IoT Plant Monitoring System

Bailey Dalton

20141483

Evan Strange

20112566

IoT Plant Monitoring System

Submitted to:

Dr. Branko Cirovic

Capstone Instructor

College of the North Atlantic, Ridge Road Campus

Ms. Donna Meade

Communications Instructor

College of the North Atlantic, Ridge Road Campus

Submitted by:

Bailey Dalton

20141483

Evan Strange

20112566

Submitted on:

August 07, 2019

flowr

1493 Torbay Road, Torbay, NL A1K 1G7
(709)765-2473
Evan.Strange66@ed.cna.nl.ca

August 7, 2019

Ms. Donna Meade
Communications Instructor
College of the North Atlantic
Engineering Technology Center
153 Ridge Road, St. John's, NL A1C 6L8

Dear Ms. Meade,

I am pleased to submit the enclosed report, which details the IoT Plant Monitoring System my partner and I designed for our Capstone Project 2019.

This report will discuss the process of creating an Internet of Things application, with the purpose of monitoring a garden remotely. It will cover the tools used to design the application, the costs associated with the finished product, and the challenges that were faced along the way. Wiring diagrams, code snippets, and screenshots of the front-end development will be featured throughout the report.

If you have any questions about the contents of this report, or the IoT garden monitoring system, please feel free to contact me by phone or by email.

Sincerely,

Evan Strange

Bailey Dalton

Enclosure: Capstone Report

EXECUTIVE SUMMARY

This report will detail the process of developing an Internet of Things plant monitoring system. This project encapsulates every aspect of Computing Systems Engineering Technology, from electrotechnology, to microcontrollers, to web development.

This product will allow the user to remotely monitor the status of their garden using a web-based application. The web app will display the temperature, humidity, light level, and soil moisture content in tabular and graphical form, with an easy-to-understand user interface.

This report will discuss the technology stack used to develop this product (React, AWS IoT, Lambda, DynamoDB, Node.JS). It walks the reader through the flow of data, from the sensors to the web app. It details the challenges the team faced when choosing the correct components, learning new technologies to better suit the project, dealing with deadlines, and much more.

Throughout this process, the two students were able to make several recommendations regarding the hardware used. These recommendations are mainly aimed at cutting costs and making the development more straight-forward.

Creating this product was very exciting, and both students thoroughly enjoyed putting their newfound knowledge to the test. The skills they learned throughout the program enabled them to work on a project that covers many different facets of computing.

ACKNOWLEDGEMENTS

This project was made possible by the knowledge the team gathered during their time in Computing Systems Engineering Technology, under the guidance of Dr. Branko Cirovic. We would like to thank Dr. Cirovic for teaching us vital building blocks of computing, so that we can easily learn other technologies, such as the ones highlighted in this report.

I would like to also thank Mr. Patrick Murphy for his knowledge of microcontrollers and steering the team in the right direction regarding sensors and modules for the Arduino microcontroller.

TABLE OF CONTENTS

LIST OF ILUSTRATIONS

1.0 INTRODUCTION

This introduction section is to provide the reader with the background information necessary to read and properly understand the report. This section will cover the purpose of the report, what will be covered throughout the report, the methodology we used to create this product, and some background about why we decided to create this product.

1.1 Purpose

The purpose of this project was to encapsulate a bit of everything learned throughout this 3-year program. An Internet of Things plant monitoring system incorporates everything from electrotechnology, microcontrollers, networking, and web development.

1.2 Scope

At first, the product was more tailored toward the newly-legalized cannabis industry. The goal was to enable the user to easily cultivate a cannabis plant, without having to monitor it full-time.

The team realized that this was severely inhibiting the products userbase, and that the same technology could be used for growing any plants. This resulted in a pivot to a more generalized product, targeting anyone who has an interest in gardening.

This report will discuss the importance of Internet of Things, the process of wiring together the sensors and modules we chose for this project, creating an intuitive React web application to display the readings from the sensors, and the flow of data from the sensors to the web application.

The team wanted to keep costs relatively low, therefore inexpensive hardware was selected, along with open-source technology. The team decided against automated watering and implementing a fan to lower temperature, as another cost-cutting measure.

1.3 Methodology

Most of the skills required to complete this project were learned during this 3-year Computing Systems Engineering Technology program. This includes, but is not limited to courses regarding electrotechnology, networking, web development, and microcontrollers.

The team had shared experience on their two work terms working with various startups, familiarizing themselves with JavaScript, and one of the team members worked with Mysa assisting in developing IoT smart thermostats.

The team purchased and completed Udemy courses specializing in React, in order to further their knowledge of JavaScript, and implement a library that would assist with creating a web-based application that looked good and was intuitive.

In order to configure the circuit, and properly wire the sensors up to the Arduino microcontroller, the team consulted various sources on the Internet, mostly the Arduino Project Hub.

2.0 BACKGROUND

The reason for developing this project is that the team wanted to build on the foundation that they had been taught throughout their 3-year program and showcase their ability to tie all of these technologies together into a functional product.

Most people like to have plants around their living space. Whether it be a garden outside, or a desk-plant. This product, in its current form, is most effective away from the elements of nature. Its best application would be in a greenhouse, or indoors. This product would be a great gift or project for a tech-savvy person who enjoys having some greenery around.

As mentioned before, initially the idea spawned from the recent legalization of cannabis, and the ability to grow up to four plants in your home. The team's interest in IoT lead them to this product idea. After a lot of thought, the team realized the best direction for a product of this nature, was to appeal to a more general audience. The sensors used in this project are vital for the life of virtually every plant.

The technology stack used to develop this project is fairly cutting-edge. React and Node.js are two open-source technologies that utilize JavaScript. React is a library for building user interfaces, developed by Facebook. Node.js is a run-time environment that executes JavaScript outside of a browser. These two technologies are extremely popular in the startup industry, due to their free, open-source nature. Also, their ability to get projects off the ground very quickly and the fact that the front and back-end are written in JavaScript is hugely attractive to a lot of up-and-coming developers.

The team implemented some Amazon Web Services, as those are gaining a lot of popularity for having no upfront investment, being very inexpensive and offering pay-as-you-go pricing. They are using AWS IoT, a platform that enables the user to connect devices to other Amazon Web Services, and to other devices. As well as secure data and interactions, process device data, and enable applications to interact with devices even if they are offline. (Amazon Web Services, Inc., 2019)

The team is also using Amazon DynamoDB with AWS Lambda. DynamoDB is a key-value and document NoSQL database that is highly scalable and very fast. It is popular due to having built-in security and being multiregional. Lambda is a Service that lets you run code without managing servers. The user pats for the compute time they consume, and there is no charge when your code is not running. This allows the user to easily run code for any type of application or backend service. The report will further detail how these technologies work together to get the data from the sensors on the microcontroller, to displaying in tabular and graphical form on the web-based application.

3.0 INTERNET OF THINGS

Internet of Things is the extension of internet connectivity into physical devices and everyday objects. These devices are able to communicate and interact with other devices, via the internet. They are also able to be remotely monitored and controlled.

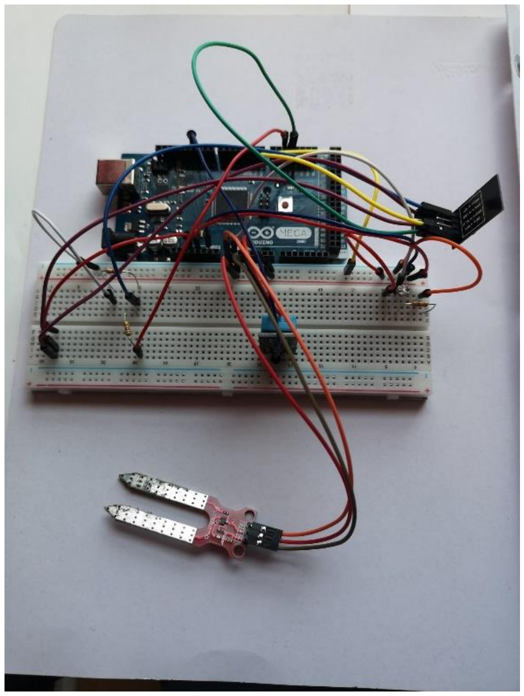
The definition of IoT has evolved in recent years due to the convergence of multiple technologies, real-time analytics, machine learning, etc.

Traditional fields of embedded systems, such as wireless sensor networks, control systems, automation, etc. all contribute to enabling the Internet of Things.

In the consumer market, IoT technology is synonymous with products pertaining to the "smart home" concept. This includes smart lighting fixtures, that turn on when you enter the room, and turn off after the room has been vacant for a period of time. Smart thermostats, which monitor what hours of the day the house is occupied, and adjust the temperature accordingly, in order to maximize efficiency, saving on power costs. There are also all sorts of kitchen appliances, home security devices, etc. Usually these devices can be manipulated or monitored using a smartphone or browser-based application.

4.0 HARDWARE

This section will detail the hardware used for this product. It will discuss the microcontroller and the different sensors that were used to read and output data. Below is a picture of the entire setup, the Arduino Mega 2560 with all 3 sensor modules and WiFi module connected.



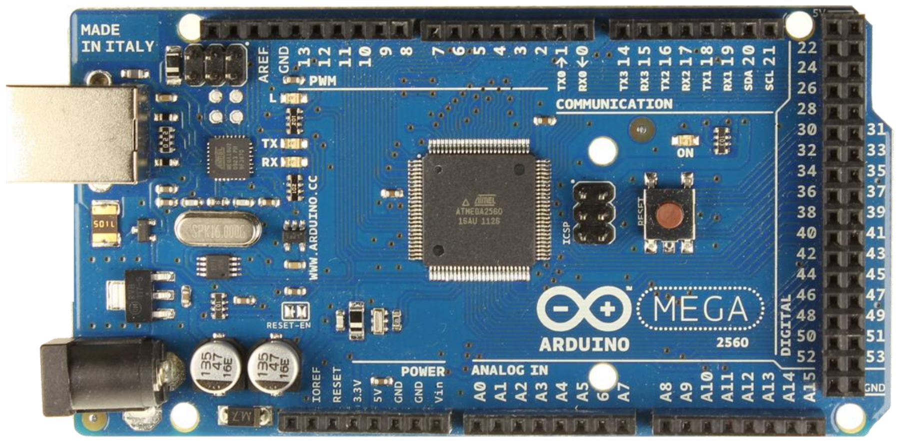
4.1 Arduino Mega 2560 Microcontroller

The Arduino Mega is designed for projects that require more Input/Output lines, more sketch memory and more RAM than its more inexpensive counterparts. This was the microcontroller that was required for DP3200 Embedded Controller Applications.

While it is much more capable and expensive than what was needed for this product, the team had already purchased two of these units for that course, so it was the least expensive option in this case.

A more cost-effective board would be the Arduino UNO, and it is just as capable for this specific application. The cost of an Uno is $14 CAD compared to the $115 CAD paid for the Arduino Mega 2560 kit at the CNA bookstore.

If cost was not an issue whatsoever, the likely candidate would be the Raspberry Pi 3, as it boasts built in Wi-fi, better processing power, and the convenience of a Linux operating system.



4.2 EK1361 Soil Moisture Sensor

The soil moisture sensor consists of two probes that measure the volume of water in the soil. The two probes allow the electric current to pass through the soil and, according to its resistance, measures the moisture level of the soil.

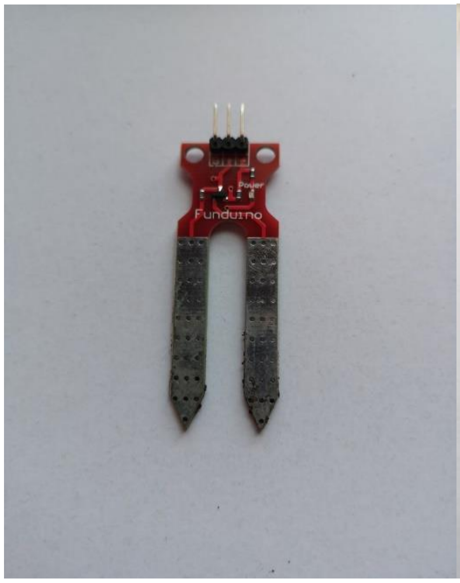
When there is more water, the soil conducts more electricity, which means that the resistance will be less, therefore the moisture level will be higher. Dry soil reduces conductivity. So, when there is less water, the soil conducts less electricity, which means it has more resistance. So the moisture level will be lower.

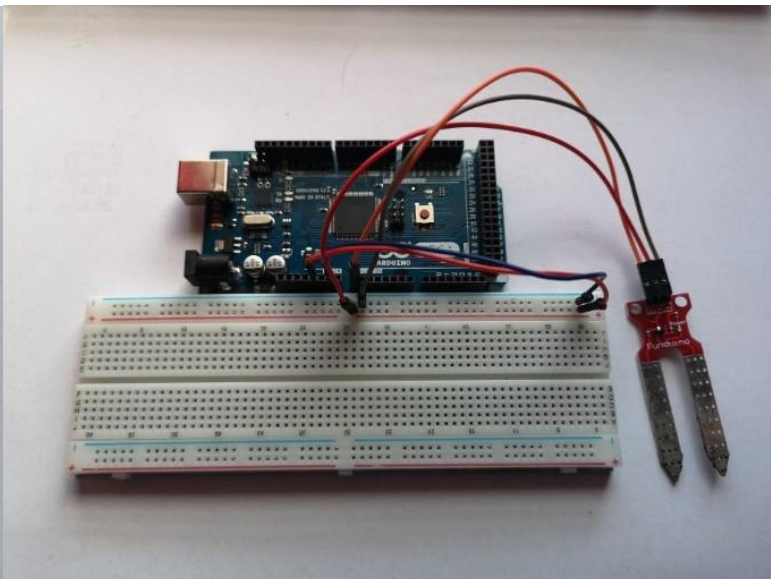
ElectroPeak (2019, April 12) *Complete Guide to Use Soil Moisture Sensor w/ Examples.* Retrieved from https://create.arduino.cc/projecthub/electropeak/complete-guide-to-use-soil-moisture-sensor-w-examples-756b1f

These sensors have a very low power requirement, and a high resolution, meaning that it could be ran off a battery for a long period of time. For this project, the Arduino will be infinitely powering the sensor, by being connected via USB through the serial port.

One thing to note about the soil moisture sensor is some units has been known to be prone to degradation due to excess moisture potentially damaging the component. However, during this project the team did not note damage to the unit during testing.

The team chose the GikFun EK1361 because they were readily available online from a North American vendor, and were quite inexpensive. A pack of 3 on Amazon.ca was a little less than $10.



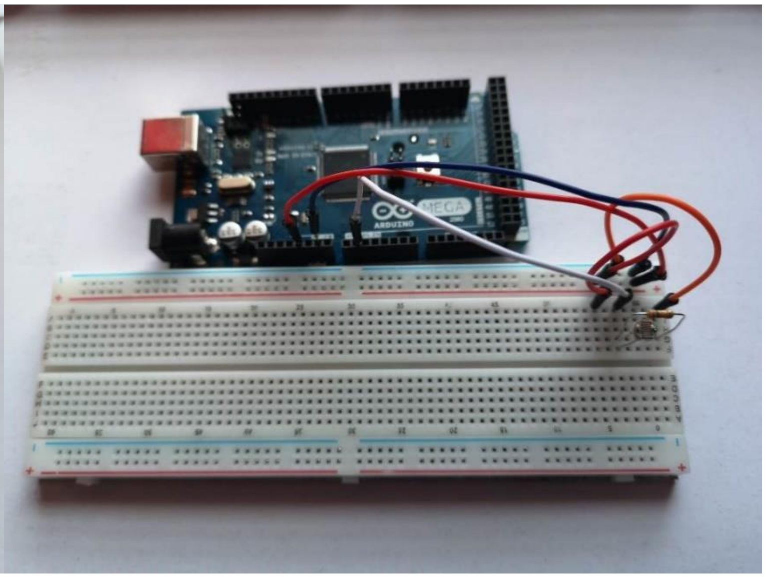


4.3 GL5516 Photoresistor

A photoresistor, or light-dependent resistor (LDR) is a light-controlled variable resistor. The resistance of the photoresistor decreases with increasing incident light intensity. This allows the user to easily measure the value of incident light intensity. It is made of a high resistance semiconductor.

These photoresistors can be used in many DIY projects pertaining to light, such as alarm devices, nightlights, solar lamps, etc.

This particular photoresistor is quite popular due to its availability and inexpensive nature. A pack of 20 on Amazon.ca was a little less than $10.

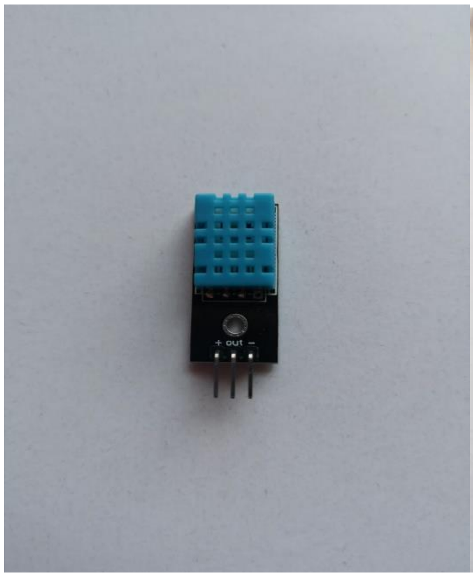
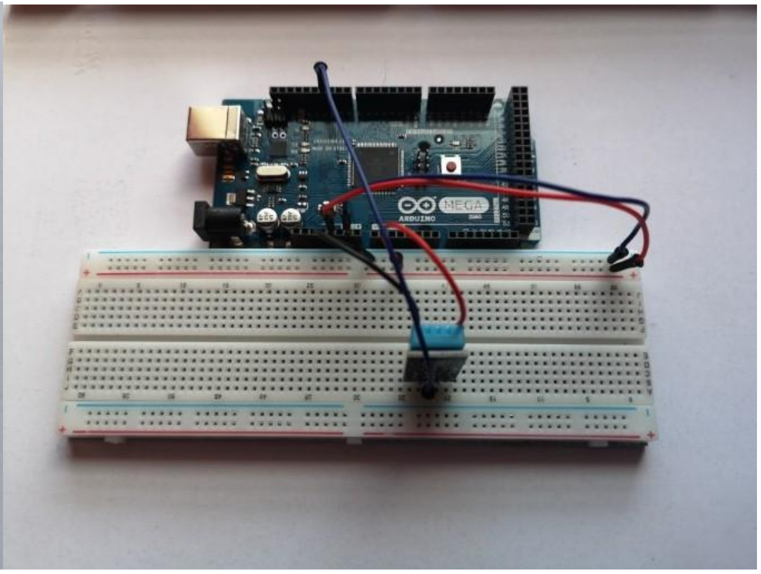


4.4 DHT11 Temperature and Relative Humidity Sensor

These sensors are made up of two parts – a capacitive humidity sensor, and thermistor. Combined with a very basic chip inside that does some analog to digital conversion and spits out a digital signal with the temperature and humidity.

This sensor connects to 5V VCC, ground and has a digital out pin that is connected to a digital pin on the Arduino. It has a sample rate of 1Hz, meaning data is sent out of the digital pin every 1 second. It can read a humidity range of 20% to 80% with an accuracy of 5%, and a temperature of 0-50 °C within ±2 degrees.

The DHT11 was selected due to its digital out simplifying the data retrieval and its inclusion in the Arduino Mega kits, saving on cost for the team.

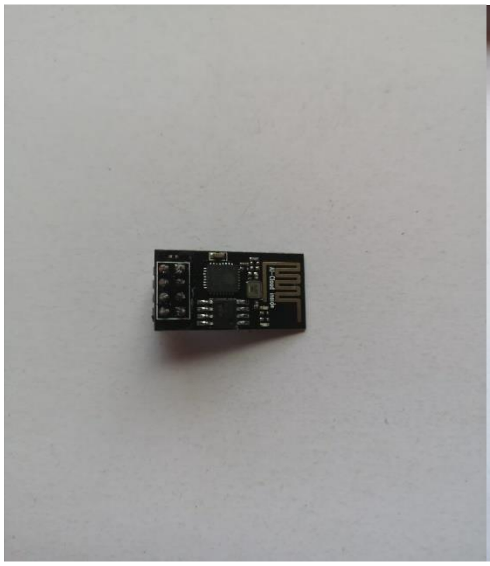



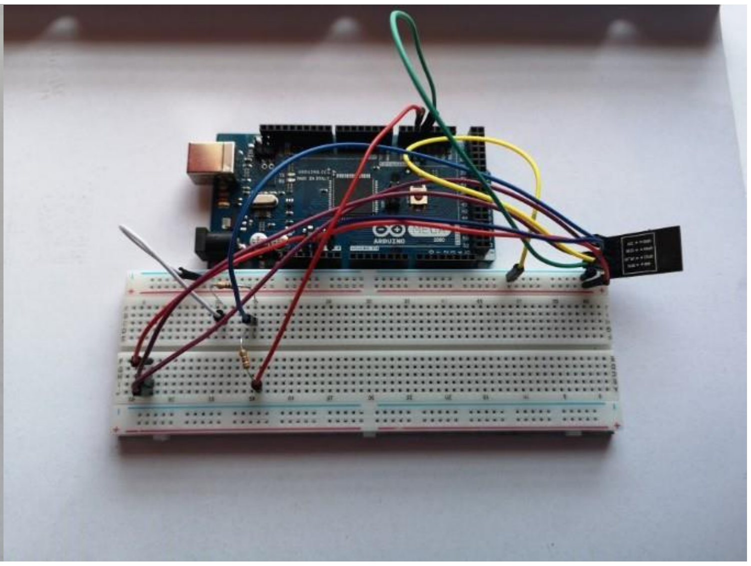
4.5 ESP8266 WiFi Module

The ESP8266 is a really useful, cheap WiFi module for controlling devices over the Internet. It comes with factory installed firmware that allows the user to control it with standard “AT commands” or you can create and upload your own code, which makes this inexpensive module very powerful and flexible for developing IoT products and solutions.

This chip is addressable, and can be configured as a web server. It can connect to your router of course, and work as a client or be configured as an access point.

This board has a huge community, due to its price and capability. This leads to an overwhelming amount of information online. A lot of this information is difficult to sort through, as there are several variations of the same chip, and some of the information is outdated, or incorrect.



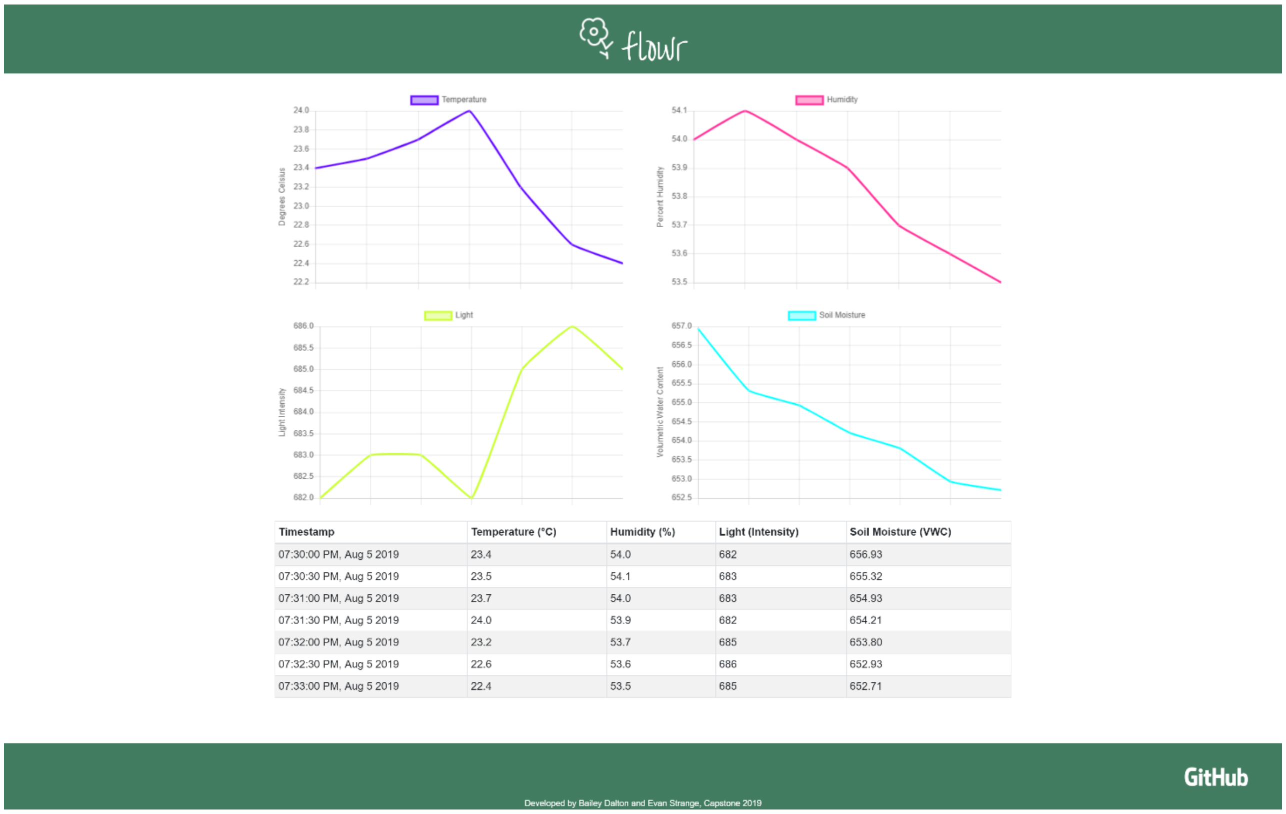


5.0 SOFTWARE

This section will discuss the software the team developed in order to transmit the data from the microcontroller and sensors to a database and web application.

5.1 Front End

The front end of the project is the visible side of the web application, or the part that the user sees and interacts with.



Timestamp	Temperature (°C)	Humidity (%)	Light (Intensity)	Soil Moisture (VWC)
07:30:00 PM, Aug 5 2019	23.4	54.0	682	656.93
07:30:30 PM, Aug 5 2019	23.5	54.1	683	655.32
07:31:00 PM, Aug 5 2019	23.7	54.0	683	654.93
07:31:30 PM, Aug 5 2019	24.0	53.9	682	654.21
07:32:00 PM, Aug 5 2019	23.2	53.7	685	653.80
07:32:30 PM, Aug 5 2019	22.6	53.6	686	652.93
07:33:00 PM, Aug 5 2019	22.4	53.5	685	652.71

Developed by Bailey Dalton and Evan Strange, Capstone 2019

5.1.1 React

React is an open-source JavaScript library tailored toward developing user interfaces, particularly for single-page web applications. It was developed by Facebook and is well-known for its ability to optimally fetch rapidly changing data that needs to be recorded.

5.1.2 Chart.js

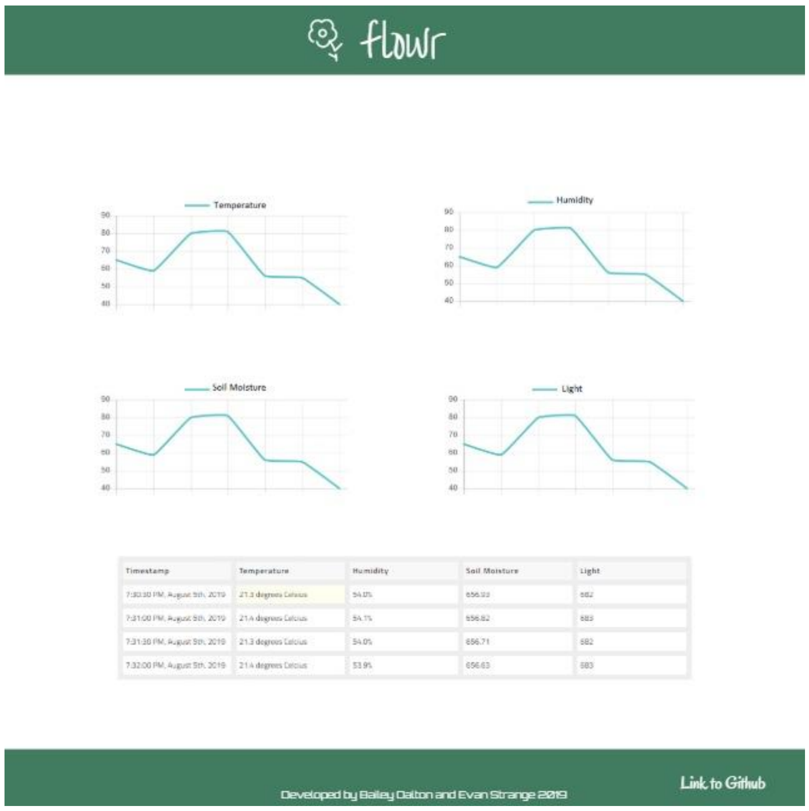
Chart.js is an open-source, community-maintained project. It allows the user to easily create responsive data visualization charts using JavaScript. These charts are rendered in HTML5 for compatibility across all modern browsers.

The team used a tool called React-Chartjs-2, which is a wrapper that makes creating these charts with React even more straight-forward.

5.1.3 HTML

HTML or Hypertext Markup Language is the standard markup language used for creating web pages or web applications. It has been around since 1993. Simply put, when creating a website or web application, HTML is the barebones layout of the website, such as where the header bar or graphs will be. Tools like JavaScript (And libraries such as React) enable the user to breathe life into these components, giving them animations, making them easily reusable, or eliminating the need to tediously style each element.

When creating this web application, the team first used an online tool called Vectr to create a simple storyboard mockup of how the website would look in the initial design.



Timestamp	Temperature	Humidity	Soil Moisture	Light
7:30:30 PM, August 5th, 2019	21.3 degrees Celcius	54.0%	656.93	682
7:31:00 PM, August 5th, 2019	21.4 degrees Celcius	54.1%	656.82	683
7:31:30 PM, August 5th, 2019	21.3 degrees Celcius	54.0%	656.71	682
7:32:00 PM, August 5th, 2019	21.4 degrees Celcius	53.9%	656.63	683

The next step was to use HTML to create the barebone layout of this page, including the header and footer elements. The charts were then created using Chart.js, and the table was created using React-Bootstrap.

5.1.4 CSS

CSS or Cascading Style Sheets was released in 1996, this language is used for describing the presentation of a document written in a markup language (HTML in our case).

This is how we are able to style websites to look exactly how we want them. Using CSS, we can change how just about any property of any attribute looks (color, size, font, etc.).

In the previous section React-Bootstrap was mentioned. Bootstrap is by far the most popular front-end component library, used to make styling elements easier. The React-Bootstrap framework is a replacement for the Bootstrap JavaScript. Each component was rebuilt from scratch as a true React component, without unnecessary dependencies such as jQuery.

5.2 Back End

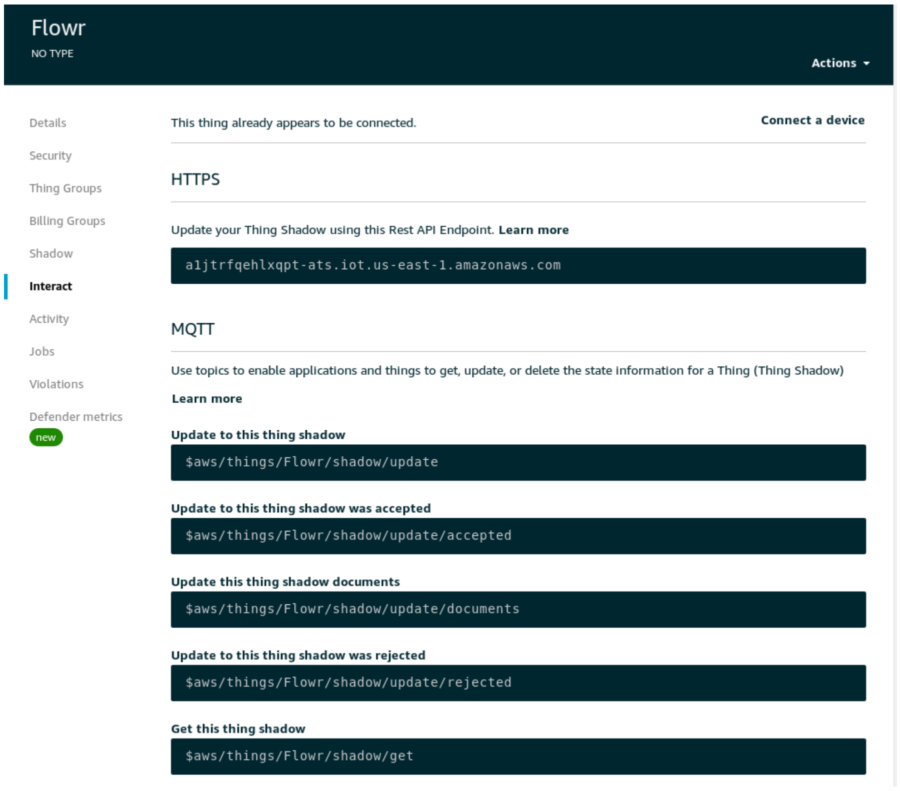
The back end or “server side” of the project is how the web application gets the data from the microcontroller to a database and vice-versa, as well as updates and changes.

5.2.1 AWS IoT

Amazon Web services IoT core allows endpoint devices to interact easily with a cloud infrastructure. It supports HTTP, Websockets, and MQTT communication protocols. It facilitates simple management of devices from the cloud.

AWS IoT Core is one of the main reasons Amazon Web Services was selected as a backend framework. It provides a gateway for the data being collected on the device to be able to be integrated easily into the cloud. Device Endpoints in IoT core are referred to as “things”.

From the cloud console, an IoT “thing” was assigned to represent the device. A “thing shadow” is a part of an IoT thing that is a JSON representation of the most recent state of the device. The thing shadow can be updated or read using MQTT communication. This interacts well with the rest of the backend stack, as the state of the thing shadow can be updated to represent the sensor readings gathered from the device, as well as retrieved when said update happens to store those readings into the database.



This thing already appears to be connected.

HTTPS

Update your Thing Shadow using this Rest API Endpoint. **Learn more**

```
a1jtrfqehlxqpt-ats.iot.us-east-1.amazonaws.com
```

MQTT

Use topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow) **Learn more**

Update to this thing shadow

```
$aws/things/Flowr/shadow/update
```

Update to this thing shadow was accepted

```
$aws/things/Flowr/shadow/update/accepted
```

Update this thing shadow documents

```
$aws/things/Flowr/shadow/update/documents
```

Update to this thing shadow was rejected

```
$aws/things/Flowr/shadow/update/rejected
```

Get this thing shadow

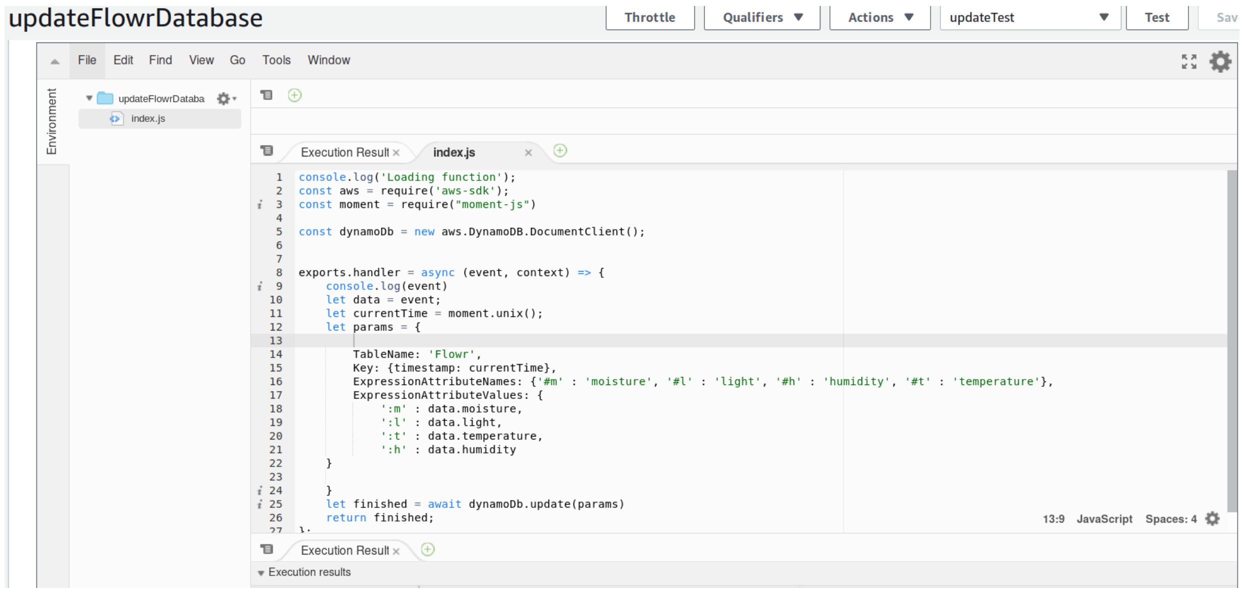
```
$aws/things/Flowr/shadow/get
```

5.2.2 AWS Lambda

AWS Lambda is a computing framework that allows for code to be run without a server. It is billed per millisecond of compute time, making it extremely cost efficient for code that only needs to run occasionally. Lambda functions also scale incredibly well, as each instance of serverless code running is provisioned automatically and separate from the rest of the invocations.

Lambda was selected for this project primarily because of its ability to integrate well with the other cloud services used. With Lambda, it is possible to trigger functions that read or store data to and from the database based on the state of other cloud services. For example, when the device sends data to AWS IoT Core, a lambda function triggers, takes the data, processes it and stores it appropriately in the database.

Secondary to integration is the cost effectiveness, as it would not make sense to pay for an entire server just to run a couple functions every half hour. Additionally, the compute time for this project was well under the threshold for the AWS Free tier, making the service effectively free.

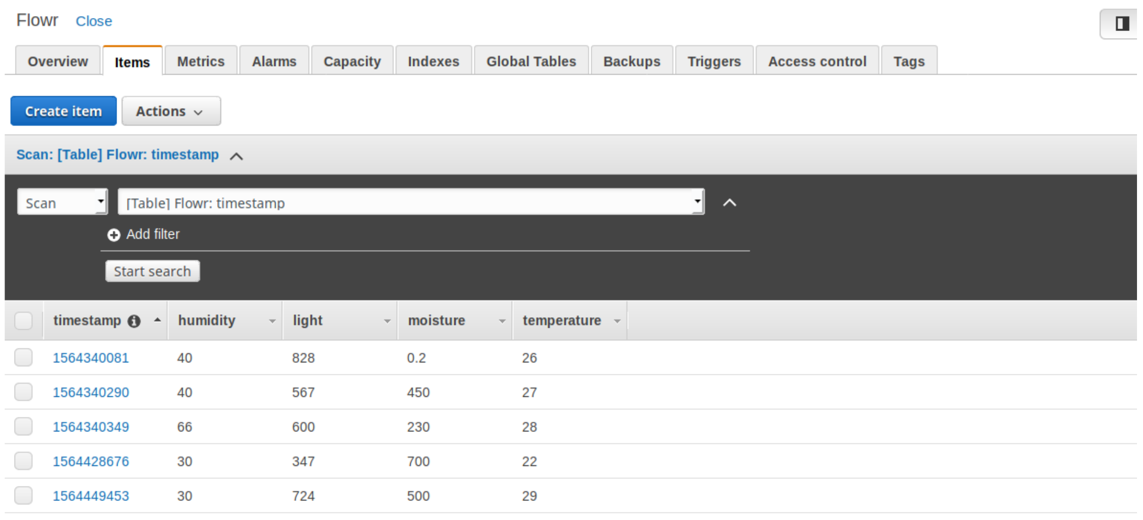


```javascript
console.log('Loading function');
const aws = require('aws-sdk');
const moment = require("moment-js")

const dynamoDb = new aws.DynamoDB.DocumentClient();


exports.handler = async (event, context) => {
    console.log(event)
    let data = event;
    let currentTime = moment.unix();
    let params = {

        TableName: 'Flowr',
        Key: {timestamp: currentTime},
        ExpressionAttributeNames: {'#m' : 'moisture', '#l' : 'light', '#h' : 'humidity', '#t' : 'temperature'},
        ExpressionAttributeValues: {
            ':m' : data.moisture,
            ':l' : data.light,
            ':t' : data.temperature,
            ':h' : data.humidity
    }

    }
    let finished = await dynamoDb.update(params)
    return finished;
};
```

5.2.3 Amazon DynamoDB

DynamoDB is a NoSQL database that relies on a partition key for entries. It scales automatically with use, and is entirely serverless. Entries are stored in a DynamoDB specific JSON schema that allows for simple create, remove, update and delete functionality to be developed.

DynamoDB was selected as it is a very straightforward and simple database to interact with. Each item in our table design has a timestamp as the partition key, to allow for chronological sorting of the readings. This allows other functions to retrieve items for a specific timeframe if it needs to. The timestamps are stored in Unix epoch time (The number of seconds elapsed since January 1st, 1970). Along with the timestamp are numerical properties for the readings gathered by the device, those being Humidity, Light, Soil Moisture, and Temperature. The table can be sorted based on any of these numerical properties, which allows the viewing of Minimum and Maximum readings for any property.



timestamp	humidity	light	moisture	temperature
1564340081	40	828	0.2	26
1564340290	40	567	450	27
1564340349	66	600	230	28
1564428676	30	347	700	22
1564449453	30	724	500	29

5.2.4 MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol that uses a publish-subscribe message pattern. Clients connect to a messaging "broker" and either publish or subscribe to a messaging topic. The maximum size for an MQTT message payload is roughly 268

Megabytes.

MQTT was selected over other protocols (such as HTTP) because of its lightweight footprint. The ESP8266 WiFi module is not an incredibly powerful microcontroller, thus the most conservative and efficient protocol is ideal. MQTT is also supported natively by AWS IoT core, and is the most widely used form of communication to and from IoT devices.

5.2.5 Node.js

Node.js is a JavaScript environment that allows for development of server-side scripts easily in JavaScript. It includes a package manager in the form of NPM that can be used to easily install and manage dependencies. It works synchronously as well as asynchronously in the form of async/await and promises.

Node was chosen for backend development for a handful of reasons. Primarily, it is a highly supported language when it comes to developing and working with Amazon Web Services, as the documentation is extremely extensive for node.

Node works well with the front end as well. React dependencies can be installed with NPM, and Node scripts can be triggered from React JSX.

5.2.6 Flow of Data

First we begin with the flow of data from hardware readings to database storage. The 4 sensors attached to the Arduino Mega collect data that is interpreted by the code running on the Arduino. This data is then passed every 30 minutes to the ESP8266 WiFi module, where the data is published as a payload to the MQTT topic representing the device's thing shadow update function in the cloud.

Once the thing shadow has been updated with data originating from the device, a Lambda function called addReading.js is triggered using the built-in triggers in IoT Core. addReading.js gets the thing shadow's current state in the event object, generates a Unix timestamp for the current time, parses it into a storable format, and then adds an item to the database with the timestamp as the key and the reading data as its properties.

On the front end is a minimal React app with two major components. The first major component displays the latest readings from the database in a tabular format. The second component displays the last 24 hours of historical reading data as a graph. Every 30 minutes the React components will update to reflect new data coming into the database. This is accomplished using a Node.js script called getReadings.js.

getReadings.js uses AWS provided authentication in the form of an access key and a secret access key. It uses the DynamoDB.DocumentClient part of the aws-sdk to perform database operations. The operation consists of a database scan, with the FilterExpression parameter set to filter on key "timestamp", where the timestamp is equal to or greater than the current Unix epoch time minus twenty-four hours in seconds. The returned item set is 48 individual readings that are parsed and formatted into a graph component. The latest reading is the first item in the set, and that is used to generate the latest reading component.

6.0 CHALLENGES

This section will describe the challenges the team faced during the development of this product. Namely the hurdles involved with learning new technologies while trying to meet deadlines and produce deliverables, and the challenge of choosing the right hardware for the job.

6.1 Research Methods

During the production of this product, team members took it upon themselves to buy Udemy courses in order to further their knowledge of JavaScript and React.

This was ultimately for the benefit of the project, but the courses were upwards of 40 hours each, so a lot of time had to be allotted to complete this project. It begs the question whether it was wise to complete these courses during development or after graduation.

6.2 Choosing Components

Although these components were procured from Amazon.ca, many of them still shipped from China. This resulted in longer-than-usual shipping times. Had we not been pleased with the performance of the product it could have set the project back weeks waiting on a replacement.

Likewise, if we had ordered one singular component, and it was to fail or malfunction, we would have to wait for another.

The WiFi adapter we chose, due to the massive cult-following and variations of this same chip, the internet is flooded with information on how to configure it for proper use. Some of this information is extremely convoluted, if not outright incorrect. Some users have reported it taking up to a week of fiddling around with the device to get it to perform properly.

This resulted in us having to develop the front end and charts using dummy data, until we could resolve the issue with the WiFi module to be able to actually access the live readings.

7.0 CONCLUSIONS

In conclusion this project encapsulated almost everything we have learned in our program. Creating an IoT plant monitoring system was a great project to solidify the concepts of wiring up a circuit, understanding a microcontroller, passing data across a network and creating a web application.

This product monitors the soil moisture, temperature, humidity and light intensity of your indoor garden and sends that data across a network using a Wi-Fi module. This data is passed to a DynamoDB using AWT IoT and Lambda before being displayed on a React web application.

8.0 RECCOMENDATIONS

The main recommendation that stands out with this project is the ability to use a much more inexpensive microcontroller. As highlighted above, the Arduino Mega kit was procured at the CNA bookstore for $115, and saved the team about $5 on a DHT11 sensor.

For that price, You could buy 5 Arduino Unos and the required sensors, and create 5 of these products that function identically. There are even cheaper alternatives that are available for only a few dollars, but these are sold through Chinese vendors so they usually take weeks-months to get here and are often unreliable. The Mega 2560 is designed for larger scale projects that require many Input/Output pins, and therefore is definitely overkill for this particular project.

LIST OF REFERANCES

Amazon Web Services, Inc. (2019). *AWS IoT Core Features - Amazon Web Services*. Retrieved from: https://aws.amazon.com/iot-core/features/

ElectroPeak (2019, April 12) *Complete Guide to Use Soil Moisture Sensor w/ Examples.* Retrieved from https://create.arduino.cc/projecthub/electropeak/complete-guide-to-use-soil-moisture-sensor-w-examples-756b1f

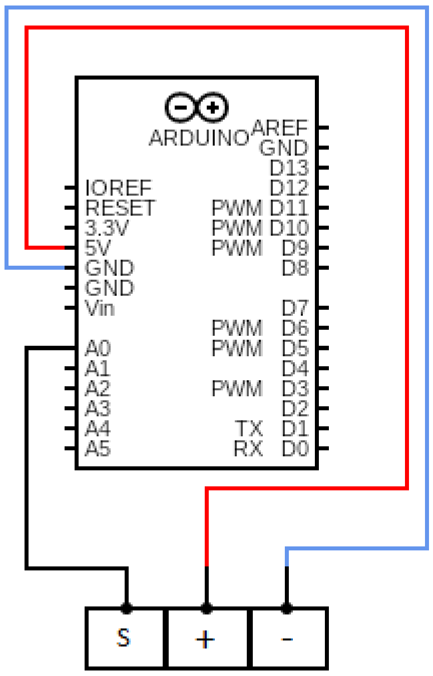
Facebook Inc. (2019) *React – A JavaScript library for building user interfaces.* Retrieved from https://reactjs.org/

Chart.js (2019, August 5) *Chart.js.* Retrieved from https://github.com/chartjs/Chart.js

Jeremy Ayerst (2019, March 28) *React-Chartjs-2*. Retrieved from https://github.com/jerairrest/react-chartjs-2

Appendix A

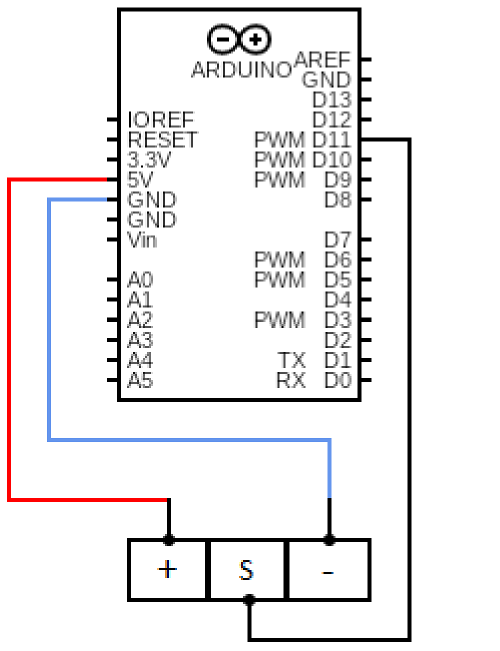
Soil Moisture Circuit



EK1361 Soil Moisture Sensor

Appendix A

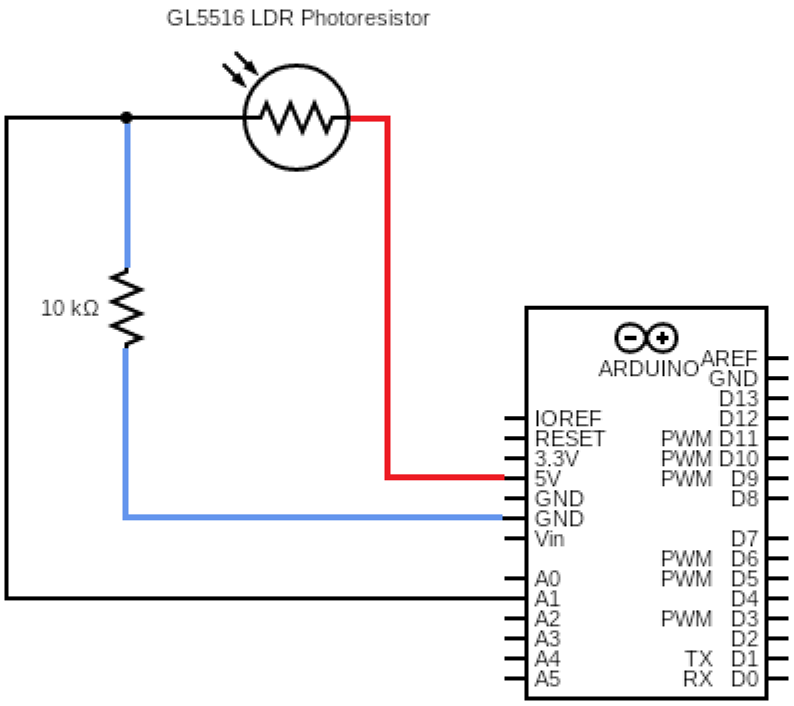
Temperature/Humidity Circuit



DHT11 Temperature/Humidity Sensor

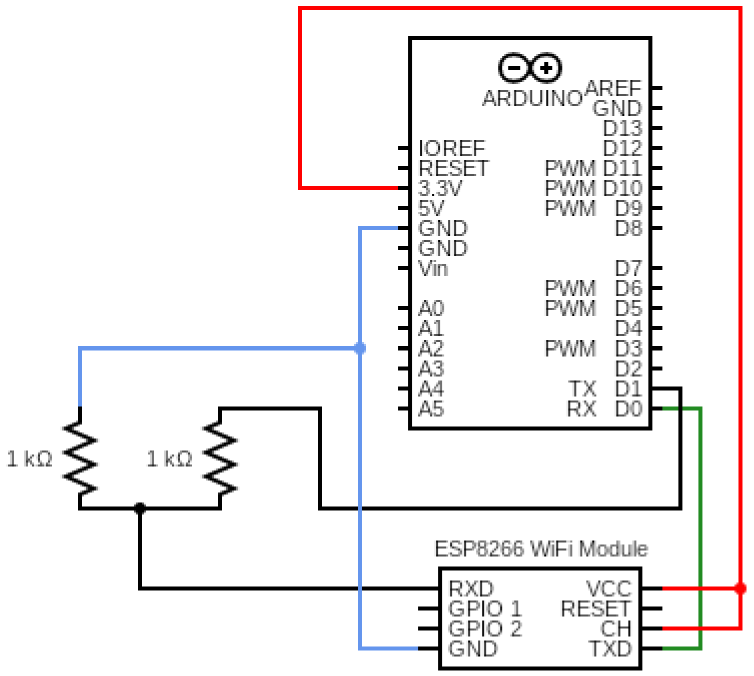
Appendix A

Photoresistor Circuit



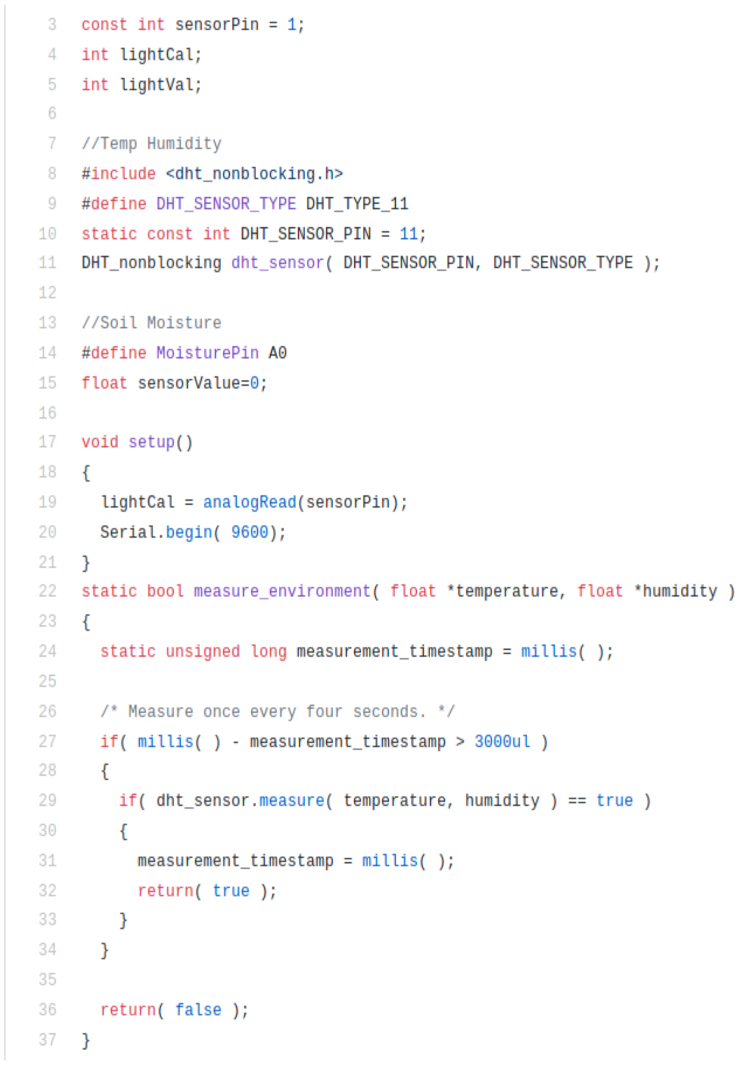
Appendix A

ESP8266 Circuit



Appendix B

Arduino Sketch



```
const int sensorPin = 1;
int lightCal;
int lightVal;

//Temp Humidity
#include <dht_nonblocking.h>
#define DHT_SENSOR_TYPE DHT_TYPE_11
static const int DHT_SENSOR_PIN = 11;
DHT_nonblocking dht_sensor( DHT_SENSOR_PIN, DHT_SENSOR_TYPE );

//Soil Moisture
#define MoisturePin A0
float sensorValue=0;

void setup()
{
  lightCal = analogRead(sensorPin);
  Serial.begin( 9600);
}
static bool measure_environment( float *temperature, float *humidity )
{
  static unsigned long measurement_timestamp = millis( );

  /* Measure once every four seconds. */
  if( millis( ) - measurement_timestamp > 3000ul )
  {
    if( dht_sensor.measure( temperature, humidity ) == true )
    {
      measurement_timestamp = millis( );
      return( true );
    }
  }

  return( false );
}
```

```
int soilMoisture(){
   for (int i = 0; i <= 100; i++)
 {
   sensorValue = sensorValue + analogRead(MoisturePin);
   delay(1);
 }
 sensorValue = sensorValue/100.0;
 Serial.println(sensorValue);
 delay(30);
}

void loop(){
  //Light/Photoresistor
  lightVal = analogRead(sensorPin);


  //Humidity/Temp
  float temperature;
  float humidity;

  /* Measure temperature and humidity.  If the functions returns
     true, then a measurement is available. */
  if( measure_environment( &temperature, &humidity ) == true )
  {
    Serial.print( "T = " );
    Serial.print( temperature, 1 );
    Serial.print( " deg. C, H = " );
    Serial.print( humidity, 1 );
    Serial.println( "%" );
    Serial.print("Light Value: ");
    Serial.println(lightVal);
    Serial.print("Soil Moisture Level: ");
    soilMoisture();
  }

}
```