# Plant.tv

CISC340 - Project #2 Report

TEAM 14
Tiffany Chan - 10181522 - tiffany.chan@queensu.ca
Garett MacGowan - 10197107 - 15gm1@queensu.ca
Michael Alarcon - 10172841 - 14ma61@queensu.ca
Quentin Petraroia - 10145835 - 13qp2@queensu.ca
Ryan Rossiter - 10177467 - 14rcr3@queensu.ca
Tyler Gawalewicz - 10135796 - 13tg19@queensu.ca

## Table of Contents

## Introduction

Plant.tv is Team 14's project that consists of Pete the Plant, who sends live updates to a web browser about the current temperature conditions he is in, and his feelings. The purpose of this project is to provide the team with a hands-on experience to apply the skills and knowledge learned from the course in relation to Internet of Things devices, hardware, and software. We achieved this purpose by collecting data, and had it processed accordingly through our written software. Our software will then output the temperature and his mood. The IOT device used for Plant.tv was Adafruit's Feather Huzzah which worked coincide with the LM35 Linear Temperature Sensor to obtain the plant's environment conditions. Once it retrieved the data collected from the sensor, the Huzzah would send the data to our cloud service platform which was Heroku. From there, the data is processed and information is given through a web browser which was our end data application.

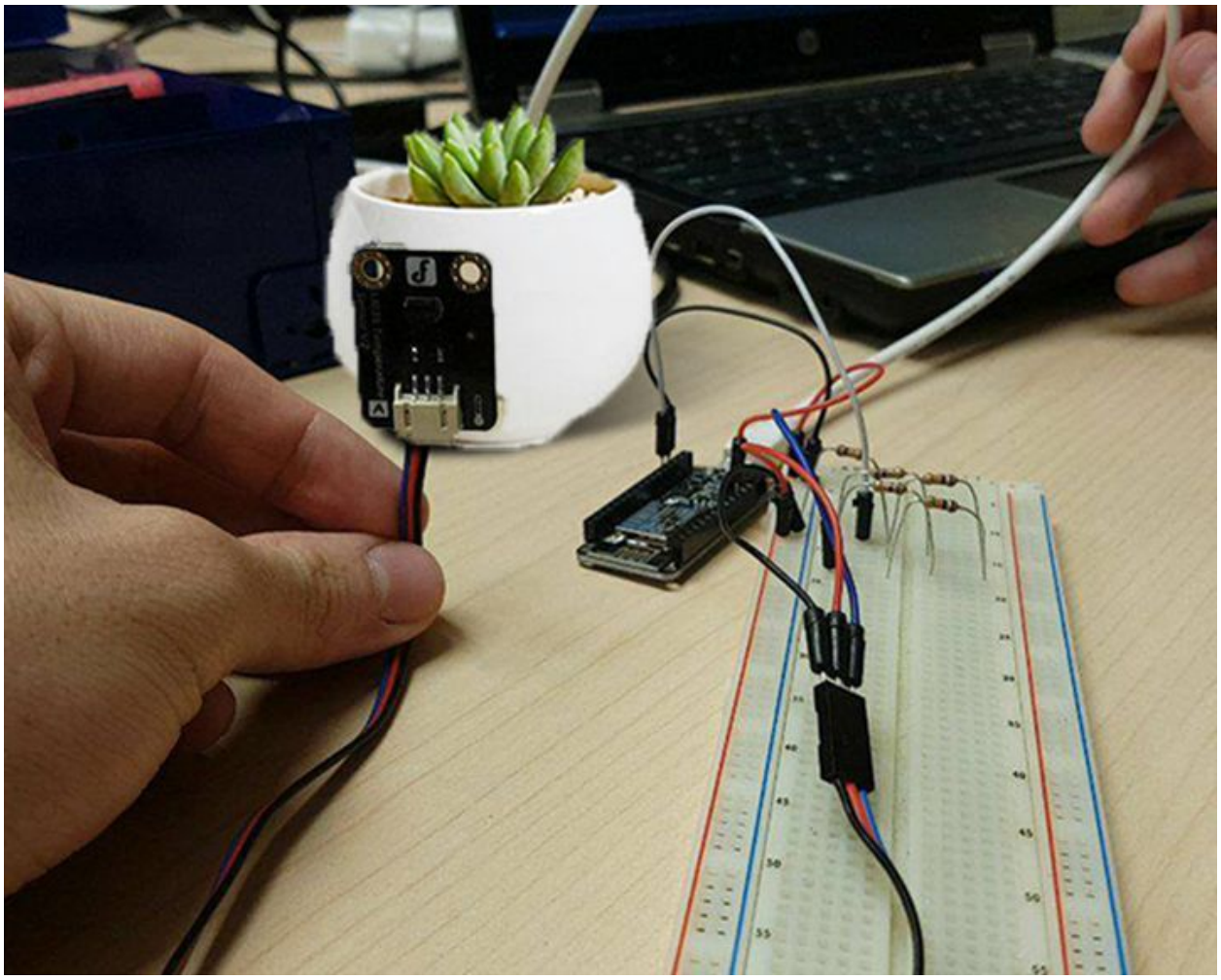

Figure 1. Plant.tv completed project

## Project Breakdown

### *IOT Device*

### Adafruit Feather HUZZAH

The Adafruit Feather HUZZAH is an ESP8266 Wi-Fi microcontroller clocked at 80MHz and at 3.3V logic and is the core to our Internet of Things project. This device contains 4 MB of flash memory (32 megabits). The HUZZAH also consists of one analog input, which has a 1.0V max, 9 GPIO pins, a built in USB, and a built in 100mA lipoly charger. The primary operating voltage of the HUZZAH is 3.3V, however, if the HUZZAH is being powered VIA usb, it will regulate the 5.0V USB to the necessary 3.3V.

The Adafruit HUZZAH communicates directly to the Heroku server via HTTP POST requests. The HUZZAH sends the POST requests with the temp data received through the LM35 Linear Temperature Sensor to the Heroku server to be utilized on the website (*as seen in the code snippet below*). However, it is important to note that at no time does the Heroku cloud server communicate back to the HUZZAH.

The code we used to send HTTP POST requests to the heroku server is in a Django Framework. Essentially, the code reads in the analog's value, created by the temperature sensor, begins the HTTP host, and sends the analog value in an HTTP POST consisting of the string "analog_value=" and the temp data stored in the variable "val." The code snippet below is how the post requests are processed (full code is shown in source code section of this report).

```
// the loop function runs over and over again forever
void loop() {
  int val = analogRead(analogPin);

  Serial.println(String("") + "Sending POST request with value " + val + ".");
  HTTPClient http;
  http.begin(String("") + "http://" + host + "/data/");
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  http.POST(String("") + "analog_value=" + val);
  http.writeToStream(&Serial);
  http.end();

  delay(5000);
}
```

## LM35 Linear Temperature Sensor (I/O device)

The LM35 Linear Temperature Sensor is powered by a 5V source. It is connected to the Adafruit Feather HUZZAH using a single analog pin. The temperature sensor's results, measured by the output voltage which is proportional to the temperature, will interact with the Adafruit HUZZAH. From there, it will connect to Wi-Fi and upload the results onto the cloud service (Heroku).

The linear temperature sensor uses two metal wires that join to create an electrical junction at changing temperatures (thermocouple). This thermocouple produces different output voltages depending on the temperature being sensed; called thermoelectric effect or the Seebeck effect. When one of the wires is cold and the other is hot, current can flow between the two wires. When both wires are at the same temperature, there would be no current. This is the basis of all thermocouples.

There are different types of thermocouples and can be configured to sense different ranges of temperatures. Since the temperature range of the LM35 is 0-150 degrees celsius, we can assume that the sensor is a type K thermocouple. Thermocouples sensors are a widespread use and popular due their versatility and ease of configuration.

The code we used to convert output voltage of the LM35 to a temperature is in Python, with the Django framework. The code reads the voltage values from the sensor 5 times and sends it to the Feather HUZZAH. The Feather HUZZAH sums up the 5 values and averages it. We then take that average value and multiply it by another value given by django (0.5, dividing the average number by 2). The code below represents the conversion from voltage to degrees celsius:

```
def home(request):
    plant_data = PlantData.objects.all().order_by('-timestamp')

    rollint_avg__max_length = 5
    analog_values = plant_data[:rollint_avg__max_length].values_list('analog_value', flat=True)
    current_temp = PlantData.to_temp(sum(analog_values) / len(analog_values))

    return render(request, 'planttv/home.html', {
        'current_temp': current_temp,
```

```
            'plant_data': plant_data
        })

def to_temp(analog_value):
            return analog_value * PlantData.ANALOG_TO_TEMPERATURE
```

## LM35 ➔ Adafruit Feather HUZZAH Communication Details

The Adafruit Feather HUZZAH receives analog output from the LM35 linear temperature sensor. The maximum analog voltage specification of the Feather HUZZAH is 1V, and the maximum analog voltage potential for the LM35 is 5V. Because of the specification inconsistency, it is necessary to use a voltage reduction circuit to keep the communicating analog signal within the bounds of the Feather HUZZAH specification.

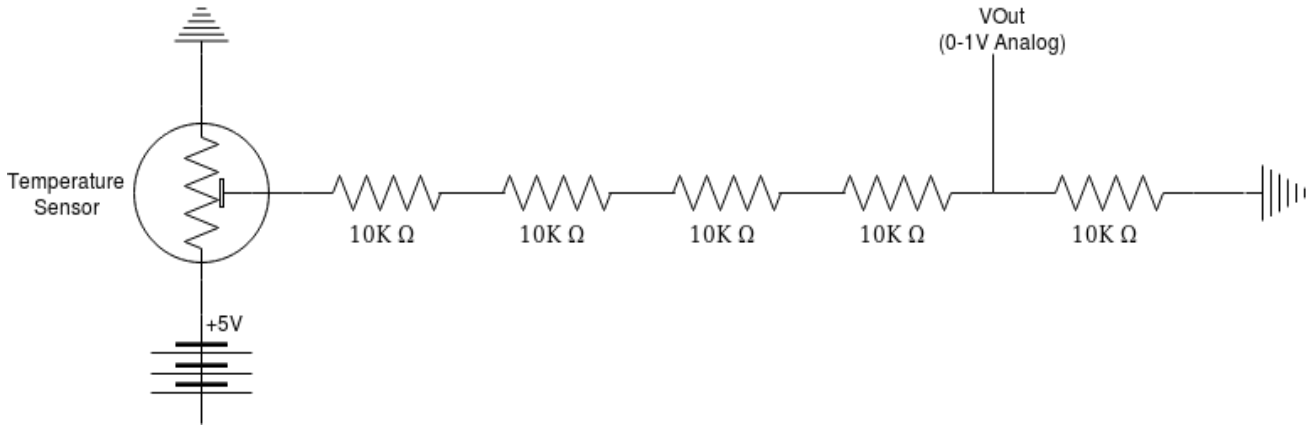


Figure 2. LM35 ➔ Adafruit Feather HUZZAH analog compatibility conversion circuit.

### *Cloud Service*

The Cloud Service we used was the Heroku platform. Heroku allows users to deploy their web apps fast with almost no overhead and minimal configuration. It was a perfect choice for our project as we needed something simple as we had little experience with cloud services.

How Heroku works is that it is based on a managed container system that has an integrated data service and a powerful ecosystem in order to allow users run their web apps. Firstly, we would deploy our code in our chosen language. Then the web app will run in smart containers called dynos on a stable and fully managed runtime environment, in which we can easily manage the app through the Heroku dashboard. By having a runtime, it enables the app to continuously run without any manual intervention. Heroku

also allows the developer to have data in web-scale backing services through add-ons. Lastly, the user of the web app will make requests which will be served by the app itself.

Heroku allows for its users to pick from a range of programming developments. Ranging from Java to Node, there was no shortage of selection. As a team we decided on Python as it was the language we were the most comfortable in. Once we picked our language we followed the Heroku documentation to get up and running. Heroku allowed us to bypass all the confusing steps that other cloud services have.

As for the interaction with Heroku between the IOT device and the end data application, it does not know or request anything about the IOT device. This is because the IOT device sends requests to the server and provides it with the information needed to be processed. For the end data application, Heroku communicates through our python app that displays the web page.

### *End Data Application*

Our end data application is written in Python using the Django framework. It is responsible for serving HTML webpages to the end user's browser. Django also managed database creation and configuration so the process to get up and running was nearly seamless.

A feature that we didn't have time to implement is a live video feed embedded in the page. We considered using Periscope to provide this service, but unfortunately they don't provide a method to embed their application directly onto a webpage.

This end data application also provided a REST endpoint for the HUZZAH to POST temperature updates to. When an update is received, the reported analog value is saved along with the current timestamp. This method of storing temperature records allows us to historically access temperature data.

## Team Issues

As a team we were lucky to run into almost no problems. We worked efficiently and quickly. By creating a list of the tasks that needed to be done, we crossed them off one by one.

However, one task that our team struggled on was deploying our application to the internet. While using Heroku we ran into an issue configuring Django to collect and serve the static files (css, images, etc); by default Django does not do this in production

mode, so we had to add additional Django middleware to perform this service. We were able to figure out the solution to this problem with Heroku's documentation. Once fixed, we were able to complete our project swiftly and have a working demo in time for our presentation.

## Source Code

This is our html code. This is the code that runs when the user goes to http://planttv.herokuapp.com/. It has three simple if statements that check the current temperature. If the temperature is below 18, it will display the cold plant as well as its corresponding message. If the the temperature is above 30, it will display the hot plant. Without this html page, there would be no way for the user of this application to see the data that the temperature sensor is outputting.

```
<!DOCTYPE html>
<html>
	<head>
	{% load staticfiles %}
	<link rel="stylesheet" type="text/css" href="{% static 'css/basicCss.css' %}"/>
	</head>
	<body>
		<center>
		<h1>Welcome to Plant.TV!</h1>
		<h3>Current average temp: {{ current_temp }}°C</h3>
		<div>
		{% if current_temp < 18 %}
			<img style="width: 100px" src="{% static 'img/cold_plant.jpg' %}" />
			<p>It's cold!</p>
		{% elif current_temp < 30 %}
			<img style="width: 100px" src="{% static 'img/happy_plant.png' %}" />
			<p>It's nice!</p>
		{% else %}
			<img style="width: 100px" src="{% static 'img/hot_plant.jpg' %}" />
			<p>It's too warm!</p>
		{% endif %}
		</div>
		</center>
	</body>
</html>
```

This is our temperature sensor code. Written in a dialect of C/C++ using the Arduino IDE. This code connects to the WIFI that was set up for use in our project. By connecting to our web app, this code posts data to our django database allowing us to manipulate the data in python and show the temperature to the user on the browser.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

const char* ssid     = "cisc340";
const char* password = "L0gica1@";

const char* host = "planttv.herokuapp.com";

const int analogPin = 17;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);

  Serial.begin(9600);
  delay(100);

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(500);
    digitalWrite(LED_BUILTIN, LOW);
    Serial.print(".");
    delay(500);
  }
```

```
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  digitalWrite(LED_BUILTIN, LOW);
}

// the loop function runs over and over again forever
void loop() {
  int val = analogRead(analogPin);

  Serial.println(String("") + "Sending POST request with value " + val + ".");
  HTTPClient http;
  http.begin(String("") + "http://" + host + "/data/");
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  http.POST(String("") + "analog_value=" + val);
  http.writeToStream(&Serial);
  http.end();

  delay(5000);
}
```

## Conclusion

Overall, the project of designing and developing Plant.tv allowed Team 14 to gain even further hands on experience with hardware as well as with Internet of Things devices. It provided us with a simulation of collaborating with others within a team and gave insight on how companies in the industry work and develop a tangible product. In order for Plant.tv to cover all three components of the project requirements, it is comprised of the Huzzah as the IOT device which is connected to a LM35 Linear Temperature Sensor through analog pins, Heroku as the cloud service, and a web browser a user can access as the end data application. For future development of this project, a soil moisture sensor could be also be applied. By doing so, the project will become more applicable to a user and in real life. We are very pleased with the final product for this project, and we feel that Plant.tv embodies the goal of this project to allow students to integrate and apply our skills gained in the course through Internet of Things.

## References


1. Industries, Adafruit. “Adafruit Feather HUZZAH with ESP8266 WiFi.” *Adafruit Industries Blog RSS*, Adafruit, ‘n.d.’, www.adafruit.com/product/2821.
2. Ada, Lady. “Adafruit Feather HUZZAH ESP8266.” *Overview | Adafruit Feather HUZZAH ESP8266 | Adafruit Learning System*, Adafruit, 25 Nov. 2015, learn.adafruit.com/adafruit-feather-huzzah-esp8266/overview.
3. Ada, Lady. “Adafruit Feather HUZZAH ESP8266.” *Power Management | Adafruit Feather HUZZAH ESP8266 | Adafruit Learning System*, Adafruit, 25 Nov. 2015, learn.adafruit.com/adafruit-feather-huzzah-esp8266/power-management.
4. https://github.com/ryanrossiter/planttv
5. Inc, OMEGA. “Thermocouple” *Omega Engineering*, ‘n.d’. https://www.omega.ca/prodinfo/thermocouples.html
6. Woodford, Chris. “How Do Thermocouples Work?” *Explain That Stuff*, 1 Jan. 2016, www.explainthatstuff.com/howthermocoupleswork.html.
7. “Deploy and Run Apps on Today's Most Innovative Platform as a Service.” *The Heroku Platform as a Service & Data Services | Heroku*, ‘n.d.’, www.heroku.com/platform.