Raspberry Pi Weather Station

Software Test Cases

Version: 1

11/26/2018

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

Version History

Date	Version	Author	Comments
11-26-2018	v1.0	Raspberry Pi Group Fall 2018	First Version
12-09-2018	v1.1	Joynal Abedin	Update AWS Test case: precondition login to AWS

Table of Contents

Functional Test Cases

View Station Health

Test Case Name	View Station Health: No Stations Connected
Test Case Id	TC - 01
Priority	Low
Pre-Conditions	Execute script DBS-01
Post-Conditions	No stations are loaded on the health stations page
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	An error message that reads “no stations available” is displayed

Test Case Name	View Station Health: Connected Station
Test Case Id	TC - 02
Priority	High
Pre-Conditions	Execute script DBS-01, DBS-02
Post-Conditions	A station health card is displayed
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	A single station health card is displayed back to the user with a green connection indicator

Test Case Name	View Station Health: Disconnected Station
Test Case Id	TC - 03
Priority	High
Pre-Conditions	Execute script DBS-01, DBS-03
Post-Conditions	A station health card is displayed

Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	A single station health card is displayed back to the user with a red connection indicator

Test Case Name	View Station Health: List Mode
Test Case Id	TC - 04
Priority	Moderate
Pre-Conditions	Execute script DBS-01, DBS-04
Post-Conditions	Station health cards are displayed on the page
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the “list” image button at the top right side of the filter input
Expected Results	All station health cards are displayed in a list format

Test Case Name	View Station Health: Grid Mode
Test Case Id	TC - 05
Priority	Moderate
Pre-Conditions	Execute script DBS-01, DBS-04
Post-Conditions	Station health cards are displayed on the page
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the “grid” image button at the top right side of the filter input
Expected Results	All station health cards are displayed in a grid format

Test Case Name	View Station Health: Good CPU Usage
Test Case Id	TC - 06
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, cpu_usage should be set at 25

Post-Conditions	Station health card is displayed with color code to CPU usage
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Text color of CPU usage will be green

Test Case Name	View Station Health: Bad CPU Usage
Test Case Id	TC - 07
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, cpu_usage should be set at 85
Post-Conditions	Station health card is displayed with color code to CPU usage
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Text color of CPU usage will be red

Test Case Name	View Station Health: Moderate CPU Usage
Test Case Id	TC - 08
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, cpu_usage should be set at 55
Post-Conditions	Station health card is displayed with color code to CPU usage
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Text color of CPU usage will be orange

Test Case Name	View Station Health: Good RAM usage
Test Case Id	TC - 09
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03

	When inserting into sensor_type table, ram_usage should be set at 25
Post-Conditions	Station health card is displayed with color code to RAM usage
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Text color of RAM usage will be green

Test Case Name	View Station Health: Bad RAM Usage
Test Case Id	TC - 10
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, ram_usage should be set at 85
Post-Conditions	Station health card is displayed with color code to RAM usage
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Text color of RAM usage will be red

Test Case Name	View Station Health: Moderate RAM Usage
Test Case ID	TC - 11
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, ram_usage should be set at 55
Post-Conditions	Station health card is displayed with color code to RAM usage
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Text color of RAM usage will be orange

Test Case Name	View Station Health: Test Station Battery Percentage
Test Case ID	TC - 12
Priority	Low

Pre-Conditions	Execute script DBS-01, DBS-03
Post-Conditions	Station health card is displayed
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Station health card display battery percentage

Test Case Name	View Station Health: Master Battery Percentage
Test Case ID	TC - 13
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-07
Post-Conditions	Station health card is displayed for master station
Test Steps	1. Click on the “health” page in the navigation bar 2. View page
Expected Results	Master station health card displays “Connected to power source” for battery

Test Case Name	View Station Health: Master Battery Percentage 2
Test Case ID	TC - 14
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-07 Instead of “Master” as the station name, use “master”
Post-Conditions	Station health card is displayed for master station
Test Steps	3. Click on the “health” page in the navigation bar 4. View page
Expected Results	Master station health card displays “Connected to power source” for battery

Test Case Name	View Station Health: View Mode is Saved
Test Case ID	TC - 15

Priority	Moderate
Pre-Conditions	Execute script DBS-04, DBS-05
Post-Conditions	View mode cookie is updated
Test Steps	1. Click on the “health” page in the navigation bar 2. Change the view mode on the health page (starts in the list view mode) 3. Click the “map” page in the navigation bar 4. Click on the “health” page in the navigation bar
Expected Results	The view mode does not change when navigating away and returning to the page

View Station Health Details

Test Case Name	View Station Health Details: User View
Test Case ID	TC - 16
Priority	Moderate
Pre-Conditions	Execute script DBS-01, DBS-02
Post-Conditions	Modal with no name change input and weather data is displayed
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Test Station” 3. View modal
Expected Results	A modal is displayed to the user and there is no text input for the station name

Test Case Name	View Station Health Details: Good CPU Usage
Test Case Id	TC - 17
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, cpu_usage should be set at 25
Post-Conditions	Station health card is displayed with color code to CPU usage
Test Steps	1. Click on the “health” page in the navigation bar

	2. Click on the station health card with the name “Test Station” 3. View page
Expected Results	Text color of CPU usage will be green

Test Case Name	View Station Health Details: Bad CPU Usage
Test Case Id	TC - 18
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, cpu_usage should be set at 85
Post-Conditions	Station health card is displayed with color code to CPU usage
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Test Station” 3. View page
Expected Results	Text color of CPU usage will be red

Test Case Name	View Station Health Details: Moderate CPU Usage
Test Case Id	TC - 19
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, cpu_usage should be set at 55
Post-Conditions	Station health card is displayed with color code to CPU usage
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Test Station” 3. View page
Expected Results	Text color of CPU usage will be orange

Test Case Name	View Station Health Details: Good RAM Usage
Test Case Id	TC - 20
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03

	When inserting into sensor_type table, ram_usage should be set at 25
Post-Conditions	Station health card is displayed with color code to RAM usage
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Test Station” 3. View page
Expected Results	Text color of RAM usage will be green

Test Case Name	View Station Health Details: Bad RAM Usage
Test Case Id	TC - 21
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, ram_usage should be set at 85
Post-Conditions	Station health card is displayed with color code to RAM usage
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Test Station” 3. View page
Expected Results	Text color of RAM usage will be red

Test Case Name	View Station Health Details: Moderate RAM Usage
Test Case ID	TC - 22
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03 When inserting into sensor_type table, ram_usage should be set at 55
Post-Conditions	Station health card is displayed with color code to RAM usage
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Test Station” 3. View page
Expected Results	Text color of RAM usage will be orange

Test Case Name	View Station Health Details: Test Station Battery Percentage

Test Case ID	TC - 23
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-03
Post-Conditions	Station health card is displayed
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Test Station” 3. View page
Expected Results	Station health card display battery percentage, total battery life and remaining battery life

Test Case Name	View Station Health Details: Master Battery Percentage
Test Case ID	TC - 24
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-07
Post-Conditions	Station health card is displayed for master station
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “Master” 3. View page
Expected Results	Master station health card displays “Connected to power source” for batter with remaining battery and total battery displayed with “-”

Test Case Name	View Station Health Details: Master Battery Percentage With ‘m’
Test Case ID	TC - 25
Priority	Low
Pre-Conditions	Execute script DBS-01, DBS-07 Instead of “Master” as the station name, use “master”
Post-Conditions	Station health card is displayed for master station
Test Steps	1. Click on the “health” page in the navigation bar 2. Click on the station health card with the name “master” 3. View page

Expected Results	Master station health card displays “Connected to power source” for battery with remaining battery and total battery displayed with “-”

Filter Historical Data

Test Case Name	Filter Historical Data: Filter By CPU Usage
Test Case ID	TC - 26
Priority	High
Pre-Conditions	Execute script DBS-06
Post-Conditions	Historical graph will re-render on the page.
Test Steps	1. Click “historical” page in the Navigation bar if 2. Click on “Filter” button on page 3. In the sensor type drop down select “CPU Usage” 4. Click “submit” button on bottom right corner of modal 5. View reloaded historical page
Expected Results	The historical page will reload with a new graph that displays CPU Usage from all stations for the last 24 hours

Test Case Name	Filter Historical Data: Filter By RAM Usage
Test Case ID	TC - 27
Priority	High
Pre-Conditions	Execute script DBS-06
Post-Conditions	Historical graph will re-render on the page.
Test Steps	1. Click “historical” page in the Navigation bar if 2. Click on “Filter” button on page 3. In the sensor type drop down select “RAM Usage” 4. Click “submit” button on bottom right corner of modal 5. View reloaded historical page
Expected Results	The historical page will reload with a new graph that displays RAM Usage from all stations for the last 24 hours

SMS Alerts With Twilio

Test Case Name	SMS Alerts With Twilio: Moderate CPU Usage
Test Case ID	TC - 28
Priority	Moderate
Pre-Conditions	Text file containing flags is set all to false and phone number the SMS will be sent to. Data text file with the following data: <0000000000000000000, {temperature}, {humidity}, {pressure}, {longitude}, {latitude}, {CPU Usage}, ,{RAM Usage}> Data is random but set CPU Usage to
Post-Conditions	An alert is sent to the user
Test Steps	1. On Master weather station, open terminal 2. Navigate to weather-station-master/client 3. Run curl.py 4. Retrieve SMS sent
Expected Results	SMS Alert with temperature, humidity, pressure, CPU usage and RAM usage and also state the abnormal

Test Case Name	SMS Alerts With Twilio: Bad CPU Usage
Test Case ID	TC - 29
Priority	Moderate
Pre-Conditions	Text file containing flags is set all to false and phone number the SMS will be sent to. Data text file with the following data: <0000000000000000000, {temperature}, {humidity}, {pressure}, {longitude}, {latitude}, {CPU Usage}, ,{RAM Usage}> Data is random but set CPU Usage to
Post-Conditions	An alert is sent to the user
Test Steps	1. On Master weather station, open terminal 2. Navigate to weather-station-master/client 3. Run curl.py 4. Retrieve SMS sent
Expected Results	SMS Alert with temperature, humidity, pressure, CPU usage and RAM

	usage and also state the there is a very high CPU usage

Test Case Name	SMS Alerts With Twilio: Moderate RAM Usage
Test Case ID	TC - 30
Priority	Moderate
Pre-Conditions	Text file containing flags is set all to false and phone number the SMS will be sent to. Data text file with the following data: <0000000000000000000, {temperature}, {humidity}, {pressure}, {longitude}, {latitude}, {CPU Usage}, ,{RAM Usage}> Data is random but set CPU Usage to
Post-Conditions	An alert is sent to the user
Test Steps	1. On Master weather station, open terminal 2. Navigate to weather-station-master/client 3. Run curl.py 4. Retrieve SMS sent
Expected Results	SMS Alert with temperature, humidity, pressure, CPU usage and RAM usage and also state the abnormal

Test Case Name	SMS Alerts With Twilio: Bad RAM Usage
Test Case ID	TC - 31
Priority	Moderate
Pre-Conditions	Text file containing flags is set all to false and phone number the SMS will be sent to. Data text file with the following data: <0000000000000000000, {temperature}, {humidity}, {pressure}, {longitude}, {latitude}, {CPU Usage}, ,{RAM Usage}> Data is random but set CPU Usage to
Post-Conditions	An alert is sent to the user
Test Steps	1. On Master weather station, open terminal 2. Navigate to weather-station-master/client 3. Run curl.py 4. Retrieve SMS sent
Expected Results	SMS Alert with temperature, humidity, pressure, CPU usage and RAM usage and also state the there is a very high RAM usage

LoRa Communication

Test Case Name	Servant device broadcasting data
Test Case Id	TC - 32
Priority	High
Pre-Conditions	Weather Station is connected to a power source.
Post-Conditions	Master device creating a file in folder EncryptedFilesReceived with encrypted message from the servant device.
Test Steps	1. Plug in the power source to the servant device. 2. Verify a solid blue light is present on the LoRa HAT 3. By default devices are sending on message on boot-up as a background process, an additional terminal can be ran and messages being sent verified. Master will receive twice as many messages during this time. 4. Verification that the correct message is being sent can be done using a decryption function with a file sent and decrypting on both ends to verify correctness.
Expected Results	Encrypted text files should be written to EncryptedFilesReceived on the master device that are exactly 128 bytes.

Test Case Name	Devices obtains current location - GPS
Test Case Id	TC - 33
Priority	Moderate
Pre-Conditions	1. Weather Station is connected to a power source. 2. Device has new production version of LoRa/GPS Hat. (early production versions have wrong L1 and C3 components) 3. External GPS antenna is securely fastened to the device. 4. Correct configurations are made to device to enable GPS component.
Post-Conditions	Latitude and Longitude of device is obtained

Test Steps	1. Plug in the power source to the servant device. 2. Verify a green light is present on the LoRa HAT. 3. Perform a quick check to see what data is coming from the GPS. To do so open terminal and enter: “sudo cat /dev/ttyS0” 4. Verify that current devices location is being obtained by checking the lines that begin with $GPGLL (latitude and longitude will be the next two pieces of information on that line)
Expected Results	Current locations matches the output of the GPS component.

Test Case Name	Master device receiving data from multiple servant devices
Test Case Id	TC - 34
Priority	Moderate
Pre-Conditions	Weather Stations are connected to a power source. Receiver/sender script is running on the appropriate devices
Post-Conditions	Master device has weather data from multiple sources
Test Steps	1. Plug in the power source to all devices being tested. 2. Verify a blue light is present on the LoRa HAT. 3. Check the EncryptedFilesRecieved folder on the master device and verify that there are different apikeys associated to the data files. 4. Compare apikeys to known apikeys to verify the information is coming from the anticipated devices
Expected Results	The master device will obtain weather information from more than one servant device

Test Case Name	Verification that encrypted message being broadcast is 128 bytes
Test Case Id	TC - 35
Priority	Moderate

Pre-Conditions	1. Weather Stations are connected to a power source. 2. Receiver/sender script is running on the appropriate devices. 3. Servant Pi is successfully encrypting the information before transit to master. 4. Master receives data and stores it into the folder named: EncryptedFilesReceived.
Post-Conditions	All data files stored into this folder will be 128 bytes due to the encryption algorithm being executed by the servant device.
Test Steps	1. Plug in the power source to the master and servant devices. 2. Verify a solid blue light is present on the LoRa HAT. 3. By default, the servant devices are broadcasting data messages on boot-up as a background process, an additional terminal can be running and the messages being sent can be verified in the payload section. **<u>If addition terminal is used:</u>** 4. Check the Length section to verify it is 128. **<u>If addition terminal is NOT used:</u>** 5. Move to the EncryptedFilesReceived folder. 6. Select a file and go to its properties and verify the size is 128 bytes.
Expected Results	All data files stored will be exactly 128 bytes in size.

Integration Test Cases

Data Handling On Raspberry Pi

Test Case Name	Post Data to Server
Test Case Id	TC - 35
Priority	High
Pre-Conditions	Master Station is connected to a power source. Directory “EncryptedFilesRecieved” on Master is not empty.

Post-Conditions	Master will receive a 200 status code return.
Test Steps	1. Open a terminal 2. Cd into directory /path/to/ weatherstation 3. Enter “python post.py” into terminal 4. Hit Enter on keyboard 5. Verify that post.py grabs the oldest file in the directory. 6. Verify if a 200 code is returned. 7. If 200 is returned, check the website to see if the changes has been made.
Expected Results	A 200 code is returned and weather changes on the front-end has been made asynchronously.

Test Case Name	Delete Data on Master Pi after posting
Test Case Id	TC - 36
Priority	High
Pre-Conditions	Master Station is connected to a power source. Directory “EncryptedFilesRecieved” on Master is not empty.
Post-Conditions	The oldest file after being posted, will be removed.
Test Steps	1. Open a terminal 2. Cd into directory /path/to/ weatherstation 3. Enter “python post.py” into terminal 4. Hit Enter on keyboard 5. Verify that post.py grabs the oldest file in the directory. 6. Verify if a 200 code is returned. 7. If 200 is returned, check the website to see if the changes has been made. 8. Open a new terminal 9. Enter Cd EncryptedFilesReceived 10. Verify that post.py has removed oldest file by looking for the oldest file.
Expected Results	Oldest file will be removed after successful posting to server. Number of files inside directory will be 1 less

Test Case Name	LoRa GPS Module
Test Case Id	TC-52
Priority	Low

Pre-Conditions	Any Raspberry Pi device with a LoRa HAT is powered on and the green light on top is blinking.
Post-Conditions	The GPS location will be listed in the output from the Weather Station.
Test Steps	1. Power on any LoRa HAT enabled Raspberry Pi 2. Wait 5 seconds for output to be produced in the /client/data folder 3. Decrypt the newest file 4. Check the latitude and longitude of output and determine correctness of output
Expected Results	Decrypted text should having the latitude and longitude of the device listed and it should be correct within a certain percentage of error.

Test Case Name	Verification of master device being connected
Test Case Id	TC - 53
Priority	Moderate
Pre-Conditions	Two master devices are online at the same time.
Post-Conditions	A node device with master capabilities converts to master and starts receiving messages from the servant devices.
Test Steps	1. Power off master device 2. Verify that the master ownership was transferred by the server
Expected Results	Master was reassigned to a device that has internet connection and continues to be the new master device.

Test Case Name	Storm Alert
Test Case Id	TC - 54
Priority	Moderate
Pre-Conditions	At least one master and one servant device currently powered and sending data to server.
Post-Conditions	An alert is sent to the user

Test Steps	1. Power a master and a servant device. 2. Manually input a file and send the file to the master that includes data outside of a threshold determined with a pre-developed data model 3. Master will send the file to the server 4. Server will use a lambda function to determine that the data is beyond a threshold of change that would suggest a storm is incoming.
Expected Results	An alert with the storm prediction is sent to the user via website or SMS.

Test Case Name	Lambda in response to Submitting sensor data
Test Case Id	TC - 41
Priority	High
Pre-Conditions	1. Visit the Amazon Web Service at https://aws.amazon.com 2. Choose Create an AWS Account. Note: If you've signed in to AWS recently, it might say Sign In to the Console. Otherwise it should say Create an AWS Account. If the option “Create a new AWS Account” isn’t displayed, first choose to Sign in to a different account, and select “Create a new AWS Account”. 3. Type the requested account information, and then choose Continue. 4. Choose Personal . 5. Type the requested personal information. 6. Read and check the AWS Customer Agreement. 7. Choose Create Account and Continue 8. Add a payment method (if you cross the free tier limit, AWS may start charging you. Here’s more detail about free tier https://aws.amazon.com/free) 9. Verify your phone number 10. Choose the Basic support plan 11. Now you have an AWS Account and Signed in to the AWS account

Post-Conditions	Data posted successfully!
Test Steps	1. Open AWS Console 2. Search for Lambda in the search bar 3. Click on the result where it says “Lambda” 4. Click on the function that says “sensordata” 5. Click on “API Gateway” 6. Take a note of the API key and the API endpoint 7. Create a text file with the following data: 2018, ffb97bc80462cbeca180, 78.68, 36.51, 1003.82, n/a, n/a, 50.0 Note: the above missing two parameters at the end 8. Make an API post request using CURL curl -H "x-api-key: your-new-api-key" aws_endpoint/post1/sensordata -d "@temp.txt" -v
Expected Results	* Trying 3.16.104.200... * TCP_NODELAY set * Connected to l4m03tep94.execute-api.us-east-2.amazonaws.com (3.16.104.200) port 443 (#0) * TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 * Server certificate: *.execute-api.us-east-2.amazonaws.com * Server certificate: Amazon * Server certificate: Amazon Root CA 1 * Server certificate: Starfield Services Root Certificate Authority - G2 > POST /post1/sensordata HTTP/1.1 > Host: l4m03tep94.execute-api.us-east-2.amazonaws.com > User-Agent: curl/7.54.0 > Accept: */* > x-api-key: KIfI3IStxB6PgOXj014cN1DwoS2kpqRh5gsFlpjT > Content-Length: 69 > Content-Type: application/x-www-form-urlencoded > * upload completely sent off: 69 out of 69 bytes < HTTP/1.1 502 Bad Gateway < Date: Mon, 26 Nov 2018 02:58:27 GMT < Content-Type: application/json < Content-Length: 36 < Connection: keep-alive < x-amzn-RequestId: 26074880-f127-11e8-a3d0-7128c8a7412e < x-amz-apigw-id: Q8wlGGqLCYcFl8A=

	 * Connection #0 to host l4m03tep94.execute-api.us-east-2.amazonaws.com left intact **{"message": "Internal server error"}**

Test Case Name	Lambda rejecting bad sensor data parameters
Test Case Id	TC - 42
Priority	High
Pre-Conditions	TC-41: Tester is logged into the AWS account
Post-Conditions	Internal server error
Test Steps	Open AWS Console Search for Lambda in the search bar Click on the result where it says “Lambda” Click on the function that says “sensordata” Click on “API Gateway” Take a note of the API key and the API endpoint Create a text file with the following data: 2018, ffb97bc80462cbeca180, 78.68, 36.51, 1003.82, n/a, n/a, 50.0, , 78.3 Make an API post request using CURL curl -H "x-api-key: your-new-api-key" aws_endpoint/post1/sensordata -d "@temp.txt" -v

Expected Results	* Trying 18.191.110.68... * TCP_NODELAY set * Connected to l4m03tep94.execute-api.us-east-2.amazonaws.com (18.191.110.68) port 443 (#0) * TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 * Server certificate: *.execute-api.us-east-2.amazonaws.com * Server certificate: Amazon * Server certificate: Amazon Root CA 1 * Server certificate: Starfield Services Root Certificate Authority - G2 > POST /post1/sensordata HTTP/1.1 > Host: l4m03tep94.execute-api.us-east-2.amazonaws.com > User-Agent: curl/7.54.0 > Accept: */* > x-api-key: KIfI3IStxB6PgOXj014cN1DwoS2kpqRh5gsFlpjT > **Content-Length: 74** > Content-Type: application/x-www-form-urlencoded > *** upload completely sent off: 74 out of 74 bytes** < **HTTP/1.1 200 OK** < Date: Sun, 25 Nov 2018 16:45:04 GMT < Content-Type: application/json < Content-Length: 27 < Connection: keep-alive < x-amzn-RequestId: 746be80b-f0d1-11e8-a006-87eb84c6cfc4 < Access-Control-Allow-Origin: * < x-amz-apigw-id: Q7WuSEeuCYcFjVw= < X-Amzn-Trace-Id: Root=1-5bfad18e-a5b25e621db219f124cf9cd6;Sampled=0 * Connection #0 to host l4m03tep94.execute-api.us-east-2.amazonaws.com left intact **"Data posted successfully!"**

Test Case Name	Successful verify weather API using Lambda
Test Case Id	TC - 47
Priority	High
Pre-Conditions	TC41: Tester is logged into AWS Account

Post-Conditions	API Verified
Test Steps	1. Open AWS Console 2. Search for Lambda in the search bar 3. Click on the result where it says “Lambda” 4. Click on the function that says “verifyKey” 5. Click on “API Gateway” 6. Take a note of the API key and the API endpoint 7. Log in to the website using Username: superuser Password: superuser123 8. Click on superuser on the right top drop down 9. Select “admin” 10. Under “Edit Station”, click on “Add” 11. Fill out the following: Name: testVerifyApi Expiration: select a date from the calender 12. Copy the API key in your clipboard 13. Click on Submit button 14. Open up terminal 15. Type the following command: curl -H "x-api-key: 7NYpat2DyO6yqh6EqXSah924XRzVeBi26TEPcwfx" https://cxdp3vrdt6.execute-api.us-east-2.amazonaws.com/dev/verifyKey -d '{"key":"e03bee689ec73759bd54"}' -v
Expected Results	* Trying 3.16.64.73... * TCP_NODELAY set * Connected to cxdp3vrdt6.execute-api.us-east-2.amazonaws.com (3.16.64.73) port 443 (#0) * TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 * Server certificate: *.execute-api.us-east-2.amazonaws.com * Server certificate: Amazon * Server certificate: Amazon Root CA 1 * Server certificate: Starfield Services Root Certificate Authority - G2 > POST /dev/verifyKey HTTP/1.1 > Host: cxdp3vrdt6.execute-api.us-east-2.amazonaws.com > User-Agent: curl/7.54.0 > Accept: */* > x-api-key: 7NYpat2DyO6yqh6EqXSah924XRzVeBi26TEPcwfx

	> Content-Length: 30 > Content-Type: application/x-www-form-urlencoded > * upload completely sent off: 30 out of 30 bytes < HTTP/1.1 200 OK < Date: Sun, 25 Nov 2018 19:12:10 GMT < Content-Type: application/json < Content-Length: 15 < Connection: keep-alive < x-amzn-RequestId: 022c935c-f0e6-11e8-a593-556048bba1d3 < Access-Control-Allow-Origin: * < x-amz-apigw-id: Q7sRoFmNiYcFrmg= < X-Amzn-Trace-Id: Root=1-5bfaf40a-e435e40b7d5cdea5d5cfcdff;Sampled=0 * Connection #0 to host cxdp3vrdt6.execute-api.us-east-2.amazonaws.com left intact **"API Verified!"**

Test Case ID	Unsuccessful verify weather API using Lambda
Test Case ID	TC - 48
Priority	High
Pre-Conditions	TC:41: Tester is logged into AWS Account
Post-Conditions	Invalid API
Test Steps	1. Open AWS Console 2. Search for Lambda in the search bar 3. Click on the result where it says “Lambda” 4. Click on the function that says “verifyKey” 5. Click on “API Gateway” 6. Take a note of the API key and the API endpoint 7. Open up terminal 8. Type the following command: curl -H "x-api-key: 7NYpat2DyO6yqh6EqXSah924XRzVeBi26TEPcwfx" https://cxdp3vrdt6.execute-api.us-east-2.amazonaws.com/dev/verifyKey -d '{"key":"gibrish12345678"}' -v

Expected Results	* Trying 18.224.254.67... * TCP_NODELAY set * Connected to cxdp3vrdt6.execute-api.us-east-2.amazonaws.com (18.224.254.67) port 443 (#0) * TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 * Server certificate: *.execute-api.us-east-2.amazonaws.com * Server certificate: Amazon * Server certificate: Amazon Root CA 1 * Server certificate: Starfield Services Root Certificate Authority - G2 > POST /dev/verifyKey HTTP/1.1 > Host: cxdp3vrdt6.execute-api.us-east-2.amazonaws.com > User-Agent: curl/7.54.0 > Accept: */* > x-api-key: 7NYpat2DyO6yqh6EqXSah924XRzVeBi26TEPcwfx > Content-Length: 25 > Content-Type: application/x-www-form-urlencoded > * upload completely sent off: 25 out of 25 bytes < HTTP/1.1 400 Bad Request < Date: Sun, 25 Nov 2018 19:17:10 GMT < Content-Type: application/json < Content-Length: 13 < Connection: keep-alive < x-amzn-RequestId: b4a7a557-f0e6-11e8-a0ab-3f674975b502 < Access-Control-Allow-Origin: * < x-amz-apigw-id: Q7tAaFPAiYcFZpQ= < X-Amzn-Trace-Id: Root=1-5bfaf535-301c1b041079e20cf56a259e;Sampled=0 * Connection #0 to host cxdp3vrdt6.execute-api.us-east-2.amazonaws.com left intact **"Invalid API"**

Non-Functional Testing

Encryption

Test Case Name	On boot-up, Weather Station generates encrypted weather data
Test Case Id	TC - 37
Priority	High
Pre-Conditions	Weather Station is connected to a power source.
Post-Conditions	Inside of directory “data” encrypted files will be generated every 5 seconds.
Test Steps	1. Open a terminal 2. Enter Cd /path/to/ “dist” directory 3. Count number of Files inside directory 4. Wait 5 seconds and count number of files again 5. Open any text file 6. Verify that the text inside are not in plain text
Expected Results	Every 5 seconds a new data will be generated with time stamp.

Test Case Name	Encryption Data at Rest on Weather Station
Test Case Id	TC - 38
Priority	High
Pre-Conditions	Weather Station is connected to a power source. Auto-start scripts to generate encrypted files has started.
Post-Conditions	Every weather data inside directory “data” will be encrypted with RSA 128.
Test Steps	1. Open a terminal 2. Enter Cd /path/to/ “dist” directory 3. Verify the size of each file generated
Expected Results	Every 5 seconds a new data will be generated with time stamp. All files generated will not be in plaintext and are also 128 bytes.

Test Case Name	Encryption Data from Servant Station to Master Station

Test Case Id	TC - 39
Priority	High
Pre-Conditions	Both Master and Servant must be connected to a power source. Auto start script must be running properly to generate encrypted files at rest for Servant.
Post-Conditions	Master will be able to decrypt using RSA 128 private key.
Test Steps	1. Open terminal and run LoRa sender script on Servant 2. Broadcast encrypted data from directory “data” on Servant using LoRa. 3. Open terminal on Master and run LoRa receiver script 4. Using LoRa write broadcasted data into directory EncryptedFilesRecieved on Master 5. Decrypt each files using RSA 128 private key by running “python pycryptoDecrypt.py”. 6. Verify the decrypted data is the same as the encrypted data before it was encrypted.
Expected Results	Decrypted data on Master will be the same as Encrypted data from Servant.

AWS

Test Case Name	Crash a server for high availability
Test Case Id	TC - 40
Priority	High
Pre-Conditions	TC-41: tester is logged into the AWS account
Post-Conditions	Server status on web page changes from “Server 1” to “Server 2”

Test Steps	1. Open AWS Console 2. Search for Lambda in the search bar 3. Click on the result where it says “Lambda” 4. Click on Create function 5. Insert Name as “Crash-Server” 6. Choose runtime “Python3.6” 7. Select “Create a custom role” 8. Execute AWS-01 script 9. Click on create function 10. Upload the AWS-02 script on Lambda 11. Click on “Test” 12. Provide an Event Name as “TestEvent1” 13. Click on create 14. Now, click on “Test” again.
Expected Results	One of the servers will crash but website will still remain up and running. On the admin page, server status will change from 1 to 2.

Test Case Name	Performance testing Lambda function
Test Case Id	TC - 43
Priority	Medium
Pre-Conditions	TC-41: Tester is logged into the AWS account
Post-Conditions	Report of actual Memory used for lambda function
Test Steps	1. Open AWS Console 2. Search for CloudWatch in the search bar 3. Click on the result where it says “CloudWatch” 4. From the left menu choose “Logs” 5. From the log group click on “aws/lambda/sensordata” 6. Click on the top most log streams that is displayed 7. At the bottom of the execution, the report will be displayed. 8. Note down the Max Memory usage
Expected Results	REPORT RequestId: 746cabab-f0d1-11e8-a2cf-2715b6b0e36a Duration: 9.31 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 24 MB

Test Case Name	Load testing Lambda function
Test Case Id	TC - 44
Priority	Medium
Pre-Conditions	TC-41: Tester is logged into the AWS account
Post-Conditions	Report of actual Duration lambda function ran for
Test Steps	1. Open AWS Console 2. Search for CloudWatch in the search bar 3. Click on the result where it says “CloudWatch” 4. From the left menu choose “Logs” 5. From the log group click on “aws/lambda/sensordata” 6. Click on the top most log streams that is displayed 7. At the bottom of the execution, the report will be displayed. 8. Note down the Duration
Expected Results	REPORT RequestId: 746cabab-f0d1-11e8-a2cf-2715b6b0e36a Duration: 9.31 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 24 MB

Test Case Name	Successful connection with public key- Data in transit encryption test for RDS instance
Test Case Id	TC - 45
Priority	Medium
Pre-Conditions	1. MySQL client is installed on the machine and usable from the terminal 2. Download the Public Key https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem.
Post-Conditions	Successful connection to mysql

Test Steps	1. Open your terminal 2. Type the following: mysql -h weatherstation_database_endpoint_URL --ssl-ca=[full path]/rds-combined-ca-bundle.pem --ssl-mode=VERIFY_IDENTITY -P 3306 -u pro1wsu -p 3. Enter the correct password [wsucapstone2018]
Expected Results	Connected to MySQL

Test Case Name	Unsuccessful connection without key- Data in transit encryption test for RDS instance
Test Case Id	TC - 46
Priority	High
Pre-Conditions	MySQL client is installed on the machine and usable from the terminal.
Post-Conditions	Unsuccessful connection to mysql
Test Steps	1. Open your terminal 2. Type the following: mysql -h weatherstation_database_endpoint_URL -P 3306 -u pro1wsu -p --ssl-mode=DISABLED 3. Enter the correct password [wsucapstone2018]
Expected Results	ERROR 1045 (28000): Access denied

Test Case Name	Verify weather token - Testing API Gateway accessibility with Invalid x-api-key
Test Case Id	TC - 49
Priority	Medium
Pre-Conditions	1. Must have an API Gateway on AWS 2. Hold of correct API Endpoint 3. Curl is installed and usable from Terminal

Post-Conditions	Forbidden
Test Steps	1. Open the terminal 2. Type the following: curl https://your_endpoint.amazonaws.com/dev/verifyKey -d '{"key":"gibrish12345678"}' -v
Expected Results	* Trying 18.220.132.95... * TCP_NODELAY set * Connected to cxdp3vrdt6.execute-api.us-east-2.amazonaws.com (18.220.132.95) port 443 (#0) * TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 * Server certificate: *.execute-api.us-east-2.amazonaws.com * Server certificate: Amazon * Server certificate: Amazon Root CA 1 * Server certificate: Starfield Services Root Certificate Authority - G2 > POST /dev/verifyKey HTTP/1.1 > Host: cxdp3vrdt6.execute-api.us-east-2.amazonaws.com > User-Agent: curl/7.54.0 > Accept: */* > Content-Length: 25 > Content-Type: application/x-www-form-urlencoded > * upload completely sent off: 25 out of 25 bytes < HTTP/1.1 403 Forbidden < Date: Sun, 25 Nov 2018 21:39:13 GMT < Content-Type: application/json < Content-Length: 23 < Connection: keep-alive < x-amzn-RequestId: 8d67cb3a-f0fa-11e8-a938-f9dfae420b05 < x-amzn-ErrorType: ForbiddenException < x-amz-apigw-id: Q8B0THTZCYcFkQA= * Connection #0 to host cxdp3vrdt6.execute-api.us-east-2.amazonaws.com left intact **{"message":"Forbidden"}**

Test Case Name	Post sensor data - Testing API Gateway accessibility with Invalid x-api-key
Test Case Id	TC - 50

Priority	Moderate
Pre-Conditions	1. Must have an API Gateway on AWS 2. Hold of correct API Endpoint 3. Curl is installed and usable from Terminal
Post-Conditions	Forbidden
Test Steps	1. Open the terminal 2. Type the following: curl curl -H "x-api-key: invalidAPIkey" aws_endpoint/post1/sensordata -d "@temp.txt" -v
Expected Results	* Trying 18.220.132.95... * TCP_NODELAY set * Connected to cxdp3vrdt6.execute-api.us-east-2.amazonaws.com (18.220.132.95) port 443 (#0) * TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 * Server certificate: *.execute-api.us-east-2.amazonaws.com * Server certificate: Amazon * Server certificate: Amazon Root CA 1 * Server certificate: Starfield Services Root Certificate Authority - G2 > POST /dev/verifyKey HTTP/1.1 > Host: cxdp3vrdt6.execute-api.us-east-2.amazonaws.com > User-Agent: curl/7.54.0 > Accept: */* > Content-Length: 25 > Content-Type: application/x-www-form-urlencoded > * upload completely sent off: 25 out of 25 bytes < HTTP/1.1 403 Forbidden < Date: Sun, 25 Nov 2018 21:39:13 GMT < Content-Type: application/json < Content-Length: 23 < Connection: keep-alive < x-amzn-RequestId: 8d67cb3a-f0fa-11e8-a938-f9dfae420b05 < x-amzn-ErrorType: ForbiddenException < x-amz-apigw-id: Q8B0THTZCYcFkQA= * Connection #0 to host cxdp3vrdt6.execute-api.us-east-2.amazonaws.com left intact **{"message":"Forbidden"}**

LoRa Communication

Test Case Name	LoRa communication range
Test Case Id	TC - 51
Priority	Medium
Pre-Conditions	At least one master device and one servant device online and at least receiver/sender scripts operational.
Post-Conditions	1. Servant device message is broadcast successfully to master device. 2. No message is received therefore range exceeded or interference occurred.
Test Steps	1. Plug in power source to both devices. 2. Move devices a desired distance away from each other. 3. To verify message is received, check EncryptedFilesReceived folder after test.
Expected Results	1. Servant device is gathering information from external sensors and is saving it into a text file. 2. The information is then broadcasted using LoRa. 3. The master device receives message from servant device successfully. 4. The master device then stores the information to prepare transfer to AWS.

Appendix

Database Scripts

Script ID	Data Description
DBS-01	Empties entire database except for one superuser. username = “testsuper” password=”testing123” email=”weatherstationtest@gmail.com” phone=’13135555555’ reset_token=’d770ddafa1786f620781’
DBS-02	“Connects” a weather station named “Test Station” which sends weather data to the backend every 5 seconds. apikey=”00000000000000000000” (20 zeros)
DBS-03	1. Adds a weather station named “Test Station” with apikey=”00000000000000000000” (20 zeros) 2. Inserts one row of weather data with random numbers into the weather table. 3. Inserts one row into the latest weather table with a foreign key to that weather row.
DBS-04	“Connects” two weather stations named “Test Station” and “Test Station 2” which send weather data to the backend every 5 seconds.
DBS-05	Empties entire database except for one super user and one regular user. username = “testuser” password=”testing123”
DBS-06	Inserts 360 rows of weather data into with random numbers into the sensor_data table that increases in time by 5 seconds for each row for apikey= “00000000000000000000” (20 zeros).

DBS-07	1. Adds a weather station named “Master” with apikey=”00000000000000000001” (19 zeros) 2. Inserts one row of weather data with random numbers into the weather table. 3. Inserts one row into the latest weather table with a foreign key to that weather row.

*Note: Each of these scripts can be completed by creating and inserting into the weather station database created when website was deployed.

AWS Scripts

Script ID	Description
AWS-01	A new role need to be created with AWS Identity Management Access. The role will include Lambda as a trusted entities. This role will allow lambda to make call to EC2 Instances and write logs to AWS Cloudwatch.

AWS-02	import boto3
	import json
	def crash_instance(id):
	  ec2 = boto3.client('ec2')
	  response = ec2.stop_instances(
	  InstanceIds=[
	  id,
	  ]
	  )
	  return response
	def lambda_handler(event, context):
	  filters = [{'Name': 'tag:Name', 'Values': ['MAIN-ELB-Instance']}]
	  response = ec2.describe_instances(Filters=filters)
	  print("EC2 RESPONSE--> ", response)
	  reservation = response['Reservations'][0]
	  instances = reservation['Instances'][0]

	id = instances['InstanceId']
	print("type-->: ", id)
	crash_instance(id)
	
	return {
	
	"headers":{'Content-Type':'application/json','Access-Control-Allow-Origin':'*'},
	"statusCode": 200,
	"body": json.dumps("Instance: "+id+" is being killed!")
	}