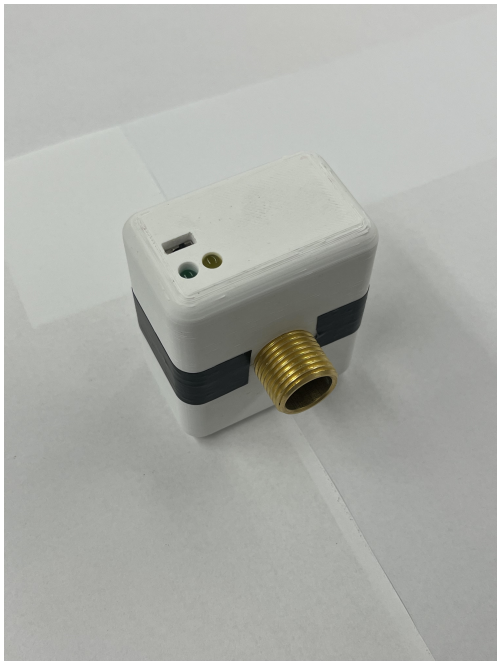


Figure 10: 3D Model of Sensor Encasing

3.1.4 Prototype Images Version 1



(a) Profile of Device

(b) Top of Device

(c) Side of Device

(d) Bottom of Device

Figure 11: Version 1.0 Sensing Device

3.1.5 Prototype Images Version 2

(a) Profile of Device

(b) Top of Device

(c) Front of Device

Figure 12: Version 1.0 Sensing Device

3.1.6 Final Prototype Drawing

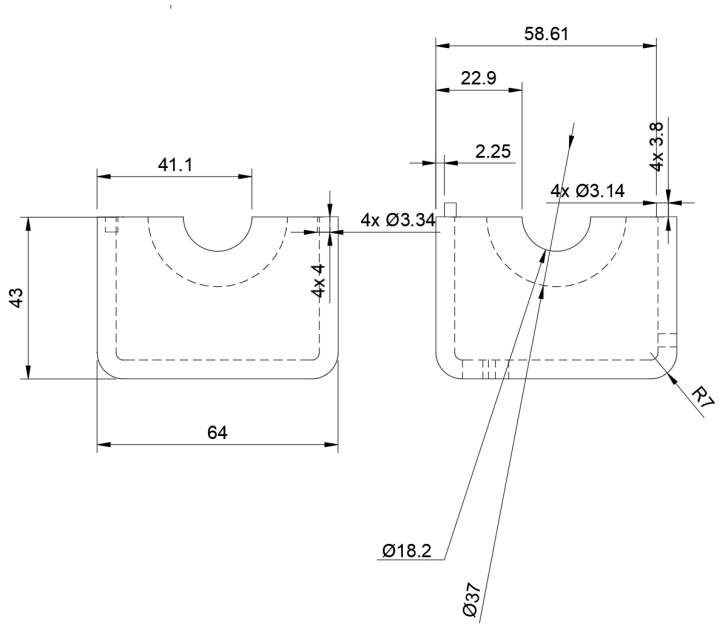


Figure 13: Side Dimensions

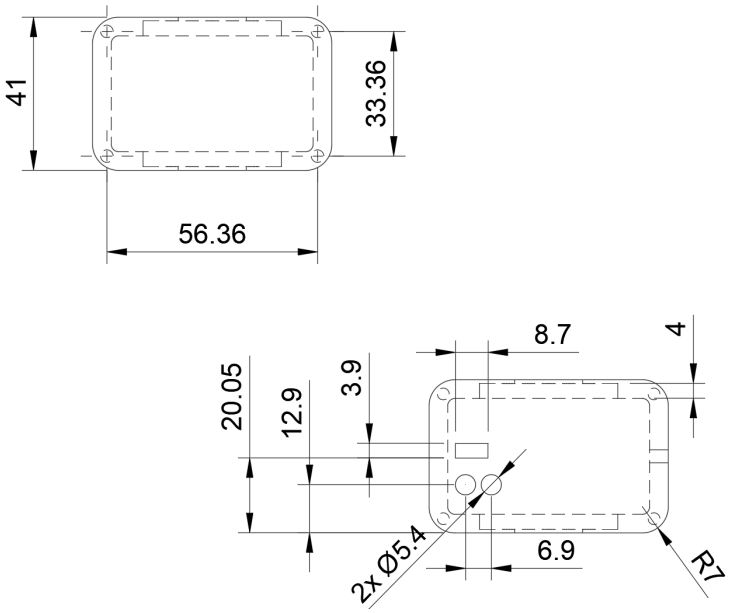
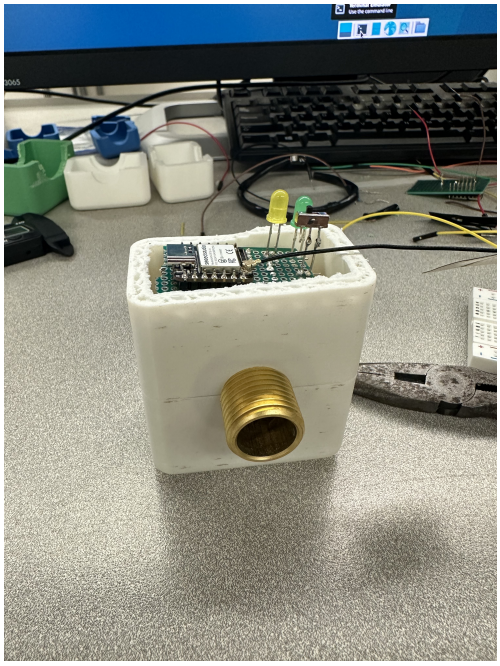
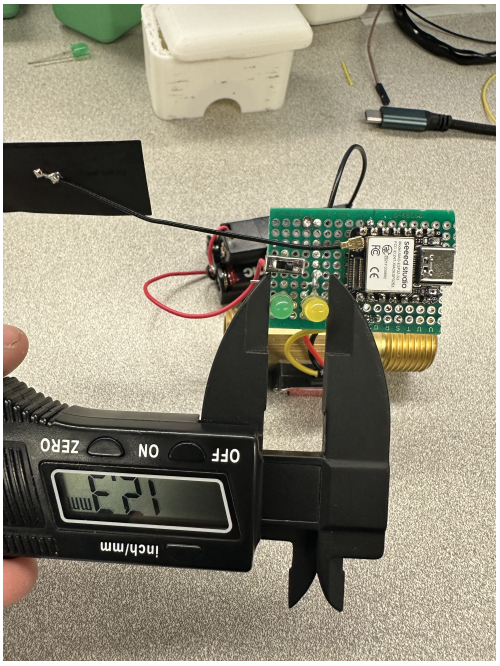


Figure 14: Top and Bottom Dimensions

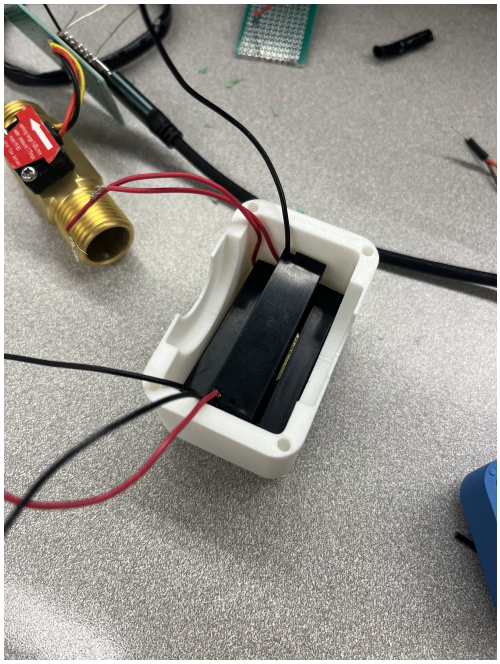
3.1.7 Prototype Hardware Images



(a) Device Hardware Case Showing



(b) Device Hardware No Case



(c) Device Battery Compartment

Figure 15: Hardware Images

3.1.8 Prototype Web Server Screenshots

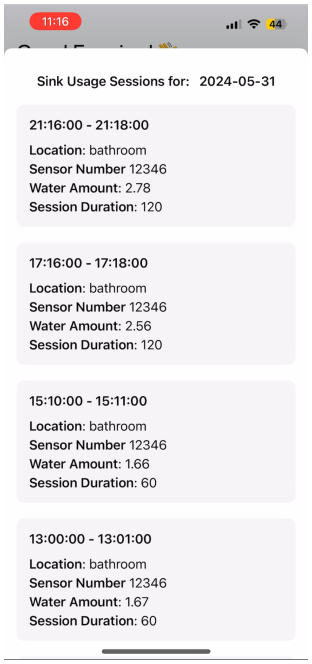
```
                List of relations
 Schema |     Name     | Type  | Owner
--------+--------------+-------+-------
 public | graph_data   | table | team1
 public | keys         | table | team1
 public | sensors      | table | team1
 public | sessions     | table | team1
 public | users        | table | team1
(5 rows)
```

Figure 16: PostgreSQL Database Setup

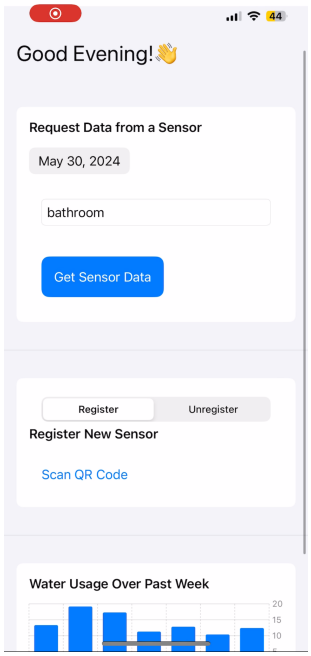
```
usage_db=# SELECT * FROM sessions;
 session_id |       sink_id        | sensor_id | water_amount | duration | start_time | end_time |    date
------------+----------------------+-----------+--------------+----------+------------+----------+------------
          1 | kitchen              |     12345 |         0.74 |       15 | 13:40:00   | 13:41:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.92 |       19 | 13:43:00   | 13:44:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.05 |        2 | 13:44:00   | 13:44:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.04 |        1 | 13:44:00   | 13:44:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.64 |       12 | 14:16:00   | 14:16:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.18 |        2 | 14:16:00   | 14:17:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.03 |        1 | 14:17:00   | 14:17:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.03 |        1 | 14:17:00   | 14:17:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.03 |        1 | 14:17:00   | 14:17:00 | 2024-05-31
          1 | kitchen              |     12345 |         0.06 |        2 | 14:17:00   | 14:17:00 | 2024-05-31
(10 rows)
```

Figure 17: Table Setup of Sessions

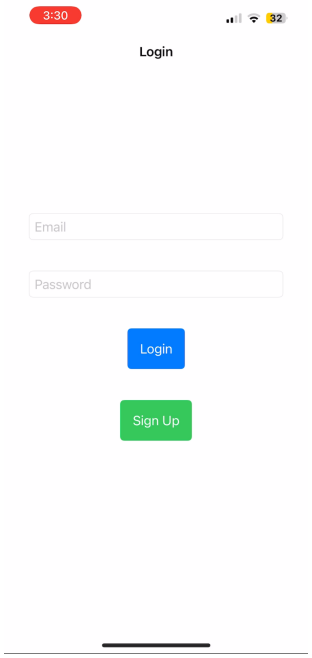
3.1.9 Prototype App Screenshots



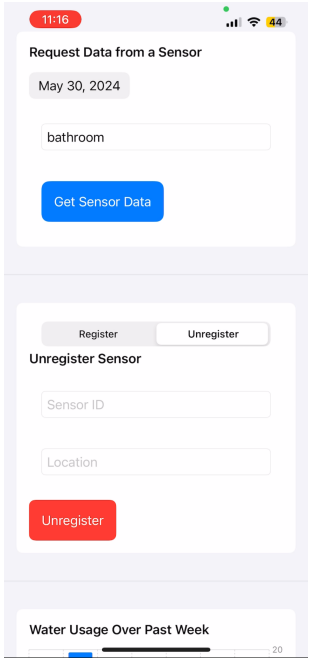
(a) Data View



(b) Home View



(c) Login View



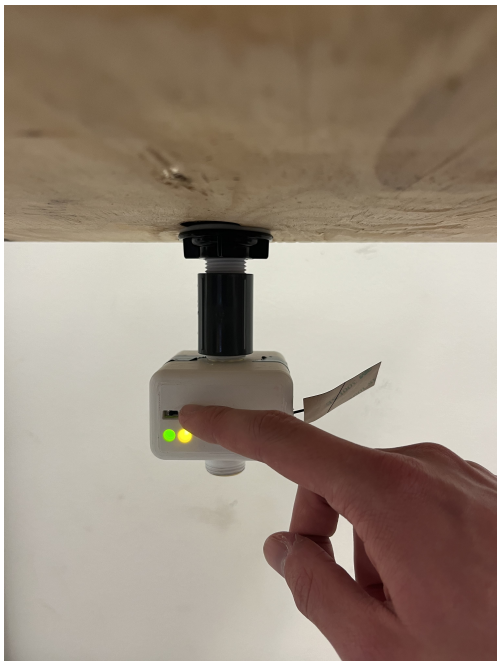
(d) Unregister Sensor View

Figure 18: App Screenshots

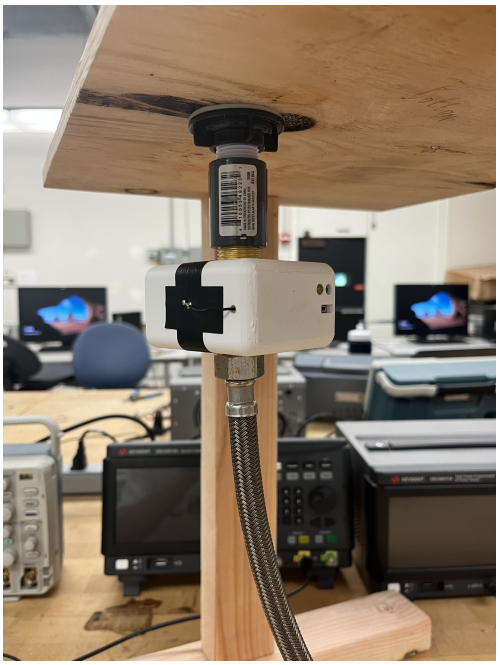
3.1.10 Prototype Test Environment Images

(a) Test Environment 1

(b) Test Environment 2



(c) Device Installed and On



(d) Device Installed

Figure 19: Test Setup Images

4 Appendix 1 - Problem Formulation

4.1 Conceptualizations

4.2 Brainstorming

- Attachment Method to faucet spout:
  - custom 3D printed threading, waterproof with plumbing tape universal clamp, attached to faucet with rubber house (one size fits all)
- Low Power Design:
  - Sleep SoC on Micro-controller, minimal hardware components
- Sensor water-proofing:
  - Temperature Probe, pokes through access hole into water chamber IC temperature sensor, cover sensitive circuitry with epoxy
- Water Conservation efforts:
  - PG-E API to get energy rates (for temp?)
  - Prompt user to input water rates for area
  - Use input rates + measured usage averages to "predict" monthly water usage
  - Provide times of day where water consumption is highest Set alerts for thresholds
- IoT system setup:
  - Multiple SoC microcontroller nodes for several faucets
  - Central computing hub to collect and format data
  - screen/touch screen on central hub to display general data?
  - Multi-threading, different thread per faucet head.
- Monitoring App:
  - Use Swift for IOS app, connects via Bluetooth to central hub to download & display data
- Housing for sensing device:
  - Use waterproof PLA plastic for 3D printing
  - Access panel for electronics, batteries
  - Waterproof sealing, rubber rings/grommets, epoxy seal
- Timeline:
  - Focus on water flow first, then move onto temperature implementation
  - Acquire sensors top priority, start testing for data collections

4.3 Decision Tables

Criteria	Units	Weight	Design 1		**Design 2**		Design 3	
			Raw	Weighted	Raw	Weighted	Raw	Weighted
Cost	$	30	20	600	18	540	15	450
Power Consumption	Volts	15	1.5	22.5	1.5	22.5	3	45
Space	in^3	10	16	160	14	140	21	210
Maintenance	Minutes	5	2	10	1	5	2	10
Precision	Error %	20	4	80	7	140	5	100
Implementation	Difficult (1-10)	10	3	30	6	60	5	50
User-Friendly	Inaccessibility (1-10)	10	1	10	3	30	7	70
				912.5		**937.5**		**935**

Decision Table Inputs

- **Cost**: The total cost for materials, construction, and technology, measured in dollars (USD).
- **Power Consumption**: The total power consumption for the flow sensing node ONLY, measured in volts (V).
- **Space**: The form-factor and the volume occupied by the sensing node design, measured in cubic centimeters (in^3).
- **Maintenance**: The maintenance requirements for the entire system, including tasks such as changing batteries and diagnosing issues, measured in frequency of maintenance per year and average maintenance time per session (hours).
- **Precision**: The accuracy of flow and temperature measurements, defined by the percentage error for flow ($\pm\%$) and temperature ($\pm°C$).
- **Implementation**: The difficulty of implementation, including the time and effort needed to prototype and manufacture, measured in a scale from 1-10.
- **User-Friendly**: The ease of use for the consumer, which affects user satisfaction and reviews, assessed qualitatively or through consumer feedback scores, on a scale from 1-10.

4.4 Morphological Charts

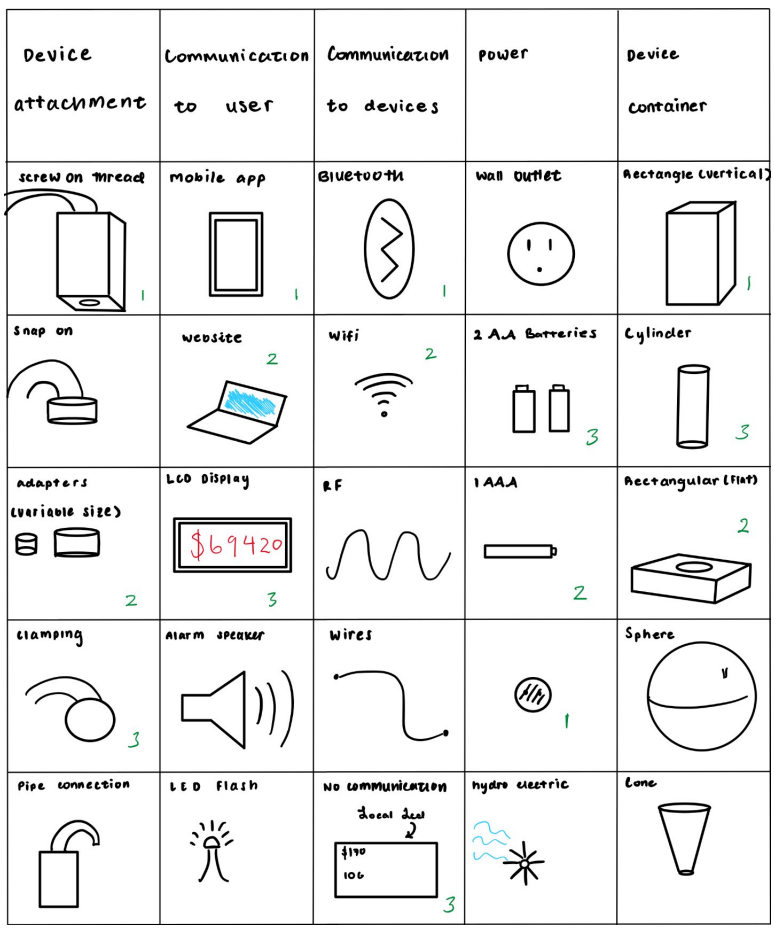


Figure 20: Morphological Chart

5 Appendix 2 - Planning

5.1 Basic Planning

5.2 Gantt Chart

Figure 21: Gantt Chart

5.3 Critical Path Analysis

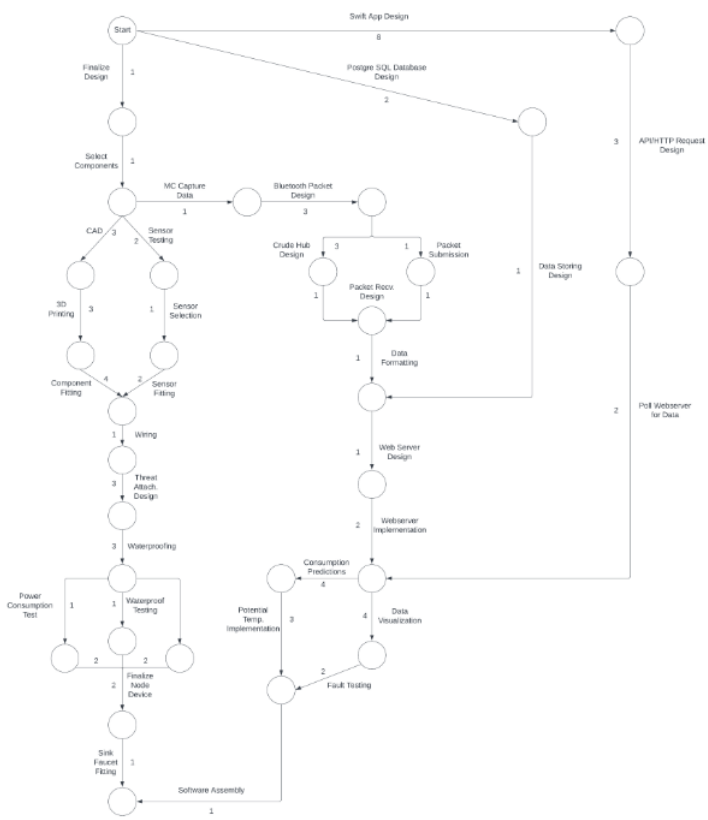


Figure 22: Critical Path

5.4 Division of Labor

The labor for the prototype was divided into 3 separate specifics: The construction of the sensing system, the development of the cloud-based web-server and accompanying database, and the creation

of the user side app. We believe that the amount of work needed for each of these sections was around the same, which made it equal in terms of workload and time spent. As the quarter progressed and new challenges/tasks popped up, someone who felt confident in tackling the task would complete it.

5.5 Collaboration

Code that was written for all 3 components of our prototype was stored in a code repository on GitHub. We made different branches to work on several parts of the project without issue. The AWS LightSail server also hosted resources and code, which was accessible to all of us using setup public-private key pairs stored in the server.

As for how tasks were assigned, each sector would have a sort of "lead" team member which was in charger of managing and maintaining sub-tasks with the other members working on that prototype section. In the end the bulk work was done in this order:

5.5.1 Sensor, Home Hub Hardware and Software

Mitchell was acting sub-team lead for this group. He did the hardware wiring and soldering for the electronics within the sensing device, as well as developed the bulk of the localized software onboard the micro-controller and the code needed to communicate between the Home Hub and the web server. Tirath aided in the development of the communication system between the sensing device and the home hub, with tests between using Bluetooth and MQTT brokers.

5.5.2 Web Server and Database System

Tim was acting sub-team lead for this group, with Kyven taking charge primarily in the database system connected to the web-server.

5.5.3 IOS App

Gary was acting sub-team lead for the app development, creating the back-end systems that communicate between the app and the web-server. Front end design and operations were mostly done between Gary and Tim.

5.5.4 3D Modeling and Case Printing

The modeling of the encasement for the sensor was done by Kyven, creating the initial design and adjusting when changes and challenges occur with the hardware design of the sensor.

6 Appendix 3 - Test Plan & Results

6.1 Introduction

Testing is a crucial aspect of prototyping and development, as results from tests and experiments act as a feedback loop to the continuous development of the product. For our capstone, we prioritized testing on all 3 fields of our design: hardware, web server, and mobile app. For both testing and the demo, we built a custom faucet rig to simulate the environment that our device would be attached to.

6.2 Test Objectives

6.2.1 Introduction

The test plan aims to validate the design and functionality of the system designed and constructed by Team 1. The testing covers hardware, mobile application, and web server components, as well as full experimentation of the entire system.

- Verify that each component of the system meets its design specifications
- Ensure the integrated system functions correctly under various real world conditions
- Identify and fix defects to ensure the reliability and usability of the system

6.3 Test Scope

6.3.1 Test Parameters

- Water Recording Accuracy: Be able to perform and accurate a quick measurement of water flow through the attached faucet, within a relatively accurate degree necessary for consumers.
- Server/App Update Speed: Be able to update the web server and subsequently mobile app down stream quickly for accurate, real time results.
- Leakage and Water Proofing: Be able to confidently install the device and verify that it does not cause any leaks due to threading or attachment points, also verify that if any water does escape, the device is water resistant and does not affect any of the internal electronics.
- Connectivity Uptime: Verify that connectivity between the sensing device and the Home Hub has a nearly perfect connectivity success rate, and that communication does not drop when data is being transferred.
- Power Consumption Testing: Verify that with the proper usage, the battery powering the device lasts its intended life cycle.

6.4 Test Environment

The test environment was designed to replicate a simple single inlet faucet. The faucet input is 1/2" threaded male, with an attached 1/2" American Standard supply hose. The hose is attached to a simple water pump in a water basin underneath the sink environment. When testing, teflon tape is wrapped around the threads of the sensing device and any other threaded attachment points, such as the inlet of the sink. This servers 2 purposes, to prevent leaks, and the follow standard plumbing practices.

Please take a look at the images displayed in the test environment sub-section of the Prototype Evaluation section.

6.5 Test Schedule

Each planned test was scheduled to be based on when its specifically based component is completed. Some tests where planned as intermediary testing to guide further development.

As the web server was being developed, constant testing on response speeds were being done, as well as verification that specific API endpoints where reaching the correct database table for storage and retrieval. In addition, helper functions were created to verify that data was stored correctly in the database, with erroneous data being injected to test security and software issues.

Water flow session accuracy tests were done in conjunction to hardware development to verify accuracy as the device was being built. When development first started, simply blowing air through the sensor was enough to verify sessions were being recorded and sent to the web server in a timely manner. Testing MQTT connection uptime led us to change our design implementation to having the device only initiate connection when a session has completed, to make sure that only the minimum amount of time of connectivity took place, lowering the chance of a dropped connection and lost data.

6.6 Test Cases & Results

6.6.1 Water Recording Accuracy Test

1. **Water Accuracy Test Before Software Adjustment**
   (a) **Trial 1:** Target: 600mL, Measurement: 570mL
   (b) **Trial 2:** Target: 600mL, Measurement: 560mL
   (c) **Trial 3:** Target: 600mL, Measurement: 567mL
   (d) **Trial 4:** Target: 600mL, Measurement: 550mL
   (e) **Trial 5:** Target: 600mL, Measurement: 563mL
2. **Water Accuracy Test After Software Adjustment**
   (a) **Trial 1:** Target: 600mL, Measurement: 591mL
   (b) **Trial 2:** Target: 600mL, Measurement: 595mL
   (c) **Trial 3:** Target: 600mL, Measurement: 590mL
   (d) **Trial 4:** Target: 600mL, Measurement: 601mL
   (e) **Trial 5:** Target: 600mL, Measurement: 595mL

6.6.2 Server/App Update Speed Test

We spent some time testing the speed at which data from the sensor propagated through our system, and concluded that when a new session was completed, the data would be present and stored in the server within 2 seconds of the sink shutting off.

6.6.3 Leakage and Waterproofing Test

To test waterproofing, we organized 3 tests: **No teflon tape**, **teflon tape on sensor alone**, **teflon tape on sensor and faucet inlet**.

- **No teflon tape**: water leaked due to short threading on output end of sensor.
- **2 layers teflon tape on sensor alone**: water leakage above sensor and female-to-female adapter, water fell onto sensor case.
- **2 layers teflon tape on sensor and faucet inlet**: No leaks detected, even after faucet turned off and pump was still active.

6.6.4 Sensor Connectivity and Uptime Test

A crucial component that required much care and thought was the verification of a dependable and reliable connection between the sensor and the home hub. In our prototype we discovered that MQTT can be somewhat finicky due to the conditions of the Baskin Lab area and eduroam, however after our initial 10 tests of water accuracy, we concluded that the sensor was able to successfully make a connection to the Home Hub 9 times out of 10.

6.6.5 Power Consumption and Lifespan Test

Power consumption is likely the most important metric to test for and validate for this type of capstone, as the reliability and operation of the sensor with the consumer depends on long battery life and infrequent charging. To test this, while testing other metrics such as water usage accuracy on a regular usage timeline, we would measure the voltage of the battery to calculate the given charge left after the usage.

Sadly we determined through this methodology that our prototype had a battery life span of 3 weeks. This was completely a subpar performance. Based on our calculations in an ideal world:

$$\text{Total Voltage} = 3 \times 1.5\,\text{V} = 4.5\,\text{V}$$

$$\text{Total Capacity} = 1000\,\text{mAh}$$

- **Active Mode Duration**: 15 seconds
- **Deep Sleep Mode Duration**: Total time between transmission - active time
- **Transmission Amount per Day**: 15 times (estimated average US household)

$$\textbf{Cycle Duration}: \frac{24 \times 3600\,\text{seconds}}{15} \approx 5760\,\text{seconds}$$

$$\textbf{Sleep Duration}: 5760 - 30 \approx 5730\,\text{seconds}$$

$$\textbf{Sleep Duration} = 50\ \text{mA} \times \frac{15\ \text{seconds}}{3600\ \text{seconds/hour}} = 0.2083\ \text{mAh}$$

$$\textbf{Sleep Energy} = 0.01\ \text{mA} \times \frac{5730\ \text{seconds}}{3600\ \text{seconds/hour}} = 0.0159\ \text{mAh}$$

$$\textbf{Total Energy Per Cycle}: 0.2083\ \text{mAh} + 0.0159\ \text{mAh} = 0.2242\ \text{mAh}$$

$$\textbf{Daily Energy}: 15 \times 0.2242\ \text{mAh} = 3.363\ \text{mAh/day}$$

- **Total Capacity** = 1000 mAh
- **Daily Consumption** = 3.363 mAh/day
- **Lifespan** =

$$\frac{1000\ \text{mAh}}{3.363\ \text{mAh/day}} \approx 297.4\ \text{days}$$

6.7 Analysis

There are many conclusions we can draw from the given tests we conducted on our capstone project. Here we discuss a bit about what we learned and fixed after each test.

- **Water Accuracy Test**: From testing the sensor purely, we learned that it was relatively accurate to the actual amount of water passing through the sensor. However to reach our goal of having an accuracy of within 10% of the actual amount of water, we added an adjustment constant to any recorded value to get a higher degree of accuracy.
- **Server/App Update Speed Test**: We learned that even though our updates were fast and practically real time, we realized that external factors such as network and server latency can occasionally cause wait times for data to post. However this is out of our control as most products like ours would also be affected.
- **Leakage and Waterproofing Test**: We concluded that for our sensor to function as intended, Teflon tape must be used at any threaded connection point. This is standard practice in plumbing, and not a design flaw on our part. We also concluded through water proof tests that the form fitting of our casing did not let water in at any seems or connection points, which shows the high accuracy of how the case was designed to fit the components with no room for error.
- **Sensor Connectivity and Up-time Test**: This connection was tricky to implement correctly, as it frequently had issues with keeping stable connections between the sensor and the Home Hub. This was using a design iteration of constant connectivity, which was not viable. By changing to an on-demand connection basis, where the sensor would only establish a connection with the Home Hub when the session ended, we achieved a higher uptime success rate between the sensor and the Home Hub, as well as lower battery usage to only use the high power communication sparingly.
- **Power Consumption and Lifespan Test**: This test was by far the most wildcard out of all of the tests, as we did not know what to expect when testing battery consumption. Though the results were abysmal with having a lifespan of around 3 weeks, we concluded that this was mainly due to our hardware code having issues with deep sleep and power savings. If implemented correctly, having the ESP go into a deep sleep mode would extend the life span closer to the calculated values. In addition through discussion with peers and the professor, we concluded that having LEDs on our prototype were not necessary since they drain power and were not needed in a consumer device that is meant to stay under your sink for its entire lifespan.

7 Appendix 4 - Review

7.1 Tim

This capstone was a fun experience and a great first look into the collaborative world of engineering design practices. During the first quarter we diligently planned out our design, conducted valid research on the best way to tackle certain components, and really made sure to hammer out any issues before they became too apparent. The second quarter of the capstone sequence became much more time consuming, however splitting the project up into 3 distinct fields, and then again into sub-fields made the progress much more manageable and graspable. If we were to redo this project again, I would really spend time to study what technologies are industry standard for a comparable product, and make an effort learning these technologies. In addition, emphasize and prioritize collaboration, as it became extremely common for work to be one sided between members of the group, and team members not pulling their weight. Create designated weekly meetings where each person went over what they did in the past week, what challenges they are facing, and how they plan on tackling these challenges. Also, incentivize asking for help, and working collaboratively.

7.2 Mitchell

Looking back on everything we accomplished over the past two quarters, I can't help but feel proud of ourselves. Overall, this capstone course was an engaging and profound learning experience that provided a glimpse into the world of engineering. My role was to develop the hardware side of the project. The first quarter consisted mainly of documentation and planning which went pretty smoothly for all of us. The second quarter is where we really began to build our prototype and see our project come to life. One of the main challenges for me was learning the new technologies we would be utilizing for the success of the prototype. Learning how an MQQT broker functions, working with a multitude of foreign sensors, and creating a reliable IoT network were all new things to me. Despite these challenges, I was able to dig through documentation and create our desired product. Having a team to communicate with and bounce questions was an important part of that process. Every time I was unsure about how to go about something, I was able to talk to my team and brainstorm ideas on how to fix them. This was especially helpful when working with the PCB. Handling the perf board was a big challenge for me and required the most time out of the entire hardware aspect. We ran into a lot of problems concerning the circuit design and the wiring which required multiple redesigns and new layouts. Through our testing and trials, we were able to create the most optimal and efficient perf board possible. I think this course is going to be a vital aspect of my career and endeavors as an engineer, and the skills I learned during this time will forever stick with me.

7.3 Gary

Overall an interesting project and one with practical application in the field. Despite its simplicity on the exterior the final product would be rather complex. My work focused on the mobile device to server communication. I had no prior experience with Swift nor XCode so this project was eye opening due to the fact this was the first time working with first party software. Additionally this was my first experience with any UI and mobile development. Despite these challenges, first party help actually made the process smooth. SwiftUI had an extensive library for displaying our server's data nicely in tiles or graphs. For server communication, the Foundation library made HTTPS requests to our server easy to write in Swift. Debugging any front-end or back-end became streamlined when I learned how to properly use SwiftUI's previews and XCode's iPhone simulators. In the end my favorite experience of this project is learning XCode; as a Macbook owner I always needed it for Git and clang but never the actual IDE itself. Finally I had the opportunity to see where Apple's money went into XCode.

7.4 Kyven

As a whole, the process of getting this project done went pretty smoothly for us. The way our prototype server, hardware, and app interact with each other is almost exactly as we had envisioned. I would say that the thing that went especially well for us was the development of the web server. We didn't run into any significant walls during development and whenever we started work on implementing

something, it was often finished within a day or two. If I were to do something differently, I would definitely get a more solid 3D model done for the encasing before actually printing. I would take much more time initially to fully understand what measurements need to be taken and are relevant to the encasing. For the first few prints, I hadn't really gotten all the precise measurements down for the model, which resulted in a lot more wasted prints that just didn't work than we really should have had.

7.5 Darren

Retrospectively, developing our water meter taught me a lot collaborating as team, not just engineering. Prior to this class, I had never done anything with front-end development. I also underestimated the complexity of databases and setting up a secure network and connection. I overlooked the complexity of these network functions, as CSE-156 (Networks Programming) taught a lot of the underlying principles of socket and local network programming. I thought I would be able to take a lot of what I learned the previous quarter, but in actuality, we had to use a handful of external services and procedures that I previously never learned about. Moving on from this project, I'd really like to study databases more in-depth and possibly use a standard panda's library to predict water usage with a range of predictive models in order to explore and study the motivations and causes for increased water usage.

7.6 Tirath

Our capstone project was a valuable and educational journey that underscored the importance of teamwork and provided significant insights into collaborative engineering design practices. My primary role involved managing the MQTT server side of the hardware and contributing to the early-stage design and rendering of the project casing. Despite the steep learning curve, particularly in mastering new technologies and protocols, we successfully navigated complex challenges in mobile device-to-server communication, web server development, and secure networking. The seamless interaction between our prototype server, hardware, and app was a notable success. This was made possible through the dedication and hard work of my fellow engineers, whose time and effort ensured our project stayed on track. Their contributions to mobile app development, API testing, database management, and hardware specifications were instrumental in our success. However, we could have improved the project with more time by incorporating Wi-Fi provisioning through QR codes and enhancing hub connections through the mobile app. This experience taught us the value of meticulous planning, effective communication, and adaptability, ultimately shaping our skills and readiness for future professional endeavors.

7.7 Gustavo

Working on the IoT side of the project was an enriching experience that pushed me to think collectively and adapt to unexpected challenges. Detailed planning in CSE123A allowed us to foresee potential obstacles and address them proactively, contributing to the project's overall success. Managing the MQTT server side of the hardware and contributing to the early-stage design of our project were key highlights of my work. Despite moments of uncertainty, this experience taught me the importance of being proactive and assertive in seeking help when needed. Implementing a backup unit with a teammate showed me how much smoother replication can be compared to designing a product from scratch. Our seamless final product was a testament to the qualifications and expertise of our engineers. Moving forward, we aim to create a custom PCB, and I look forward to applying the skills and lessons learned from this experience in future projects.