

Raspberry Pi Weather Station

Product Design Specification

Version: 1

10/20/2018

Joynal Abedin, Justin Roy, Leng Lee, Alungur Uddin, Naim Cekaj



Version History

Date	Version	Author	Comments
10-18-2018	v1.0	Raspberry Pi Group Fall 2018	First Version

Table of Contents

1 Introduction	4
1.1 Purpose	4
2 General Overview and Design Guidelines/Approach	5
2.1 Assumptions / Constraints / Standards	5
2.1.1 Assumptions	5
2.1.2 General Constraints	5
3 Architecture Design	6
3.1 Hardware Architecture	6
3.2 Software Architecture	7
3.3 Infrastructure Architecture	8
3.4 Security Architecture	9
3.5 Communication Architecture	13
3.6 Raspberry Pi Data Storage Architecture	14
4 System Design	15
4.1 Use Cases	15
4.2 Use Case Diagrams	25
4.3 Sequence Diagrams	27
4.4 Data Flow Diagram	29
4.5 Database Design	31
4.6 Class Diagrams	32
4.7 Application Program Interfaces	34
4.8 User Interface Design	42
5 Product Design Specification Approval	53
A Appendices	55
Appendix A: References	55
Appendix B: Key Terms	56
Appendix C: Use Case/Sequence Diagram Mapping	58

1 INTRODUCTION

1.1 PURPOSE

The purpose of the Product Design Specification document is give a better understanding of the architecture and system design for the software to be developed. This document will include an architecture design section that consists of all diagrams required for hardware, software, security and communications architectures to achieve a comprehensive description of the architecture. Followed by the architecture design section, a system design section will describe all use cases for the additions to previous semesters website with corresponding sequence diagrams, data flow diagrams, database schema, and class diagrams. The final section of the document is the Product Specification Approval section where upon approval the development team for the Raspberry Pi Weather Station will sign and utilize this document for the implementation of the project.

2 GENERAL OVERVIEW AND DESIGN GUIDELINES/APPROACH

2.1 ASSUMPTIONS / CONSTRAINTS / STANDARDS

2.1.1 Assumptions

It is assumed that the master unit will always have a stable internet connection to establish communication with the other weather stations. It is also assumed that when the weather stations are deployed the communication between them will be near instantaneous and successful. Lastly, it is assumed that the Raspberry Pi Weather Stations will be pre configured with all libraries and dependencies needed.

2.1.2 General Constraints

With LoRaWAN technology being used as the main source of communication between the weather stations the use of the LoRa/GPS Long Range Transceiver HAT will be necessary. A potential constraint is due to the terrain and environment, if the HAT was to break, communication to that specific weatherstation would be lost.

With the master weather station having access to an internet source it will have the capability to allow remote access. The constraint is that once the rest of the weather stations are deployed there will be no remote access to them.

Thirdly, testing the network at a high scale is not possible due to the lack of weather stations available. Having only three Raspberry Pi weather stations it will not be possible to stress test our communication system to replicate a real world experience.



Figure 2.1.2: LoRa/GPS Long Range Transceiver HAT

3 ARCHITECTURE DESIGN

3.1 HARDWARE ARCHITECTURE

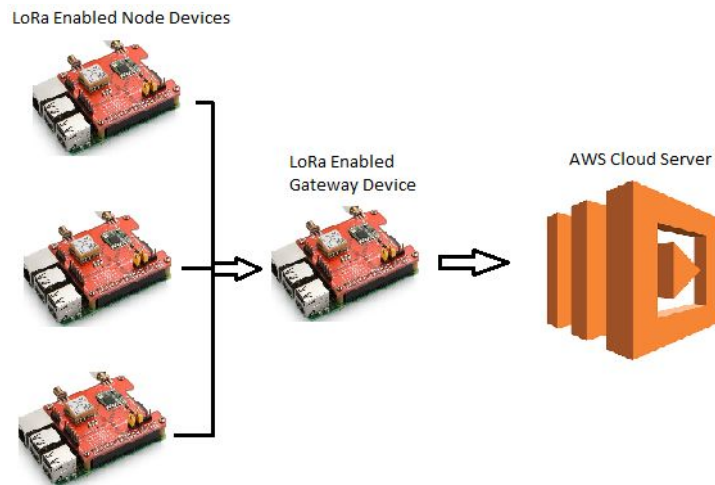


Figure 3.1.1: Hardware architecture

Communication to the AWS cloud server will be done through a LoRa Enabled Gateway device. This particular device will have a constant source of internet connection. Connecting to the LoRa gateway will be multiple LoRa enabled node devices. All LoRa enabled devices in the system will be Raspberry Pi's. The Raspberry Pi that is being used for the devices is version 3 Model B. This specific model includes a quad core 1.2 GHz Broadcom BCM2837 64-bit CPU, 1 GB of Ram, BCM348 on board wireless LAN, and 40-pin extended GPIO. The operating system and data will be both stored on a SanDisk Ultra 32GB microSDHC card. Each device will contain the LoRa/GPS Long Range Transceiver HAT to enable long distance communication between the node devices and gateway. Each Hat contains a built-in temperature sensor, low battery indicator, and a gps base. The LoRa Hat works on a 915 MHZ frequency band and is based on the SX1276/SX1278 transceiver with an additional L80 GPS.

With the LoRa enabled node devices not having a source of power when deployed a portable power supply will be used. Benchmarking of battery life is being done with a large and small battery. The large being an Anker PowerCore 20000mAh power pack portable charger and the small being a Jackery 6000mAh. A smaller power pack is being tested to efficiently test modification made to hardware and software to improve battery life of the LoRa devices.

3.2 SOFTWARE ARCHITECTURE

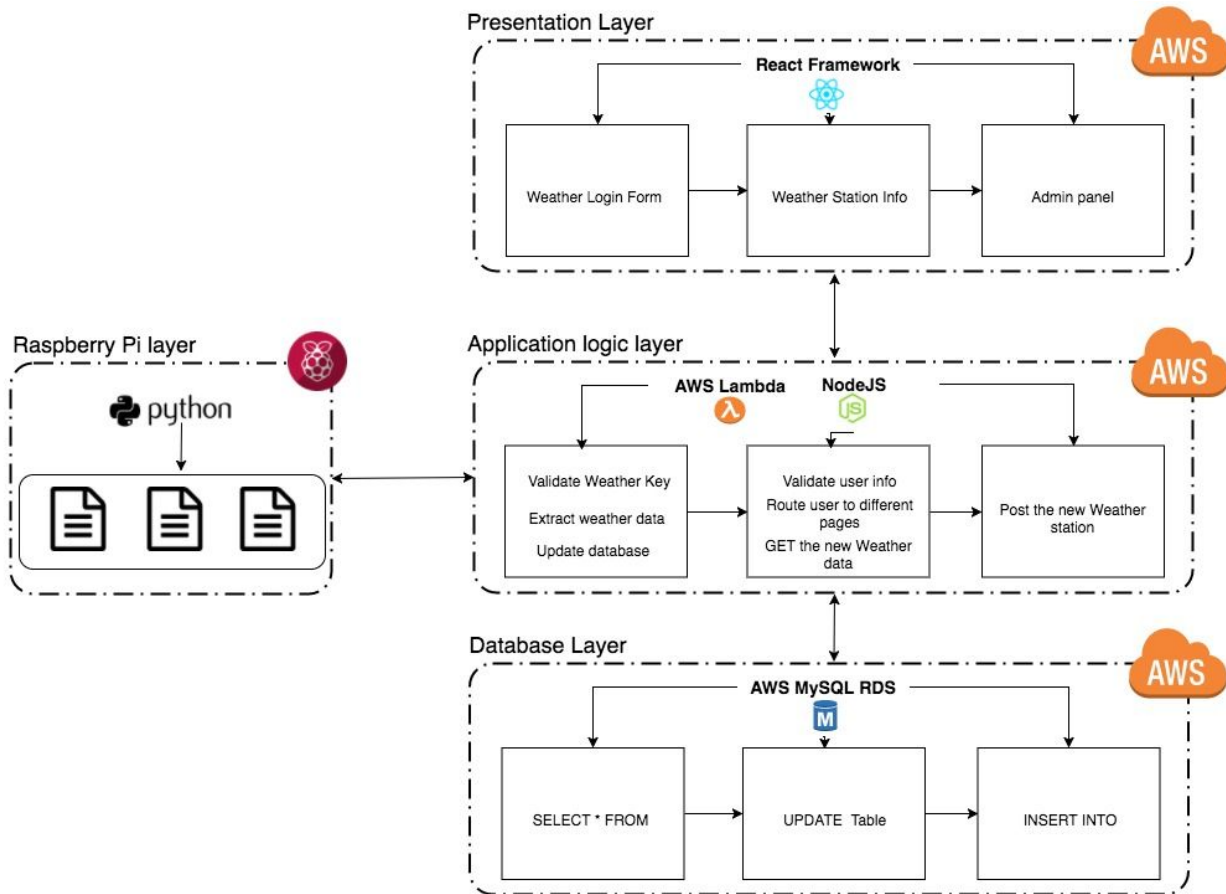


Figure 3.2.1 Software architecture

In the presentation layer, the user interface is designed using React framework. This layer will translate the tasks and results to a friendly form which the user can understand. The website front-end consists of four main components: HTML, CSS, Bootstrap, Javascript. React is a javascript library which is being used to build the user interface. It retrieves information by making API call to the Application logic layer and present the rendered information the user.

The Application logic layer consists of NodeJS and AWS Lambda. NodeJS is being used as a run-time environment that executes Javascript code in the backend. Express.js is a web application framework for NodeJS which is used to route user to different pages as well as implement a MVC architectural pattern in the backend. AWS Lambda resides on the cloud wait for API invocation from the Raspberry Pi layer. When the raspberry pi layer makes API request to AWS Endpoint, it triggers Lambda which verifies weather key as well as extract data and insert into database. MySQL database resides on an AWS RDS instance which performs all the SQL queries.

3.3 INFRASTRUCTURE ARCHITECTURE

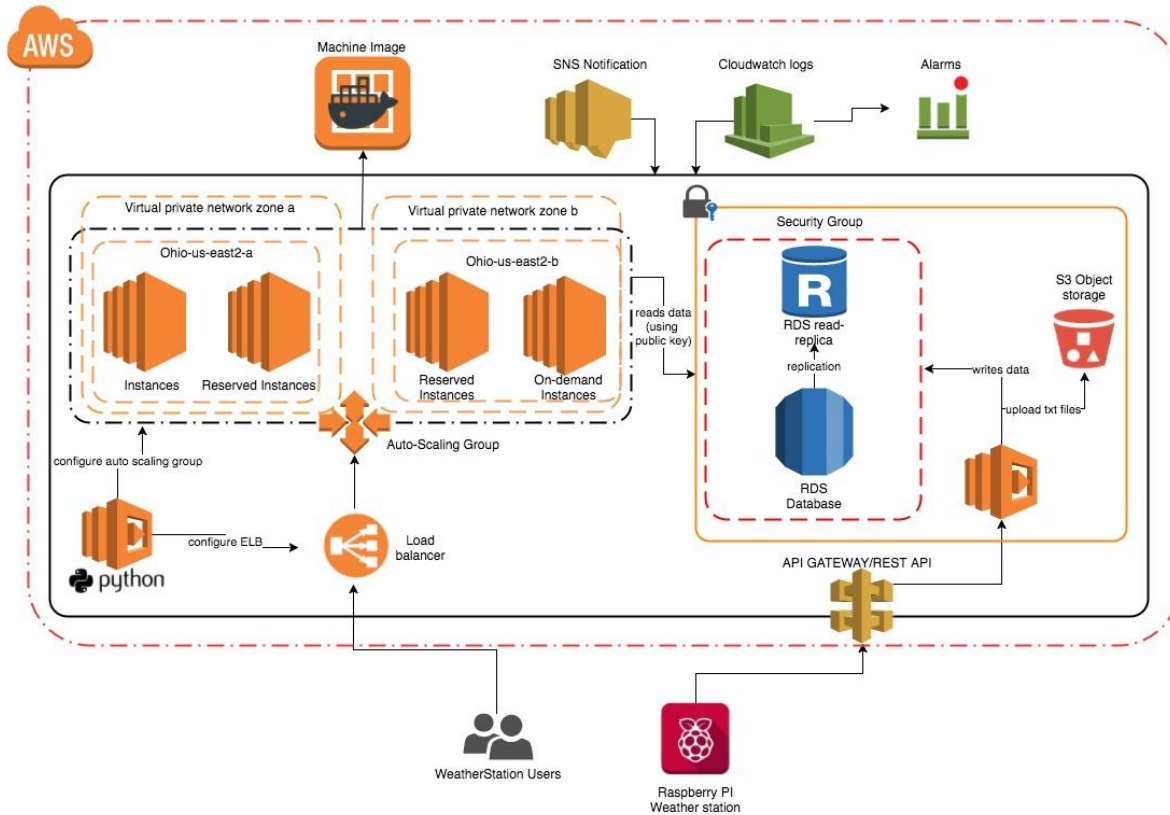


Figure 3.3.1 Cloud architecture

The elastic load balancer will be the single point of access for all incoming traffic to the server instances. The server instances are grouped into an Autoscaling group (ASG) which has a launch configuration. Load balancer looks into the launch configuration which points to the auto scaling groups. In case of a server failure, the health check will fail and ASG will automatically replace it with a new instance. While creating the new instance, it pulls a new image off the machine image storage which is pre-configured using dockerfile, nginx, and pm2 services. When a newly instance is spawned, the elastic load balancer will start distributing traffic to the new instance.

On the other side, raspberry pi will connect to AWS endpoint through an API-gateway which acts like a gatekeeper, however, also wakes up lambda function to do processing on the received information from client-side. When lambda completes processing, it will keep a copy of the file on the s3 bucket and writes the data to the RDS database.

The RDS instances are replicated automatically across multiple region to maintain high availability. All the RDS instances are encrypted at rest. Any communication to RDS will be encrypted as well. For example, server will be using a public key to encrypt data in transit.

3.4 SECURITY ARCHITECTURE

Security is an important feature that must be thought thoroughly. In this case, Rivest-Shamir-Adleman (RSA) will be used to generate a public key and private key for our use. With RSA, each WeatherStation will be able to encrypt any data before sending it to the cloud. Since it is an public-private key system, anything encrypted with the public key can only be decrypted with the private. The private key will reside on Master and this is where the data will be decrypted and sent to the server.

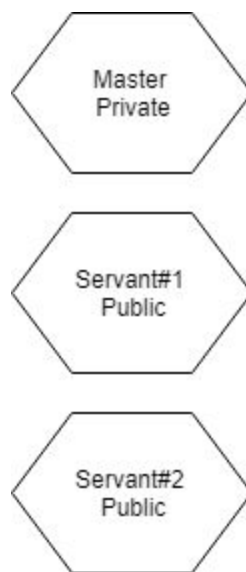


Figure 3.3.2 Public - Private Key locations

The private key will reside on the Master, where it would only be accessible by the Master. This method helps hide the private key from each WeatherStations. In the case that if any of the stations were stolen or if communications were being eavesdropped on, the data will stay encrypted because the stations themselves will not know what the private key is and therefore decryption of data would not be possible.

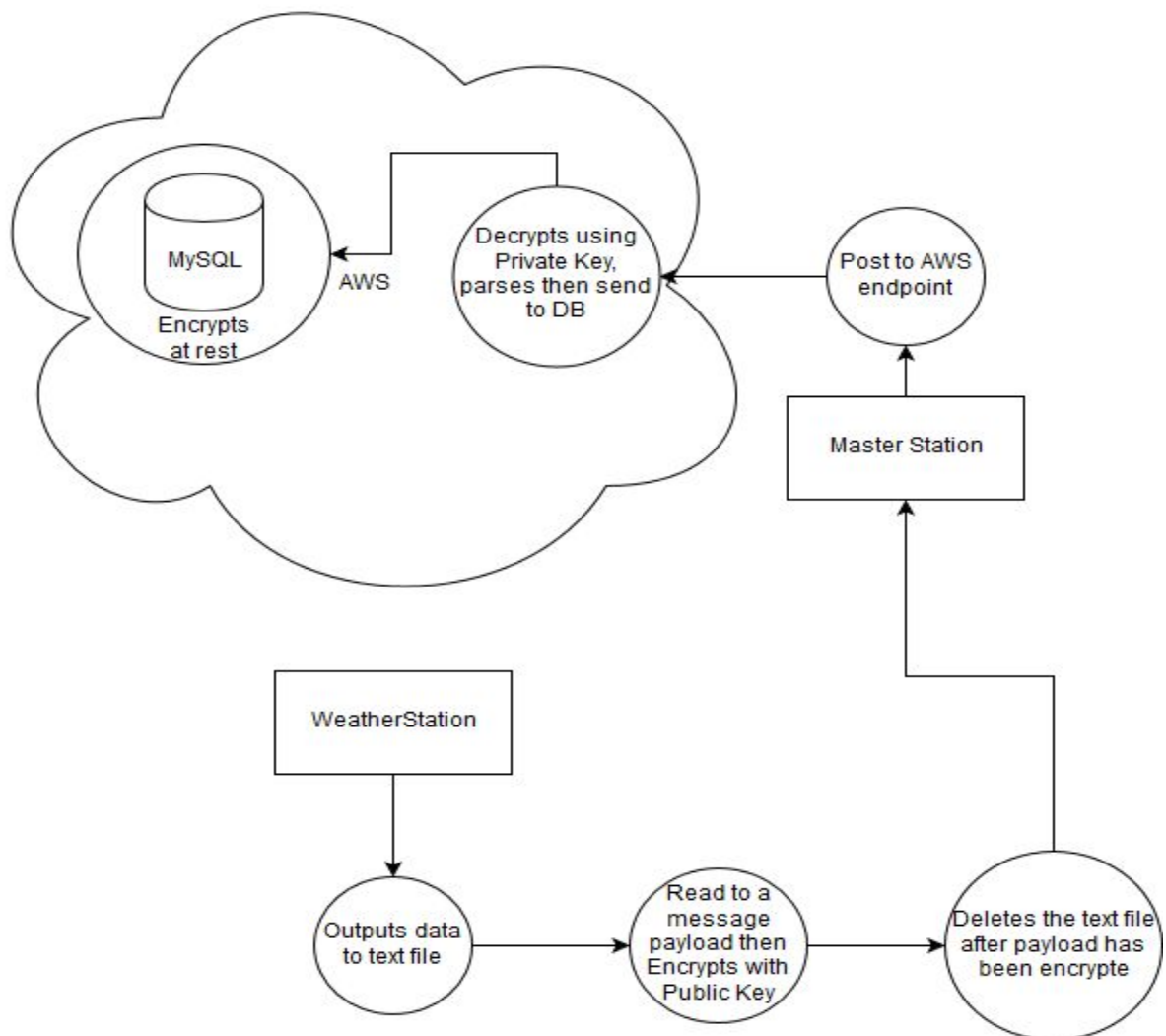


Figure 3.3.3 Security Flow Diagram

When the sensors gather the current weather data it is then written to a text file as a comma separated string. Once the data is written to the text file it is then read, parsed, and sent to the database. To ensure our data cannot be seen a similar yet modified approach will be taken.

The modification that will take place will incorporate encryption to the current method. Once the data is written to the text file the whole text file will be encrypted using AES. After the encryption is complete, the encrypted data will be sent to the master station. As soon as the data is sent, the servant station will completely delete the encrypted data off of its disk and repeat.

The Master Station is the only one who will have network connection, hence it will be the one to send the encrypted file to the server. The server will then decrypt the data

using the private key, and then store in the database where it will be encrypted at rest.

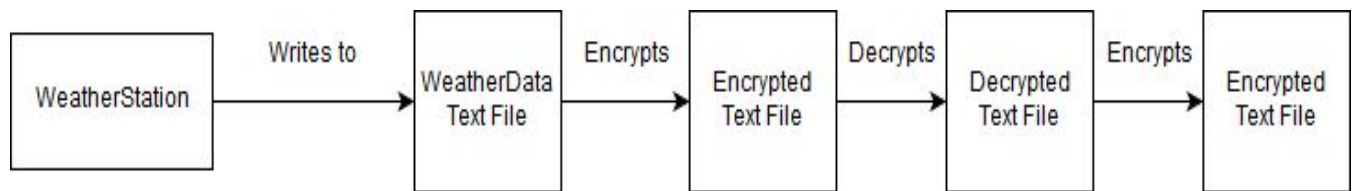


Figure 3.3.5 Data-At-Rest Station

Data-at-Rest means data when its not in transit. Security protocol needs to be taken to ensure its integrity and that unauthorized personals can't view the data. Whenever the servant weather stations lose connection to the master they will have data-at-rest that needs encryption. In this case, weather data is being stored every 4 seconds onto a text file. The text file will be encrypted after successfully writing data to it. Afterwards, when 4 seconds has elapsed and new data needs to be written to the file, the text needs to be decrypted. If it is not decrypted, data will not be able to be written to it because you can't write plaintext to encrypted data. After decryption of the text file happens and the new data has been written, the same method of encryption will be used to encrypt the file again. This process will loop until the station establishes connections. Once connection has been re-established, it will be sent using the communication protocols described above.

3.5 COMMUNICATION ARCHITECTURE

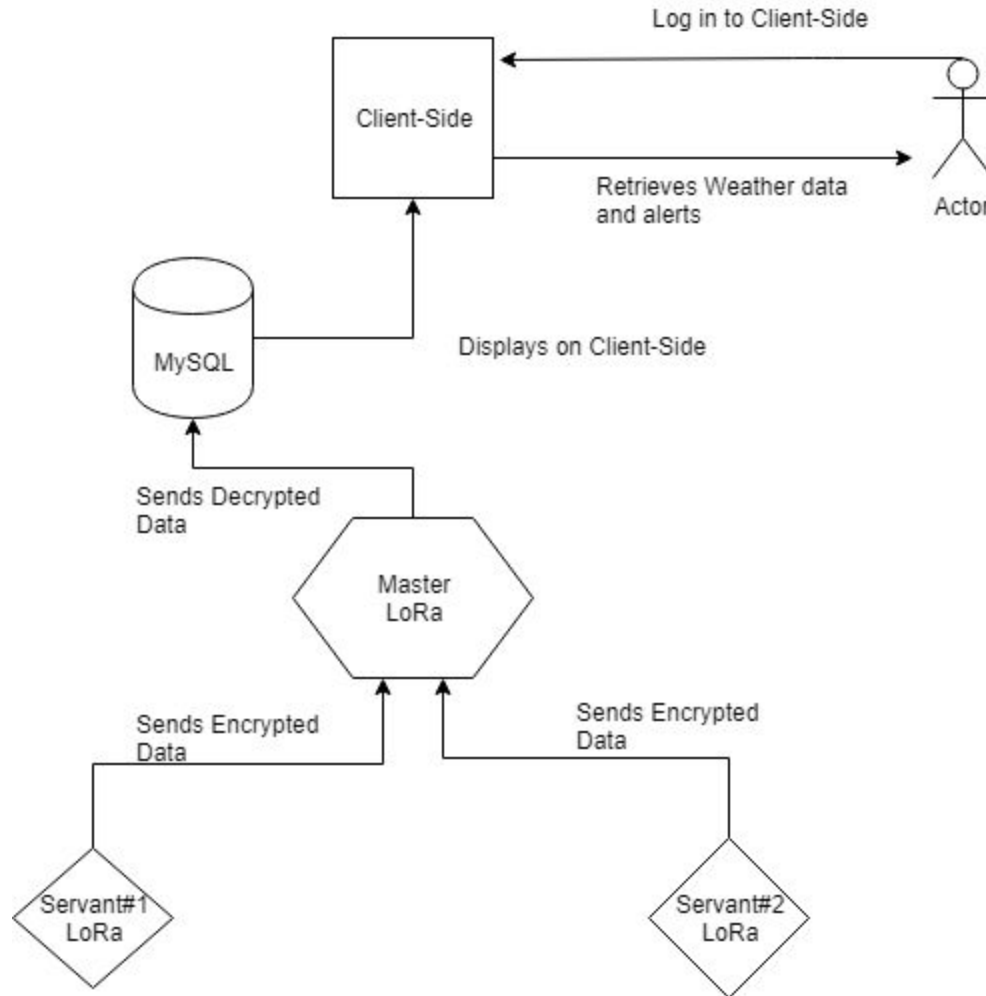


Figure 3.4.1 Communication

The communication of the system will be LoRa and requests. By using a combination of the listed technologies above, it would allow a communication to flow from the stations to the master and to the AWS Server.

LoRa will be the main source for communications between WeatherStations. Each station who are the servant pi will send weather data via LoRa to the Master Station. To connect these devices together on an offline network, LoRa will allow communication between the stations.

Python's request library will be utilized to make requests to the server. Once the master receives the data from servant stations, it will make a request to the AWS Server. It will first validate the station api key with the server to ensure that there is a valid station in

the database. If successful, the master can now make post requests to the server to post onto the AWS endpoint, in which it will be stored in the database.

3.6 RASPBERRY PI DATA STORAGE ARCHITECTURE

Weather and health data storage on the Raspberry Pi will be handled by text files. Each servant station will send data to a text file containing weather and health data related to that specific station. As the station retrieves data, the data will be encrypted and then stored into a text file. Then the text file will be sent to the master which will then send it to AWS and store into the database. In the case where the communication is broken between the master and servant Raspberry Pis, the data will be stored into a text file to be sent later when communication is back up. When the communication between master and servant have been restored and the master station has retrieved all files from the servant stations, the client will sequentially read through and send the text file data to the AWS database storage for the time that was lost due to communication in reverse chronological order. Within each text file, each column of the weather and health data will be comma separated and a new line will be created for each collection time, every 3 seconds. Data formatting for the weather and health collection are as followed:

```
<<created_at, apiKey, temperature, humidity, pressure, latitude, longitude, cpu_usage, battery_life,
ram>>
```

Figure 3.5.1 is an example of how the data will look before being encrypted.

```
2018-09-18 17:35:02.066199, 28cbe87809185a040a5d, 86.58, 50.83, 994.97, 42.362495442, -83.071719216, 25.03, 01:35:02.066199, 1500
2018-09-18 17:36:17.033398, 28cbe87809185a040a5d, 86.3, 50.1, 994.9, 42.362495442, -83.071719216, 26.20, 01:36:17.066199, 1502
2018-09-18 17:37:39.523833, 28cbe87809185a040a5d, 86.52, 50.67, 994.84, 42.362495442, -83.071719216, 24.87, 01:37:39.066199, 1504
2018-09-18 17:38:35.843926, 28cbe87809185a040a5d, 87.52, 58.94, 994.8, 42.362495442, -83.071719216, 24.95, 01:38:35.066199, 1506
```

Figure 3.5.1 Data Storage Architecture

4 SYSTEM DESIGN

4.1 USE CASES

UC-1	Add Custom Alert
Actors:	Operator
Description:	The operator will create alerts based on either CPU usage, estimated battery level, or RAM that will notify them via email, SMS, or the webpage based on changes to the given values.
Trigger:	The operator clicks “Add” on the alert page
Precondition:	The operator is logged in and on the alert page
Postcondition:	A new alert is displayed to the operator on the alerts page
Normal Flow:	<ol style="list-style-type: none"> 1. The operator clicks the “Add” button on the alerts page. 2. The system displays the add alert window which contains a form to submit the new alert details. 3. The operator selects the alert interval, weather station, data type, value key and value within the form. 4. The operator clicks “Create Alert” in the add alert window. 5. The system adds the alert data to the database. 6. The system updates the operator’s alert list.
Alternative Flow:	
Use Frequency	Low - operator will add or update their alert preferences from time to time
Assumptions:	The operator is on the alert page with weather stations available to select from

UC-2	Receive Health Alerts
Actors:	Operator

Description:	The operator will receive health alerts based on the triggers they set in UC-1. The operator will be able to sign up for SMS, email, or webpage alerts. When one trigger from the operator's alerts is activated, the operator will receive an alert in each form they have opted in for.
Trigger:	One or more of the operator's health alert triggers have been activated.
Precondition:	The operator has an activated account, they have set at least one customer alert trigger, and they have signed up for at least one method of receiving alerts (SMS, email, or webpage)
Postcondition:	The operator has received an alert for each method they have opted in for. The alert repeats the alert trigger back to the operator and gives the current health for that station.
Normal Flow:	<ol style="list-style-type: none"> 1. The system tests each operator's alert triggers based on historical data stored from the weather stations. 2. The system matches a operator's alert trigger with historical weather data. 3. The system looks at which alert types the operator has activated, and sends the alert(s) through each method. 4. The operator receives alerts via each method they opted in for. Each alert reads the trigger that the operator initially set and provides the historical data that triggered the alert in a list format.
Alternative Flow:	<ol style="list-style-type: none"> 3a. The system is unable to find any alert types that the operator has activated. 4a. The operator does not receive any alert.
Use Frequency	Moderate - health alerts will be sent out depending on the health of the weather station, but they will be sent out to many operators and in multiple different formats.
Assumptions:	There is an active internet connection, the operator already has an activated account, and they have signed up for at least one alert trigger and one alert type.

UC-3	View Health Monitor
Actors:	Operator

Description:	The operator will view a list of all currently or previously connected stations. Each connected station will be a card that displays CPU usage, estimated battery level, RAM, and a connection quality indicator. If the station is disconnected, the quality indicator will be red and the station will be moved to the bottom of the list. If it is a green plug icon the station is connected and sending data every five seconds.
Trigger:	The operator navigates to the stations page.
Precondition:	The operator is logged into their account and there is at least one connected station.
Postcondition:	The operator will see a list of all currently or previously connected stations along with each stations' most recent health data.
Normal Flow:	<ol style="list-style-type: none"> 1. The operator arrives at the stations page 2. The system retrieves a list of all stations. 3. The system retrieves CPU usage, estimated battery life, and RAM for each station. 4. The system displays a list of all stations. Each weather station is displayed as a card which shows CPU usage, estimated battery level, RAM, and a connection quality indicator.
Alternative Flow:	<ol style="list-style-type: none"> 2a. The system retrieves no stations. 3a. The system displays an alert that reads "There are no currently connected stations"
Use Frequency	High - This page will be one of main point of contact for logged in operators and will be used most frequently to display station health data.
Assumptions:	There is an active internet connection, the operator has already successfully created an account, and their account has been approved by an admin.

UC-4	Filter Health Monitor
Actors:	Operator

Description:	The operator will use the input box above the list of health station cards to filter the list. As the operator types in the box, the list will automatically filter to match the value being typed.
Trigger:	The operator starts typing into the filter input box on the health page.
Precondition:	The operator is on the health page and there is no value typed into the input box.
Postcondition:	The list of health station cards is filtered to only display the stations that match the value of the filter input box
Normal Flow:	<ol style="list-style-type: none"> 1. The operator begins typing into the filter input box above the station cards. 2. The system displays the station cards with names that match the value of the filter input.
Alternative Flow:	2a. The system displays an alert that says "No stations match the filter."
Use Frequency	Moderate - This use case will be used occasionally when the operator wants to view a particular station or needs to filter a larger number of stations.
Assumptions:	There is an active internet connection and the operator is on the health page.

UC-5	View Health Station Details
Actors:	Operator, Admin, Superuser
Description:	The operator will click on an individual health station card and see additional station data such as estimated total battery life and estimated remaining battery life in a popup window. The operator will also see the station's current CPU usage, estimated battery level, and RAM in this same window.
Trigger:	The operator clicks on a health station card on the health page.
Precondition:	The operator is logged in and is viewing the health page.
Postcondition:	The operator will see a popup window with additional health

	station data such as estimated total battery life and estimated remaining battery life in a popup window. The operator will also see the station's current CPU usage, estimated battery level, and RAM in this same window.
Normal Flow:	<ol style="list-style-type: none"> 1. The operator clicks on an individual weather station card. 2. The system retrieves the data for the clicked weather station. 3. The system displays a popup window with the clicked weather station's name, CPU usage, estimated battery level, RAM, estimated battery life and estimated remaining battery life.
Alternative Flow:	3a. If the operator has admin privileges, the system displays a popup window with the clicked weather station's CPU usage, estimated battery level, RAM, estimated battery life, estimated remaining battery life, and an input box with the station name as an editable value.
Use Frequency	Moderate - This use case will be used occasionally when the operator wants to view an individual station's battery information such as estimated total battery life and estimated battery life remaining, or change the station's name.
Assumptions:	There is an active internet connection and the operator is on the health page.

UC-6	Edit Station Name
Actors:	Admin
Description:	The admin will click a weather station health card to bring up the station detail window. This window will display an input box with the current station name as value. The admin will be able to edit this value and click "Save Changes" at the bottom of the window for the changes to take effect.
Trigger:	The admin clicks on an individual weather station health card.
Precondition:	The admin clicks on a weather station card on the health page and is currently an admin.

Postcondition:	The weather station's name is changed.
Normal Flow:	<ol style="list-style-type: none"> 1. The admin clicks on a weather station card 2. If the operator has admin privileges, the system brings up the station detail window with an input box for the station name value. 3. The admin changes the value of the station name. 4. The admin clicks "Save Changes" 5. The system updates the value of the station name in the database. 6. The system updates the health page to display the new station name.
Alternative Flow:	<ol style="list-style-type: none"> 2a. If the operator does not have admin privileges, the system will bring up the station detail window with only text for the station name value. 3a. The operator is unable to change the value of the station name. 5a. The system displays an error saying "This station name already exists"
Use Frequency	Low - This use case will be used rarely and will only be allowed to occur for admins.
Assumptions:	There is an active internet connection and the operator already has admin permissions.

UC-7	View Historical Data
Actors:	Operator
Description:	The operator will view a graph of historical data from both the weather stations as well as Open Weather API. The default graph displays temperature for each connected station over the last 24 hours.
Trigger:	The operator navigates to the historical page.
Precondition:	The operator is logged into their account and there is weather/health data from at least one connected weather station.
Postcondition:	The operator will see a line graph containing the historical temperature data from all currently connected stations.
Normal Flow:	<ol style="list-style-type: none"> 1. The operator clicks on the "historical" link in the

	<p>navigation bar.</p> <ol style="list-style-type: none"> The system redirects the operator to the historical page. The system displays a line graph of the historical temperature data saved for the previous 24 hours from all connected weather stations.
Alternative Flow:	3a. There is no data stored by connected stations so the system displays an alert saying "No Historical Data."
Use Frequency	High - Besides the stations and health page, this will be a common page for operators to get information about previous weather and health for each station.
Assumptions:	There is an active internet connection, the operator has already successfully created an account, and their account has been approved by an admin.

UC-8	Filter Historical Data
Actors:	Operator
Description:	Once on the historical page, the operator will be able to click the filter button which will bring up the filter window. From this window, the operator will be able to select the type of data, the date range, and which stations they would like displayed on the line graph.
Trigger:	The operator clicks the filter button on the historical page.
Precondition:	The operator is on the historical page and has clicked the filter button.
Postcondition:	The operator will see a line graph of historical data based on their selected filters.
Normal Flow:	<ol style="list-style-type: none"> The operator clicks on the filter button. The system brings up the filter window. The operator selects one or more of the following filters: type of data, the date range, and station names. The operator clicks the save changes button. The system displays a line graph based on the selected filters.
Alternative Flow:	4a. The operator decides they no longer wish to filter the

	graph so they click cancel instead of save changes. 5a. There is no data stored by connected stations so the system displays an alert message saying “No Historical Data.”.
Use Frequency	High -In order to get more historical data, the operators will be filtering the graph fairly often.
Assumptions:	There is an active internet connection, the operator has already successfully created an account, and their account has been approved by an admin.

UC-9	Send Weather Station Data (servant)
Actors:	Weather Station
Description:	The weather station sends data to a text file which is encrypted and then sent to the master weather station every five seconds.
Trigger:	The station is communicated with the master weather station and is running the client code.
Precondition:	The station has been added to the website, either has the Sense Hat or Individual sensors connected to it, LoRaWAN hat for communication and has the client installed on it.
Postcondition:	The text file containing data is sent to the master weather station.
Normal Flow:	<ol style="list-style-type: none"> 1. The weather station is turned on. 2. The weather station reads the sensor or sensors. 3. The data is sent to the master weather station.
Alternative Flow:	<ol style="list-style-type: none"> 3a. There is no communication to master weather station. 4a. Communication is restored to the master weather station 5a. Send text file containing collected data to master weather station.
Use Frequency	High
Assumptions:	The client has been correctly installed on the weather station, weather station has communication with master weather station that has an internet connection.

UC-10	Send Weather Station Data (Master)
Actors:	Weather Station
Description:	The weather station sends weather and health data to the website asynchronously through an API endpoint every five seconds from all servant stations including itself
Trigger:	The station is connected to the website and is running the client code.
Precondition:	The station has been added to the website, either has the Sense Hat or Individual sensors connected to it, LoRaWAN hat for communication and has the client installed on it.
Postcondition:	The website is updated every five seconds with weather data from the station and servant stations.
Normal Flow:	<ol style="list-style-type: none">1. The weather station is turned on.2. The weather station reads the sensor or sensors.3. The weather station gets data from servant stations4. The data is sent to the website via API endpoint.5. The new data is stored in the sites database.6. The website updates the stations and health page with the new data.
Alternative Flow:	<ol style="list-style-type: none">3a. There is no internet connection so the data is saved to a text file.4a. The internet connection is restored.5a. The stored weather data is sent to the site via API endpoint.
Use Frequency	High
Assumptions:	The client has been correctly installed on the weather station and has an internet connection

UC-11	Historic Alerts
Actors:	Operator

Description:	The operator has the option of filtering historic alerts by alert or by date. When one of the historic alerts is clicked it will display the weather and health data at the time the alert was triggered in a modal.
Trigger:	The operator is on the alerts page.
Precondition:	An alert has been triggered.
Postcondition:	The alerts rendered on the page are changed depending on how the operator has set the filters.
Normal Flow:	<ol style="list-style-type: none"> 1. The operator selects a day from the calendar filter 2. The operator clicks on an alert and views the weather and health data from that alert.
Alternative Flow:	1a. The operator selects an alert from the alert filter
Use Frequency	Low
Assumptions:	The operator is logged in. There is at least one alert that has been triggered.

UC-12	Deployment of Device
Actors:	Operator
Description:	Deployment of the device will include turning on the device, and placing/dropping the device within the range necessary to be within distance of the master device.
Trigger:	The operator is tasked to deploy a set number of Weather Station devices in a set area.
Precondition:	<ol style="list-style-type: none"> 1. The operator is in possession of at least 1 master device and 1 servant device. 2. All devices in possession have been loaded with proper libraries, software and boot-up sequence. 3. Devices has a source of power. 4. Server may or may not be ready to receive data.
Postcondition:	The devices power on, fulfilling the role they have been designed to do (master versus servant) and start sending data to master Pi or server depending on role.
Normal Flow:	<ol style="list-style-type: none"> 1. Operator powers on each device with a battery or

	<p>alternative power source.</p> <ol style="list-style-type: none">2. A red light should be glowing on both the LoRa HAT and the Raspberry Pi device. On the Sense HAT only a single LED exists.3. Optional: Operator can verify data being transmitted to server using Health Status UI or database historical data screen.4. Operator deploys Pi's in designated location within range of each other.5. Operator verifies data is being sent to master and from master to server to verify range and communication by visiting Health Monitoring UI or checking historical database data for recent input.
Alternative Flow:	
Use Frequency	Low to Medium
Assumptions:	<ol style="list-style-type: none">1. The operator is in possession of 1 or more servant devices and 1 or more master devices.2. The operator will have an internet connection for the master device3. The operator will have a power source for all Pi devices.

4.2 USE CASE DIAGRAMS

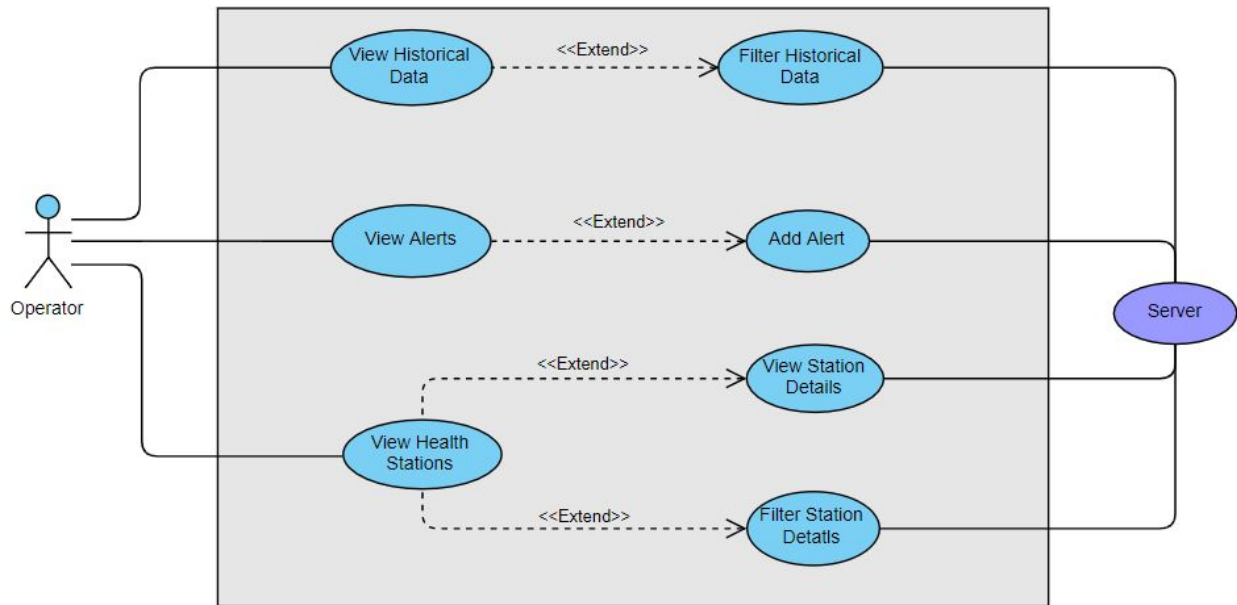


Figure 4.2.1: Operator Website Use Case Diagram

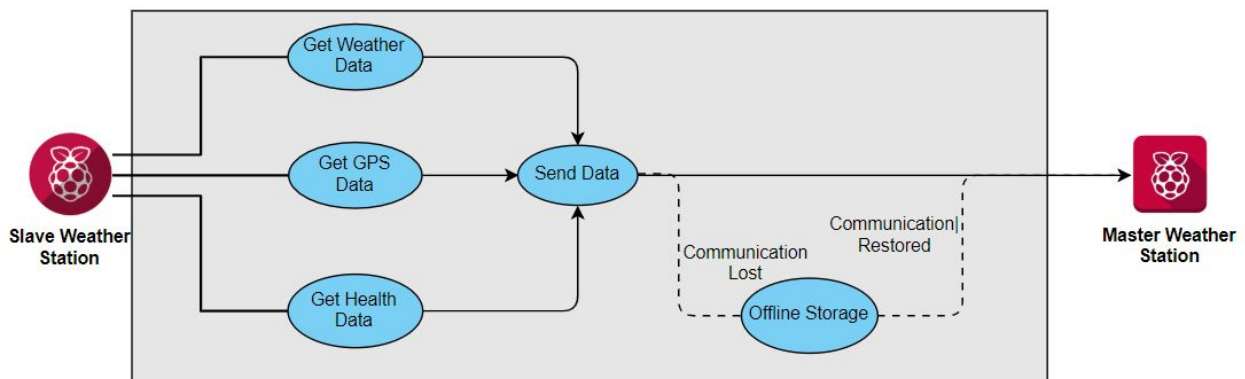


Figure 4.2.2: Servant Weather Station Use Case Diagram

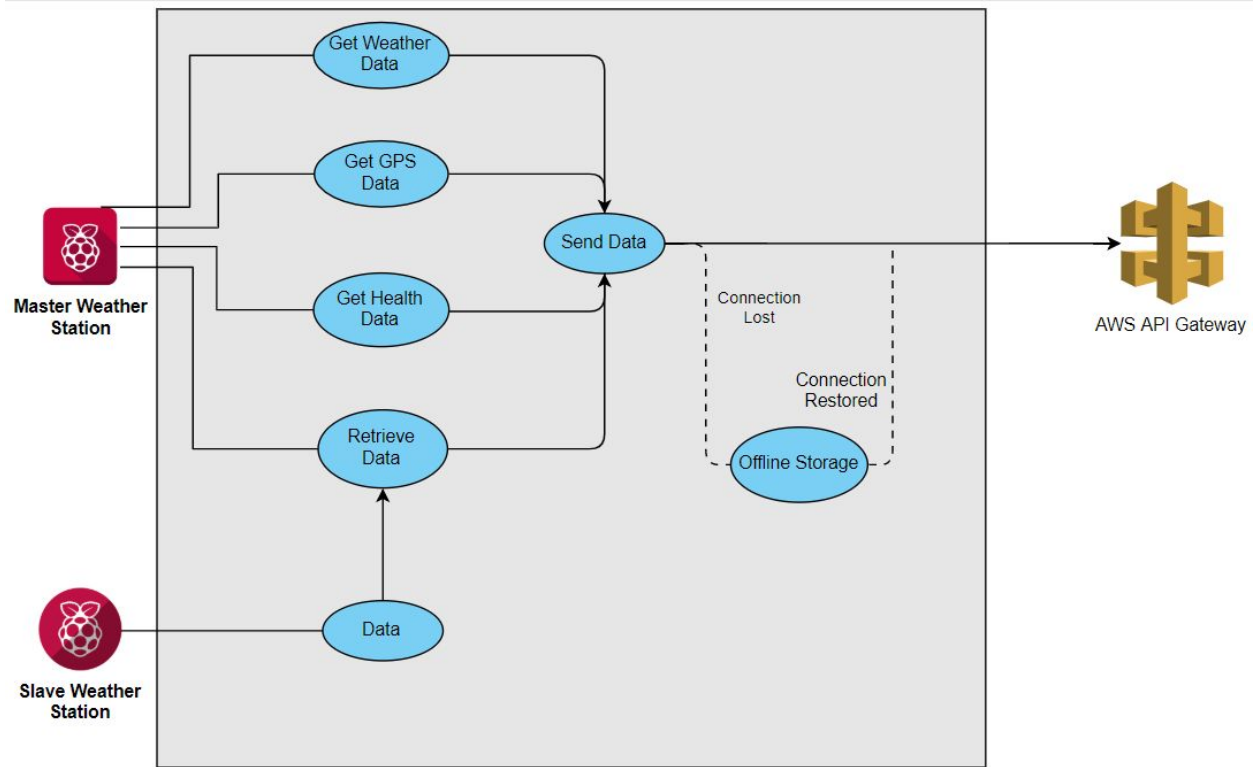


Figure 4.2.3: Master Weather Station Use Case Diagram

4.3 SEQUENCE DIAGRAMS

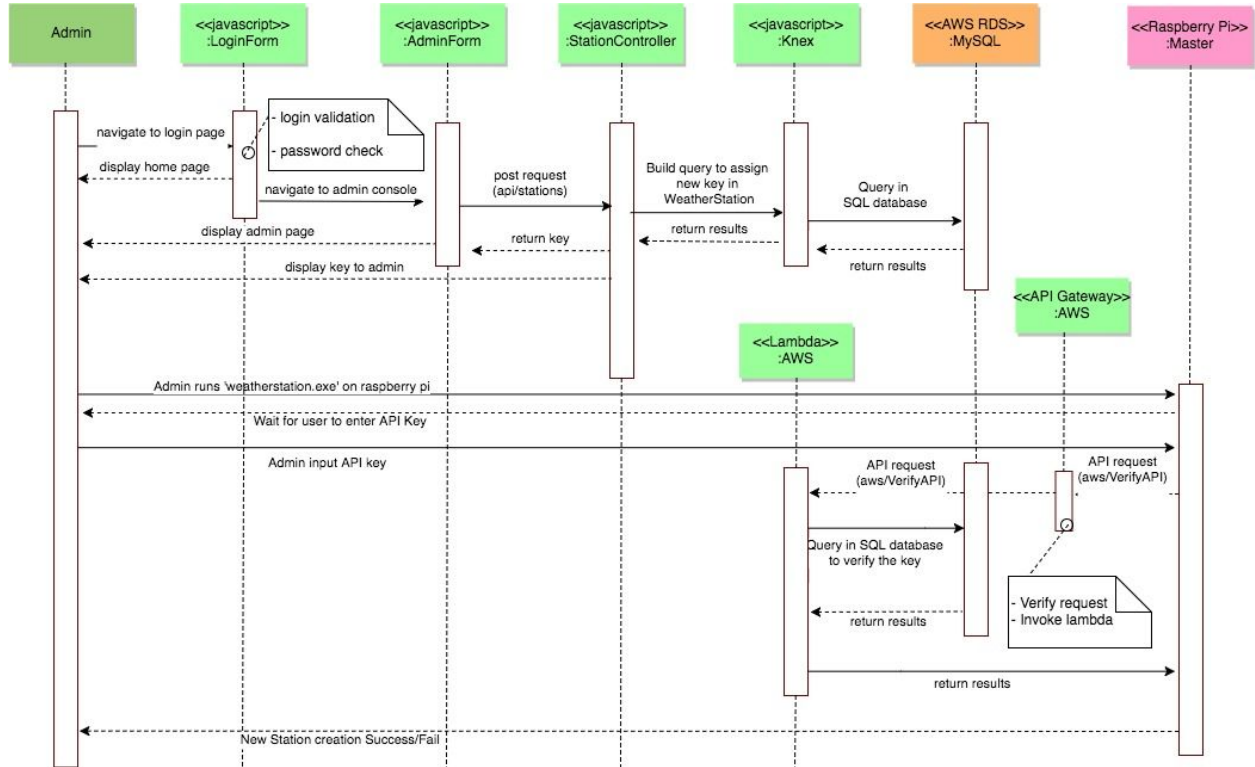


Figure 4.3.1: Admin adds new station

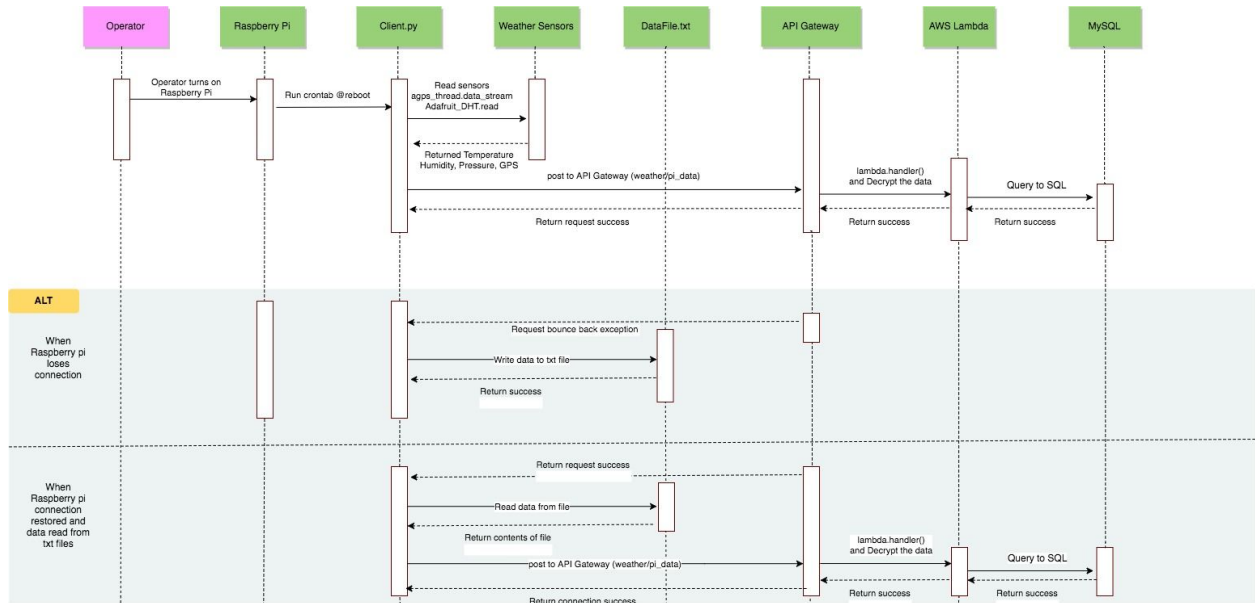


Figure 4.3.2: Raspberry pi sending data

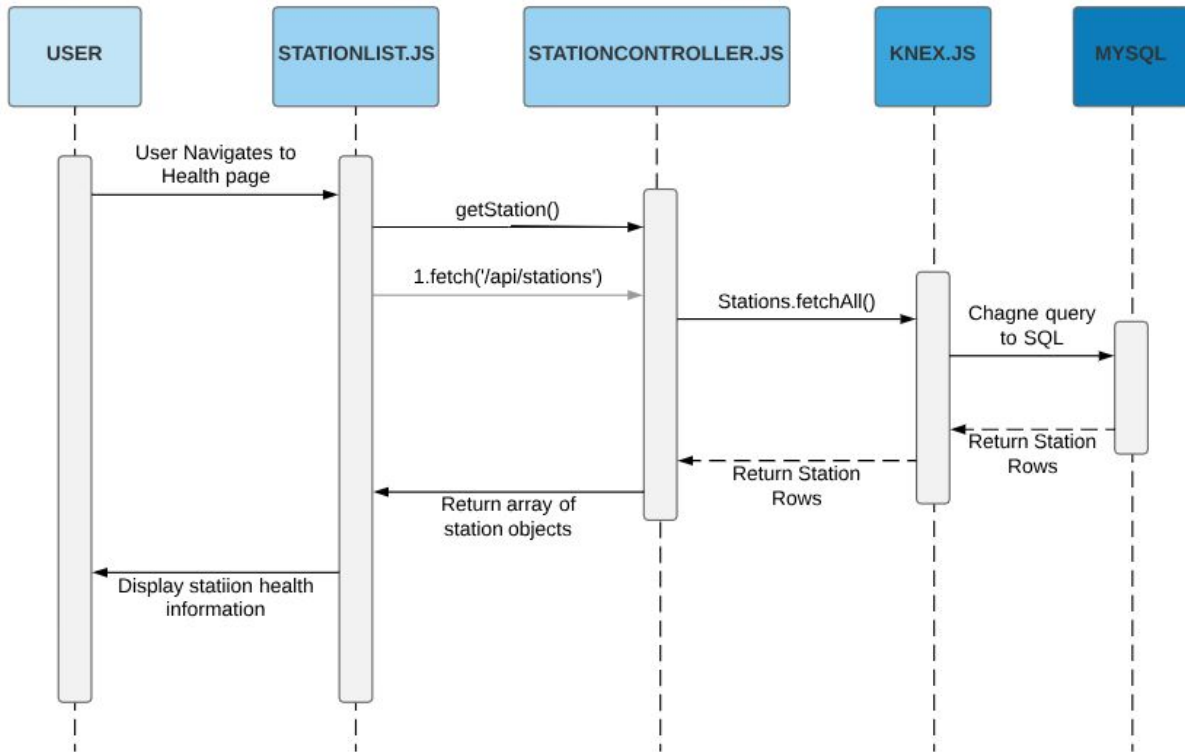
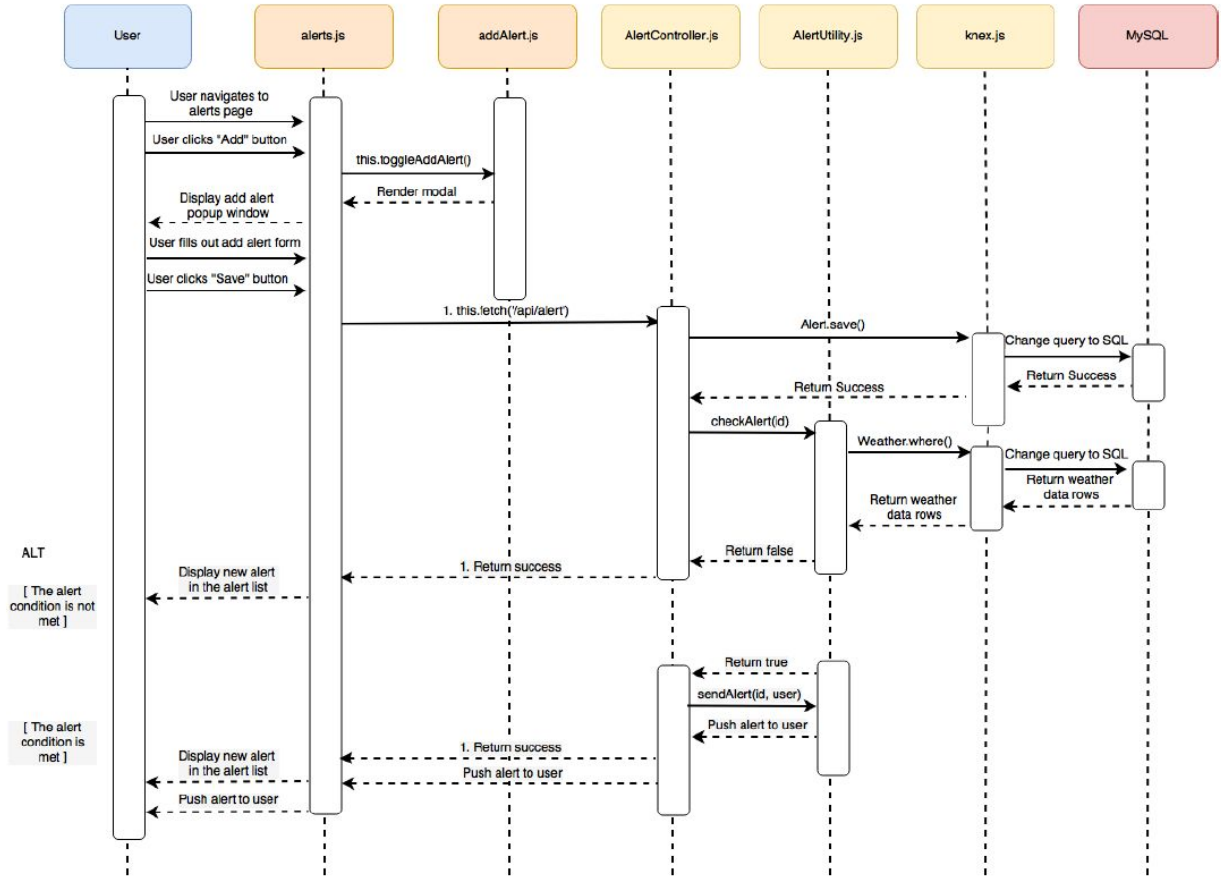
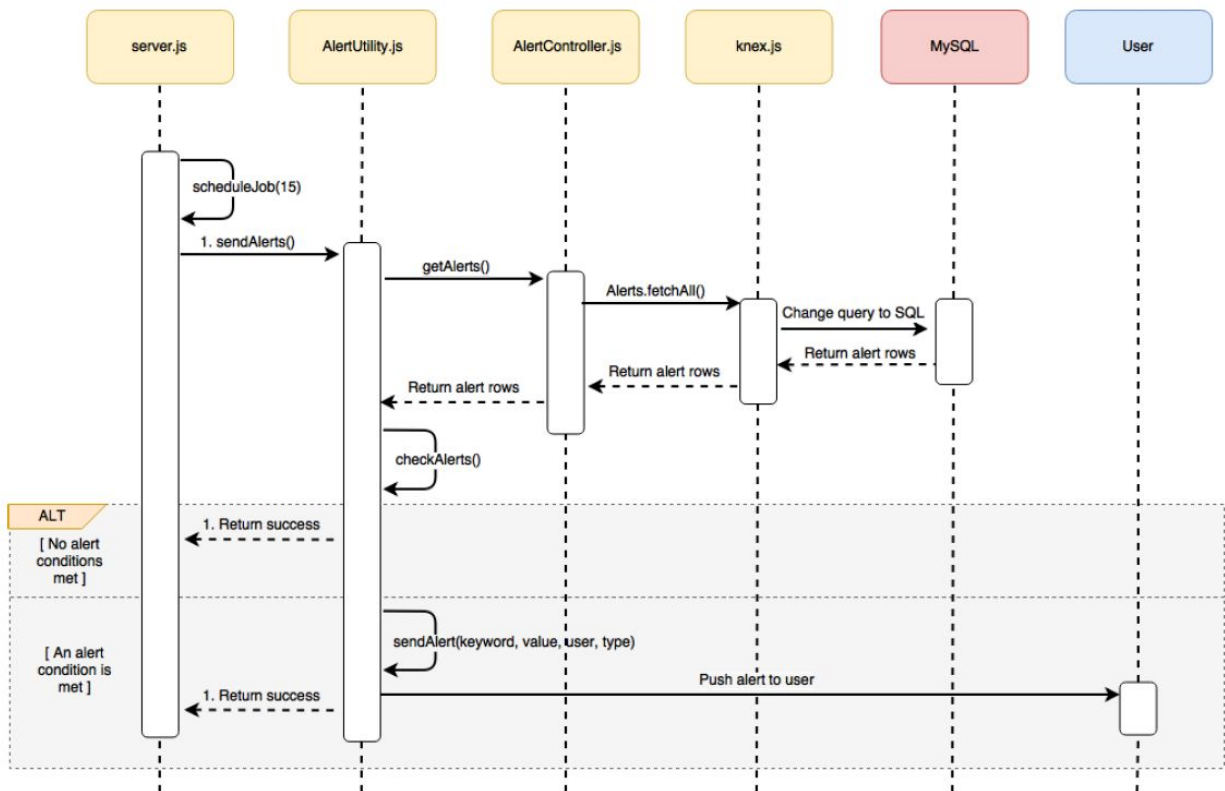


Figure 4.3.3: View Stations Health



(Jackson et al., 2018) Figure 4.3.4: Add Alert Trigger



(Jackson et al.,2018) Figure 4.3.5: Receive Alert Trigger

4.4 DATA FLOW DIAGRAM

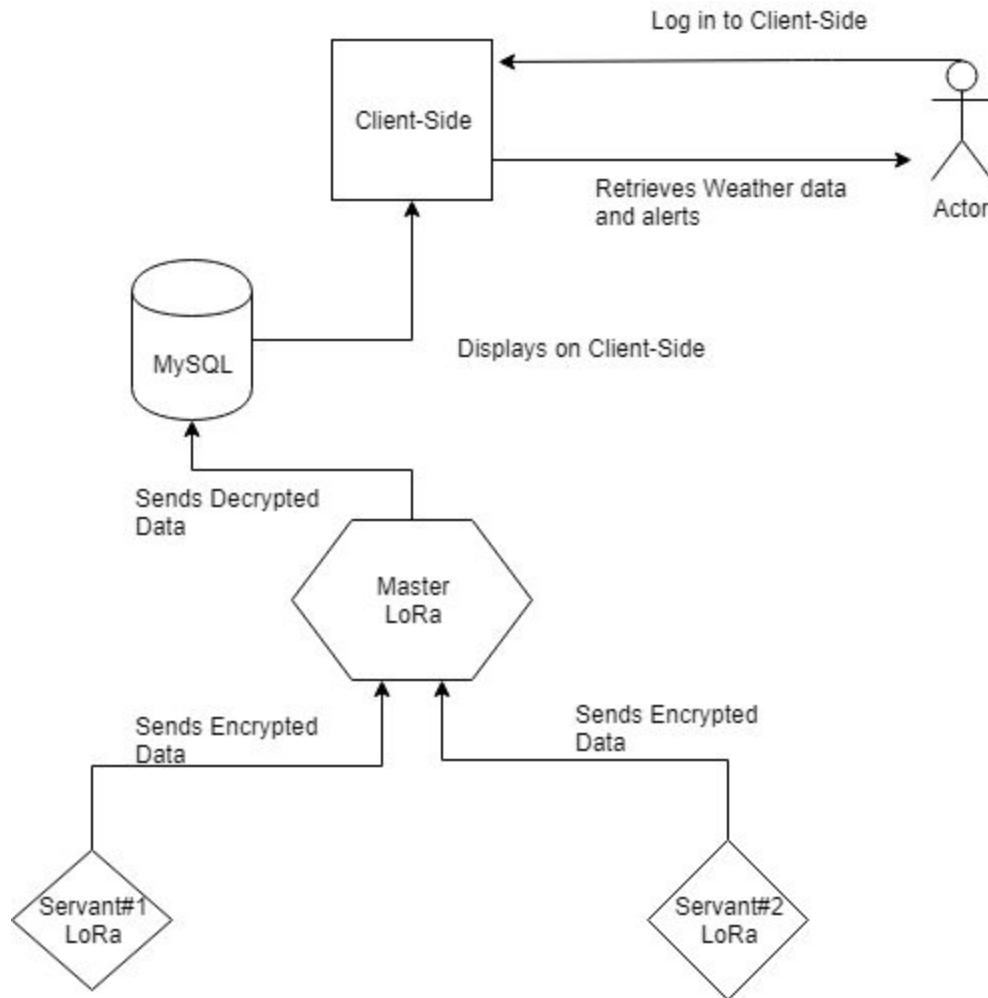
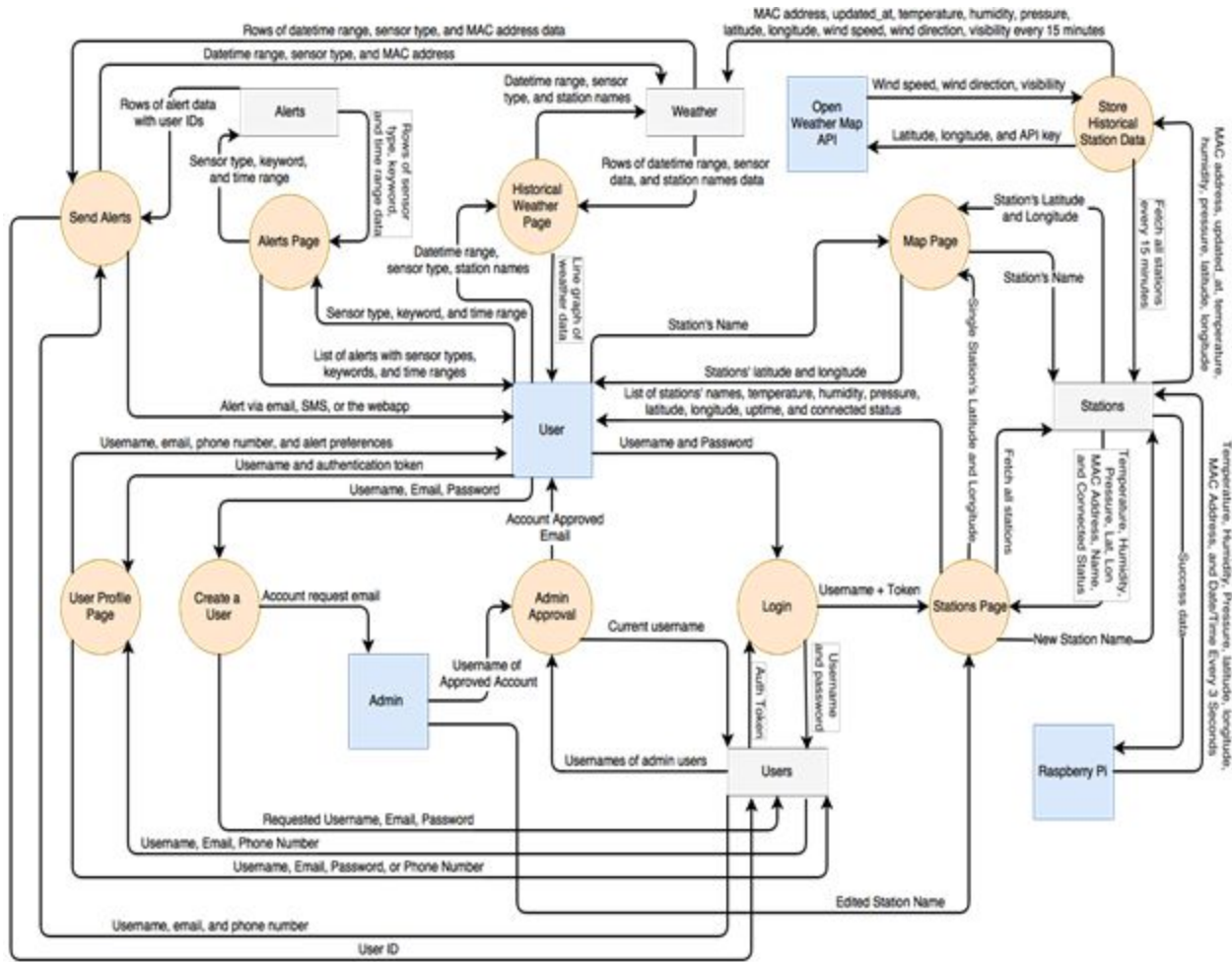


Figure 4.4.1: Weather System's Data Flow Diagram

Figure 4.4.1 shows how the data flows through the weather station system for the user to view. The weather data is first gathered from the sensors on the weather station Raspberry Pi's. The data is then encrypted and sent to the master. From there the Master Pi will check the broker for new data and retrieve any if there is. It will then keep the data encrypted and send it to the AWS server. It will finally be sent to the database where the client-side can access the information to display to the user.



(Jackson et al., 2018) Figure 4.4.2: Previous semester's Client-Side Data flow

This is the client-side web application data flow and it was completed by the previous semesters group. Since the goal of the project is to be an extension of the weather station system it was decided to have this current way of data flow on the client-side remain the same. Figure 4.4-2 shows the additional data flow that happens before it reaches the client-side web application.

4.5 DATABASE DESIGN

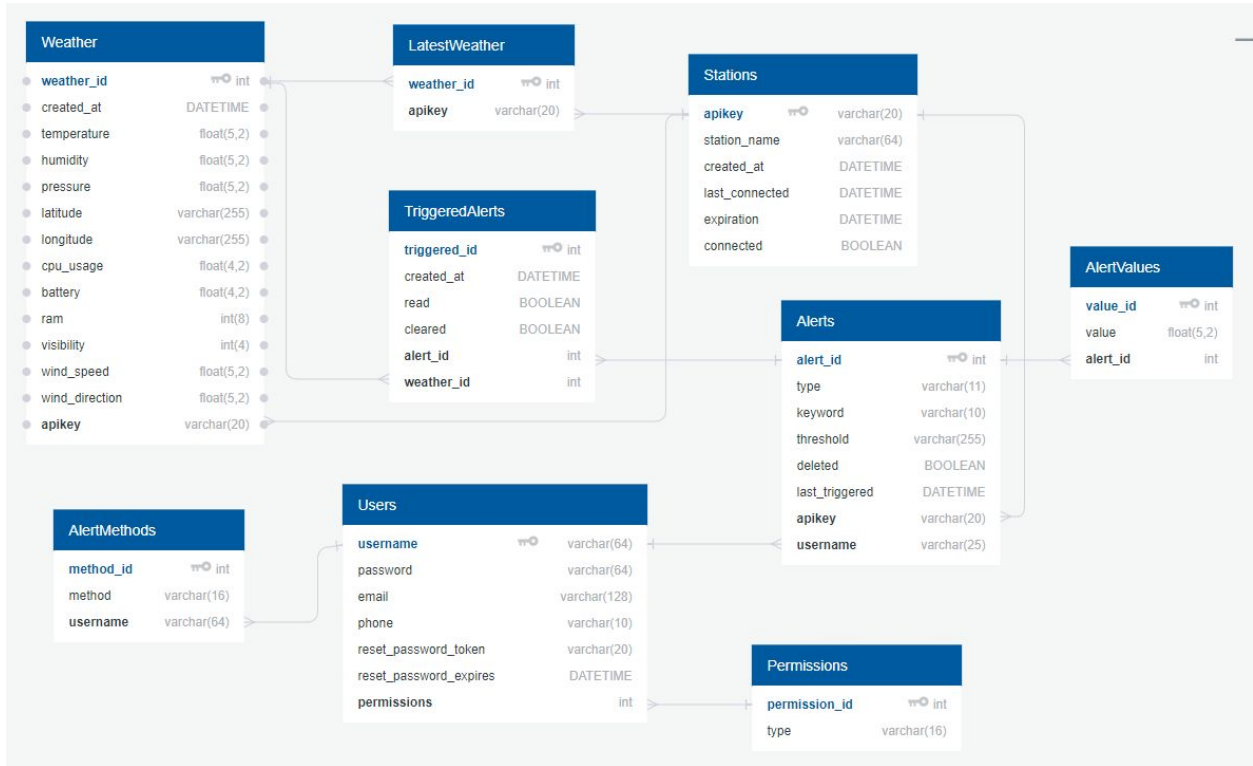


Figure 4.5.1: Database Diagram

4.6 CLASS DIAGRAMS

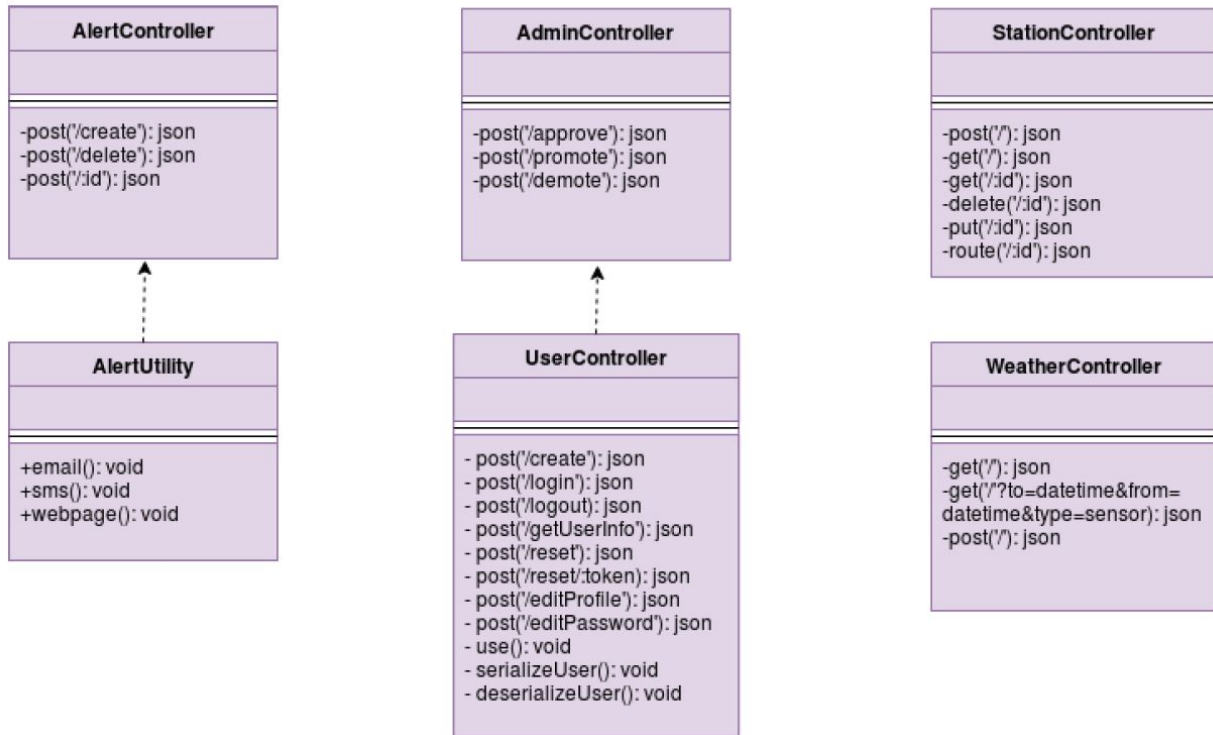


Figure 4.6.1: Controller Class Diagram

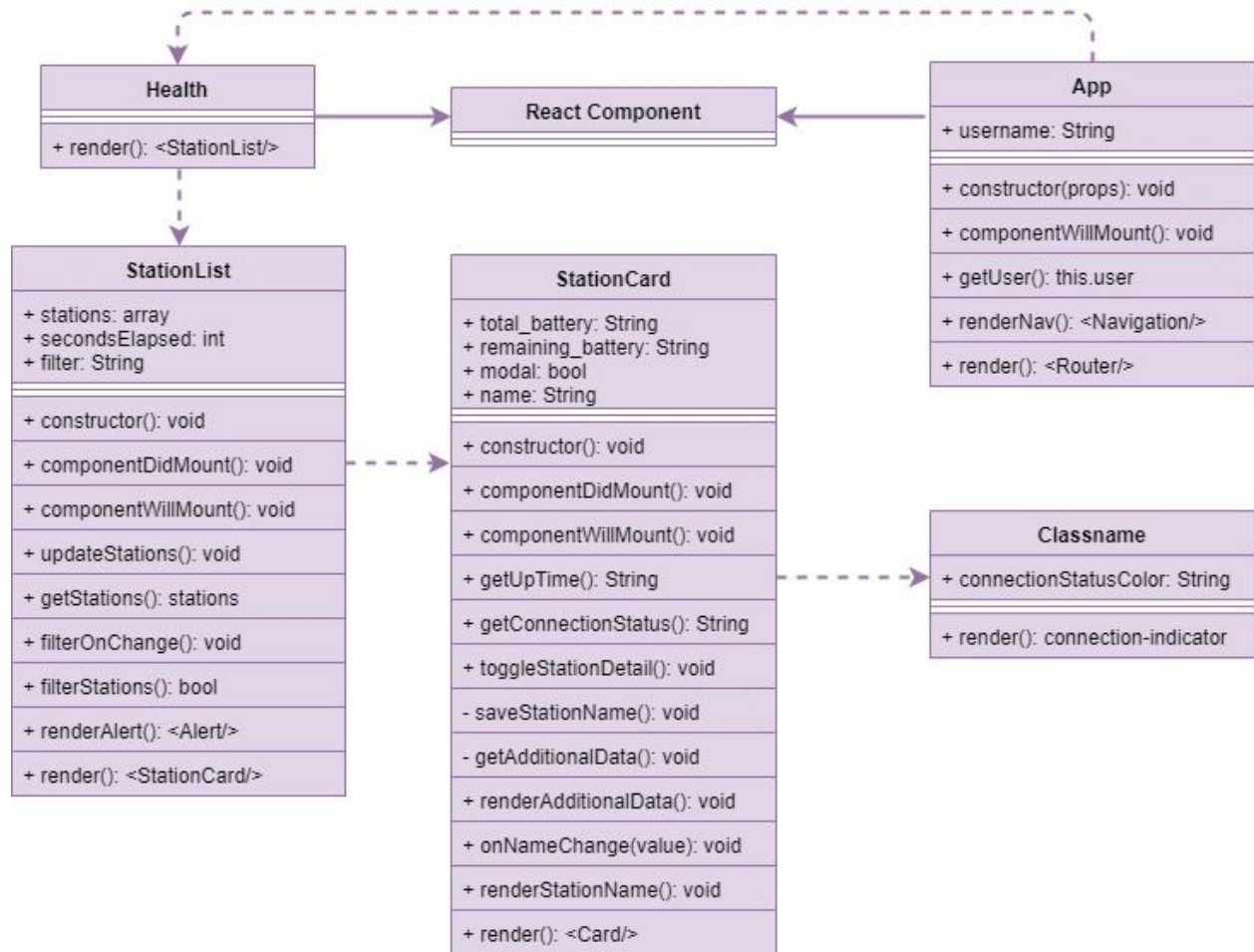


Figure 4.6.3: Classes Added to React Component Class

4.7 APPLICATION PROGRAM INTERFACES

Title:	Create User
URL:	/api/user/create
Method:	POST
Parameters:	username[string], email[string], password[string], confirmPas[string],
Returns:	response [string]

Title:	Login
URL:	/api/user/login
Method:	POST
Parameters:	username[string], password[string]
Returns:	response [string]

Title:	Logout
URL:	/api/user/logout
Method:	POST
Parameters:	username[string]
Returns:	response[string]

Title:	Get User Information
URL:	/api/alert/getuserInfo
Method:	POST
Parameters:	
Returns:	username[string], email[string], phone[string], permission[string]

Title:	Request Forgotten Password Reset
URL:	/api/user/reset

Method:	POST
Parameters:	email[string]
Returns:	response[string]

Title:	Edit Password
URL:	/api/user/editPassword
Method:	POST
Parameters:	currPass[string], newPass[string], confirmPass[string],
Returns:	response[string]

Title:	Add Alert
URL:	/api/alert/create
Method:	POST
Parameters:	keyword[string], type[string], value1[int] value2[int], to[string], from[string],
Returns:	response[string]

Title:	Delete Alert
URL:	/api/alert/delete
Method:	POST

Parameters:	alertID[int]
Returns:	response[string]

Title:	Edit Alert
URL:	/api/alert/:id
Method:	PUT
Parameters:	keyword[string], type[string], value1[int], value2[int] to[string], from[string],
Returns:	response[string]

Title:	Admin Approve/Deny user
URL:	/api/admin/approve
Method:	POST
Parameters:	username[string], approved[boolean]
Returns:	response[string]

Title:	Promote To Admin
URL:	/api/admin/promote
Method:	POST
Parameters:	username[string]
Returns:	response[string]

Title:	Demote Admin
URL:	/api/admin/demote
Method:	POST
Parameters:	username[string]
Returns:	response[string]

Title:	Add Station
URL:	/api/stations
Method:	POST
Parameters:	name[string], key[string] expiration[string], connected[boolean], username[string]
Returns:	response[string]

Title:	Get All Stations
URL:	/api/stations
Method:	GET
Parameters:	
Returns:	stations[array]

Title:	Get Single Station
URL:	/api/stations/:id
Method:	GET, DELETE

Parameters:	
Returns:	response[string]

Title:	Update Station
URL:	/api/stations/:id
Method:	PUT
Parameters:	updated_at[datetime], mac_address[string], temperature[float], humidity[float], pressure[float], latitude[string], longitude[string], wind_speed[float], wind_direction[int], visibility[int], connected[boolean]
Returns:	response[string]

Title:	Update Station Health
URL:	/api/health/:id
Method:	PUT
Parameters:	updated_at[datetime], mac_address[string], cpu_usage[float], ram_usage[int], battery[float], connected[boolean]
Returns:	response[string]

Title:	Get All Weather Station Data
URL:	/api/weather
Method:	GET
Parameters:	
Returns:	updated_at[datetime], temperature[float], humidity[float], pressure[float], latitude[string], longitude[string], wind_speed[float], wind_direction[int], visibility[int], cpu_usage[float], ram_usage[int], battery[float]

Title:	Filter Weather Data
URL:	/api/weather?to=datetime&from=datetime&type=sensor
Method:	GET
Parameters:	
Returns:	updated_at[datetime], temperature[float], humidity[float], pressure[float], latitude[string], longitude[string], wind_speed[float], wind_direction[int], visibility[int], cpu_usage[float],

	ram_usage[int], battery[float]
--	-----------------------------------

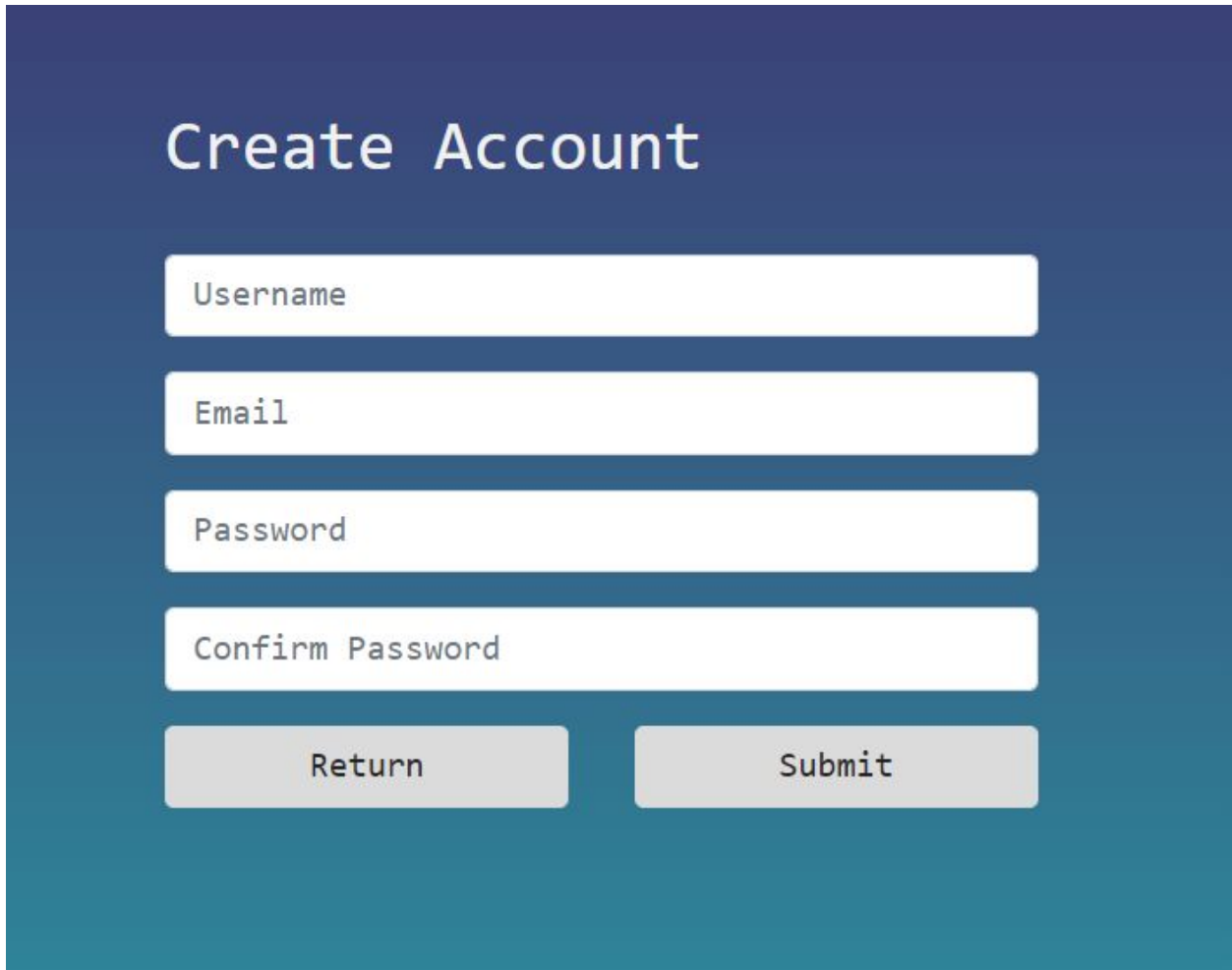
Title:	Add Weather Data
URL:	/api/weather
Method:	POST
Parameters:	updated_at[datetime], temperature[float], humidity[float], pressure[float], latitude[string], longitude[string], wind_speed[float], wind_direction[int], visibility[int], cpu_usage[float], ram_usage[int], battery[float]
Returns:	response[string]

4.8 USER INTERFACE DESIGN



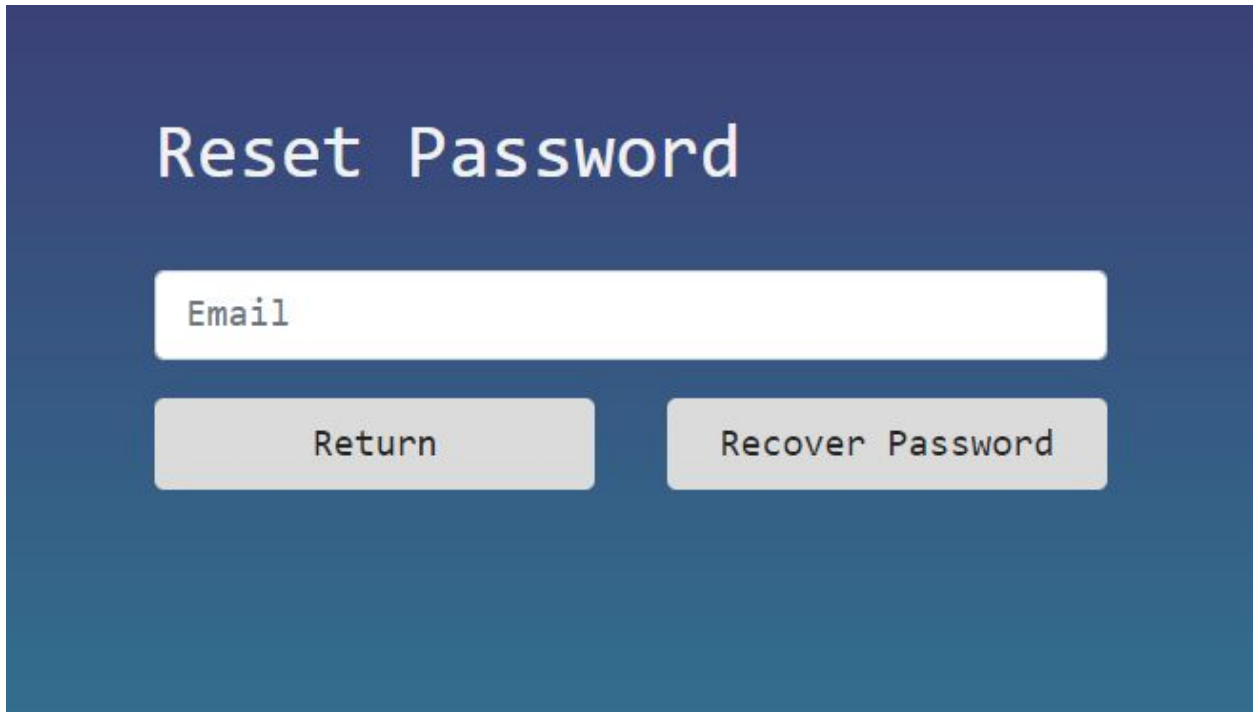
Forgot password?

Login View

A screenshot of a 'Create Account' web form. The form is set against a dark blue gradient background. It features four white input fields stacked vertically, each with a placeholder label: 'Username', 'Email', 'Password', and 'Confirm Password'. Below these fields are two light gray buttons with rounded corners, labeled 'Return' and 'Submit'.

Create Account

Create Account View



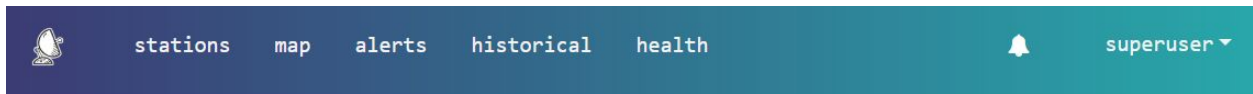
The image shows a web interface for resetting a password. It has a dark blue gradient background. At the top, the text "Reset Password" is displayed in a large, white, sans-serif font. Below this, there is a white rectangular input field with the placeholder text "Email". Underneath the input field, there are two light gray rectangular buttons with rounded corners. The first button is labeled "Return" and the second button is labeled "Recover Password".

Forgot Password View



The image shows a web interface for resetting a password. It has a dark blue gradient background. At the top, the text "Reset Password" is displayed in a large, white, sans-serif font. Below this, there are two white rectangular input fields. The first input field is preceded by the label "Password:" and contains the placeholder text "Password". The second input field is preceded by the label "Confirm Password:" and contains the placeholder text "Confirm Password". Below these two input fields, there is a single light gray rectangular button with rounded corners labeled "Submit".

Reset Password View



Filter

• Station #1 09/18/18 13:35:02

temperature: 86.58 °F	visibility: m
pressure: 994.97 hPa	wind speed: mph
humidity: 50.83%	wind direction: °

Stations View

Station Detail View

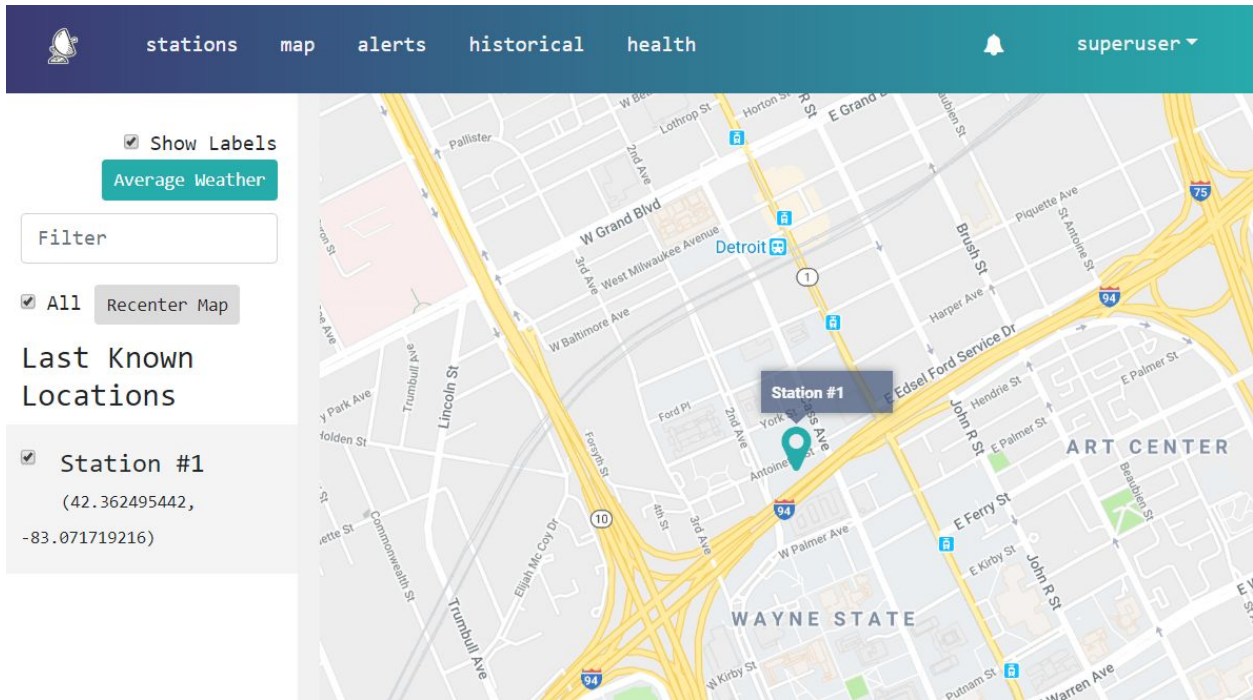
Station #1

Temperature	86.58 °F
Pressure	994.97 hPa
Humidity	50.83 %
Last Known Latitude	42.362495442
Last Known Longitude	-83.071719216

Map data ©2018 Google Terms of Use Report a map error

Save Changes Cancel

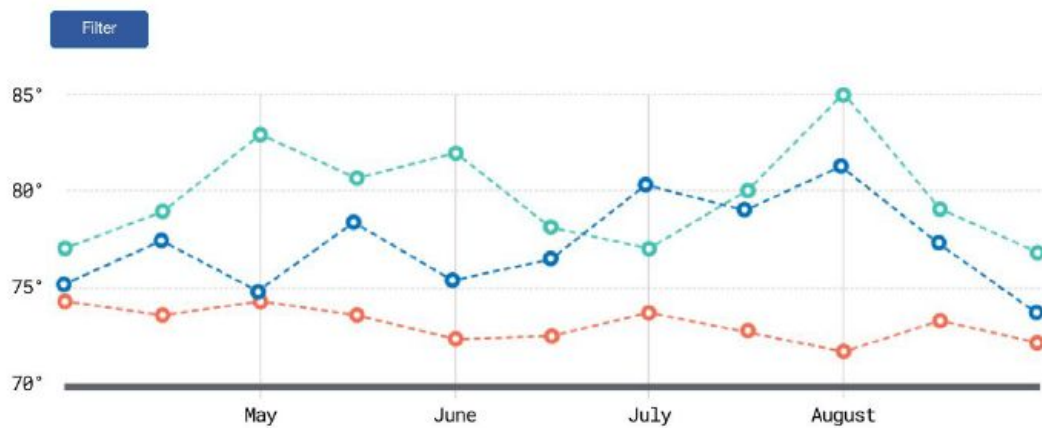
Stations Additional Information Modal



Station Map View



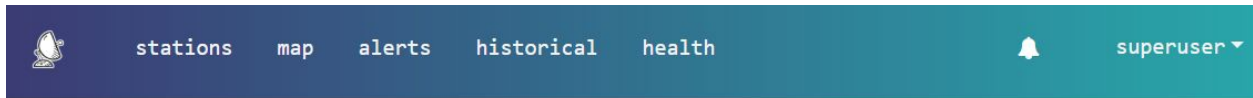
Historical Data



Historical Data View



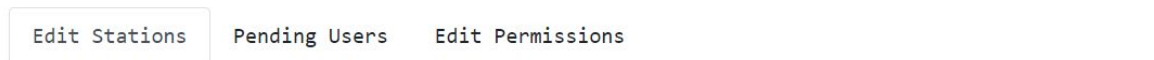
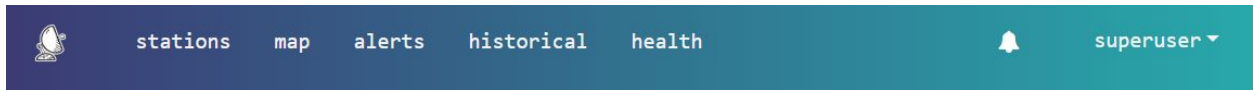
Historical Data Filter Modal



User Profile

Username	<input type="text" value="superuser"/>
Email	<input type="text" value="superuser1073543@gmail.com"/>
Phone	<input type="text" value="Phone Number"/>
Password	<input type="button" value="Change"/>
<input type="button" value="Save Changes"/>	

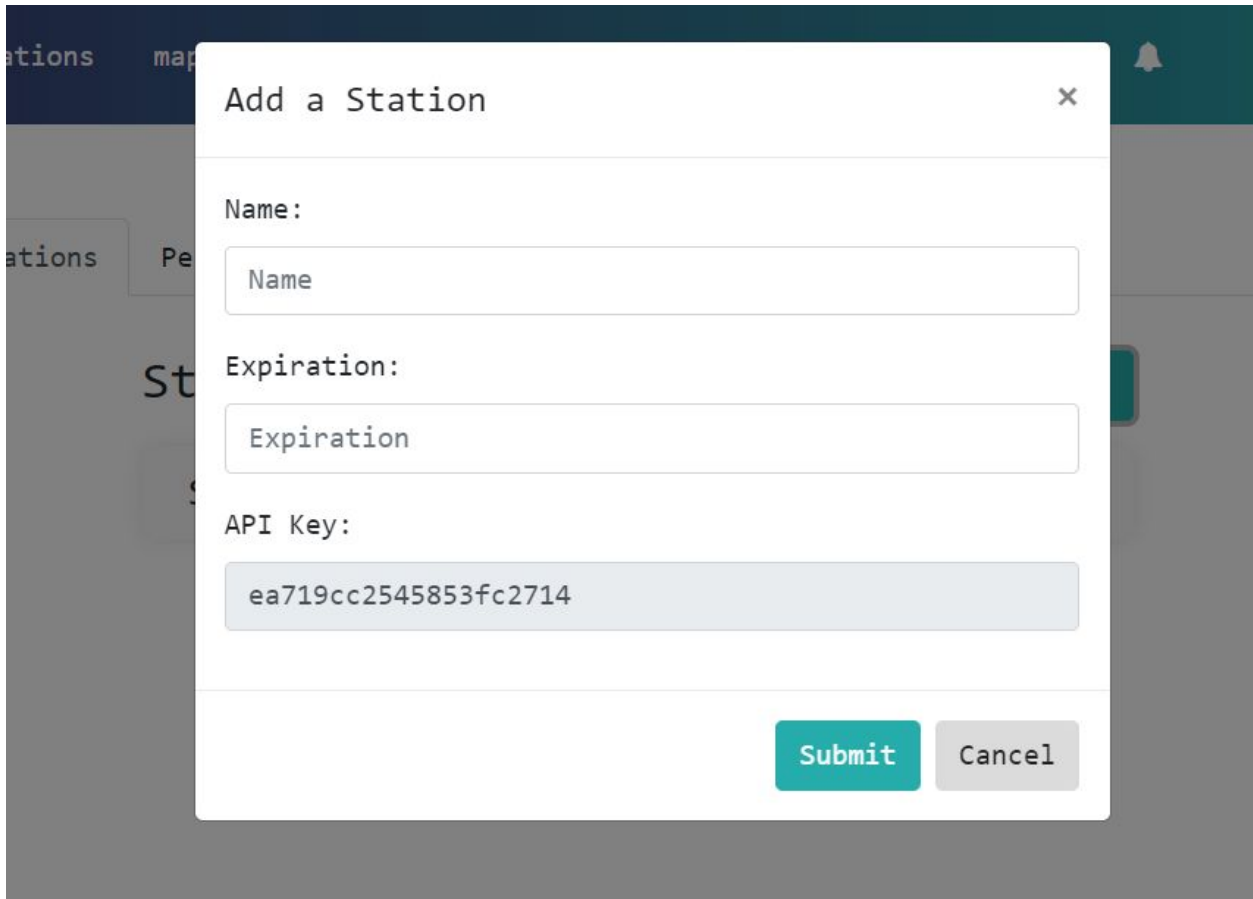
User Profile View



Stations

[Download Client](#)[Add](#)

Admin View



The image shows a web application interface with a modal dialog titled "Add a Station". The modal has a close button (X) in the top right corner. It contains three input fields: "Name:" with a placeholder "Name", "Expiration:" with a placeholder "Expiration", and "API Key:" with a placeholder "ea719cc2545853fc2714". At the bottom right of the modal are two buttons: "Submit" (teal) and "Cancel" (grey). The background of the application is dark teal with some text like "ations", "map", "ations", "Pe", "St", and "s" visible.

ations map

ations Pe

St

s

Add a Station

Name:

Name

Expiration:

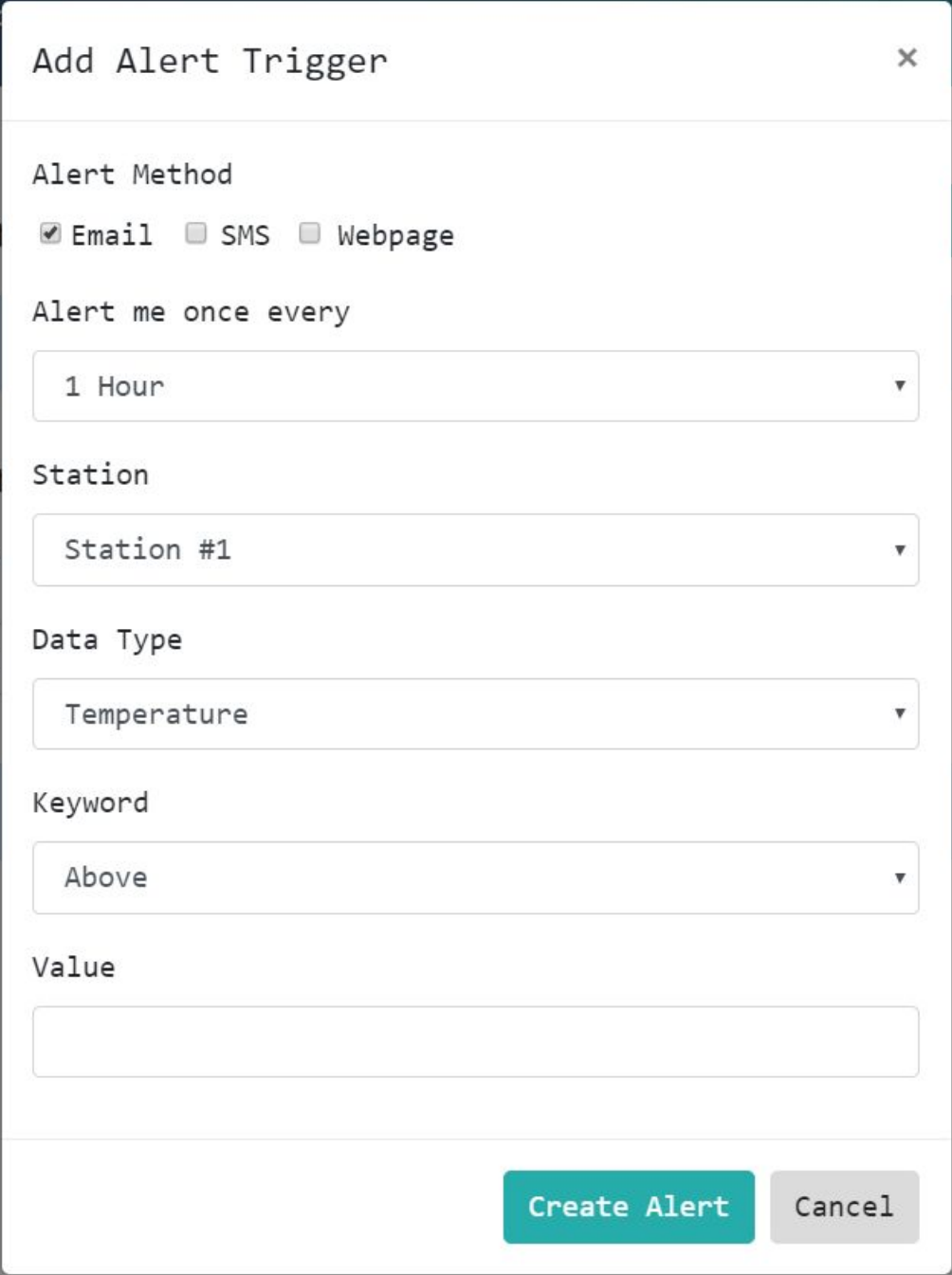
Expiration

API Key:

ea719cc2545853fc2714

Submit Cancel

Add/Edit Station View



The image shows a mobile application interface with a modal dialog box titled "Add Alert Trigger". The dialog has a close button (X) in the top right corner. It contains several form fields: "Alert Method" with radio buttons for "Email" (selected), "SMS", and "Webpage"; "Alert me once every" with a dropdown menu showing "1 Hour"; "Station" with a dropdown menu showing "Station #1"; "Data Type" with a dropdown menu showing "Temperature"; "Keyword" with a dropdown menu showing "Above"; and "Value" with an empty text input field. At the bottom right of the dialog are two buttons: "Create Alert" (teal) and "Cancel" (grey). The background of the app is dark teal with some blurred text and icons.

ns map

Add Alert Trigger

Alert Method

☒ Email ☐ SMS ☐ Webpage

Alert me once every

1 Hour

Station

Station #1

Data Type

Temperature

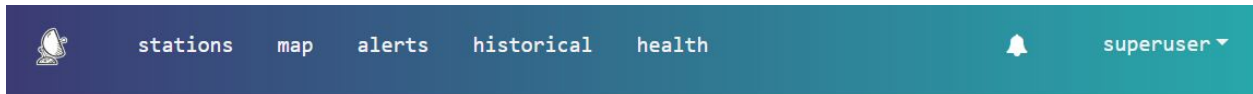
Keyword

Above

Value

Create Alert Cancel

Add Alert Trigger View



Alert me when...

Add

Station #1's cpu_usage is above 70

email

every: 1 hour

Station #1's ram_usage is between 5000 and 8000

email

every: 1 hour

Alert history:

Filter 2018-10-21

All alerts ▼

There are no alerts for this date.

Alert Triggers View



Filter

● Station #1

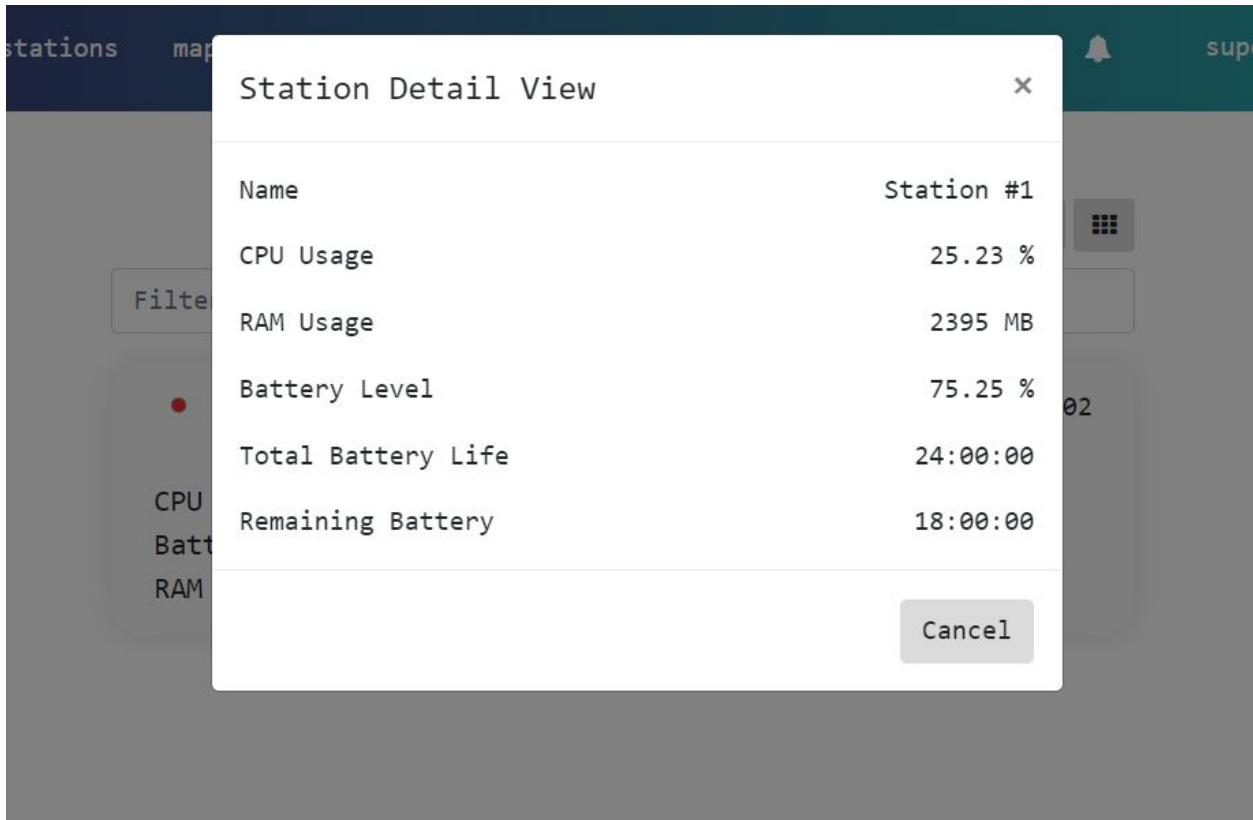
09/18/18 13:35:02

CPU Usage: 25.23%

Battery: 75.25%

RAM Usage: 2395mb

Station Health View



Station Health Additional Information Modal

5 PRODUCT DESIGN SPECIFICATION

APPROVAL

The undersigned acknowledge they have reviewed the Weather Station **Product Design Specification** document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

Print Name: _____ Date _____

Signature: _____

Title: _____

Role: _____

Print Name: _____ Date _____

Signature: _____

Title: _____

Role: _____

Print Name: _____ Date _____

Signature: _____

Title: _____

Role: _____

Print Name: _____ Date _____

Signature: _____

Title: _____

Role: _____

Print Name: _____ Date _____

Signature: _____

Title: _____

Role: _____

A APPENDICES

APPENDIX A: REFERENCES

Jackson, B. *Weather Station Software Requirements Specifications* (Vol. V1.3). Wayne State University.

Description	Location
Raspberry Pi 3 Model B Motherboard	https://www.amazon.com/Raspberry-Pi-RASPBERRYPI3-MODB-1GB-Model-Motherboard/dp/B01CD5VC92
LoRa/GPS HAT	https://www.robotshop.com/en/lora-gps-long-range-transceiver-hat-915-mhz-north-america.html
Anker PowerCore 20100mAh	https://www.amazon.com/Portable-Charger-Anker-PowerCore-20100mAh/dp/B00X5RV14Y
Jackery 6000mAh Portable Charger Bar	https://www.amazon.com/Jackery-Portable-Pocket-sized-Emergency-Flashlight/dp/B00DTXA578?th=1
Sandisk microSD	https://www.amazon.com/SanDisk-microSDHC-Standard-Packaging-SDSQU NC-032G-GN6MA/dp/B010Q57T02
915MHz LoRa Antenna	https://www.sparkfun.com/products/14876

APPENDIX B: KEY TERMS

Term	Definition
LPWAN	Low power wide area network is a type of wireless telecommunication network designed to allow long range communication
LoRa WAN	A media access control (MAC) protocol for wide area networks. It is designed to allow low-powered devices to communicate with Internet-connected applications over long range wireless connections
LoRa Hat	An expansion module for LoRa WAN and GPS for use with a Raspberry Pi
MQTT	Message Queuing Telemetry Transport is an ISO standard publish-subscribe based on messaging protocol. It is designed to be used in remote locations or where network bandwidth is limited
Mosquitto	An open source message broker that implements the MQTT protocol
Broker	A broker is an IP that a publisher will send data to queue while waiting for a subscriber to retrieve
Subscriber	A subscriber is a device that will reach out to a broker to retrieve any data queued. Subscriber sends no data to the broker, but just checks it for queued data
Publisher	A publisher is a device that sends data to a broker, where it will be retrieved by a subscriber
Raspberry Pi	An affordable computer chip that is made up of one serial board
API	Application Programming Interface, we use this to grab information from another website and set up URLs the Raspberry Pi can use to send data
RDS	Relational Database Service
AWS RDS	Relational Database Service allows you to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks

VPC	Amazon Virtual Private Cloud lets you launch AWS resources, such as Amazon RDS DB instances, into a virtual private cloud
Internet Gateway	Allow resources in a VPC to communicate with the internet by creating and attaching an internet gateway to the VPC
MySQL	An open-source relational database management system
HTTP	HyperText Transfer Protocol. Generally used to send data across the web
Server	Program that waits for a message to either push or pull data. The server determines if the request should be allowed access to its data, and responds if the request is allowed
Client	Program that sends a message to a server. There can be multiple clients sending messages to the server at the same time.
Asymmetric Encryption	Known as public key encryption. The public key is used by anyone to encrypt a message and the private key is kept secret by the recipient.
Encryption	The process of converting information/data into a code to prevent unauthorized access
Decryption	The process of transforming data that has been rendered unreadable through encryption back to its unencrypted form
Hashing	It is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string.
Public Key	It is a large numerical value that can be used to encrypt data which is intended for a particular recipient, so that the encrypted data can be deciphered only by using a second key that is known only to the recipient
Private Key	The second key that is only known to the recipient of encrypted information to be used for decryption of the data.
React	React is a frontend Javascript framework that allows small parts of each web page to be built using components. These components make it easier to split up a web page and keep each individual piece bug free.
Python	An interpreted high-level backend programming language.

Node	Node is a backend runtime environment, so that Javascript can be executed on the server side. This means that while the Node server is either retrieving or sending data, it will be able to continue working instead of being blocked as that interaction is happening.
Advanced Encryption Standard (AES)	A symmetrical block-cipher encryption and decryption algorithm
Rivest-Shamir-Adleman (RSA)	A cryptosystem for public and private keys used during encryption and decryption

APPENDIX C: USE CASE/SEQUENCE DIAGRAM MAPPING

Description	Use Case	Sequence Diagram
Add Custom Alert	UC-1	4.3.4
Receive Health Alerts	UC-2	4.3.5
View Health Monitor	UC-3	4.2.1
Filter Health Monitor	UC-4	
View Health Station Details	UC-5	
Edit Station Name	UC-6	
View Historical Data	UC-7	
Filter Historical Data	UC-8	
Send Weather Station Data (Servant)	UC-9	4.2.2
Send Weather Station Data (Master)	UC-10	4.2.3
Historic Alerts	UC-11	