

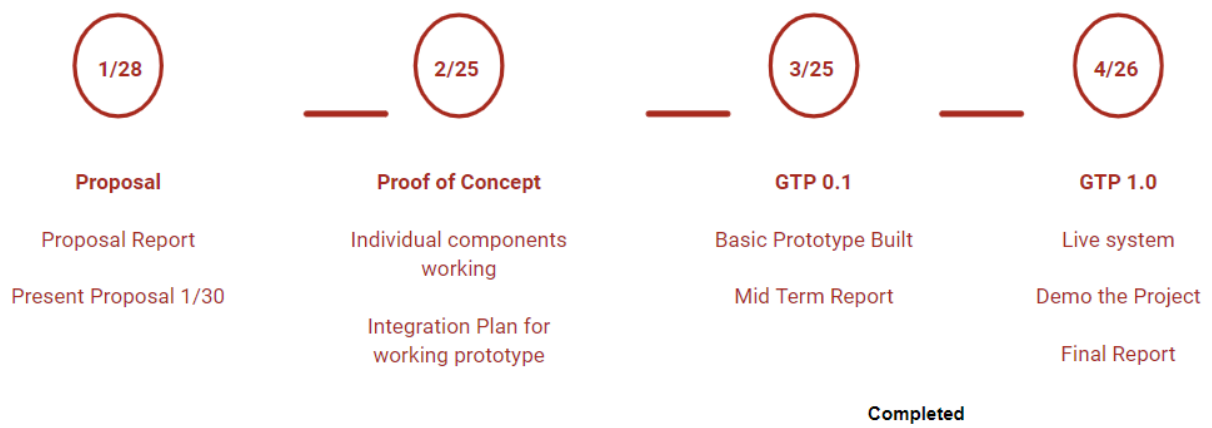
Green Thumb Project

By: Adarsh Gopalakrishnan, John Cutone, Tracy Keys

Background and Objectives

The Green Thumb Project built a working model of a system for smart “hot houses” to utilize IoT (Internet of Things) to remotely manage UV light and air humidity and temperature data of plants in a wholesale hot house business. Our objective was to learn how to create, monitor and control an end to end system from physical sensors connected via an electronic circuit board to a Raspberry Pi, which then connects over wifi to the internet to a Firebase real-time database and web application. This report summarizes the experience of building the prototype, our challenges and achievements, and what future iterations might include.

Timeline



The Green Thumb Project idea

The Green Thumb Project (GTP) system is composed of three units: the Raspberry Pi (with its electronic circuit board and plant environment sensors), the cloud Firebase Database and the GTP Web Application (Web App) built on bubble.io. All three units are connected via the internet. Figure 1 shows the GTP system conceptual diagram.

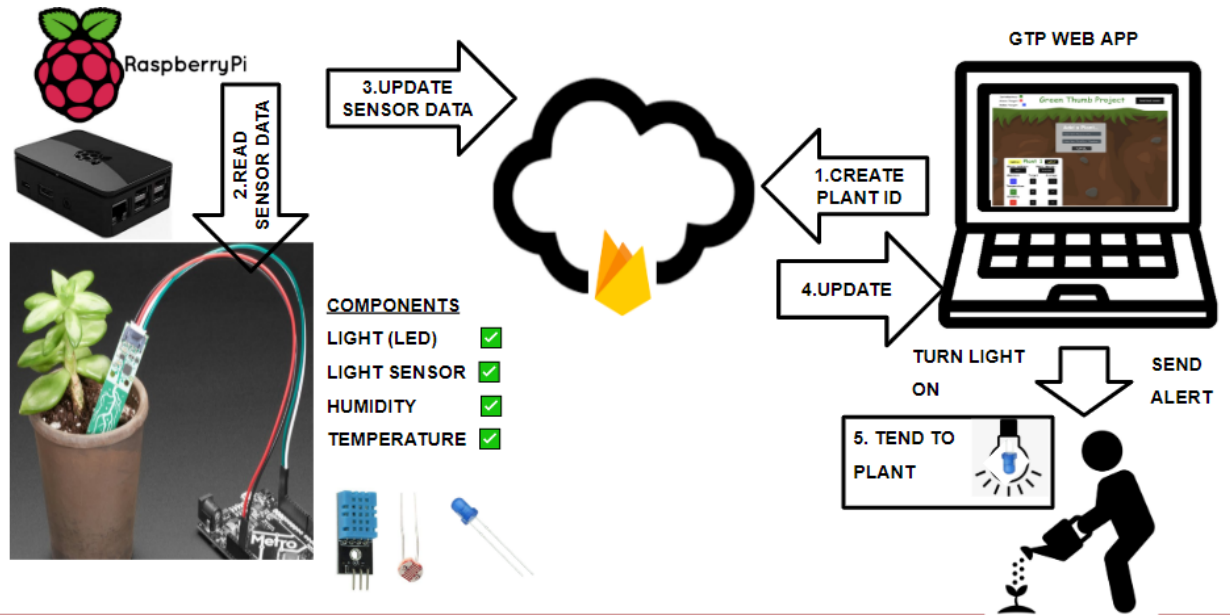


Figure 1 GTP System conceptual diagram

The complete list of deliverables and responsibilities are shown in the table below.

| <u>Deliverable</u> | <u>Responsibility</u> | <u>Completed</u> |
|---|-----------------------|------------------|
| | ? | |
| Product requirements | Tracy | ✓ |
| Raspberry Pi components and design sourced | Tracy | ✓ |
| Raspberry Pi system set up and programmed | Tracy | ✓ |
| Failed deployment of GreenPiThumb code [1] | Tracy | ✗ |
| Sensors connected and programmed individually in Raspberry Pi | | ✓ |
| Temperature | Adarsh | ✓ |
| Humidity | Adarsh | ✓ |
| Light LED | Tracy | ✓ |
| Light Dependant Resistor sensor | Tracy | ✓ |
| Sensor programs integrated into one GTP.py | Adarsh | ✓ |
| Firestore Database built | Adarsh | ✓ |
| Raspberry Pi and Firestore connected | Adarsh | ✓ |
| Web app platform sourced and selected | John | ✓ |
| Web App designed and built | John | ✓ |
| Firestore and Web App connected | Adarsh/John | ✓ |

| | | |
|--|-------------|---|
| Functionality to send notification emails | John | ✓ |
| Functionality to add a plant and manual control of LED in Web App/Firebase | John/Adarsh | ✓ |
| Manual LED override programming in Raspberry Pi | Adarsh | ✓ |
| Reports, documentation and Presentations writing | Tracy | ✓ |

Raspberry Pi Plant monitoring device and components

The Raspberry Pi (RPi) is a micro-processor connected to a electronic circuit board (called the breadboard) and 3 electronic sensors: the light dependent resistor (LDR), an LED light, and a DHT11 air humidity and temperature sensor. These are set up to monitor the plants environment in the hothouse. The RPi code GTP.py is shown in Appendix 1.

The Raspberry Pi runs on Python code, and has its own integrated development environment (IDE). Its basic packages such as Python 3, Adafruit DHT11, RPi.GPIO packages and other required packages were downloaded from websites and github repositories to set it up to begin programming the GTP Project. Inspiration and tutorial videos used in the GTP project are acknowledged in the References section [2][3][4][1][5, p. 11][6].

The Raspberry Pi (RPi) is physically connected to the sensors via the breadboard and three types of wires: data output connections, 3V or 5V power connections and ground connections to the GPIO board in the RPi. The LED circuit is made up of the LED, a resistor to ensure the LED doesn't burn out, and the three types of wires. The red LED data wire is connected to GPIO Pin 21 (board pin 40), and does not have a seperate power wire (see Figure 2 for screen shot).

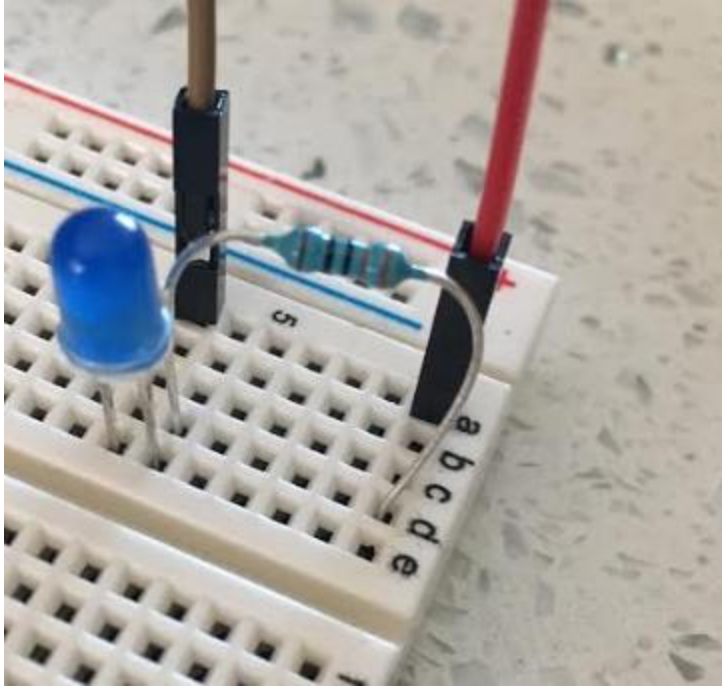


Figure 2 Close up of the LED, resistor and ground and output wires on breadboard

The LDR circuit includes the LDR itself, and a 10 ohm capacitor that charges up when the plant environment is light and discharges when it grows dark, essentially measuring the resistance of the LDR. Therefore the higher the reading, the darker it is. The project team sourced a 10 ohm rather than a 1 ohm capacitor so the reading threshold had to be set at 5000 rather than 400 as originally designed. It is powered by GPIO Pin 1 the 3V power on the RPi, and data comes via GPIO pin 13 (see Figure 3 for screenshot)

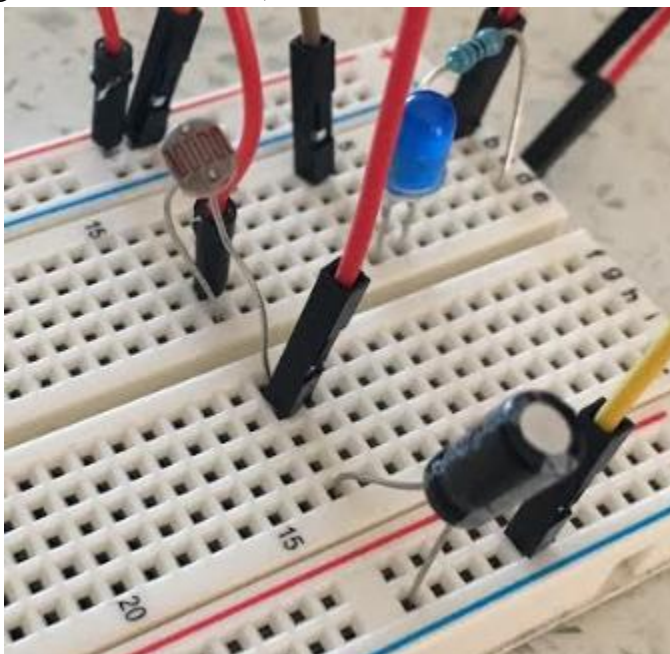


Figure 3 Bronze LDR, black 10 ohm capacitor, and wires with LED in the background

DHT11 temperature and humidity sensor connects directly to the Raspberry Pi, rather than using the breadboard, with a 5V positive connection (grey wire) connected to board pin 2 on the RPi, a ground/negative connection (black wire) connected to board pin 9 and a data out connection (white wire) connected to board pin 7 (GPIO pin 4). The image shown in Figure 4 has different wires but the color scheme needed to be swapped once there were a lot of components attached for ease of connecting and management.



Figure 4 DHT11 connection to Raspberry Pi

The RPi is programmed using Python 3 to regularly takes readings from these components: it takes the reading of the the level of light/dark in the plant environment from the LDR, it reads whether the LED Light is on or off, and it reads the air humidity and temperature surrounding the plant using the DHT11 sensor (see Figure 5a and 5b for screenshots). These readings are then sent over the internet to the Firebase database.

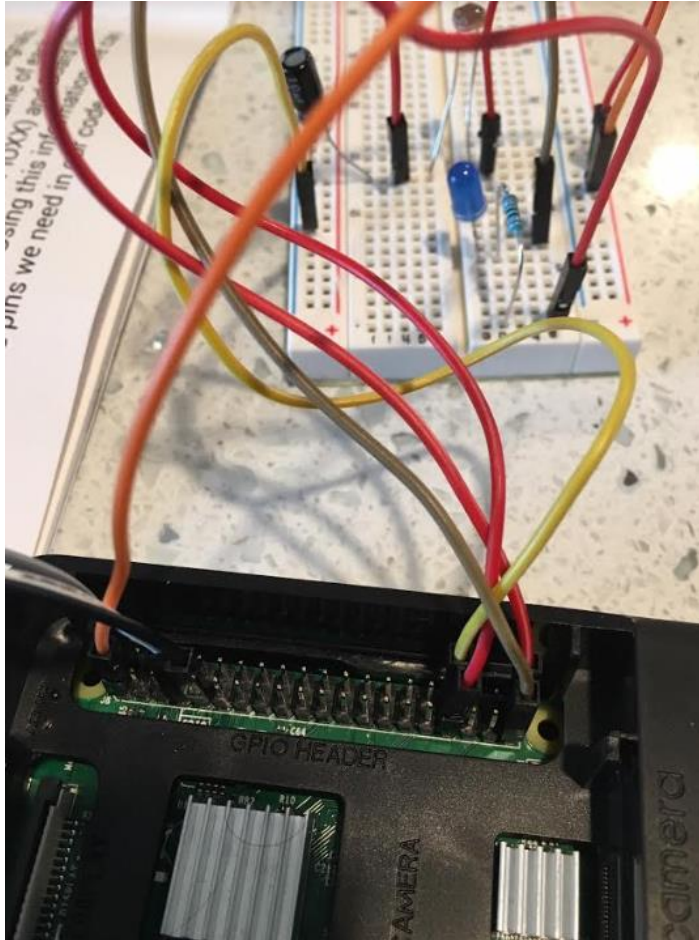
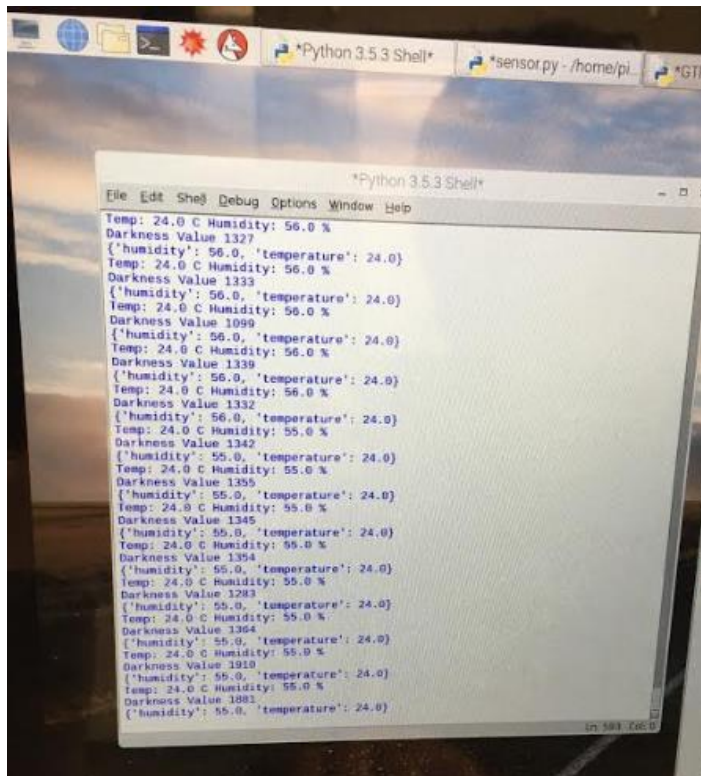


Figure 5a Close up of Raspberry Pi circuit

The sensors are connected to the GPIO Header, which is comprised of 40 General Purpose Input Output (GPIO) pins as shown in Figure 5a. The entire Raspberry Pi setup is not that mobile, as you can see in Figure 5c. A monitor, keyboard, mouse and stand were required plus many cables for power and connection.

GREEN THUMB PROJECT PROPOSAL



```
Python 3.5.3 Shell
File Edit Shell Debug Options Window Help
Temp: 24.0 C Humidity: 56.0 %
Darkness Value 1327
{'humidity': 56.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 56.0 %
Darkness Value 1333
{'humidity': 56.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 56.0 %
Darkness Value 1099
{'humidity': 56.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 56.0 %
Darkness Value 1339
{'humidity': 56.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 56.0 %
Darkness Value 1332
{'humidity': 56.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1342
{'humidity': 56.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1355
{'humidity': 55.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1345
{'humidity': 55.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1354
{'humidity': 55.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1364
{'humidity': 55.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1364
{'humidity': 55.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1319
{'humidity': 55.0, 'temperature': 24.0}
Temp: 24.0 C Humidity: 55.0 %
Darkness Value 1381
{'humidity': 55.0, 'temperature': 24.0}
```

Figure 5b The RPi readings print out

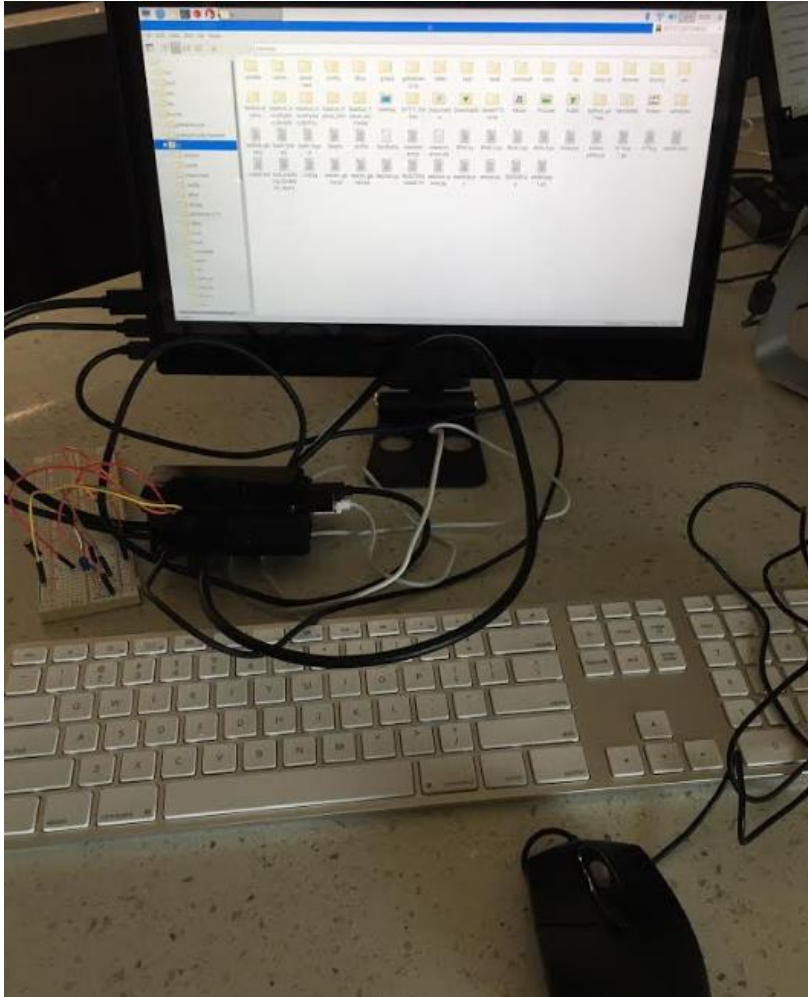


Figure 5c Raspberry Pi, breadboard and peripherals

There is also automation of light for the plant, with the RPi programmed so that when the LDR readings are greater than 5000, the LED automatically turns on and remains on whilst the LDR reading is at this level (see Figure 6 for screen shot). In addition, the LED can be controlled manually by the Web App, overriding the LDR readings. The default is set to turn off gain after ten seconds but can also be turned off by the Web App.

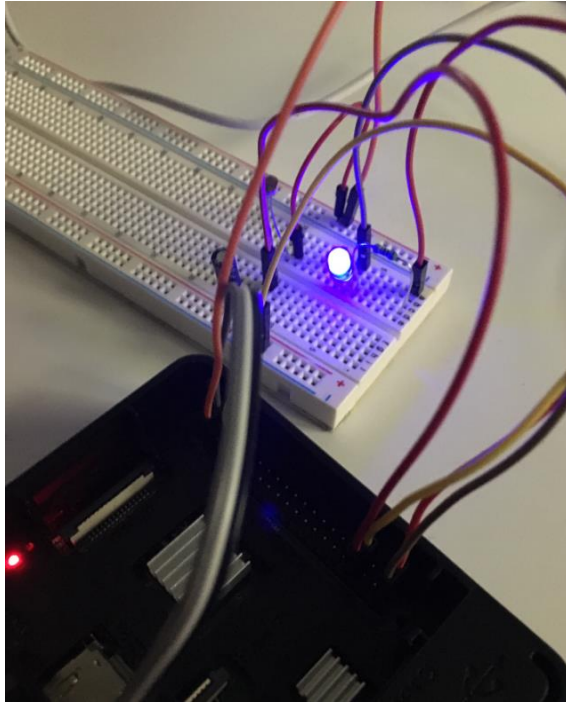


Figure 6 LED automatically turning when LDR reaches 5000

The programming of the Raspberry Pi took a lot of time, although in the end appears rather simple. Initially, Tracy had sourced code from GreenPiThumb project [1] and had procured all the sensors to use this as a base, and installed all the software on the device (Adafruit and Ansible packages mostly) (see Figure 8 for screenshot of file folders on the Pi).

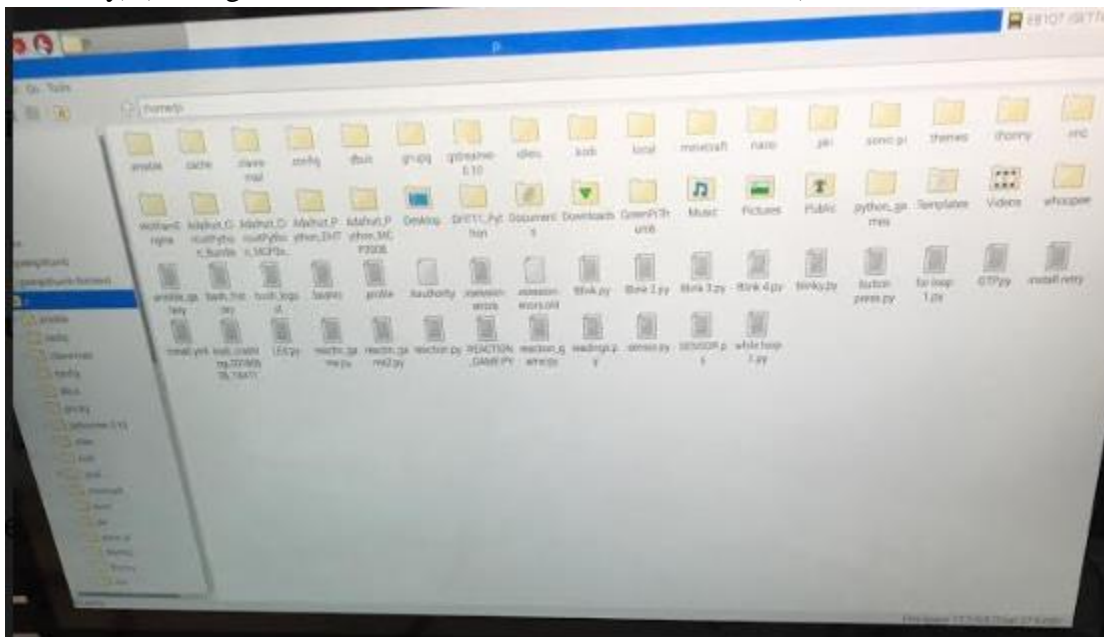


Figure 8 RPi File Folders

However, the teams knowledge of Ansible for remote control and deployment wasn't up to the task, and indeed Ansible was completely unnecessary as it was possible to connect remotely into

the RPi using ssh. The GreenPiThumb project also used a Sparkfun soil moisture sensor which we had procured, but we ran out of time to add this as well. Therefore the decision was made to ditch the GreenPiThumb approach and use basic RPi code for the programming.

The python code was developed as two individual programs, one for the DHT11 [5, p. 11] and one for the LED [7] and LDR sensor circuit. Adarsh then integrated both programs into one, which is shown in Appendix 1 and uploaded to Blackboard also.

The GTP Firebase Database

The GTP database is a Firebase database in the cloud, the console is located [here](https://gtpdatabase.firebaseio.com). The Firebase Database URL is <https://gtpdatabase.firebaseio.com>. The database receives regular updates of sensor readings from the Raspberry Pi via the internet (whenever the RPi is on and connected to the internet). The structure and updated readings from the sensors are shown in Figure 9 below. Firebase stores the last 10 readings for each plant, as well as permanent information about the plant such as where it is located and what type of plant it is. Latest readings are kept in their own field to enable easy display in the web app. The Firebase database also stores the status for the LED light as updated from the Web App, 1 for on and 0 for off, which RPi then reads as part of the program to manually operate the LED.



Figure 9 GTP Firebase Cloud based real time database

The GTP Web App

The GTP Web App uses the Bubble.io platform, which was selected because it supports building web apps without code. The Web App is located [here](#).



Figure 10 Screenshot of the Web App

The web app has a number of capabilities: the system administrator can add plants using the Web App, it displays the control panel for each plant showing current readings, target, and calculates and displays the last 10 readings average. This control panel displays to the user whether the plant environment is on or outside the target range using color. Lastly, the user can manually control the plants LED light overriding the automatic system, using the Light On or Light Off button.

The GTP web app uses an API built by bubble for Firebase. This enables it to pull the latest readings, the last 10 readings for each plant from the Firebase database, and also to add a Plant

object to the database. The light on and off functionality of the Web App uses the API to update a field in the Firebase Database which the RPi then reads to turn the light on or off.

Lastly, the web app has the capability to send users a notification email, with the latest status report for each plant (see Figure 10 for screenshot).

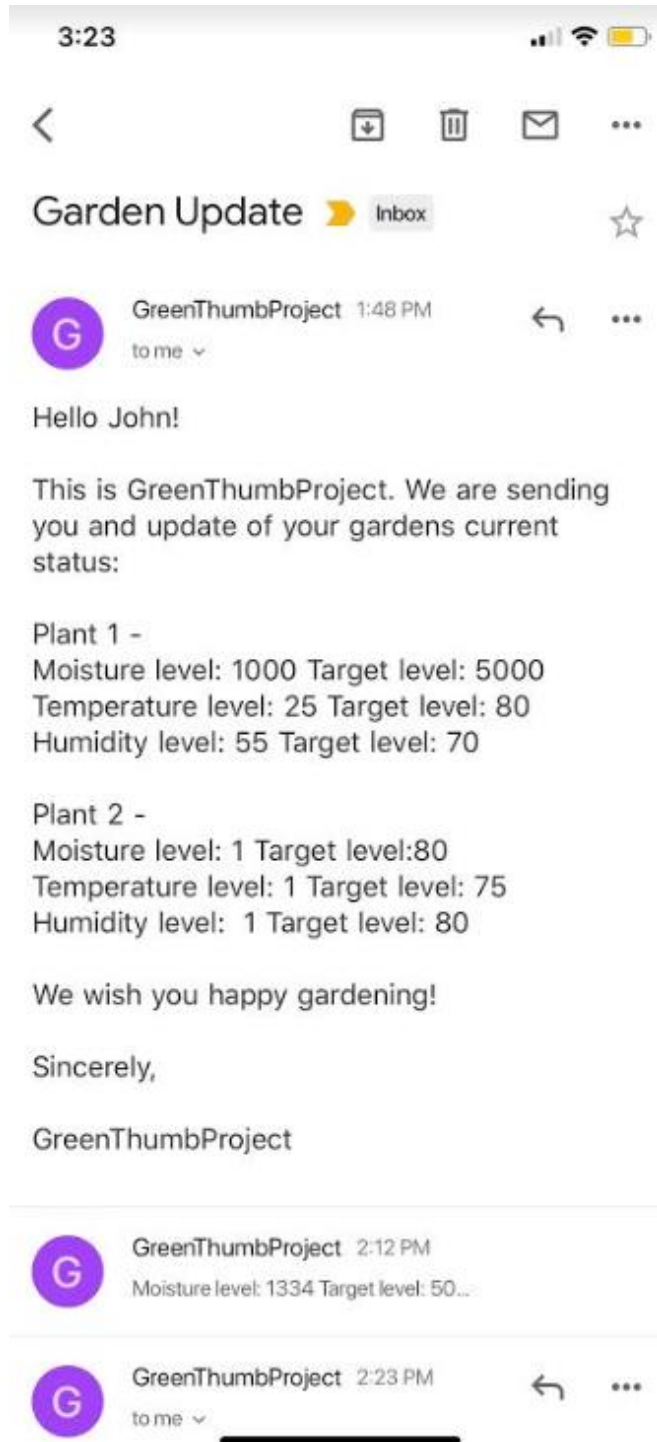


Figure 10 User notification email

Responsibilities and Work of each team member

Our team worked very collaboratively and worked together on concept design, planning, end to end testing and pressure testing whether the project had met the requirements. In accordance with the original plan, Tracy was responsible for product requirements, report writing and the RPi procurement, set up and programming of the Raspberry Pi, sensor code, set-up and testing. John was responsible for selecting a Web application tool, and the complete design and build of the GTP web application (app) and its user control and display features, sending commands to the Firecase Database (DB) and receiving updates from the Firebase DB. Adarsh was responsible for bringing it all together: all the more complex pieces of RPi code, overall system integration and testing, and setting up the Firebase Database and integrating it with both the Web App and the Raspberry Pi. In particular Adarsh developed the RPi manual override code, and integrated the code for each sensor into one working program, GTP.py.

In Summary

This was a very fun project, unlike anything any of the team had worked on before and quite novel amongst the INF551 class of Spring 2019. None of the team were familiar with electronics, programming Raspberry Pis, building a Web App, or creating and running a live cloud based database. We learnt a lot of techniques such as testing each unit separately and then integrating them, troubleshooting each component, as well as utilising Youtube and Google resources.

As a result we were able to achieve everything we set out to do, despite facing major hurdles and needing to pivot our entire program design to something a lot simpler but still effective. Future iterations of the GTP system could include the Sparkfun Soil Moisture sensor, and the RPi is even capable of being a media center so we could play the plants soothing music so they grow faster.

References

- [1] "GreenPiThumb: A Raspberry Pi Gardening Bot," *mtlynch.io*, 27-Jun-2017. [Online]. Available: <https://mtlynch.io/greenpithumb/>. [Accessed: 08-Feb-2019].
- [2] "Turning on an LED with your Raspberry Pi's GPIO Pins," *The Pi Hut*. [Online]. Available: <https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins>. [Accessed: 11-Apr-2019].
- [3] Gus, "How to Setup a Raspberry Pi Light Sensor," *Pi My Life Up*, 30-Jan-2016. [Online]. Available: <https://pimylifeup.com/raspberry-pi-light-sensor/>. [Accessed: 22-Mar-2019].
- [4] "JeetShetty/GreenPiThumb," *GitHub*. [Online]. Available: <https://github.com/JeetShetty/GreenPiThumb>. [Accessed: 13-Feb-2019].
- [5] C. B. | R. Pi | 140, "How to Set Up the DHT11 Humidity Sensor on the Raspberry Pi," *Circuit Basics*, 13-Dec-2015. [Online]. Available: <http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-the-raspberry-pi/>. [Accessed: 20-Mar-2019].
- [6] *Python code to use the MCP3008 analog to digital converter with a Raspberry Pi or BeagleBone black.*: *adafruit/Adafruit_Python_MCP3008*. Adafruit Industries, 2019.
- [7] "Turning on an LED with your Raspberry Pi's GPIO Pins," *The Pi Hut*. [Online]. Available:

<https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins>. [Accessed: 26-Mar-2019].

Appendix 1 Raspberry Pi Code

```
 -*- coding: utf-8 -*-
#!/usr/bin/python

import pyrebase
import sys
import Adafruit_DHT
import json, requests
import RPi.GPIO as GPIO
import time

#Raspberry Pi GPIO pins have two different naming conventions- setting to BCM
GPIO.setmode(GPIO.BCM)

#create function to check LDR sensor reading contributed by Tracy
def LSval (LSpin):
    reading = 0
    GPIO.setup(LSpin,GPIO.OUT)
    GPIO.output(LSpin,GPIO.LOW)
    time.sleep(1)
    GPIO.setup(LSpin,GPIO.IN)
    while (GPIO.input(LSpin) == GPIO.LOW):
        reading += 1
    return reading

#Set the data pins based on their GPIO number in RPi
#Light sensor =LS Light is LED
LED=21
LS=13

#contributed by Adarsh

GPIO.setup(LED,GPIO.OUT)
config = {
    "apiKey": "insertkeyhere",
```



```

    "authDomain": "gtpdatabase.firebaseio.com",
    "databaseURL": "https://gtpdatabase.firebaseio.com",
    "projectId": "gtpdatabase",
    "storageBucket": "gtpdatabase.appspot.com",
    "messagingSenderId": "791376230162"
  }

  firebase = pyrebase.initialize_app(config)
  db = firebase.database()

  #url = "https://gtpdatabase.firebaseio.com/temperature.json"
  while True:
    humidity, temperature = Adafruit_DHT.read_retry(11,4)
    print("Temp: {0:0.1f} C Humidity: {1:0.1f} %".format(temperature, humidity))
    lightres = LSval(LS)
    updatedata = {"Temperature": temperature,
                  "Humidity": humidity,
                  "Light Resistance": lightres}
    db.child("plant1").child("Latest Readings").update(updatedata)
    #data = {'humidity': humidity, 'temperature': temperature}
    print(updatedata)

    #Read the database for LED status
    led = db.child("plant1").child("LED Switch").get().val()

    #check the number returned from the function and turn LED on or off as needed
    if led == 1:
        GPIO.output(LED,True)
        time.sleep(10)
    elif led == 0.5:
        if lightres>5000:
            GPIO.output(LED,True)
        else:
            GPIO.output(LED,False)
    else:
        GPIO.output(LED,False)
        time.sleep(10)

    db.child("plant1").update({"LED Switch": 0.5})
    #Update the database
    humidity_readings = db.child("plant1").child("Readings").child("Humidity").get()
    humidity_readings = humidity_readings.val()[1:] + [humidity]

```

```
temp_readings = db.child("plant1").child("Readings").child("Temperature").get()
temp_readings = temp_readings.val()[1:] + [temperature]
lightres_readings = db.child("plant1").child("Readings").child("Light Resistance").get()
lightres_readings = lightres_readings.val()[1:] + [lightres]
db.child("plant1").child("Latest Readings").update(updatedata)
update_readings = { "Humidity": humidity_readings,
                    "Temperature": temp_readings,
                    "Light Resistance": lightres_readings}
db.child("plant1").child("Readings").update(update_readings)
#requests.post(url, json.dumps(data))
#Python 3.5.3 (default, Jan 19 2017, 14:11:04)
#[GCC 6.3.0 20170124] on linux
#Type "copyright", "credits" or "license()" for more information.

GPIO.cleanup()
```

Appendix 2 Web App code for the manual LED override

```
curl -X PATCH -d '{"LED Switch": 1}'
"https://gtpdatabase.firebaseio.com/plant1.json"
curl -X PATCH -d '{"LED Switch": 0}'
"https://gtpdatabase.firebaseio.com/plant1.json"
```

Appendix 3 Links to the Web App and Firebase Console

<https://greenthumbproject.bubbleapps.io/>
<https://console.firebase.google.com/u/0/project/gtpdatabase/database/gtpdatabas>
[e/data/](#)