West Texas A&M University™

College of Engineering

Kirk Blood

Hector Ceballos

Dale Andre Descallar

Preston DeShazo

William Kolath

CS4390 I Senior Capstone Project

TerraTek

Supervised by:

Dr. Mohammad Faridul H. Siddiqui. Ph.D.

Fall 2024

Department of Computer Science

Abstract

This project presents the development of TerraTek, an IoT-based monitoring system designed to assist a customer in tracking environmental data. The system utilizes Arduino-controlled sensor clusters to gather data on key parameters such as rainfall, water pH, water level, and so on, which are then transmitted to a centralized server. A responsive web interface allows users to visualize data trends and generate reports to aid decision-making processes. Through data analysis, the project demonstrates improved statistics by utilizing historical and real-time data. Future work will involve extending the system's capabilities to include real-time camera monitoring based on sensor data.

Plagiarism Declaration

With the exception of any statement to the contrary, all the material presented in this report is the result of my own efforts. In addition, no parts of this report are copied from other sources. I understand that any evidence of plagiarism and/or the use of unacknowledged third party materials will be dealt with as a serious matter.

Signed

William Kolath

[illegible] [illegible]

[illegible]

[illegible]

[illegible]

Contents

List of Figures

List of Tables

Chapter 1

Software Requirement Specification

1.1 Introduction

1.1.1 Purpose

The purpose of this document is to provide a detailed description of the TerraTek Sensor System (TSS). It will explain the intended scope, purposes and features, constraints, and interactions between its systems. This document is intended for all stakeholders and developers of the system.

1.1.2 Project Scope

The intended scope of the TSS is to provide a web interface for the user to observe telemetry data. Implementing clusters of sensors that will collect and aggregate data to a central database which can be parsed and graphed through a web interface. This will be implemented on small, household scale with the ability to expand to scale.

1.2 Overall Description

1.2.1 Product Perspective

The TSS will encapsulate a backyard consisting of water tanks and garden areas. Both areas will have two sensor clusters each with each cluster collecting data depending on the area. This is a small scale project to serve as a foundation for larger scale implementation.

1.2.2 Product Features

TSS include the five features listed below:

Feature 1: Tank Monitoring

Feature 2: Garden Monitoring

Feature 3: Remote access to data

Feature 4: Data visualizations

Feature 5: Remote camera

1.2.3 User Classes and Characteristics

There are two main classes of users for our project.

- Owner of RWH system
- Other users

The owner of the RWH is primarly focused on being able to understand the current and historical status of the RWH system. They will also use the garden monitoring functionality to make changes to their watering practices as needed. They also must be able to make configuration changes to the TerraTek system to suit their changing needs.
Other users will be primarily focused on learning about how a RWH system works as well as seeing different weather data from the area visualised.

1.2.4 Operating Environment

The server will use Ubuntu Server Version 24.04.1 LTS, the most current version as of writing. The server will utilize MySQL for database operations. The sensors will be programmed with Arduino to communicate with a Raspberry Pi, which will aggregate and send all the data to the server.

1.2.5 Design and Implementation Constraints

A significant portion of the system will be deployed outdoors. Some portion of the system will deployed in water tanks, either submerged or directly above water. Water and weather proofing will be a significant constraint we must work around in this project. Another constraint is longevity. The system needs to be designed to operate for long periods of time without replacing hardware components or performing maintenance on the software.

1.2.6 User Documentation

We will create a user manual for the TerraTek system. There will also be a small online tutorial for users of the website, explaining what the system is monitoring and how to operate the website.

1.2.7 Assumptions and Dependencies

We are making the following assumptions:

- All sensors will be programmable on Arduino
- Each cluster contains only a singular sensor of its type
- The Tanks will never completely empty nor completely fill

We are dependent on the following systems:

- Linux Server
- Home WiFi of the RWH owner
- Pre existing RWH system

1.3 System Features

Feature 1: Tank Monitoring

Each of the RWH tanks on the property needs to be monitored using various sensors. This is a very high priority feature.

REQ-1.1: The system must monitor tank level.
REQ-1.2: The system must monitor tank water PH.
REQ-1.3: The system must monitor tank TDC.
REQ-1.4: The system must monitor tank temperature.

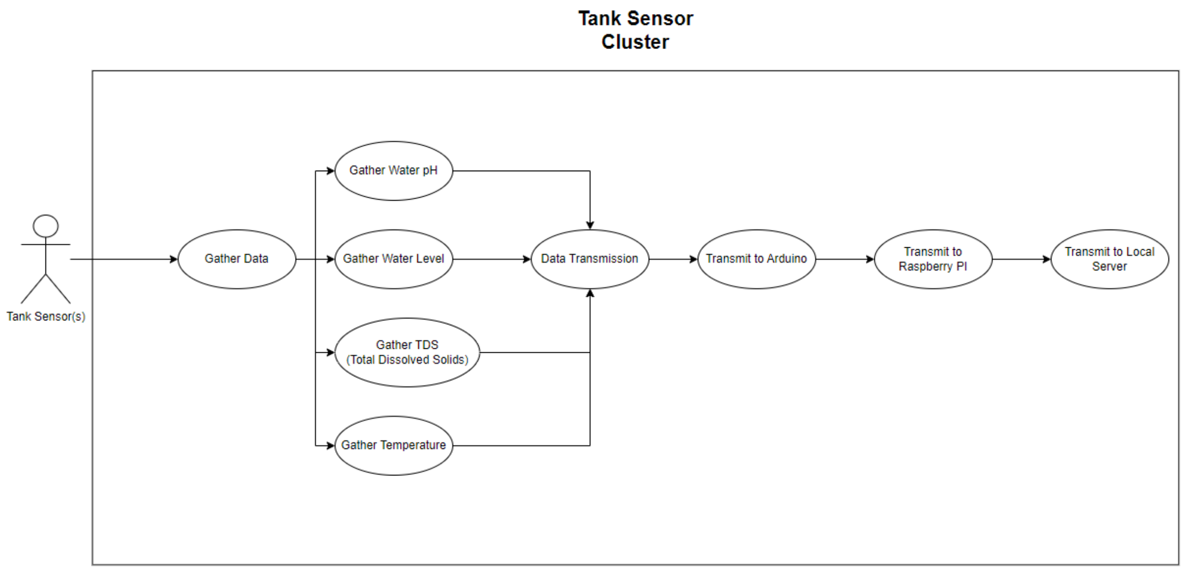

Figure 1.1: Use Case Diagram for Tank Sensors

Feature 2: Garden Monitoring

The garden area will have sensors installed to monitor the health of the soil and current weather conditions. This will allow the owner of the garden to make informed decisions regarding watering and planting. This is a high priority feature.

REQ-2.1: The system should monitor soil moisture in multiple locations of the garden.

REQ-2.2: The system must measure rainfall in the garden.

REQ-2.3: The system should measure atmospheric temperature at multiple locations in the garden.

REQ-2.4: The system should measure atmospheric pressure at multiple locations in the garden.

REQ-2.5: The system could measure wind speed in the garden.

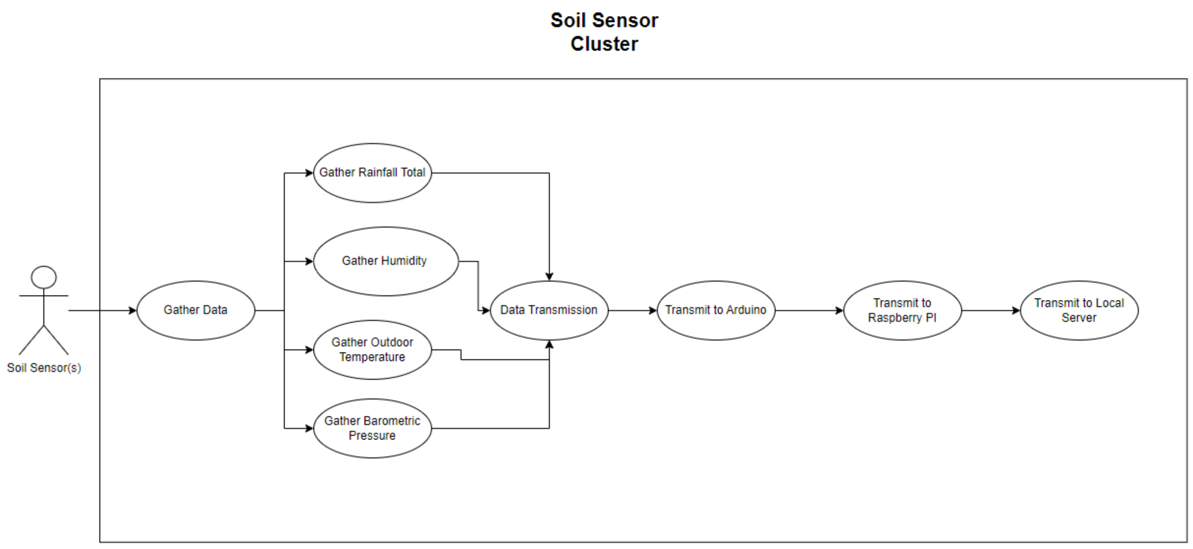

Figure 1.2: Use Case Diagram for Soil Sensors

Feature 3: Remote access to data

Users of the system must be able to access all of the gathered data from anywhere in the world. This feature enables the owner of the RWH system to see its current status even if they are not home. It also allows users interested in leaning about RWH systems to see one in action remotely. This is a very high priority feature.

REQ-3.1: The user must be able to access all current data using a web interface.

REQ-3.2: The user must be able to access historical data using a web interface.

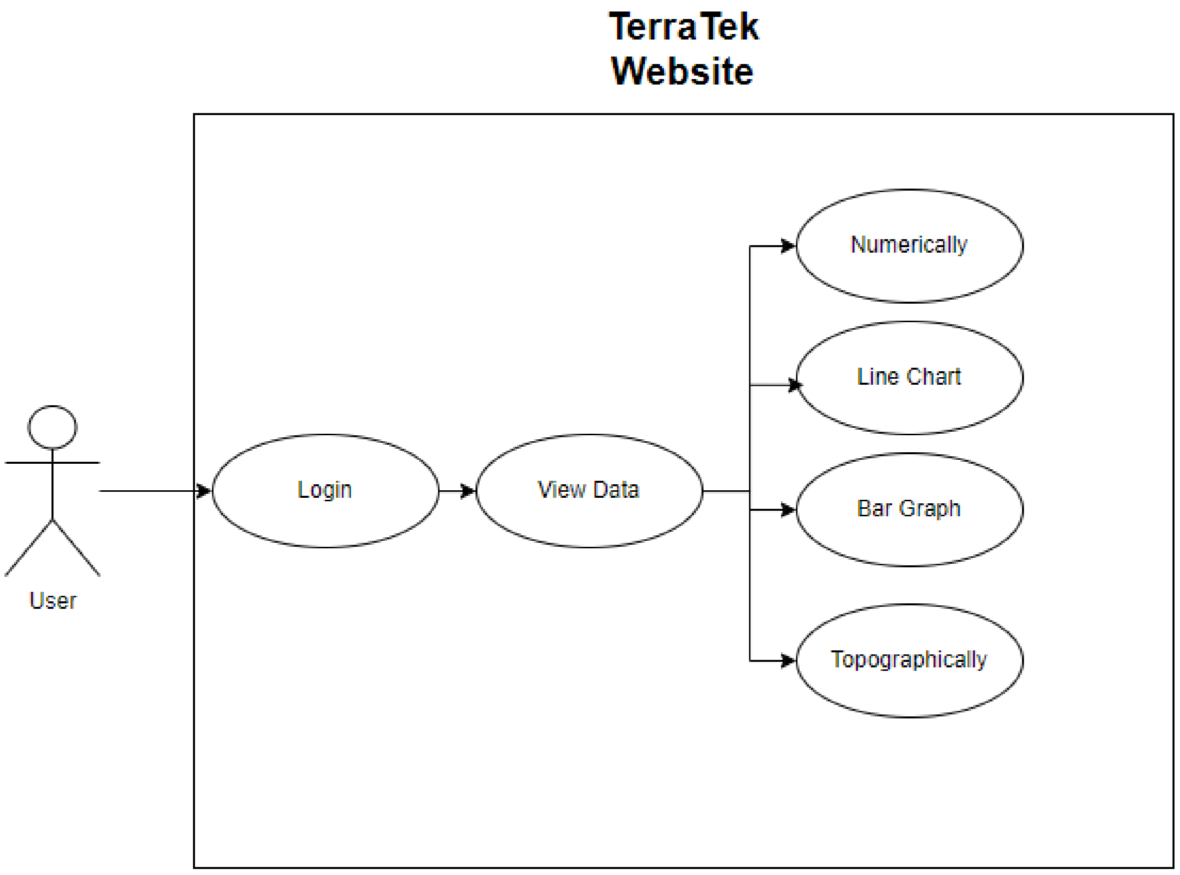

Figure 1.3: Use Case Diagram for Website

Feature 4: Data visualizations

Users of the system should be able to use data visualizations to read current and historical data. This is a medium priority feature.

REQ-4.1: The user should be able to use data visualisations to see current data.

REQ-4.2: The user should be able to use data visualisations to see historical data.

REQ-4.3: The user should be able to run custom reports for user defined sensors and time frames.

Feature 5: Remote camera

Users of the system could be able to show the users live views of the tanks and surrounding property. This enables users interested in RWH systems to actually see one in action. This is a low priority feature.

REQ-5.1: The user could be able to view live images of the tanks

REQ-5.2: The user could be able to view live images of the property

1.4 External Interface Requirements

1.4.1 User Interfaces

Users will interface with the system using a web browser to access the website. The system will support Firefox, Safari, and Chromium based browsers.

1.4.2 Software Interfaces

There will be many software interfaces. The Sensors use many different predefined interfacing protocols such as I2C. The sensor clusters will communicate to the central hub using a wireless data communication standards, such as WiFi or LoRa. The central hub will then send the aggregated sensor data to the server using a standard internet protocols.

1.5 Other Nonfunctional Requirements

1.5.1 Performance Requirements

REQ-6.1: The website must load the homepage in less than 5 seconds

REQ-6.2: The server must be able to handle at least 50 users accessing the website at the same time.

REQ-6.3: The system should be able to create historical reports in less than 10 seconds

REQ-6.4: The system should be built with no dependency on subscription based or free tier cloud services.

1.5.2 Software Quality Attributes

REQ-7.1: The website must be easy to use.

REQ-7.2: The website must remain accessible even if portions of the sensor network fail.

REQ-7.3: The system must be secure

Chapter 2

Software Design

2.1 Research

2.1.1 Arduino Sensors

DFRobot Gravity Analog Electrical Conductivity Meter (DFR0300)

The DFRobot Gravity Analog Electrical Conductivity Meter is designed to measure water's electrical conductivity, providing insight into the dissolved salts and overall water quality. This sensor is particularly useful in applications such as hydroponics and aquarium management. Using the DFRobot Gravity library, developers can easily calibrate the sensor and collect data efficiently, enhancing the user's ability to monitor water quality in real time [16].

Weather Meter Kit (SEN-15901)

The Weather Meter Kit is a comprehensive package that includes sensors to monitor weather conditions, such as wind speed, wind direction, and rainfall. These measurements are essential for setting up weather stations, where real-time data can be analyzed to predict weather trends. Integration with this kit varies depending on the specific sensors, as they may require unique libraries or standard analog and I2C communication protocols to process data accurately [20].

ME007YS Waterproof Ultrasonic Distance Sensor (SEN0312)

The ME007YS is a robust ultrasonic sensor designed for distance measurement in outdoor and industrial environments, even underwater. This sensor operates by emitting ultrasonic waves and capturing reflections. The NewPing library simplifies handling ultrasonic pulses and interpreting distance measurements, making this sensor reliable for applications in challenging environments [2].

Weather-proof Ultrasonic Sensor (SEN0208)

Similar to the ME007YS, the SEN0208 sensor provides reliable distance measurements in outdoor settings and is weatherproof. Utilizing the NewPing library, this sensor is capable of accurate data acquisition in harsh weather, making it suitable for long-term outdoor installations [21].

Waterproof DS18B20 Temperature Sensor Kit (KIT0021)

The DS18B20 is a popular waterproof temperature sensor known for its durability in harsh conditions. Using the OneWire protocol, multiple sensors can connect to a single data pin, a unique feature beneficial for applications requiring distributed temperature monitoring. The DallasTemperature library enables easy data acquisition from this sensor, allowing for seamless integration in temperature-critical monitoring setups [3].

Arduino MKR Environmental Shield (ASX00029)

The Arduino MKR Environmental Shield includes sensors for temperature, humidity, barometric pressure, and light intensity. This shield is particularly suitable for comprehensive environmental monitoring applications. The Arduino-MKRENV library simplifies data handling across multiple sensors, streamlining the process of integrating environmental metrics into projects [18].

Gravity Non-contact Liquid Level Sensor (SEN0204)

This non-contact sensor is ideal for environments where the liquid should not be touched directly, such as in sterile or hazardous conditions. The DFRobot-Sensor library offers straightforward methods for obtaining liquid level data, allowing for quick and safe implementation in sensitive monitoring applications [15].

MODBUS-RTU RS485 4-in-1 Soil Sensor (SEN0604)

This Soil Sensor Provides 4 different data outputs; Soil Moisture, Temperature, PH, and EC. This sensor is also built to last buried in the soil it is measuring, allowing it to be non-invasive and low maintenance. The ArduinoModbus Library allows for straightforward data gathering from the sensor to allow for quick setup and easy deployment. [4]

2.1.2 Website Framework

React.JS

ReactJS is a JavaScript library developed by Facebook for building user interfaces, primarily focused on component-based development, DOM(Document Object Model) manipulation, and various other functionalities [7]. It allows developers to create reusable UI components that handle the view layer of web applications efficiently. React updates and renders only the components that change, which improves performance and user experience. It also features a virtual DOM, which speeds up updates by minimizing direct manipulation of the real DOM.

NextJS

Nextjs is a React-based framework for building web applications [7]. It adds powerful features such as server-side rendering (SSR), static site generation (SSG), and dynamic routing, which enhance performance and SEO(Search Engine Optimization). With Nextjs, developers can render pages on the server before sending them to the client, making the initial load faster and more SEO-friendly. It also integrates well with APIs and data fetching techniques for modern web applications.

2.1.3 LoRaWAN

LoRa

LoRa, which stands for long range, is a wireless transmission protocol designed for long range, low power, applications. LoRa is capable of transmitting up to 5 kilometers in urban environments and up to 10 kilometers in rural environments with line of sight between devices [1]. The reason that LoRa is capable of transmitting such long distances is that it uses a radio modulation technology known as Chirp Spread Spectrum or (CSS) [11]. CSS works by sending series of chirps that encode data. Other radio protocols such as amplitude modulation (AM) and frequency modulation (FM) are highly susceptible to noise and obstructions and require high powered transmission devices to achieve the distances that they do. CSS, on the other hand, is very resilient to noise and obstructions, which makes it an ideal choice for urban environments [22]. The other thing that makes LoRa stand out is its incredibly low power usage. Unlike other radio protocols, it takes very little power to transmit and receive LoRa. This enables the use of batteries and solar in remote end-nodes [17], which makes LoRa a perfect choice for rural and farming applications. This is also what makes LoRa a great choice for our project. LoRa operates on license free sub-gigahertz bands, in the United States a common frequency choice is 915 MHz [14]. This means that there fewer costs and complexity that would be associated with operating on other bands. The major drawback to LoRa is data transmission rates, you can expect to see transmission speeds ranging from 250bit/s to 11kbit/s, however, for end nodes collecting data every couple of minutes this does not pose a significant issue [8].

LoRaWAN

LoRaWAN acts as a media access control (MAC) layer for the end nodes on the LoRa network [10]. LoRaWAN is a star network topology with many end nodes connecting to a gateway. Each gateway can support hundreds of devices, depending on how often they transmit data [17]. The gateway is responsible for collecting the packets sent by the end nodes and forwarding them to a network server.

LoRaWAN network server

A network runs software that manages one or more gateways. The network server typically stores the packets it receives from the various end nodes in a database and allows users to integrate their projects with the server. This entire architecture is a star of stars model and allows one network to cover a vast area with many gateways [22]. There are two main categories of networks servers, public and private. The main difference between the two categories is the entity operating the server and how access to the network is configured [5].

Public network servers (such as The Things Network) allow any end node that has been registered with the network to connect to any gateway on the network and have their packets forwarded to the users application. Public servers typically offer a tiered subscription model, many offer a free tier with support for a limited number ov devices and features [12]. It is worth noting that any end node can connect to any gateway, however, only end nodes that have been configured with the same network server will have their packets routed correctly.[10] All communication is encrypted and only the person who registered the end node is able to read the data, even if they do not own the gateway that the end node is connected to [10].

The other type of network servers are private network servers. These servers are hosted by the network owner either locally or on a web hosting platform and allow for more secure and private handling of data [5]. For many beginners, using a public network server is very convenient, all that they need to do is configure their gateway to forward packets to a public server like The Things Network, and they are ready to start connecting their end nodes. It is even possible that someone else nearby has a gateway that can be used instead of acquiring their own gateway. Setting up a private network server is a more complex undertaking and requires setting up a server to run the software that will control the gateways [5]. For this project we will be creating our own private network server to host our application.

ChirpStack

For people wanting to configure their own private network servers, there are several open-source projects that package all the network server components into one place. One of these projects is ChirpStack [19]. ChirpStack even has a dedicated Raspberry Pi OS that is ready to be configured as a network server. ChirpStack also includes a web interface for easy configuration and end node management. Additionally, ChirpStack provides a python package that enables easy integration using the gRPC API [6]. For these reasons, we believe that ChirpStack will work well for our project.

Gateways

As mentioned above, the gateway is responsible for collecting packets from end nodes and forwarding them via the internet to the network server. This typically occurs using UDP or a similar protocol. There are many gateways available with many different applications. One interesting option that we are currently investigating for our project is creating our own gateway using a Raspberry Pi and a gateway hat. These gateway hats, such as Elecrow's LR1302 HAT, connect directly to the Raspberry Pi and allow for a lot of configuration and flexibility [23]. The same Raspberry Pi running the gateway can also run the network server, this is a very compact and elegant solution that offers a lot of flexibility. The main concern is range, this is something that we need to test to determine if it is a viable solution for our project.

End Nodes

There are numerous end nodes that work using LoRa. The one that we have chosen for our project is the Arduino MKR WAN 1310. This board comes with the advantage of having all the support, documentation, and libraries that Arduino provides for all of its products. This board provides an easy way to integrate all the required sensors and connect to a gateway using LoRa [9].

2.1.4 Database

A database solution was required as for the implementation of the TSS would require a way to both store and organize the data collected from all sources.

SQL

SQL was decided as the database solution for this specific implementation. It was chosen as all members were familiar with the language as well as being perfectly suited for processing and storing text outputs collected from sensor data. How SQL expresses data is through mathematical relations [13]. The data collected from the raspberry pi will be expressed as the following relation:

ER Diagram

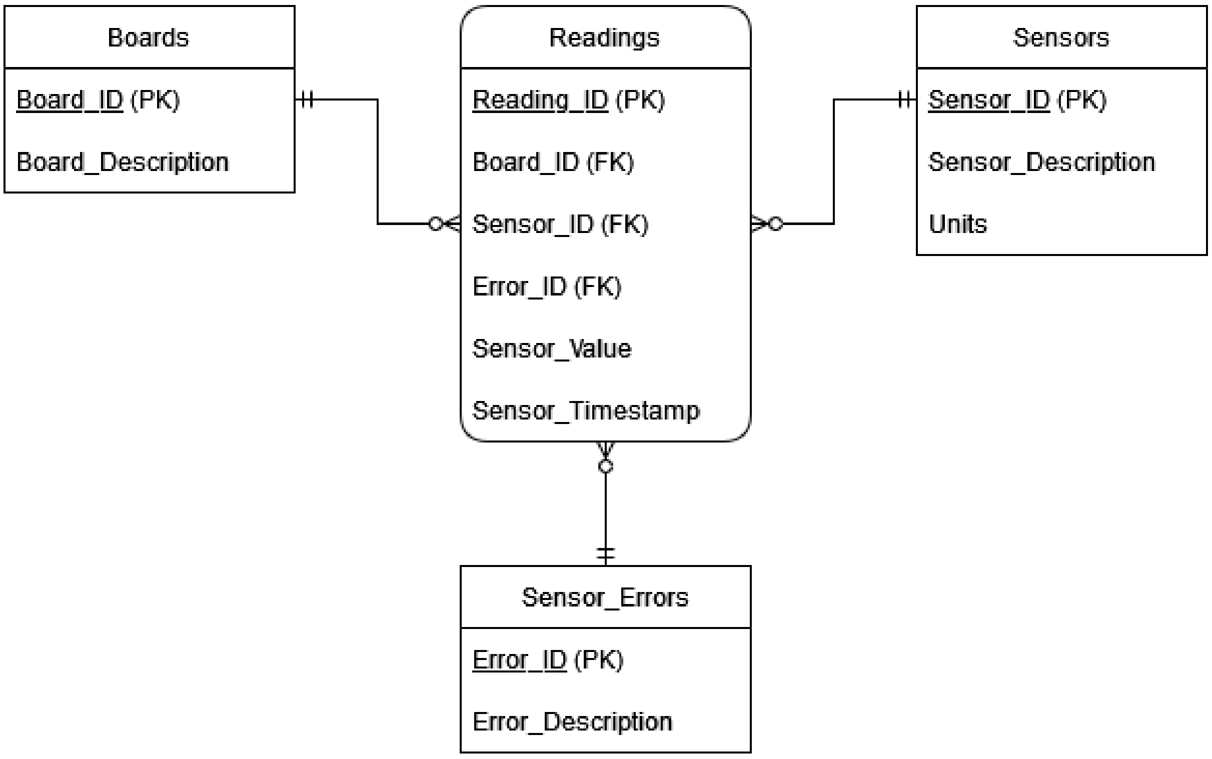

Figure 2.1: ER Diagram

This will serve as our schema for the database. Boards, Sensors and Readings serving as the entities present within. The Sensors table will represent individual sensor readings and the kind of sensor that is reading the data. The Boards table will represent a Cluster of sensors as well as that Cluster's location. Finally the Readings table will take the Board_ID and Sensor_ID from the Boards and Sensors tables in order to express the readings from any given sensor, which Cluster that sensor belongs too, and the time that the sensor has taken that specific reading.

2.1.5 Deployment Location Mapping

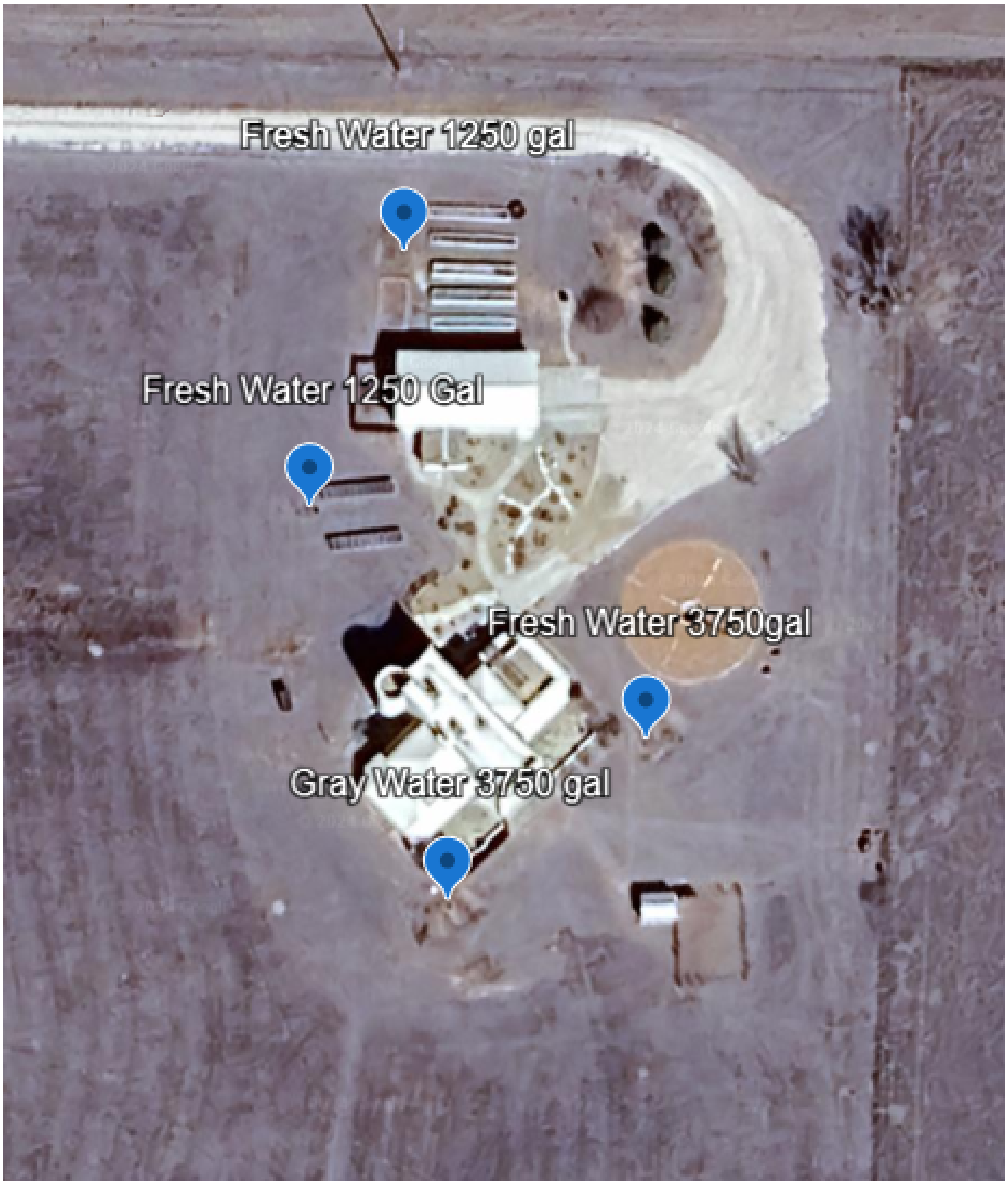

Figure 2.2: Water Tank Map

Water Tanks

There are 4 water tank locations as shown above in the photo. The two southern-most are fed by the house, fresh water from the roof catching rain, and gray water from the house appliances after use. The fresh water from the house is used to feed all of the houses water use, and the gray water is used to water plants in the area surrounding the house. The 2 northern tanks are fed by either side of the garage with fresh rainwater, and this water is used mostly to water the gardens they are adjacent to.

Garden Plots

There are many garden plots that we could attempt to gather data on, but the overhead of running so many sensors may prove to be extensive. To mitigate this the project will likely only be working on a select few of the garden plots to gather data. There are 4 raised garden boxes on the northern side of the garage, these 4 are used to grow annual vegetables.

2.2 Diagrams

2.2.1 Use Case Diagram

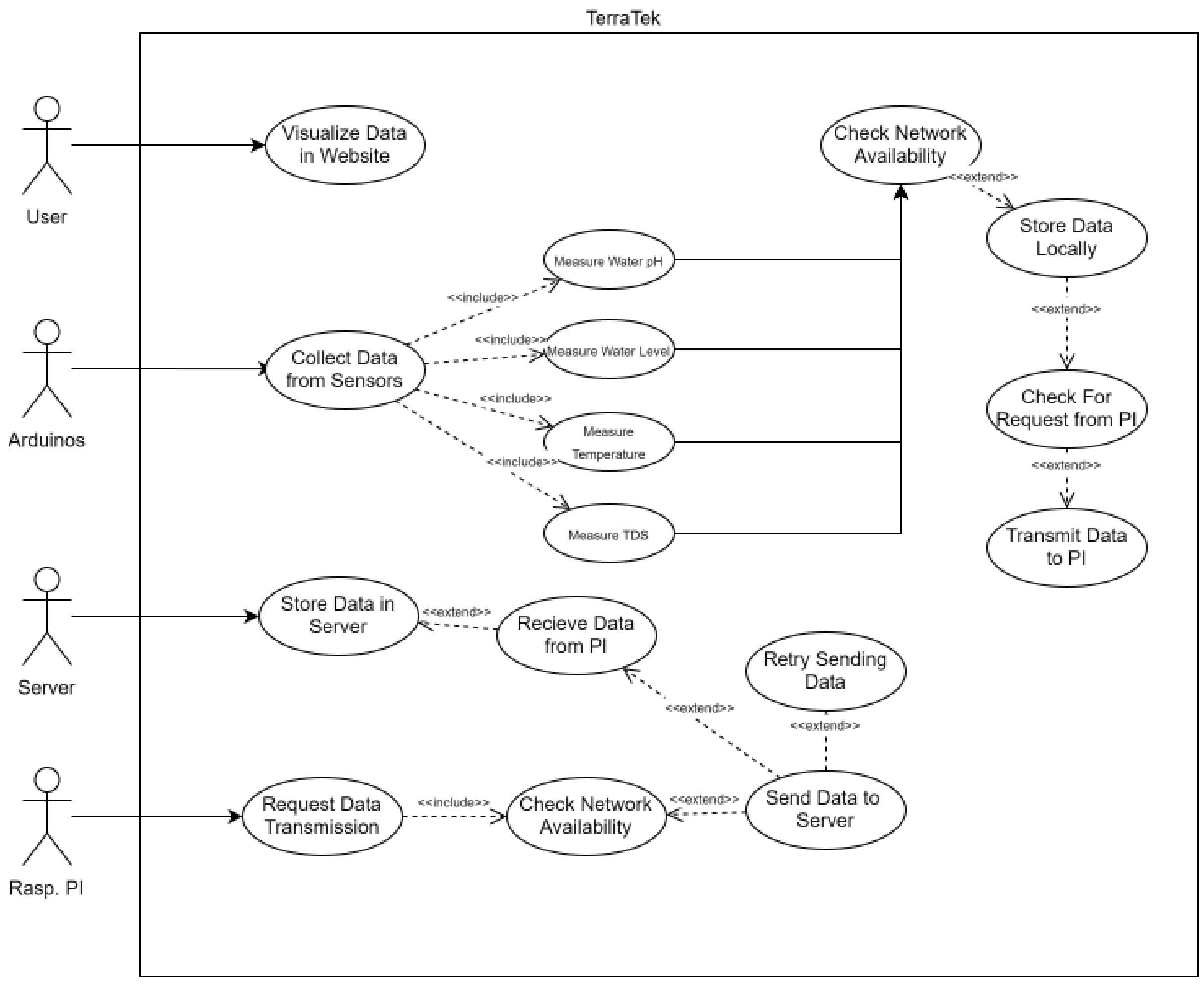

Figure 2.3: Use Case Diagram

2.2.2 Flow Diagram

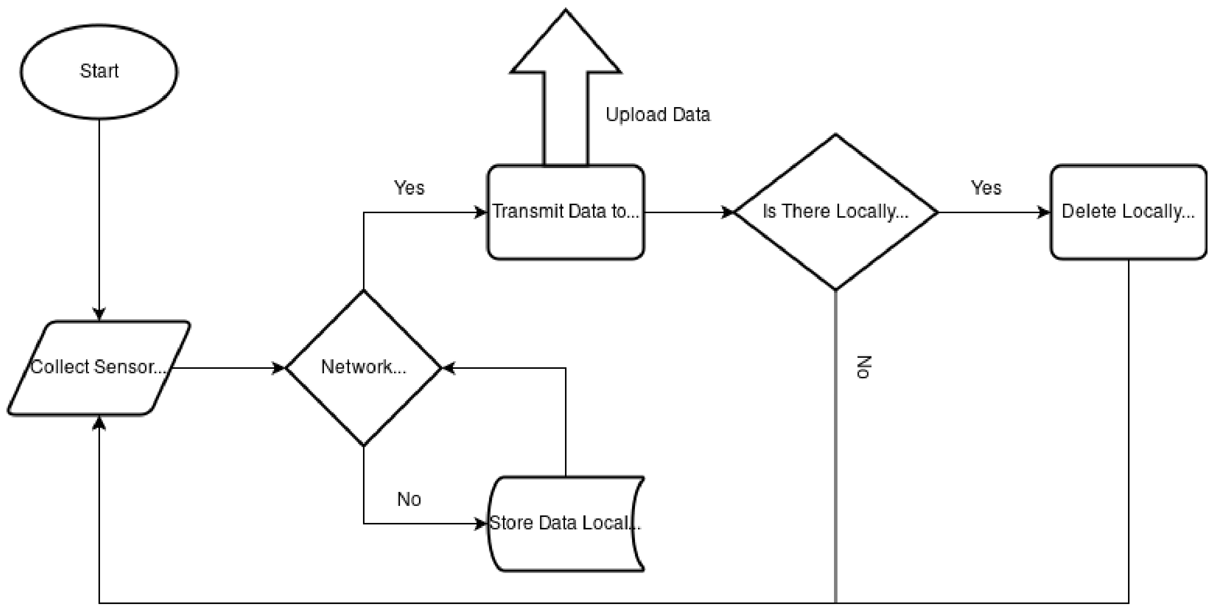

Figure 2.4: Flow Diagram

2.2.3 ER Diagram

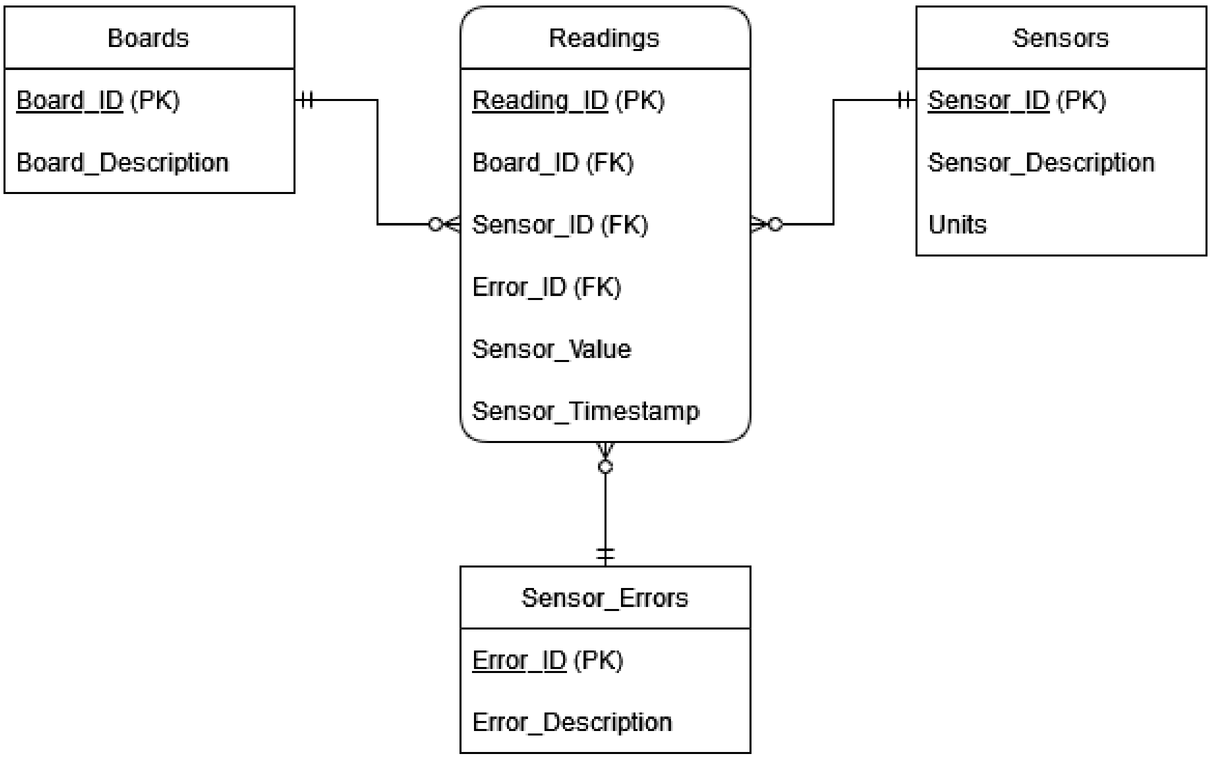

Figure 2.5: ER Diagram

2.2.4 Class Diagram

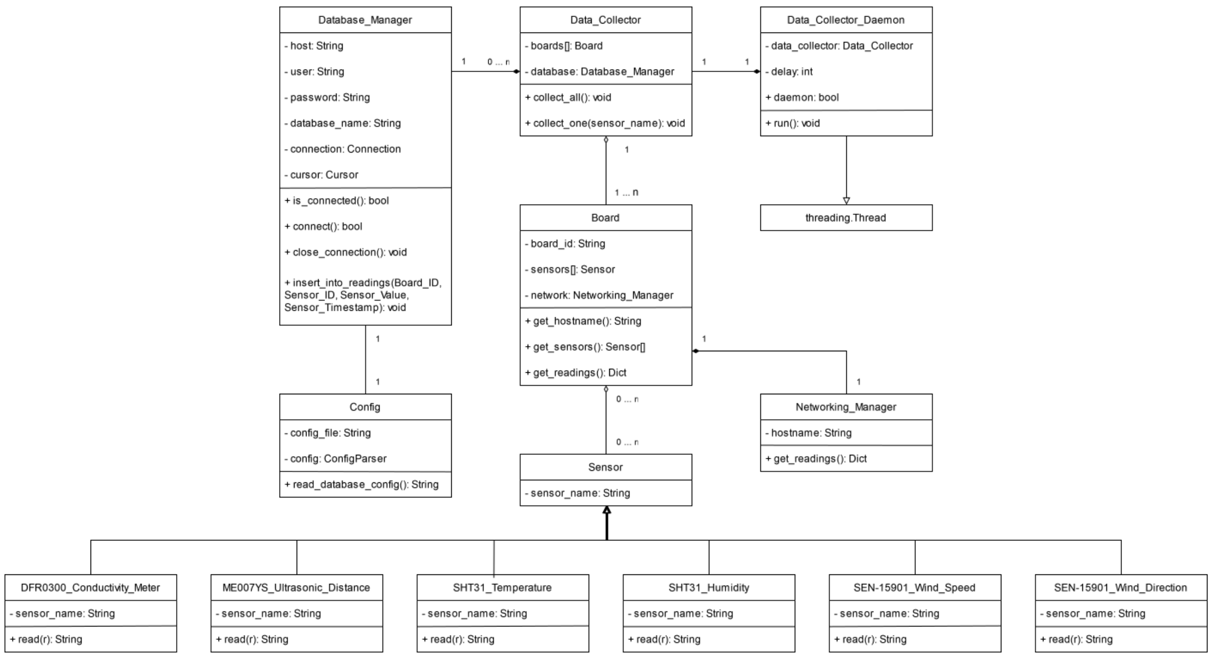

Figure 2.6: Class Diagram (Raspberry Pi)

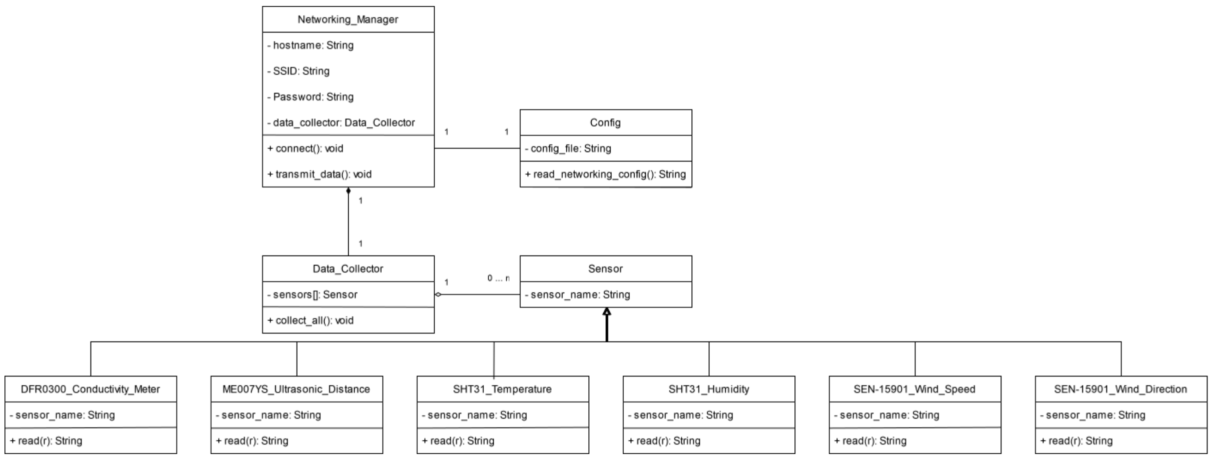

Figure 2.7: Class Diagram (Arduino)

2.2.5 Object Diagram

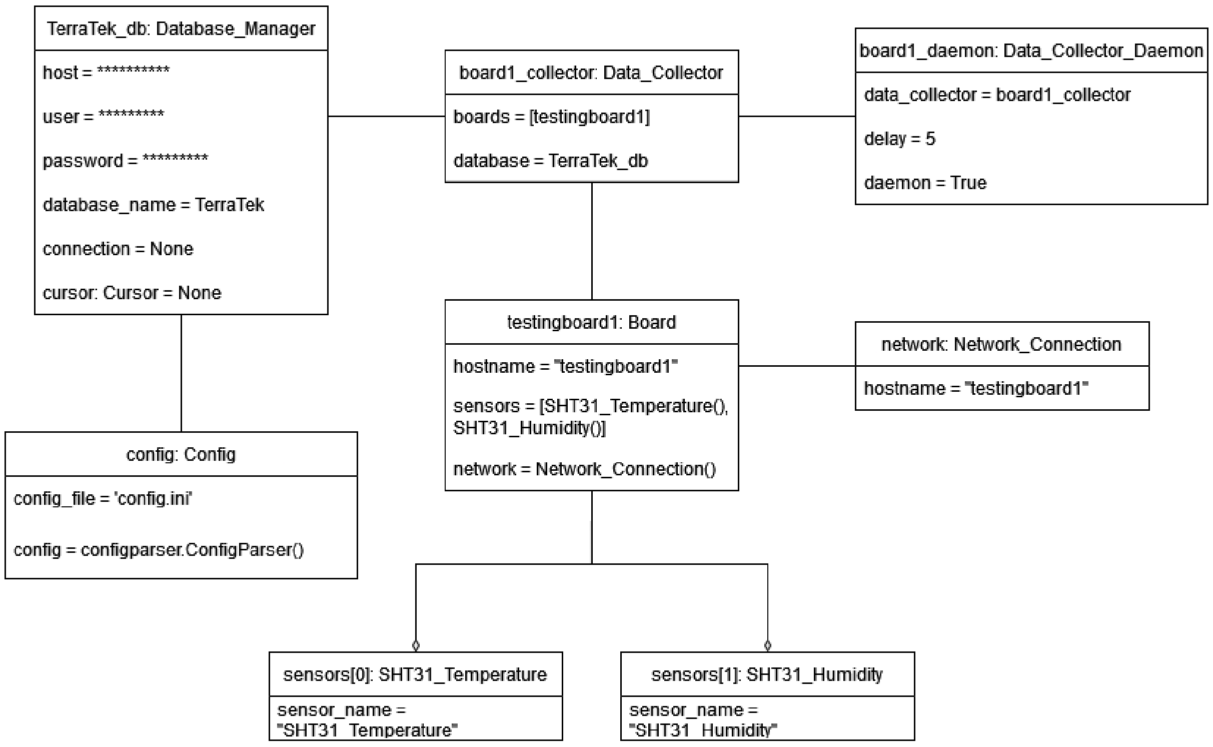
TerraTek_db: Database_Manager
host = **********
user = *********
password = *********
database_name = TerraTek
connection = None
cursor: Cursor = None
board1_collector: Data_Collector
boards = [testingboard1]
database = TerraTek_db
board1_daemon: Data_Collector_Daemon
data_collector = board1_collector
delay = 5
daemon = True
config: Config
config_file = 'config.ini'
config = configparser.ConfigParser()
testingboard1: Board
hostname = "testingboard1"
sensors = [SHT31_Temperature(), SHT31_Humidity()]
network = Network_Connection()
network: Network_Connection
hostname = "testingboard1"
sensors[0]: SHT31_Temperature
sensor_name = "SHT31_Temperature"
sensors[1]: SHT31_Humidity
sensor_name = "SHT31_Humidity"

Figure 2.8: Object Diagram

2.2.6 Sequence Diagram

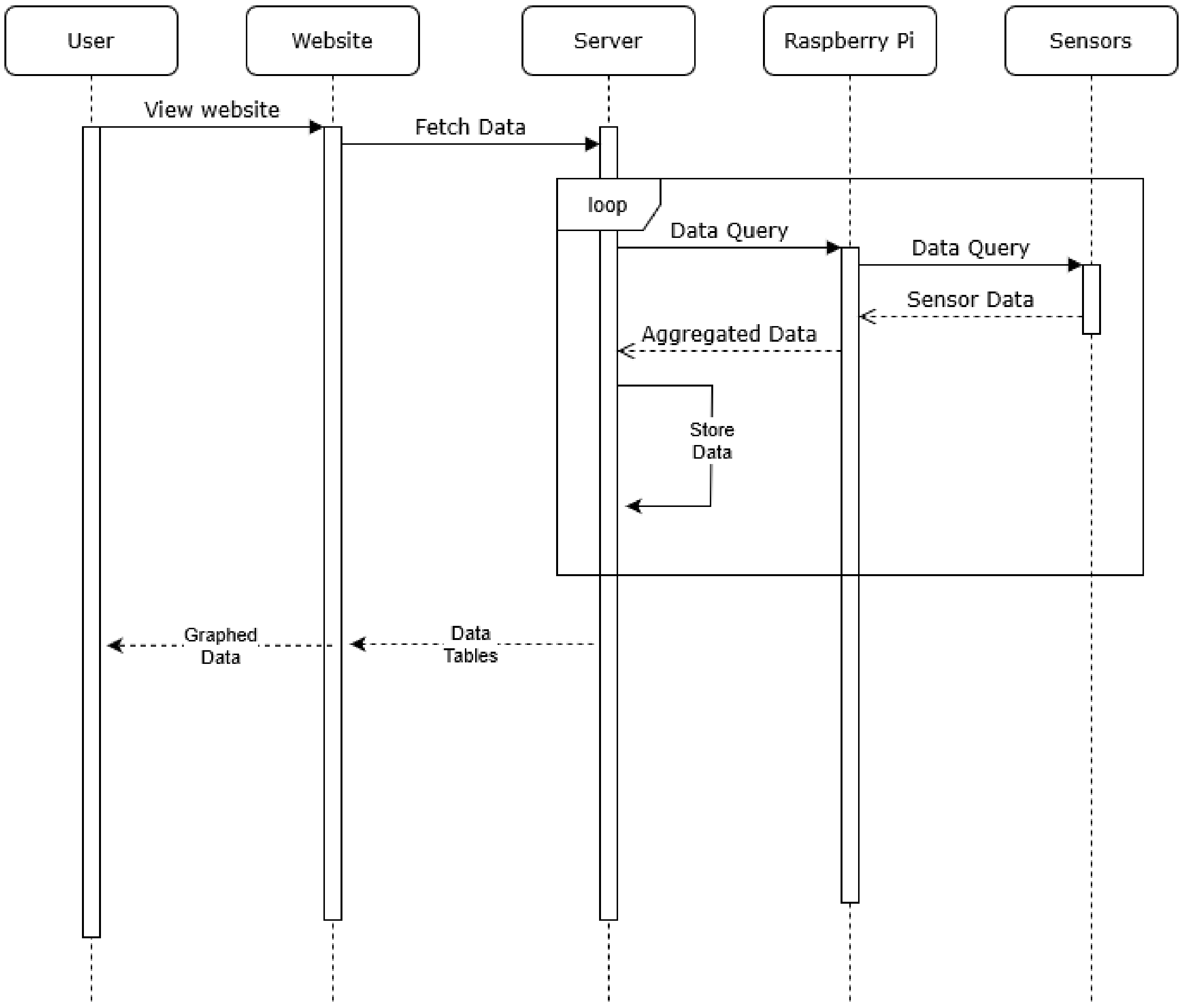

Figure 2.9: Sequence Diagram

2.2.7 DFD Level 0 Diagram

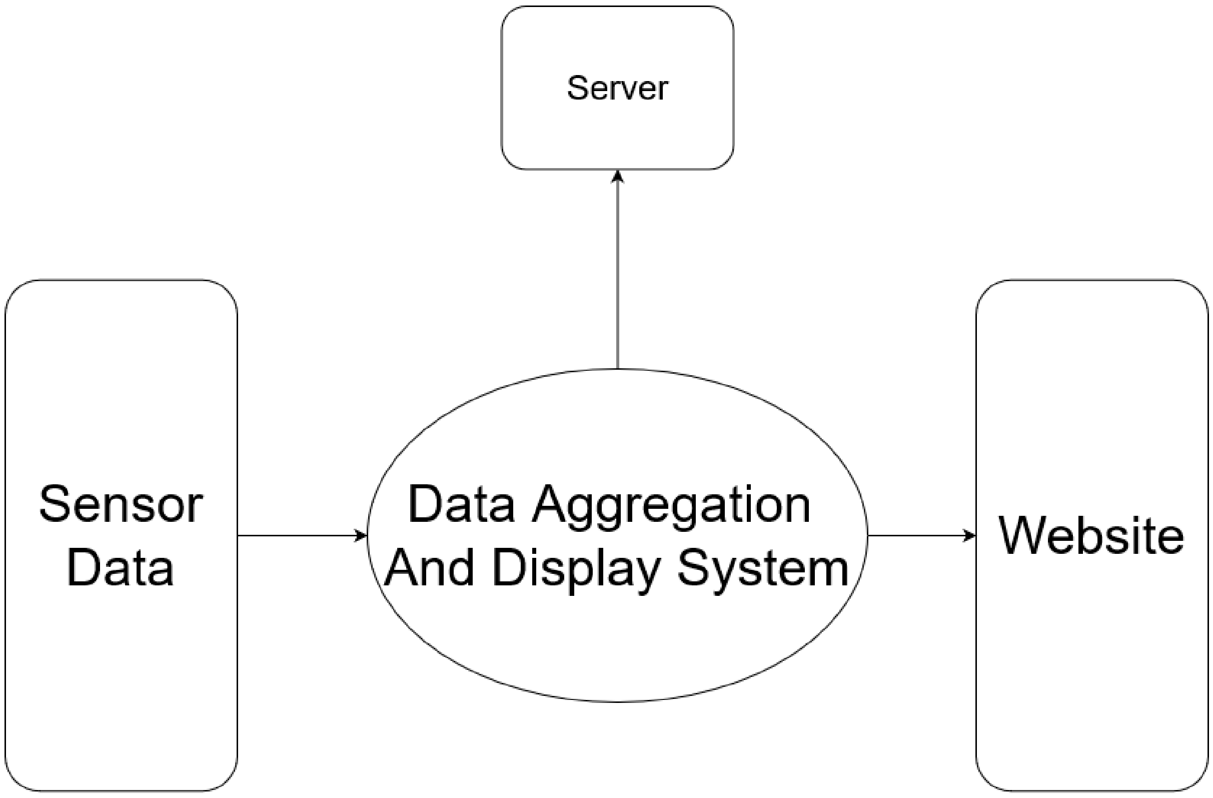

Figure 2.10: Data Flow Diagram Level 0

2.2.8 DFD Level 1 Diagram

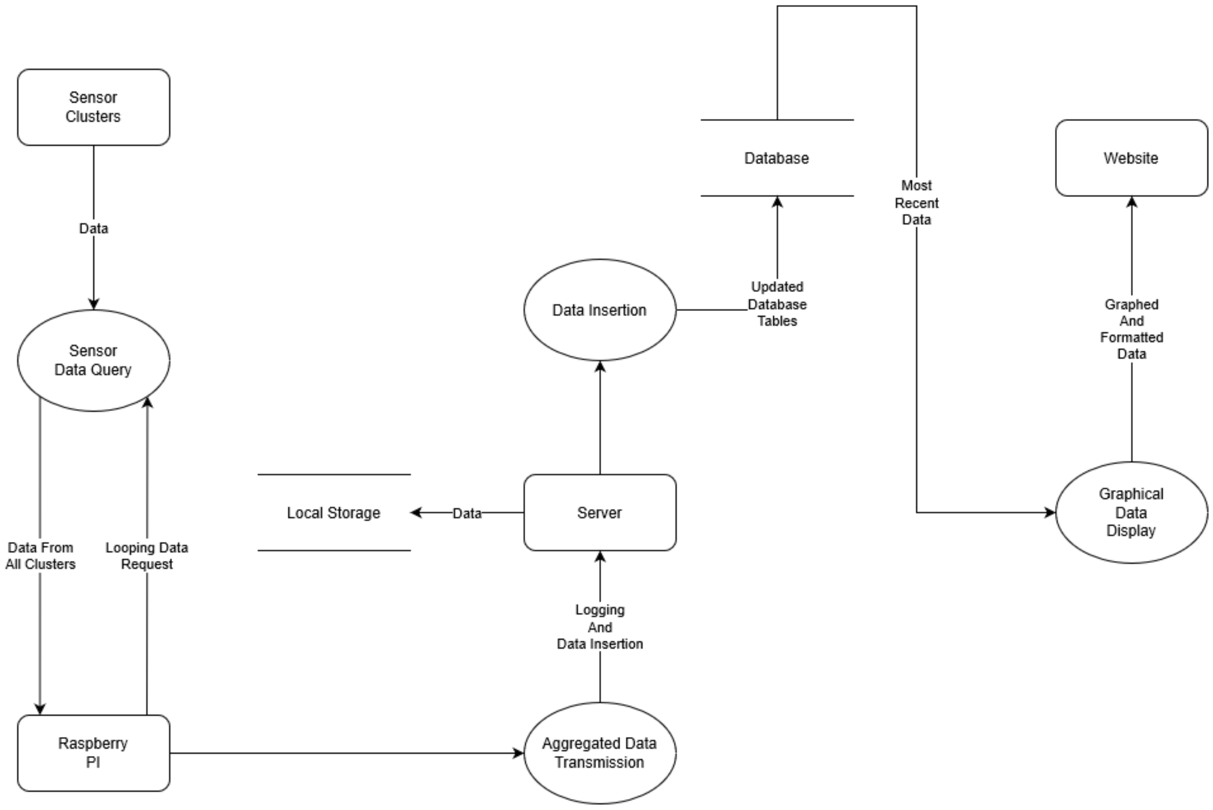

Figure 2.11: Data Flow Diagram Level 1

2.2.9 Raspberry Pi State Machine Diagram

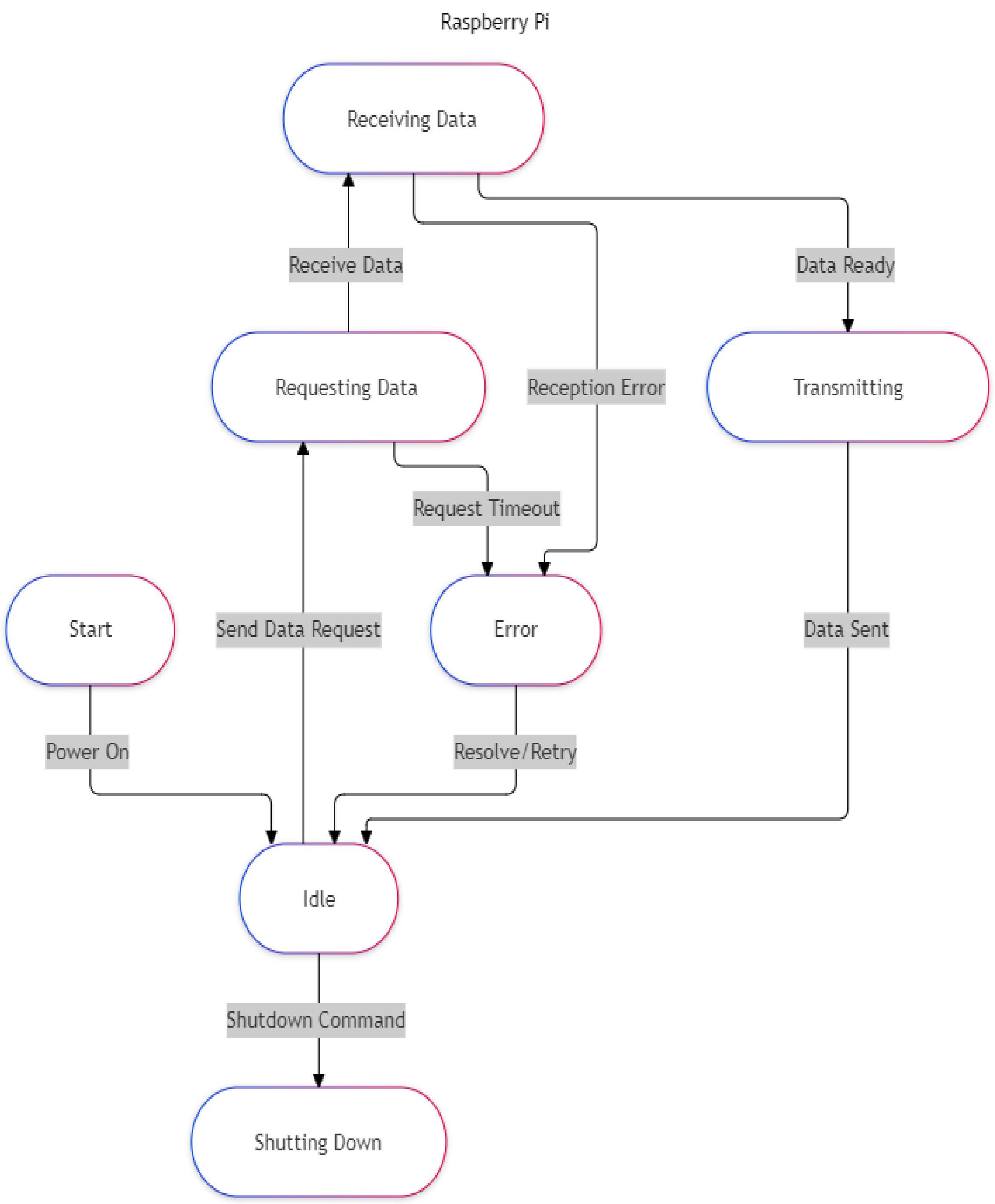

Figure 2.12: State Machine For Raspberry Pi

2.2.10 Sensor Cluster State Machine Diagram

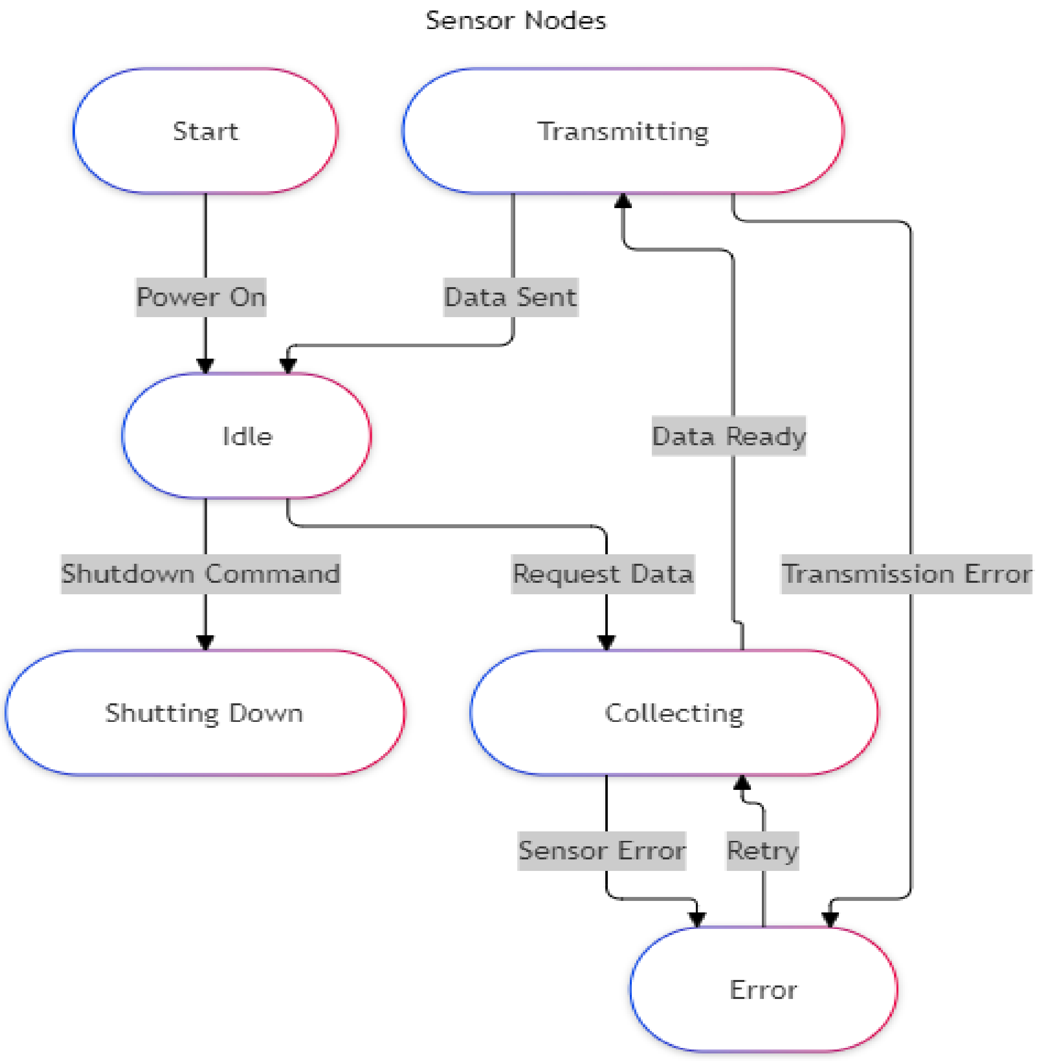

Figure 2.13: State Machine For Sensor Clusters

2.2.11 Server State Machine Diagram

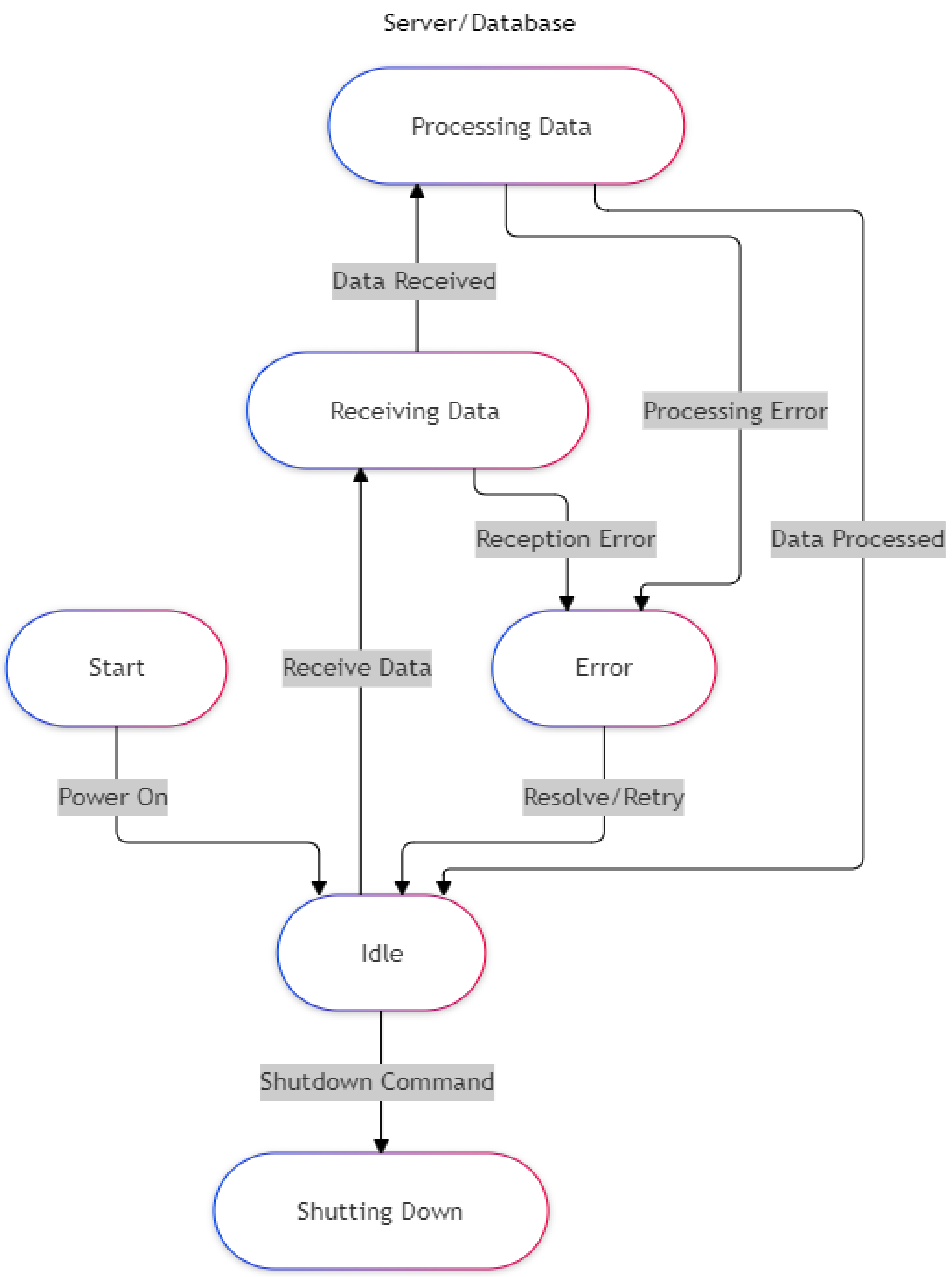

Figure 2.14: State Machine For Server

Website State Machine Diagram

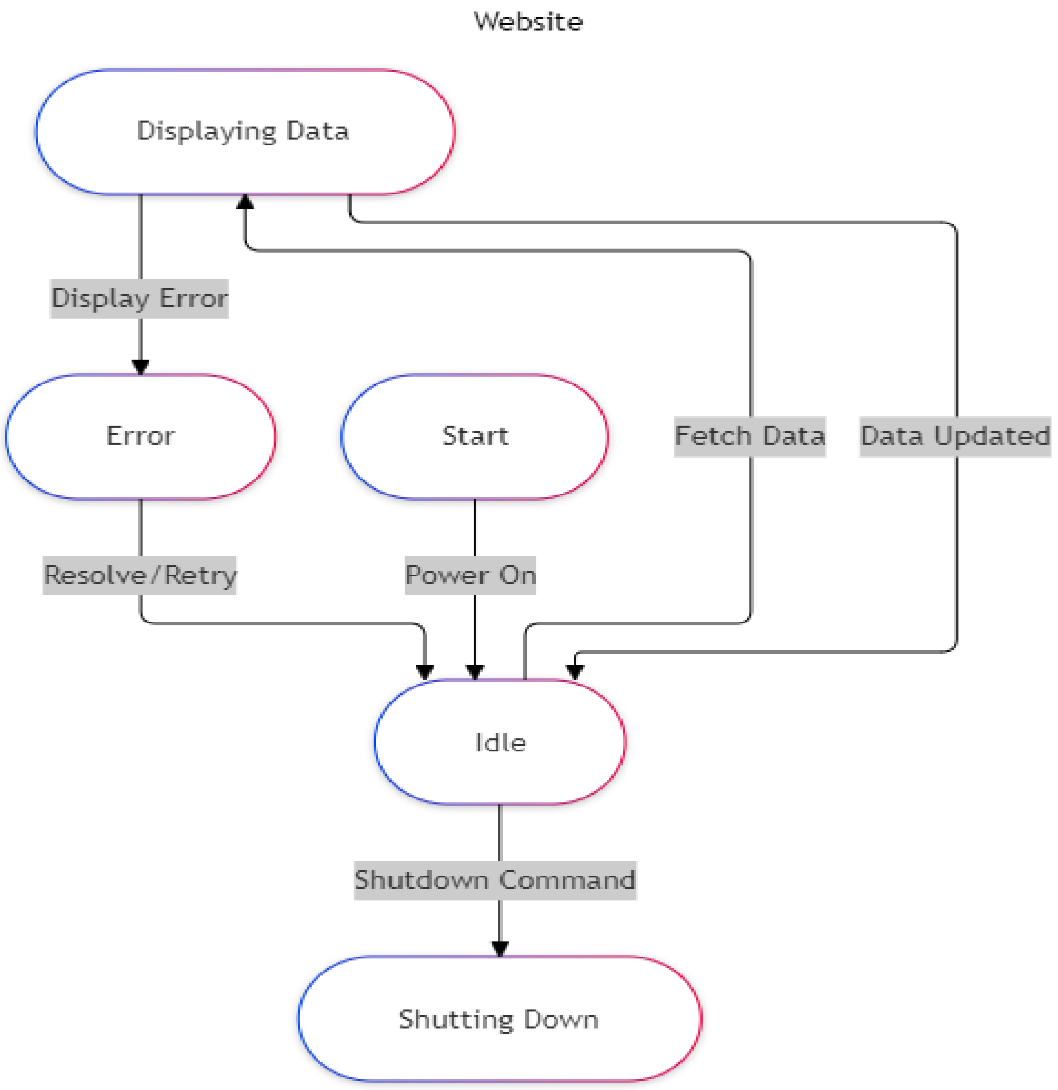

Figure 2.15: State Machine For Website

2.2.12 Component Diagram

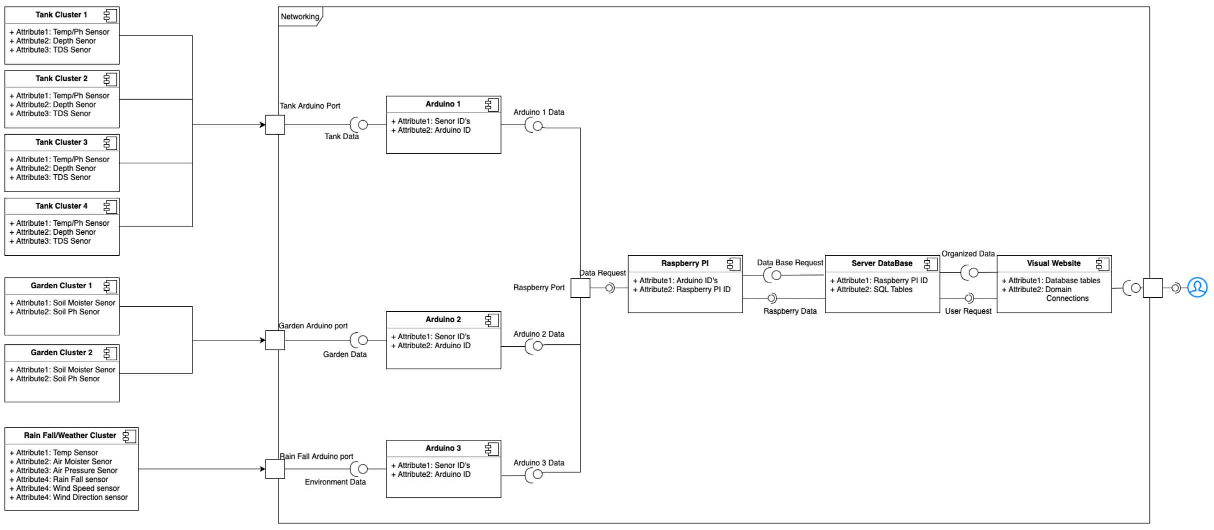
Tank Cluster 1
+ Attribute1: Temp/Ph Sensor
+ Attribute2: Depth Senor
+ Attribute3: TDS Senor
Tank Cluster 2
+ Attribute1: Temp/Ph Sensor
+ Attribute2: Depth Senor
+ Attribute3: TDS Senor
Tank Cluster 3
+ Attribute1: Temp/Ph Sensor
+ Attribute2: Depth Senor
+ Attribute3: TDS Senor
Tank Cluster 4
+ Attribute1: Temp/Ph Sensor
+ Attribute2: Depth Senor
+ Attribute3: TDS Senor
Garden Cluster 1
+ Attribute1: Soil Moister Senor
+ Attribute2: Soil Ph Senor
Garden Cluster 2
+ Attribute1: Soil Moister Senor
+ Attribute2: Soil Ph Senor
Rain Fall/Weather Cluster
+ Attribute1: Temp Sensor
+ Attribute2: Air Moister Senor
+ Attribute3: Air Pressure Senor
+ Attribute4: Rain Fall sensor
+ Attribute4: Wind Speed sensor
+ Attribute4: Wind Direction sensor
Networking
Tank Arduino Port
Tank Data
Arduino 1
+ Attribute1: Senor ID's
+ Attribute2: Arduino ID
Arduino 1 Data
Garden Arduino port
Garden Data
Arduino 2
+ Attribute1: Senor ID's
+ Attribute2: Arduino ID
Arduino 2 Data
Rain Fall Arduino port
Environment Data
Arduino 3
+ Attribute1: Senor ID's
+ Attribute2: Arduino ID
Arduino 3 Data
Raspberry Port
Data Request
Raspberry PI
+ Attribute1: Arduino ID's
+ Attribute2: Raspberry PI ID
Data Base Request
Raspberry Data
Server DataBase
+ Attribute1: Raspberry PI ID
+ Attribute2: SQL Tables
Organized Data
User Request
Visual Website
+ Attribute1: Database tables
+ Attribute2: Domain Connections

Figure 2.16: Component Diagram

Chapter 3

Software Implementation and Testing

3.1 Test Cases

3.1.1 Test Case 1: Arduino Properly Transmits Data to LoRaWAN Gateway

Purpose:	
The purpose of this test case is to ensure that the Arduino can properly transmit data over the LoRaWAN connection to the gateway.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
William Kolath	In order for this test to be performed, the gateway must be running. The gateway must be configured to receive on the US915_1 band. The gateway must be registered with the local Chirp Stack application server.
Notes:	

Table 3.2: Test Case 1 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Vali-dated	Pass /Fail
1	Run the 'FirstConfiguration.ino' example sketch on the Arduino	Arduino outputs device EUI to serial monitor			
2	On the Chirp Stack web interface create a new device named "test" using the MKRWAN 1310 template.	New device named "test" has been created.			
3	On the device configuration page enter the device EUI from step one.	The "test" device EUI matches the output on the serial monitor.			
4	On the device configuration page enter "0000000000000000" for the APP EUI and generate a random number for the APP KEY.	The "test" device has "0000000000000000" for the APP EUI and a random hex number for the APP KEY.			
5	On the serial monitor enter 1 to enter OTAA Join.	After entering 1, the Arduino will ask for the APP EUI.			
6	On the serial monitor enter "0000000000000000" for the APP EUI.	After entering the APP EUI, the Arduino will ask for the APP KEY.			
7	On the serial monitor enter the APP KEY generated in step 4.	The Arduino will join the LoRaWAN network and print "Message sent correctly!" to indicate that the device setup was successful.			

Table 3.4: Arduino Properly Transmits Data to LoRaWAN Gateway

3.1.2 Test Case 2: Raspberry Pi is Able to Request Data from LoRaWAN Application Server

Purpose:	
The purpose of this test is to ensure that the Raspberry Pi hub is able to properly request data from the LoRaWAN Application server.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
William Kolath	Chirp Stack must be installed and running on the gateway. At least one device must be connected and configures to transmit data in order to test data transfer.
Notes:	

Table 3.6: Test Case 2 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Create a API key in the Chirp Stack web interface.	API key created.			
2	Enter the API key created in step one in the ”config.ini” file on the Raspberry Pi.	The ”config.ini” properly reflects the API key created in step 1.			
3	Run the ”ChipStack-DataCollector.py” python script.	The most recent data sent to the LoRaWAN gateway will be printed to the terminal periodically.			

Table 3.8: Raspberry Pi is Able to Request Data from LoRaWAN Application Server

3.1.3 Test Case 3: Raspberry Pi Properly Sends Data to Database

Purpose:	
The purpose of this test is to ensure that the Raspberry Pi hub is able to insert data in to the database.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
William Kolath	The server must be running and port 3306 must be forwarded. The database must be running and the tables must match the ER Diagram.
Notes:	

Table 3.9: Test Case 3 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Enter the database IP address, username, password, and database name in the ”config.ini” file on the Rasspberry Pi.	The ”config.ini” properly reflects database configuration.			
2	Run the ”Database-DataInserter.py” python script.	A message will be displayed on the terminal periodically indicating that data has been inserted in the database.			
3	On MySQL workbench, run ”DisplayTables.sql”	The output shows the data that the raspberry pi inserted in step 2.			

Table 3.11: Raspberry Pi Properly Sends Data to Database

3.1.4 Test Case 4: Arduino Handles Invalid Sensor Values from Ultrasonic Sensor

Purpose:	
The purpose of this test is to ensure that the arduino knows how to handle the invalid results from the ultrasonic sensor ranges	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Hector Ceballos	The sensor must be properly wired up to the ENV SHIELD which is on the MKR WAN board. The probe must be free of any obstacles to get proper readings.
Notes:	

Table 3.12: Test Case 4 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Sensor must be powered up and running	The lights on the boards should be on			
2	As sensor reads data and outputs data, check if values are properly being read	Values ranging from 21 to 600 should be outputted			
3	If range reads at a constant 600 no matter what , there is an error	If true, then proceed to error handling			
4	Output error handler for the ultrasonic sensor	Output error codes for the ultrasonic sensor			

Table 3.14: Arduino Handles Invalid Sensor Values from Ultrasonic Sensor

3.1.5 Test Case 5: Arduino Handles Invalid Sensor Values from Temperature Probe

Purpose:	
The purpose of this test is to ensure that the arduino knows how to handle the invalid results from the temperature probe sensor	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Hector Ceballos	The sensor must be properly wired up to the ENV SHIELD which is on the MKR WAN board. The probe must be free of any obstacles to get proper readings.
Notes:	

Table 3.15: Test Case 5 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Sensor must be powered up and running	The lights on the boards should be on			
2	As sensor reads data and outputs data, check if values are properly being read	Values ranging from -125 to 196 should be outputted			
3	If temperatures reading -125 and 196, there is an error in reading the temperature	If true, then proceed to error handling			
4	Output error handler for miscalculated temperatures	Output error codes for the temperature probe			

Table 3.17: Arduino Handles Invalid Sensor Values from Temperature Probe

3.1.6 Test Case 6: Arduino Handles Invalid Sensor Values from Electrical Conductivity Sensor

Purpose:	
The purpose of this test case is to verify that the Arduino can identify and handle invalid or extreme values from the Electrical Conductivity Sensor	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Kirk Blood	The Sensor must be properly wired up to the ENV Shield and PCB. The PCB must have an external EEProm chip installed. Sensor must be submerged in the target liquid.
Notes:	

Table 3.18: Test Case 6 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Disconnect the Electrical Conductivity sensor or send out of range data to simulate an error.	Arduino recognizes invalid data and outputs an error message on the monitor			
2	Verify Arduino catches the invalid data and outputs the proper error code.	The Arduino outputs a invalid data error code and blocks the transmission of bad data.			
3	Reconnect the sensor and check calibration	Sensor Powers on and either logs proper data or needs calibration			
4	Calibrate the sensor if needed, using proper calibration solutions.	Sensor transmits proper data			

Table 3.20: EC Sensor Test Case

3.1.7 Test Case 7: Arduino Handles Invalid Sensor Values from Ph Sensor

Purpose:	
The purpose of this test case is to verify that the Arduino can identify and handle invalid or extreme values from the PH Sensor	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Kirk Blood	The Sensor must be properly wired up to the ENV Shield and PCB. The PCB must have an external EEProm chip installed. Sensor must be submerged in the target liquid.
Notes:	

Table 3.21: Test Case 7 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Disconnect the PH sensor or send out of range data to simulate an error.	Arduino recognizes invalid data and outputs an error message on the monitor			
2	Verify Arduino catches the invalid data and outputs the proper error code.	The Arduino outputs a invalid data error code and blocks the transmission of bad data.			
3	Reconnect the sensor and check calibration	Sensor Powers on and either logs proper data or needs calibration			
4	Calibrate the sensor if needed, using proper calibration solutions.	Sensor transmits proper data			

Table 3.23: PH Sensor Test Case

3.1.8 Test Case 8: Arduino Handles Invalid Sensor Values from Rainfall Sensor

Purpose:	
The purpose of this test case is to verify that the Arduino can identify and handle invalid or extreme values from the rainfall sensor.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Preston DeShazo	The rainfall sensor must be connected to the Arduino, and the Arduino should be programmed to read values from this sensor.
Notes:	

Table 3.24: Test Case 8 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Disconnect the rainfall sensor or send an out-of-range voltage to simulate an error.	Arduino recognizes invalid data and outputs an error message on the serial monitor.			
2	Observe if the Arduino logs or flags the error in the system, avoiding incorrect data storage.	Arduino logs the error, and data is not sent to the LoRaWAN gateway.			
3	Reconnect the rainfall sensor with valid input values and verify normal operation.	Arduino resumes normal data reading and transmits correct data.			

Table 3.26: Arduino Handles Invalid Sensor Values from Rainfall Sensor

3.1.9 Test Case 9: Arduino Handles Invalid Sensor Values from Wind Direction Sensor

Purpose:	
The purpose of this test case is to verify that the Arduino can identify and handle invalid or extreme values from the Wind Direction sensor.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Preston DeShazo	The wind direction sensor must be connected to the Arduino, and the Arduino should be configured to read values from this sensor.
Notes:	

Table 3.27: Test Case 9 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Disconnect the wind direction sensor or provide an out-of-range signal.	Arduino detects invalid data and outputs an error message on the serial monitor.			
2	Verify that the Arduino prevents the transmission of erroneous data to the LoRaWAN gateway.	The Arduino blocks data transmission and logs the error locally.			
3	Reconnect the sensor and provide valid data to check system recovery.	Arduino resumes normal data reading and transmits correct data.			

Table 3.29: Arduino Handles Invalid Sensor Values from Wind Direction Sensor

3.1.10 Test Case 10: Arduino Handles Invalid Sensor Values from Wind Speed Sensor

Purpose:	
The purpose of this test case is to verify that the Arduino can identify and handle invalid or extreme values from the Wind Speed sensor.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Preston DeShazo	The wind speed sensor must be connected to the Arduino, and the Arduino should be set up to read and interpret values from this sensor.
Notes:	

Table 3.30: Test Case 10 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Disconnect the wind speed sensor or simulate an error by sending out-of-range values.	Arduino recognizes the error and displays a message on the serial monitor.			
2	Observe if the Arduino prevents the faulty data from being logged or transmitted.	The Arduino blocks erroneous data and logs the incident without forwarding it.			
3	Reconnect the wind speed sensor with valid data and check system restoration.	Arduino resumes correct data logging and transmission with valid sensor input.			

Table 3.32: Arduino Handles Invalid Sensor Values from Wind Speed Sensor

3.1.11 Test Case 11: Arduino Handles Invalid Sensor Values from Atmospheric Humidity Sensor

Purpose:	
The purpose of this test case is to verify that the Arduino can identify and handle invalid/valid values from the Humidity sensor.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Hector Ceballos	The sensor must be properly wired up to the ENV SHIELD which is on the MKR WAN board. The probe must be free of any obstacles to get proper readings.
Notes:	

Table 3.33: Test Case 11 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Sensor must be powered up and running	The lights on the boards should be on			
2	As sensor reads data and outputs data, check if values are properly being read	Values ranging from 0 to 100 should be outputted			
3	If range reads abnormal values there is an error	If true, then proceed to error handling			
4	Output error handler for the humidity sensor	Output error codes for the humidity sensor			

Table 3.35: Arduino Handles Invalid Sensor Values from Atmospheric Humidity Sensor

3.1.12 Test Case 12: Arduino Handles Invalid Sensor Values from Barometric Pressure Sensor

Purpose:	
The purpose of this test case is to verify that the Arduino can identify and handle invalid/valid values from the barometric sensor.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Hector Ceballos	The sensor must be properly wired up to the ENV SHIELD which is on the MKR WAN board. The probe must be free of any obstacles to get proper readings.
Notes:	

Table 3.36: Test Case 12 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Sensor must be powered up and running	The lights on the boards should be on			
2	As sensor reads data and outputs data, check if values are properly being read	Values ranging from 260kPA to 1260 kPA should be outputted			
3	If range reads abnormal values there is an error	If true, then proceed to error handling			
4	Output error handler for the humidity sensor	Output error codes for the humidity sensor			

Table 3.38: Arduino Handles Invalid Sensor Values from Atmospheric Humidity Sensor

3.1.13 Test Case 13: Data from sensor failure transmitted to Database

Purpose:	
This test case is to verify the behavior of the database if the raspberry pi transmits incorrect data to the database	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Dale Andre Descallar	Raspberry Pi is receiving data from all sensors and is transmitting data to the database
Notes:	

Table 3.40: Test Case 13 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Disconnect Rainfall Sensor	Hardware error code returned as reading			
2	Disconnect wind speed Sensor	Hardware error code returned as reading			
3	Disconnect Wind Direction Sensor	Hardware error code returned as reading			
4	Disconnect Soil Sensor	Arbitrarily large or small number returned as reading			
5	Disconnect Shield	Hardware error code returned as reading			
6	Disconnect Conductivity Meter	Hardware error code returned as reading			
7	Disconnect Temperature Sensor	Hardware error code returned as reading			
8	Disconnect Soil Sensor	Hardware error code returned as reading			

Table 3.42: Data from sensor failure transmitted to Database

3.1.14 Test Case 14: No data is transmitted to Database

Purpose:	
This test case is to validate the behavior of the database in the event that it stops receiving data from raspberry pi in the event of hardware failure from the pi itself.	
Test Run Information:	
Tester Name / Date:	Prerequisites / Required Configuration:
Dale Andre Descallar	Raspberry Pi is receiving data from all sensors and is transmitting data to the database
Notes:	

Table 3.44: Test Case 14 Description

TEST SCRIPT STEPS/RESULTS					
Step	Test Step/Input	Expected Results	Actual Results	Req Validated	Pass /Fail
1	Power off Raspberry Pi	Database will stop storing data from that moment			
2	Wait for 5 minutes and power on Raspberry Pi	Database will resume data insertion at the moment the raspberry pi resumes communication			

Table 3.46: No data is transmitted to Database

References

[1] *About Lorawan.* Lora Alliance. URL: https://lora-alliance.org/about-lorawan/ (visited on 10/25/2024).

[2] Random Nerd Tutorials Arduino. *ME007YS Waterproof Ultrasonic Distance Sensor (SEN0312).* 2024. URL: https://www.arduino.cc/en/Guide/ME007YS.

[3] Arduino Chip Wired. *Waterproof DS18B20 Temperature Sensor Kit (KIT0021).* 2024. URL: https://www.arduino.cc/en/Guide/DS18B20.

[4] DFRobot. *MODBUS-RTU RS485 4-in-1 Soil Sensor (SEN0604).* 2024. URL: https://wiki.dfrobot.com/RS485_Soil_Sensor_Temperature_Humidity_EC_PH_SKU_SEN0604#target_1.

[5] Radek Fujdiak, Konstantin Mikhaylov, Jan Pospisil, Ales Povalac, and Jiri Misurec. "Insights into the Issue of Deploying a Private LoRaWAN". In: *Sensors* 22.5 (Mar. 2022), pp. 2042–2042. DOI: https://doi.org/10.3390/s22052042. URL: https://www.mdpi.com/1424-8220/22/5/2042.

[6] Python Package Index. *ChirpStack API.* 2024. URL: https://pypi.org/project/chirpstack-api/ (visited on 10/25/2024).

[7] Mochammad Fariz Syah Lazuardy and Dyah Anggraini. "Modern front end web architectures with react. js and next. js". In: *Research Journal of Advanced Engineering and Science* 7.1 (2022), pp. 132–141.

[8] *Limitations.* The Things Network. URL: https://www.thethingsnetwork.org/docs/lorawan/limitations/ (visited on 10/25/2024).

[9] *LoRaWAN*. Arduino. 2024. URL: https://docs.arduino.cc/arduino-cloud/hardware/lora/ (visited on 10/25/2024).

[10] Alireza Maleki, Ha H. Nguyen, Ebrahim Bedeer, and Robert Barton. "A Tutorial on Chirp Spread Spectrum Modulation for LoRaWAN: Basics and Key Advances". In: *IEEE Open Journal of the Communications Society* 5 (2024), pp. 4578–4612. DOI: 10.1109/OJCOMS.2024.3433502.

[11] Biswajit Paul. "An Overview of LoRaWAN". In: *WSEAS TRANSACTIONS ON COMMUNICATIONS* 19 (Jan. 2021), pp. 231–239. DOI: https://doi.org/10.37394/23204.2020.19.27. URL: https://wseas.com/journals/articles.php?id=1302.

[12] *Pricing Plans*. The Things Network. URL: https://www.thethingsindustries.com/stack/plans/#:~:text=%24230%2Fmonth&text=Our%20powerful%20LoRaWAN%20Network%20Server,renowned%20companies%20of%20all%20sizes. (visited on 10/25/2024).

[13] Shamkant B. Navathe Ramez Elmasri. *Fundamentals of Database Systems*. Pearson Education, 2015.

[14] *Regional Parameters*. The Things Network. URL: https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/ (visited on 10/25/2024).

[15] Juan De Dios Santos. *Gravity Non-contact Liquid Level Sensor (SEN0204)*. 2024. URL: https://www.dfrobot.com/wiki/index.php/Gravity:_Non-contact_Liquid_Level_Sensor_SKU:_SEN0204.

[16] Juan De Dios Santos and Chip Wired. *DFRobot Gravity Analog Electrical Conductivity Meter (DFR0300)*. 2024. URL: https://wiki.dfrobot.com/Gravity__Analog_Electrical_Conductivity_Meter_V2_SKU__DFR0300.

[17] Semtech. *LoRa® and LoRaWAN®*. 2024. URL: https://www.semtech.com/uploads/technology/LoRa/lora-and-lorawan.pdf (visited on 10/25/2024).

[18] Adafruit Learning System. *Arduino MKR Environmental Shield (ASX00029)*. 2024. URL: https://learn.adafruit.com/arduino-mkr-environmental-shield.

[19] *The ChirpStack project.* ChirpStack. URL: https://www.chirpstack.io/docs/ (visited on 10/25/2024).

[20] Random Nerd Tutorials. *Weather Meter Kit (SEN-15901).* 2024. URL: https://randomnerdtutorials.com/weather-meter-kit/.

[21] Random Nerd Tutorials. *Weather-proof Ultrasonic Sensor (SEN0208).* 2024. URL: https://randomnerdtutorials.com/weather-proof-ultrasonic-sensor/.

[22] *What are LoRa and LoRaWAN?* The Things Network. URL: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/ (visited on 10/25/2024).

[23] Elecrow Wiki. *LR1302 LoRaWAN HAT for RPI$_P$RD.* URL: https://www.elecrow.com/wiki/lr1302-lorawan-hat-for-rpi-prd.html (visited on 10/25/2024).

Appendix A

Backlog

A.1 User Stories for Semester 1

S. No.	Epic / User Stories	Effort	Priority	Sprint	Owner
E1	**Research and Requirements**	**8**	**Very High**		
U1.1	Research and List Required Sensors For Data Gathering	2	5	2	Kirk
U1.2	Research Arduino and Required Libraries	2	5	4	Preston, Kirk
U1.3	Research Networking and Connectivity	4	4	4	William, Andre
U1.4	Research Databases and Required Hardware For Server	5	4	3	Preston
U1.5	Research Cameras and Raspberry PI for Image and Video Collection	3	1		
U1.6	Casing landscape For Final/Future Deployment	5	1	3	Kirk
U1.7	Research Server Requirements	3	4	2	Preston
U1.8	User Manual	3	4	6-7	Dale
U1.8	Research Similar Papers	3	4	6-7	Hector
E2	**Hardware Functionality**	**8**	**High**		

U2.1	Arduino enclosure	3	3	6	Preston
U2.2	Arduino Power	3	3	6	Kirk
U2.3	Raspberry Pi Network Hub Enclosure and Power	2	3		
U2.4	Raspberry Pi Camera Enclosure and Power	2	1	2	William
U2.5	Rain Gauge Functionality	3	4	5	Preston, William
U2.6	Wind Direction Functionality	3	2	5	Preston
U2.7	Wind Speed Functionality	3	2	5	Preston
U2.8	Tank Level Functionality	3	2	5	Hector, Andre, William
U2.9	Water Quality Sensors Functionality (TDS/pH)	3	4	5-6	Kirk
U2.10	MKR Shield	3	4	5	Hector
U2.11	LoRaWAN Gateway	3	4	5	William
U2.12	Raspberry Pi Camera Functionality	5	1	2	William
U2.13	PCB Design	5	1	6	William
E3	**Server Setup**	**3**	**High**		
U3.1	Gather and Assemble Server Hardware	3	3	2	Preston
U3.2	Implement and Deploy Sever onto Network	5	4	2	Preston
U3.3	Design a simple Script for Website	2	2	2	Hector
U3.4	Deploy a Simple Website For Data Visualization	5	2		
E4	**Networking**	**5**	**Mid**		
U4.1	Hardware Data Transmission to Server	5	3		
U4.2	Server to Internet Communication	3	2		
U4.3	Data Transmission on the Same Network to Server	5	4		
U4.4	Data Transmission Over Distance on Different Networks to The Server	8	5		
U4.5	Peer to Peer Communication Between Sensors and Data Transmission Device	5	2	2	William

U4.6	Cybersecurity for Security From the Unwanted Individuals	8	1		
U4.7	Arduino LoRaWAN functionality and finalized code	5	8	7	Preston
U4.8	Raspberry Pi LoRaWAN functionality and finalized code	5	8	7	William
E5	**Website Functionality**	**2**	**Very Low**		
U5.1	Receive Data From Database	4	5	4	Hector
U5.2	Displaying Received Data	2	2	6	Hector
U5.2	Website wireframe	2	2	7	Hector
E6	**Database Design**	**8**	**Mid**		
U6.1	Plan Database Structure	5	5	2	Andre
U6.2	Design and Write Database Schema	5	5	3	Andre, Hector, William
U6.2	Error handling and final structure	5	5	7	Andre, William