IOT PROJECT
SMART GARDEN

mokxf16@sp.edu.sg

SCHOOL OF COMPUTING (SOC) Singapore Polytechnic

Table of Contents

Section 1
Overview of project

A. Tutorials

The tutorial is linked here: https://www.hackster.io/mokxf16/smart-garden-raspberry-pi-arduino-65c7b7

Github link: https://github.com/chowzzzz/smartgarden

B. What is the application about?

The smart garden monitors the temperature, humidity, light levels and soil moisture of the plant. It has an automated system that waters the plant when the soil is too dry and switches on the light when it is too dark. This maintains an ideal and consistent soil condition for the plant, and makes it convenient for those who tend to forget to water their plants regularly. Also, the plant can continuously photosynthesize even when there is no sunlight.

We will be using an Arduino and a Raspberry Pi to receive data from the sensors and control the different actuators. The surrounding temperature, air humidity and brightness values will be recorded, as well as the soil moisture levels. These values will then be displayed on the LCD screen, which allow users to know the environmental conditions of the plants when they check on them.

When the soil moisture level goes above 500 (for our soil moisture sensor, the higher it is the drier the soil), the red LED will light up as a warning to show that the plant needs water. Also the water pump will start to run and pump water into the soil automatically. This is very convenient for users as they do not need to water their plants every time but instead let the system water their plants automatically based on the moisture level of the soil.

As for the automated light, when the LDR records a value higher than 300, the yellow LED will light up and act like the sun, to allow continuous photosynthesis to occur for the plants.

The temperature, humidity, light levels and soil moisture values will also be published to DynamoDB. Through a server (Raspberry Pi), the data will be displayed onto a flask web page where it shows real-time data coming from the sensors. This will allow users to view the real-time environmental conditions of the plants on the go (the latest 15 records through a graph).

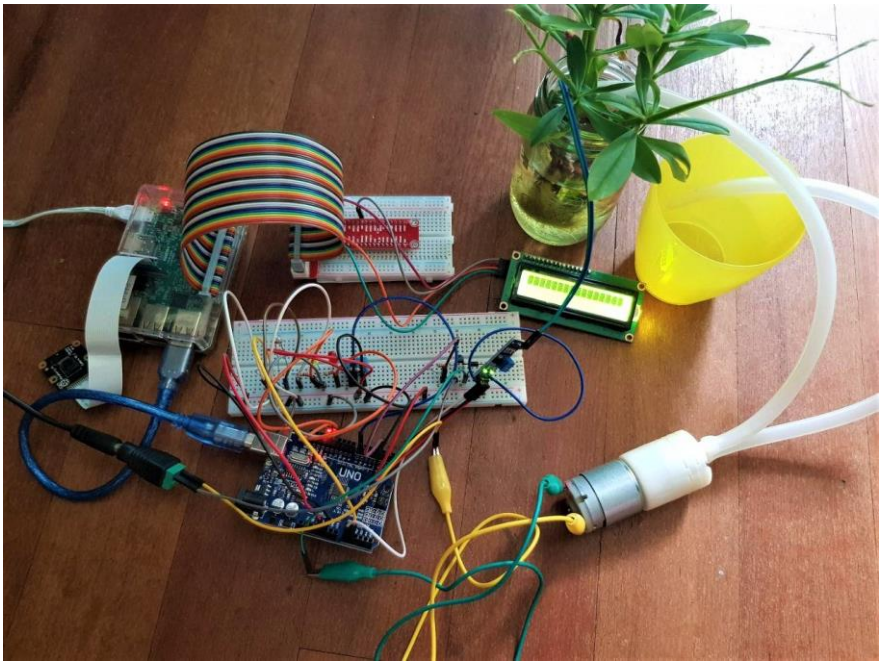
The web page will also allow users to control the water pump and decide whether they wish to water the plants automatically or manually. They can turn on or off the water pump

whenever they wish to, thus making it very convenient if users wish to water their plants even when they are not around.

C. Summary of the Steps

	Section	Description
1)	Overview	Overview of application
Sections 2 to 8 provides the step-by-step instructions to set up the application		
2)	Hardware Requirements	Provides overview of hardware required
3)	Hardware Setup	Setting up of hardware – Smart Garden (3.1) & Lock System (3.2)
4)	Software Setup	Downloading of packages on Raspberry Pi and creation of 3rd party software accounts
5)	Setting Up Amazon Web Service (AWS) account and DynamoDB	Set up AWS account and create DynamoDB Database
6)	Coding the Application	Write the necessary codes
7)	Running the Application	Guides user how to run the application
8)	Outputs of application	Web Interface

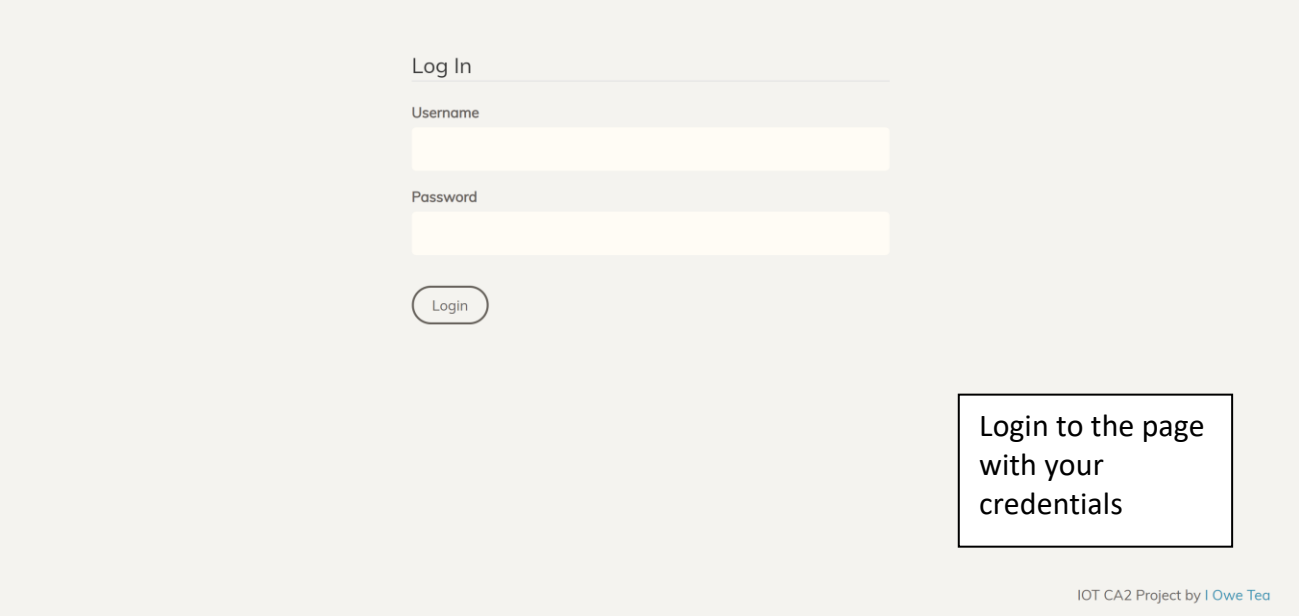
D. Final Setup



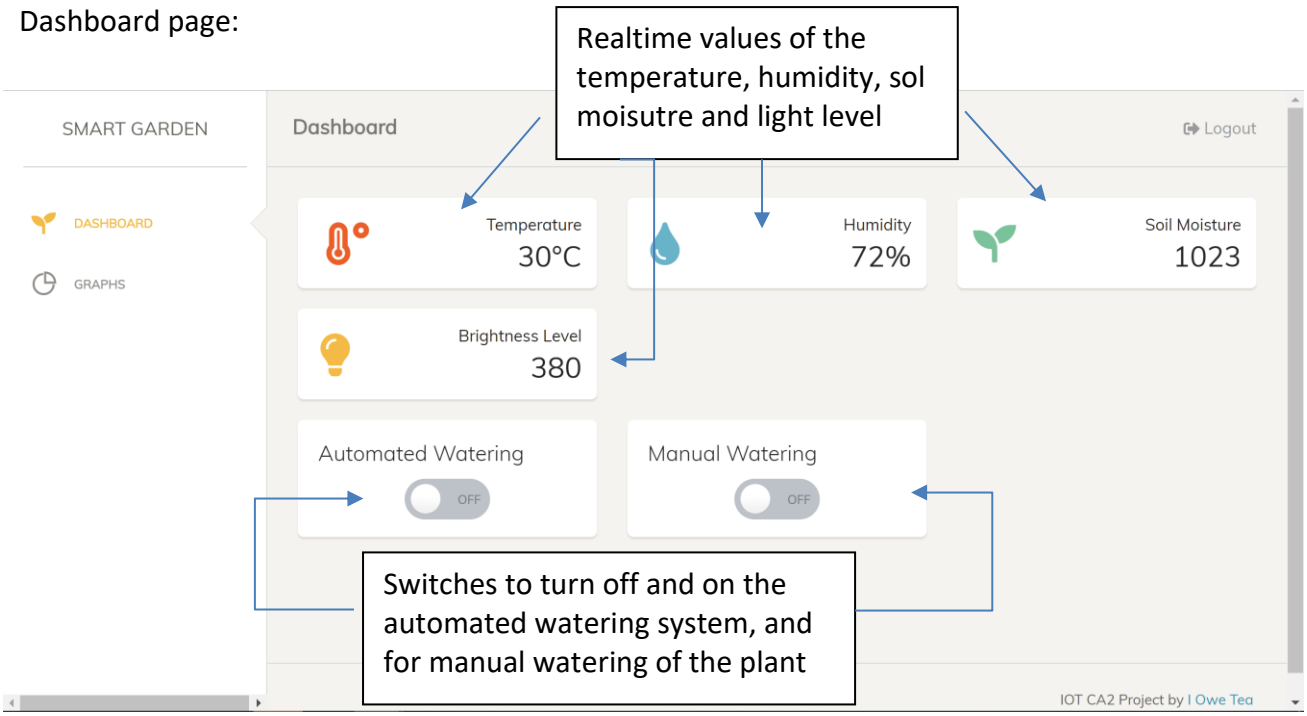
E. How does the web application look like?

Flask web app

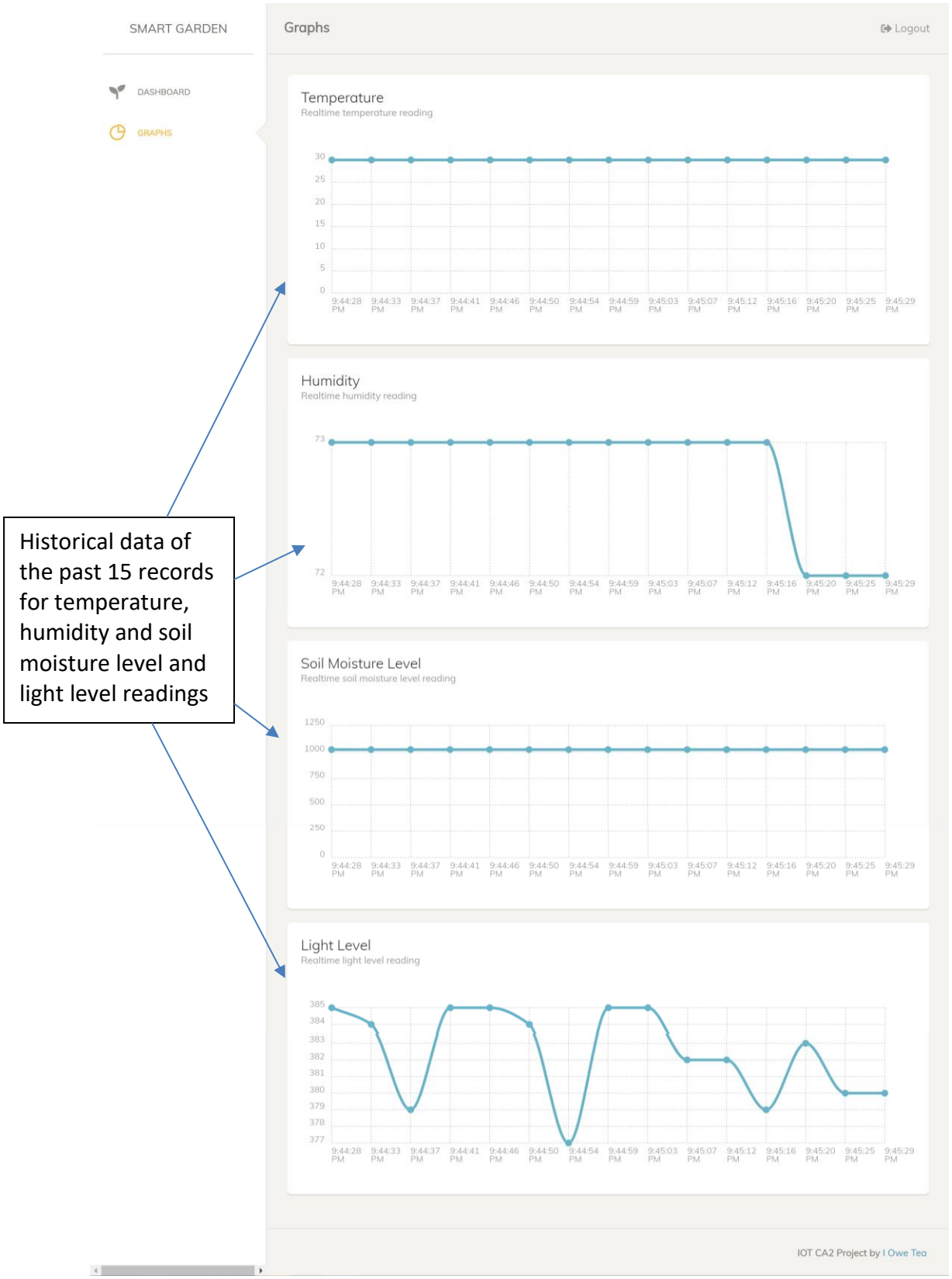
Login page:



Dashboard page:



Graphs page:


SMART GARDEN
DASHBOARD
GRAPHS
Graphs
Logout
Temperature
Realtime temperature reading
Humidity
Realtime humidity reading
Soil Moisture Level
Realtime soil moisture level reading
Light Level
Realtime light level reading
IOT CA2 Project by I Owe Tea
Historical data of the past 15 records for temperature, humidity and soil moisture level and light level readings

Section 2
Hardware Requirements

A. Hardware checklist

Smart Garden

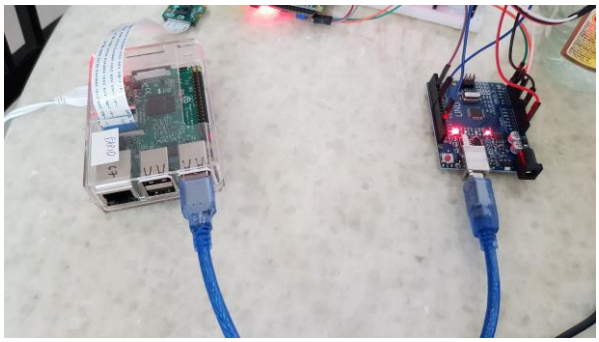
	Item	Quantity
1)	Raspberry Pi 3 Model B	1
2)	T-Cobbler kit	1
3)	Breadboard	1
4)	Arduino UNO	1
5)	DHT11 Temperature & Humidity Sensor	1
6)	Soil Moisture Sensor	1
7)	Water Pump (5V DC Motor)	1
8)	Silicone Tubes	2
9)	LED (red)	1
10)	LED (yellow)	1
11)	i2c LCD Screen (16x2)	1
12)	Light-Dependant Resistor (LDR)	1
13)	PN2222 Transistor	1
14)	1N4001 Diode	1
15)	220 Ω Resistor	3
16)	10k Ω Resistor	2
17)	Jumper wires	26
18)	Alligator jumper wires	2
19)	USB 2.0 Cable	1

Section 3
Setting up the hardware

A. Connect Arduino to Raspberry Pi

Task

a) Connect Arduino to Raspberry Pi via a USB 2.0 Cable as shown in the figure.

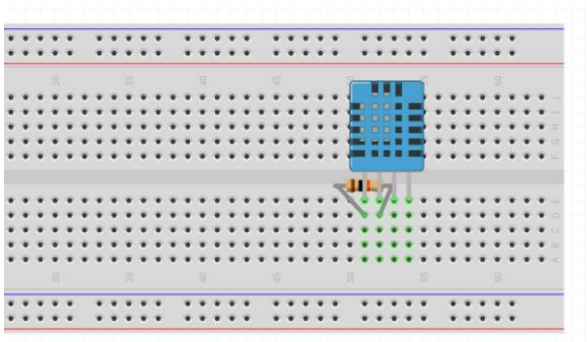


B. Connect DHT11 Sensor

Task

a) Insert the DHT11 sensor in the middle of the breadboard.

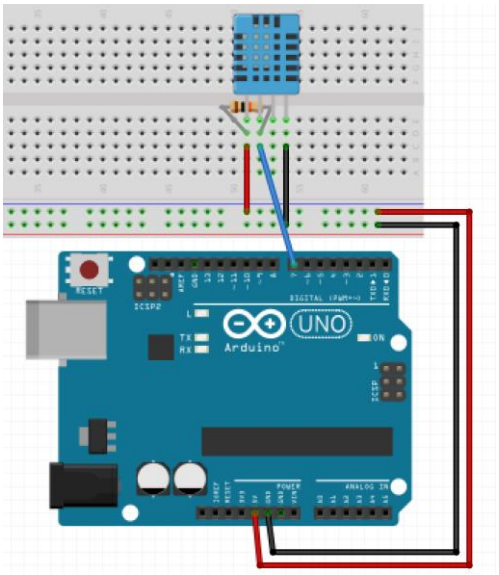
Add in a 10k ohms resistor in the DATA and VCC line as shown in the figure.



Task

b) Connect them to the Arduino pins with the corresponding color jumper cables as shown in the diagram below.

DHT11 Sensor	Arduino Pin	Jumper color
VCC	5V	Red
DATA	D7	Blue
NC		
GND	GND	Black



C. Connect LED

Task

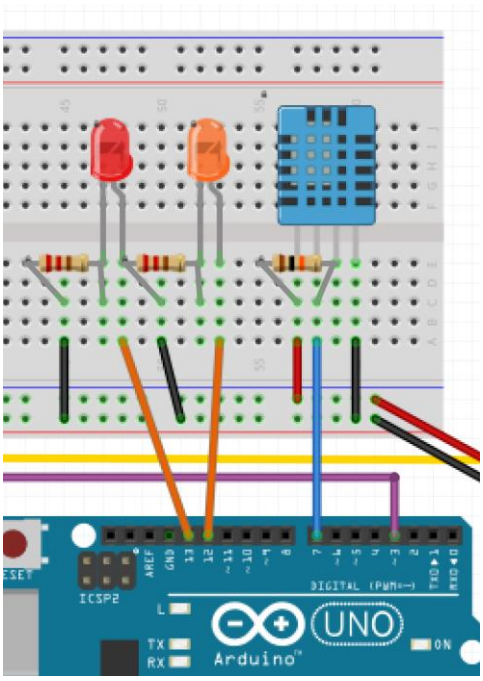
a) Insert the 2 LEDs in the middle of the breadboard.

Add in 2 220 ohms resistors for each of the LEDs, with one end connected to the longer end of the LED.

Connect them to the Arduino pins with the corresponding color jumper cables as shown in the diagram below.

LED (red)	Arduino Pin	Jumper color
Long-end	D13	Orange
Short-end	GND	Black

LED (red)	Arduino Pin	Jumper color
Long-end	D12	Orange
Short-end	GND	Black

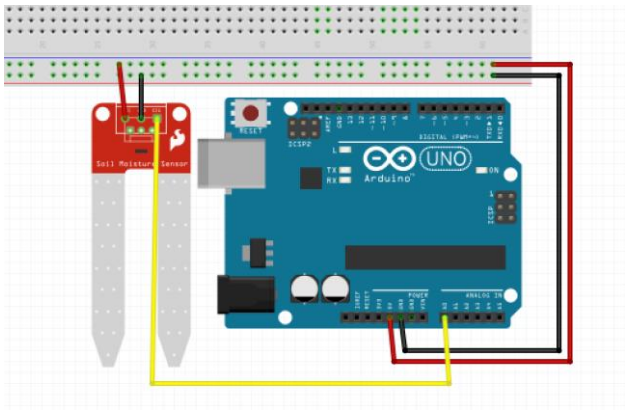


D. Connect Soil Moisture Sensor

Task

a) Connect the soil moisture sensor to the Arduino pins with the corresponding color jumper cables as shown in the diagram below.

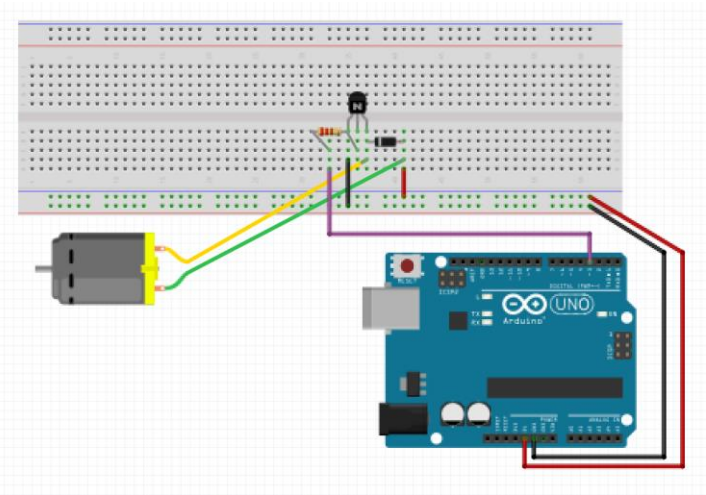
Soil Moisture Sensor	Arduino Pin	Jumper color
VCC	V5	Red
GND	GND	Black
SIG	A0	Yellow



E. Connect DC Motor

Task

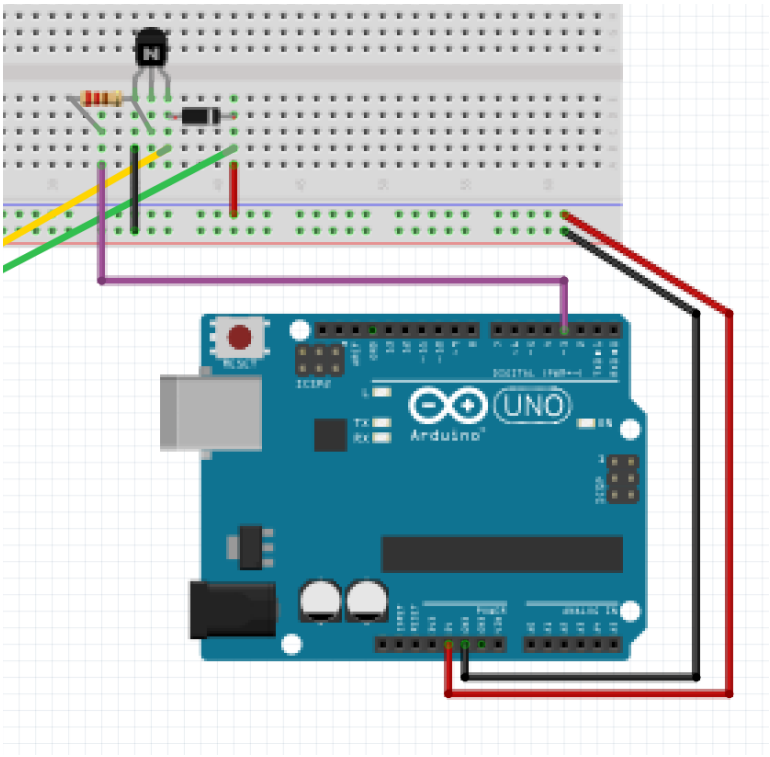
a) Connect the DC Motor to the breadboard as shown in the figure using alligator jumper cables. The motor can be connected either way around.



Task

b) Add a 220 ohms resistor in the breadboard as shown.

One end of the resistor should connect to the D3 pin of the Arduino. The other end should be connected to the base (middle pin) of the transistor.

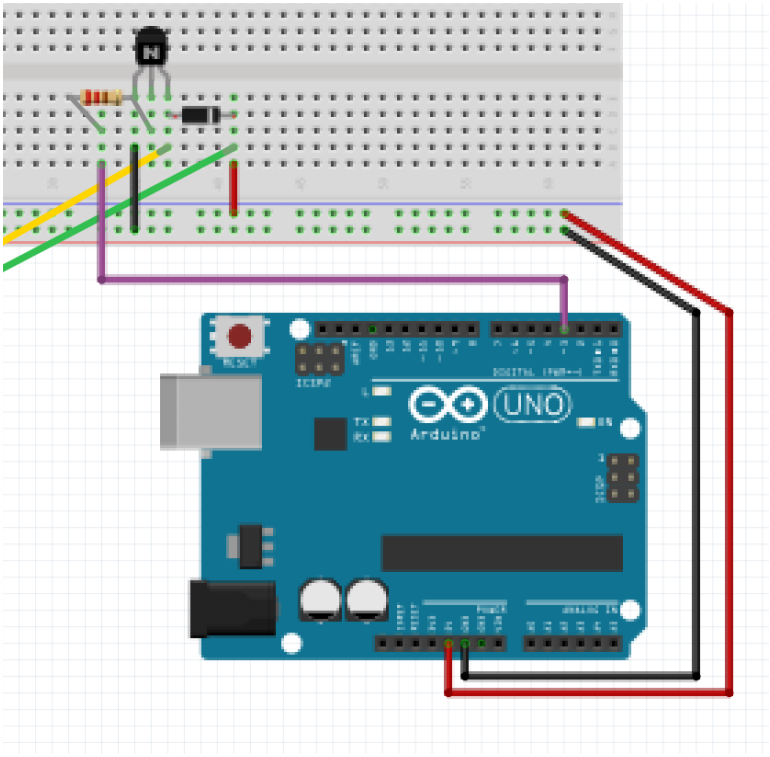


c) Add a PN2222 transistor in the breadboard as shown in the figure.

The emittor of the transistor should connect to the GND pin of the Arduino.

The base (middle pin) should connect to one end of the resistor.

The collector should connect to the cathode of the diode.

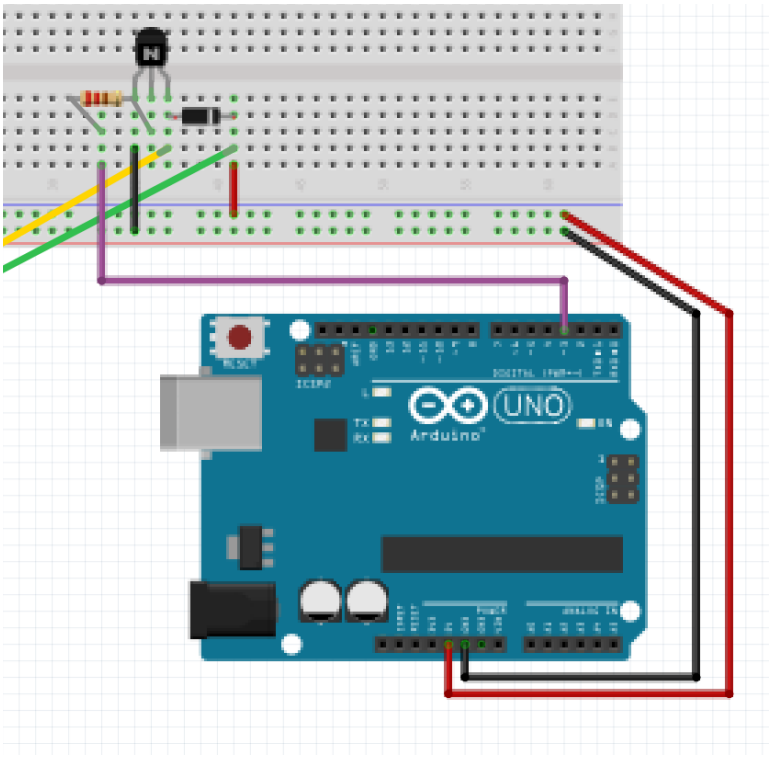


Task

d) Add a 1N4001 diode to the breadboard as shown in the figure.

The cathode (white end) of the diode should connect to one end of the motor and the 5V pin of the Arduino.

The anode pin of the diode should connect to the collector of the transistor and the other end of the DC motor.

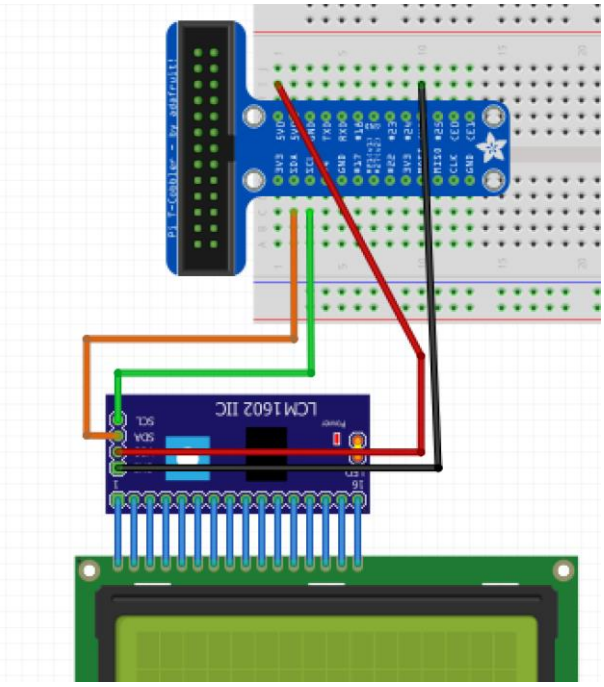


F. Connect i2c LCD

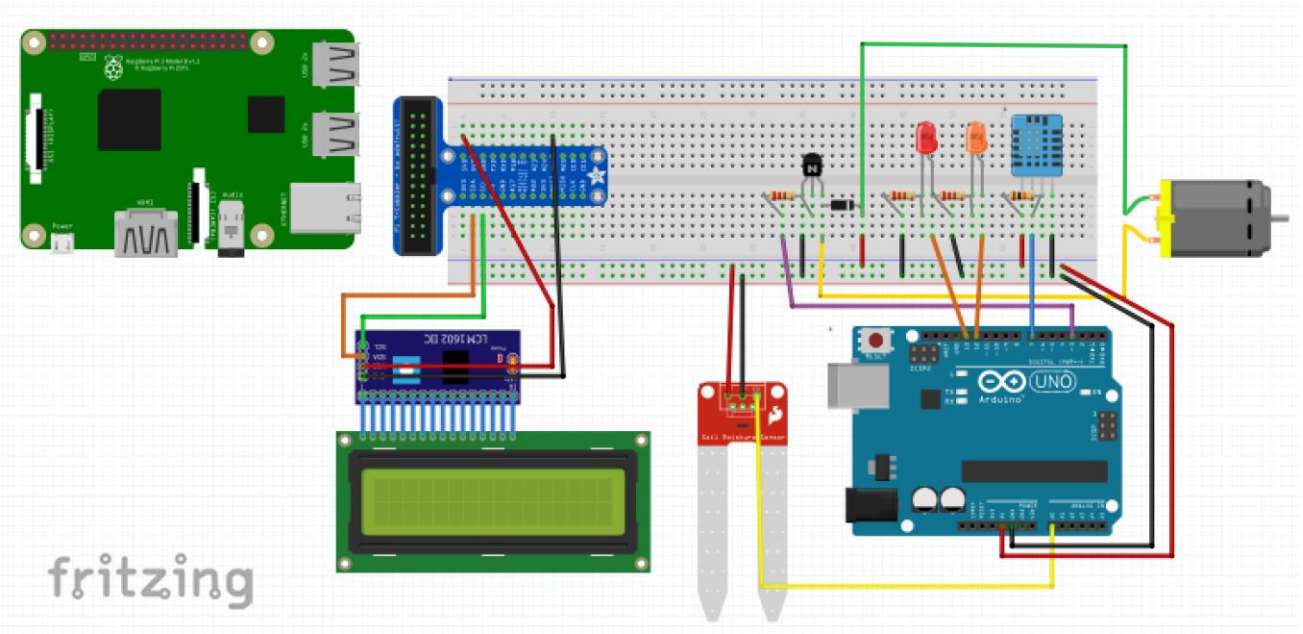
Task

a) Connect the i2c LCD screen to the RPi with the corresponding color jumper cables as shown in the diagram below.

LED	Arduino Pin	Jumper color
GND	GND	Black
VCC	V5	Red
SDA	A4	Orange
SCL	A5	Green



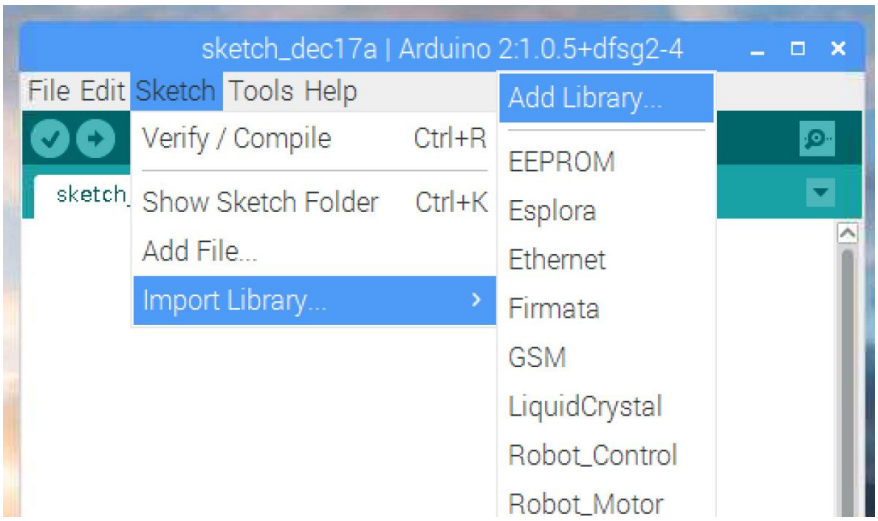
G. Completed Fritzing Diagram



Section 4
Software Setup

It is important to install and setup essential packages on the Raspberry Pi and Arduino UNO before we proceed with the programming section of the application.

A. Installing Arduino Library

Task	
a)	As we will be using the DHT11 Sensor, we will have to install the DHTLib library to the Arduino through the Raspberry Pi. The DHTLib library will be used to read the temperature and humidity values from the DHT11 and can be downloaded from: https://github.com/RobTillaart/Arduino/tree/master/libraries/DHTlib
b)	Once the zip files are downloaded, open up the **Arduino IDE** go to Sketch > Include Library > Add Library and select the DHTLib.zip files. 

B. Installing Packages & Libraries on RPi

Task

a) Install the required packages on the **RPi** by using the terminal.

```
sudo apt-get install python3-pip
sudo apt-get install python-pip
sudo pip3 install Rpi.GPIO
sudo pip install AWSIoTPythonSDK
sudo pip install paho-mqtt
sudo pip install boto3
sudo pip install awscli
sudo pip install flask
sudo pip install rpi-lcd
```

C. Prepare folders

Task

b) Create new folders for us to work with:

```
mkdir ~/smartgarden/
mkdir ~/sketchbook/smartgarden
```

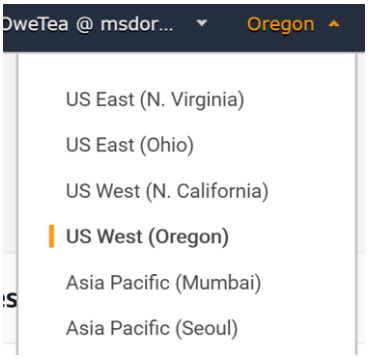
Section 5.1
Setting Up Amazon Web Service (AWS) account

A. Sign in to the AWS IoT Console

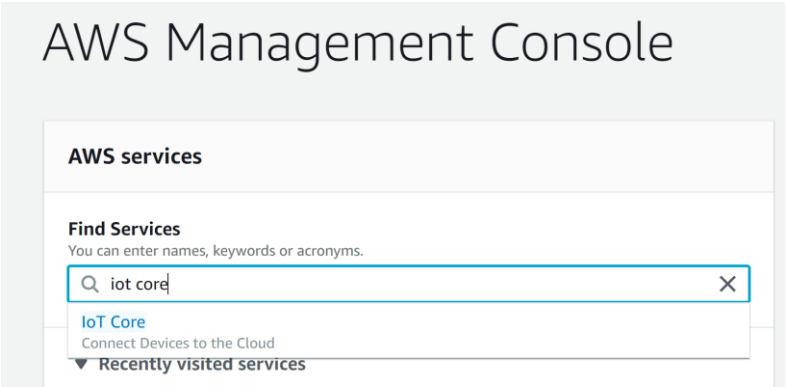
Task

a) Sign in to your AWS console at https://aws.amazon.com

b) Make sure you change your location to Oregon (us-west-2):

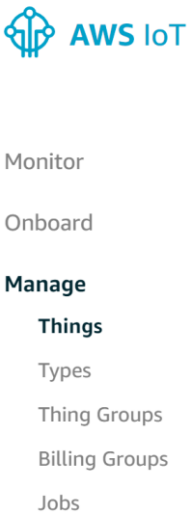
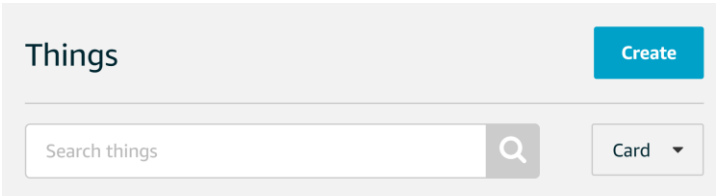
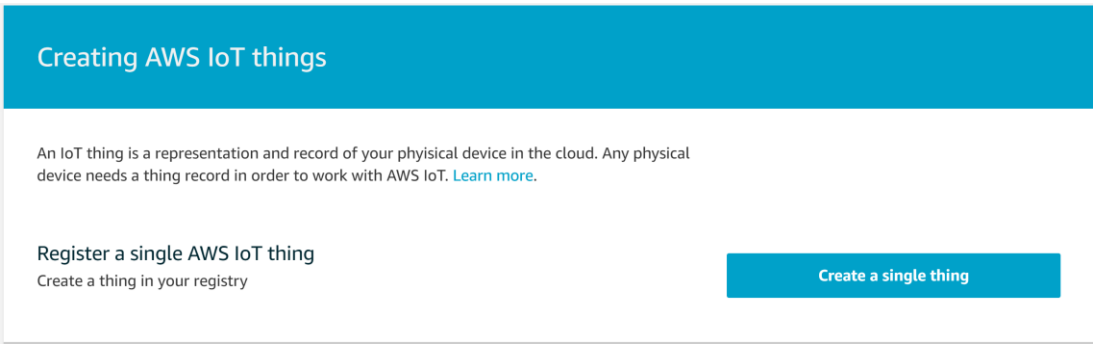


c) In the AWS Management Console, search for "IoT Core" to access the AWS IoT service.

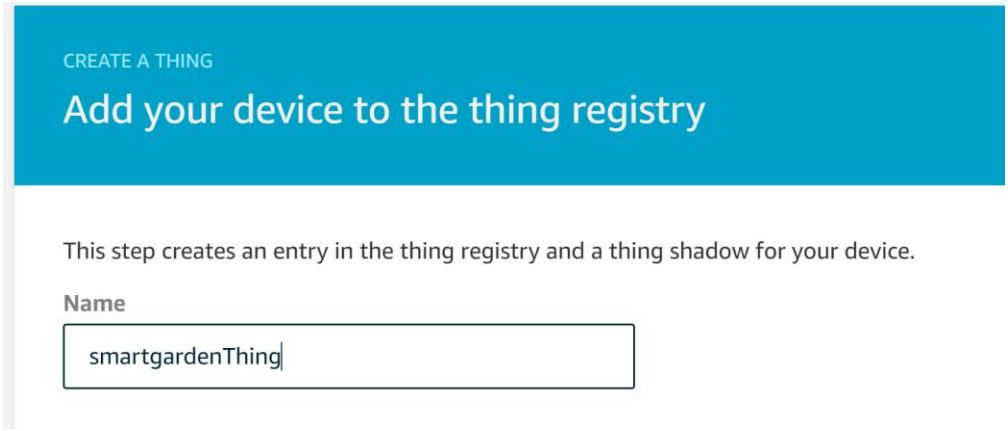


d) On the Welcome page, click on the "Get started" button.

B. Create and register your "Thing"

	Task
a)	In the left navigation bar, click the "Manage" option to expand it, and select "Things". 
b)	Click on "Create" to create a thing. 
c)	Click "Create a single thing". 
d)	Name your thing with whatever name you want, here we will name it "smartgardenThing". Then click next.

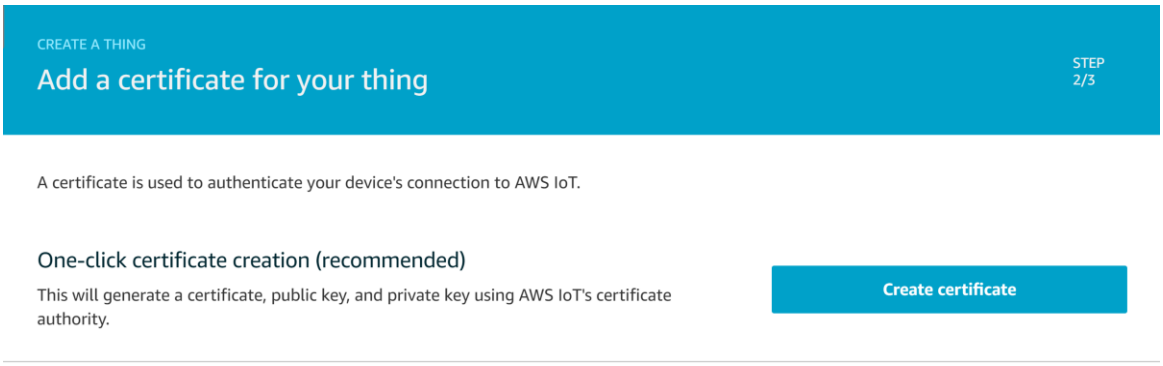
Task



C. Create Certificates

Task

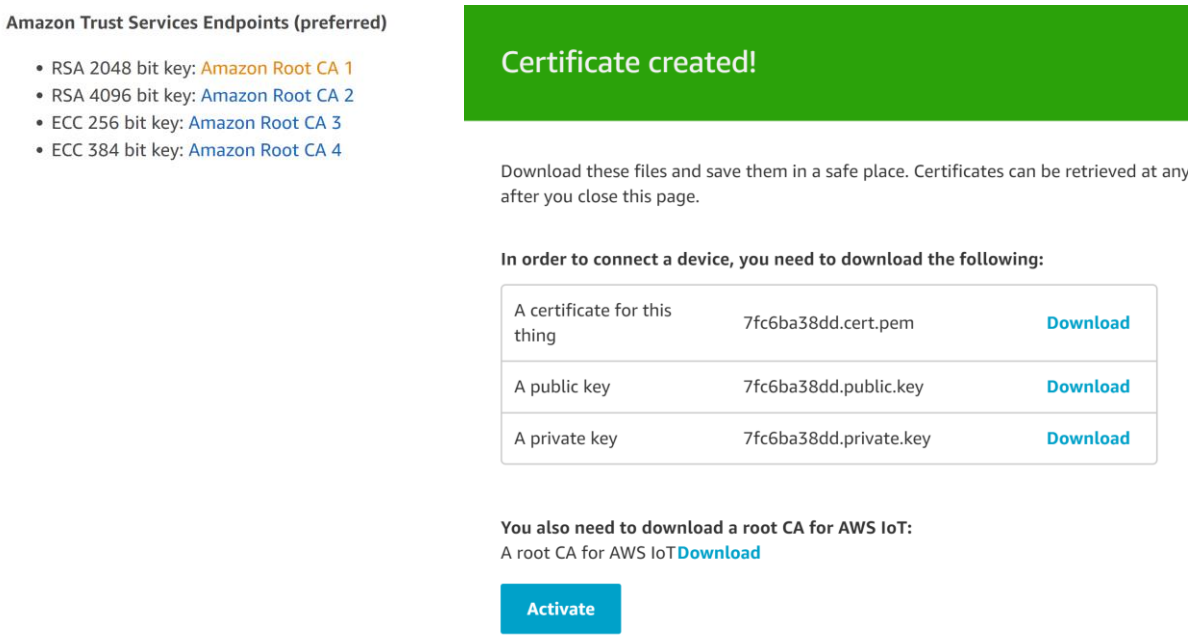
a) Next let's create a certificate for you thing by clicking the "Create certification" button for "One-click certification creation".



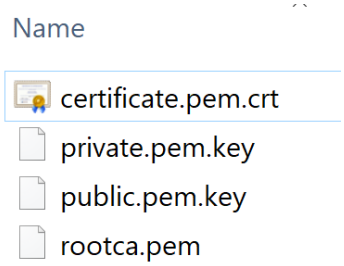
b) Your thing is now created and you would be brought to this page. You will have to download the **4** different links.

For root CA, choose Amazon Root CA1.

Task

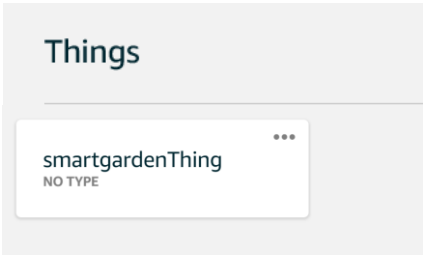


c) Create a directory called “certs” in your computer and download those files here, renaming them with the following:



d) Next, click the “Activate” button to activate your certificate.

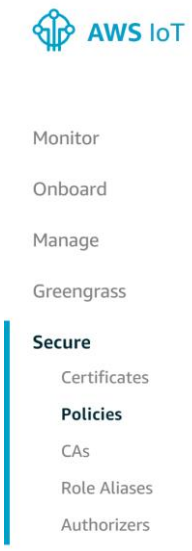
e) Once you are done, click on the “Done” button and you will be brought to this page where it shows your newly created thing



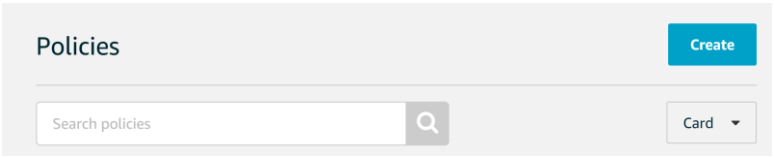
D. Create a Security Policy for you RPi

Task

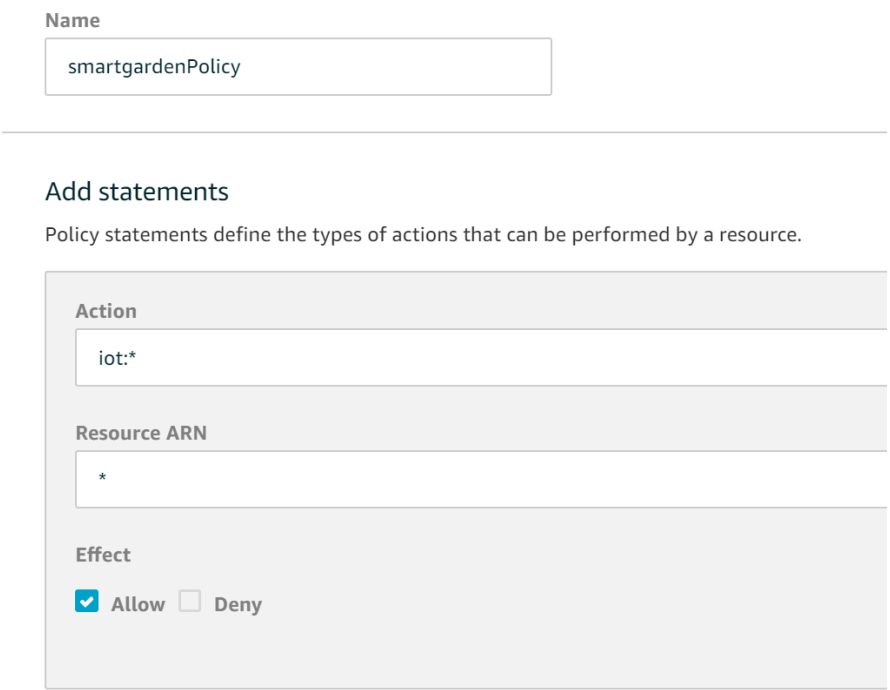
a) In the left navigation bar, click the “Secure” option to expand it, and select “Policies”.



b) Click on the “Create” button.



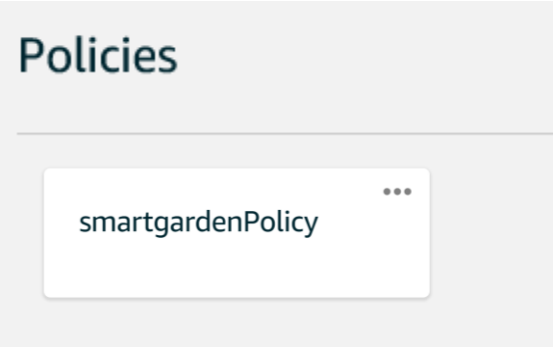
c) Fill in the fields as shown below:



Task

d) Click "**Create**".

You now have a Security Policy that allows all access to IOT Core services

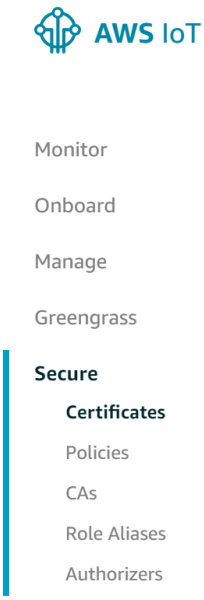


E. Attach Security Policy and Thing to your Cert

In this section, you will attach both your security policy and your Thing to your certificate

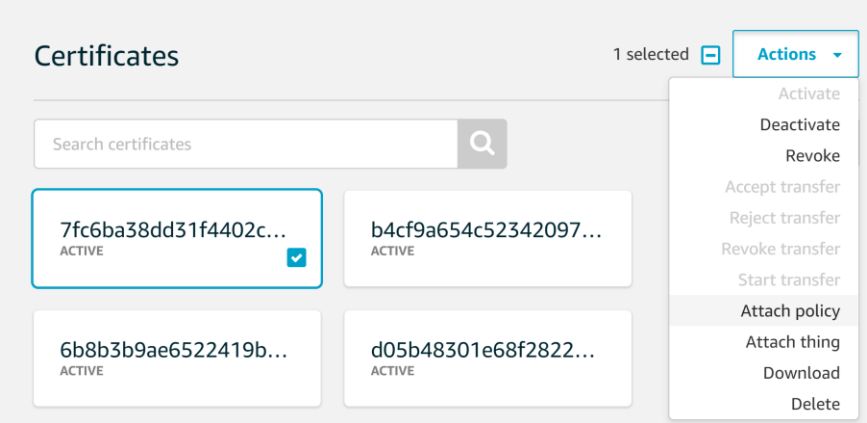
Task

a) In the left navigation bar, click the "Secure" option to expand it, and select "Certificates".

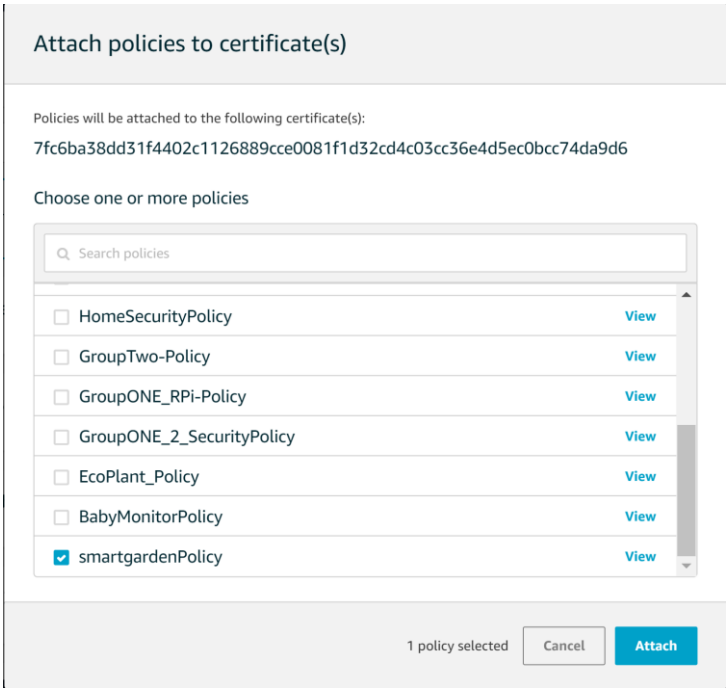


b) The certificate that you created earlier will be shown. Check on the certificate by ticking the checkbox and click on the "Actions" button and select "Attach Policy".

Task

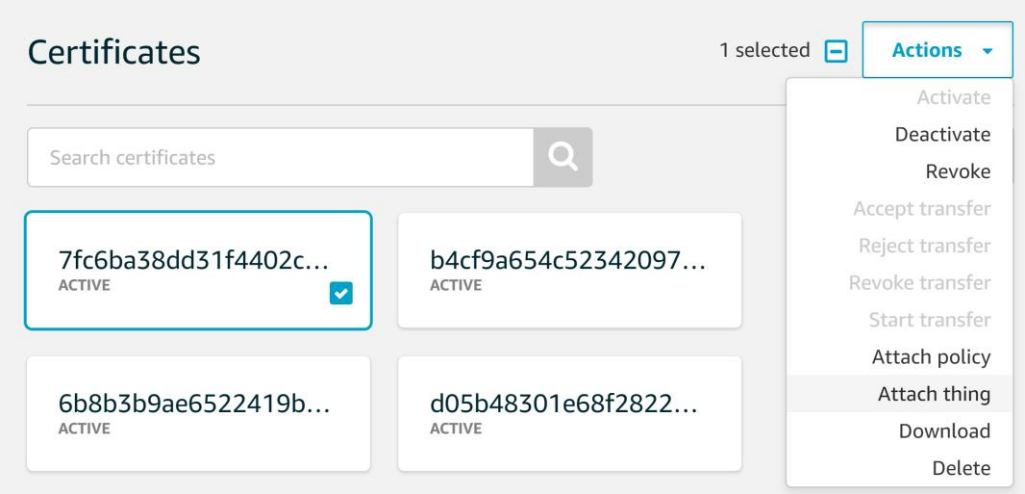


c) Select the policy you created earlier by checking the “smartgardenPolicy” and click the “Attach” button.

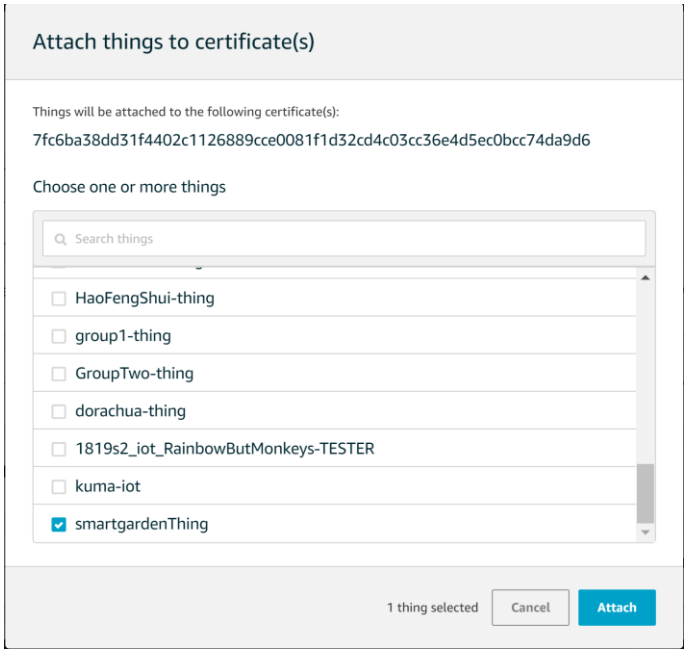


d) Next, let’s attach our “Thing” to this certificate.
Check on the certificate by ticking the checkbox and click on the “Actions” button and select “Attach Thing”

Task



e) Select the thing you created earlier by checking the "smartgardenThing" and click the "Attach" button.

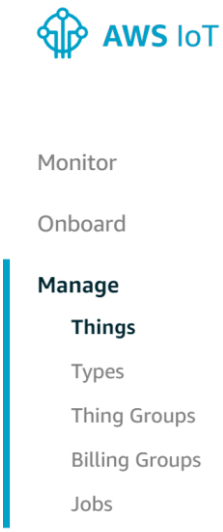


F. Save REST API endpoint

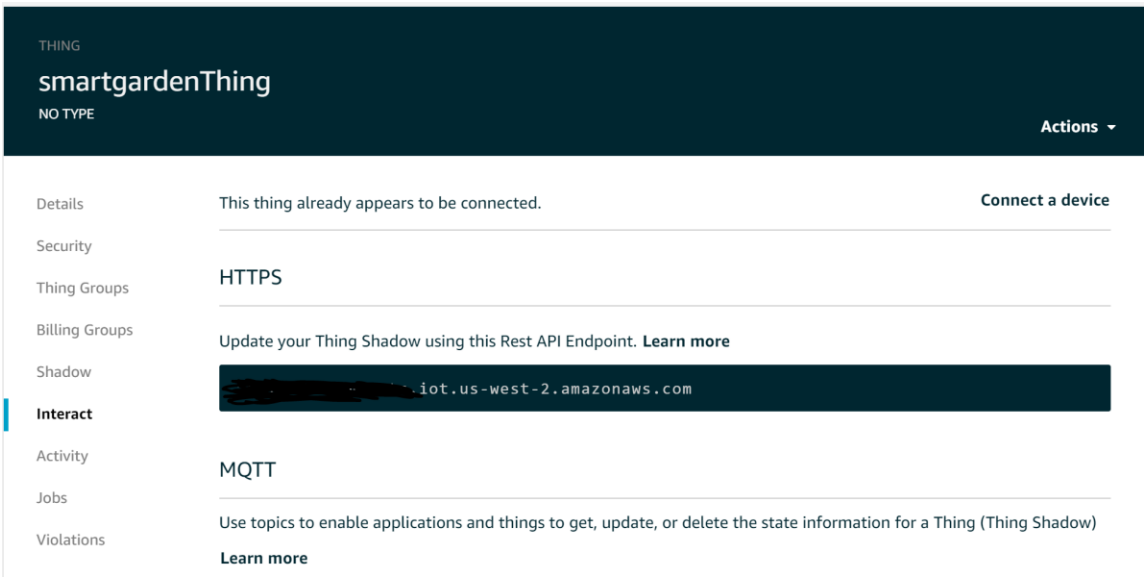
Task

a) In the left navigation bar, click the "Manage" option to expand it, and select "Things".

Task



b) Click into the Thing you created and navigate to the “Interact” tab.
Copy the REST API Endpoint and save it somewhere, you will need it later.



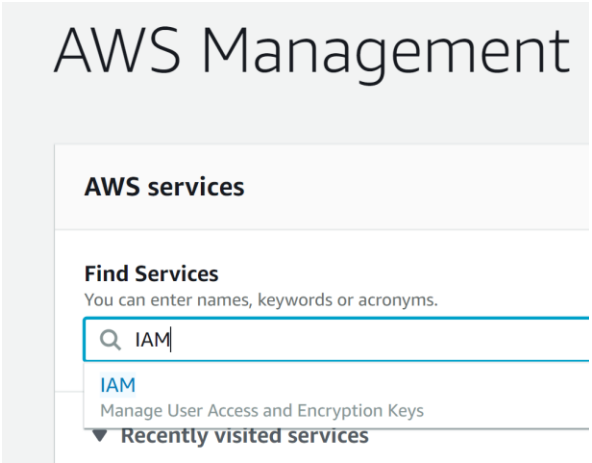
Section 5.2
Setting Up DynamoDB

A. Create AWS Role

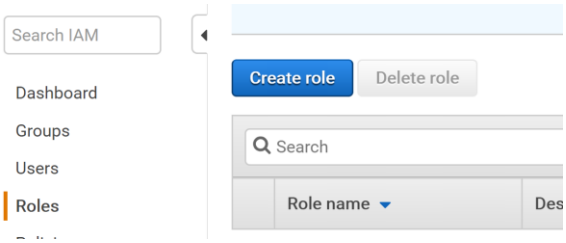
If you do not have a paid AWS account, you should continue with the steps for this section, else skip to the next.

Task

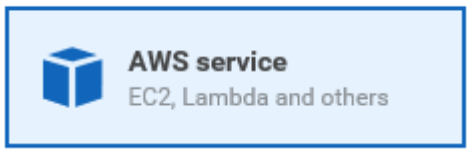
a) Back at the AWS Maganement Console, search for “IAM”.



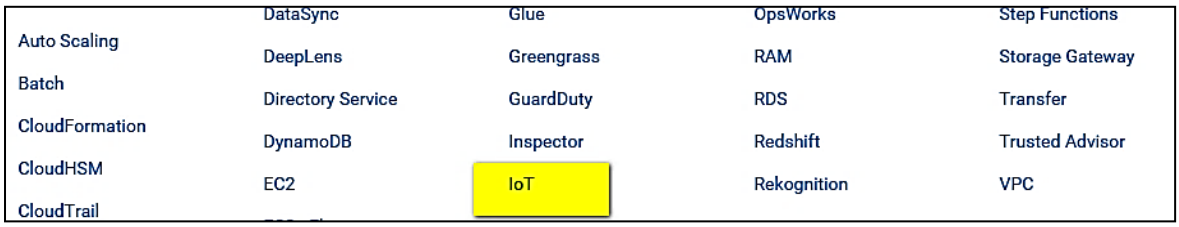
b) In the left navigation bar, select “Roles” and click on “Create role”.



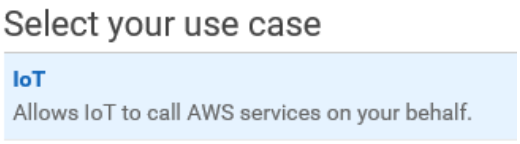
c) Next, choose “AWS service”, then “IOT”



Task



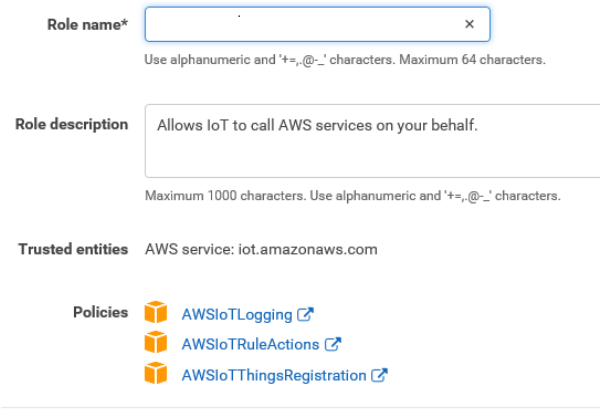
d) Under “Select your use case”, select IoT.



e) Click “Next->Permissions”. You will be brought to another page.
Do not do anything on the new page, but just click “Next->Tags”
You will be brought to another new page.
Do not do anything. Just click “Next->Review”

f) You will see a page that requires you to input a name for your Role.

Key in a rolename.

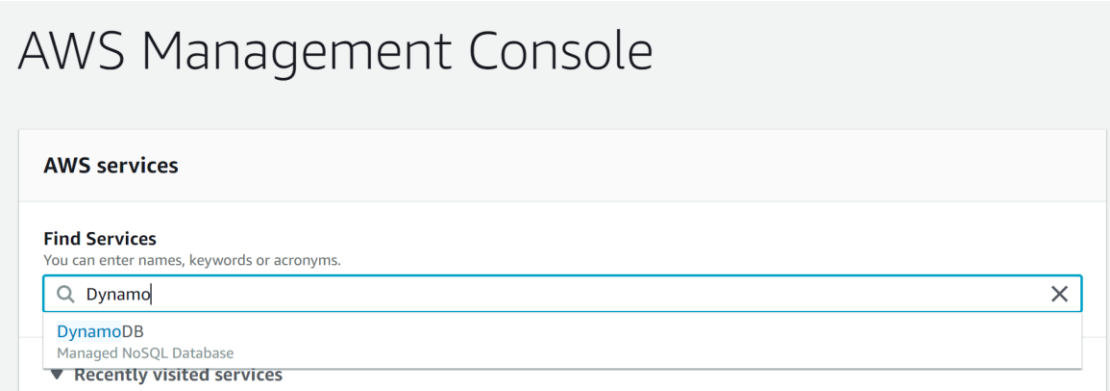


B. Create a DynamoDB table

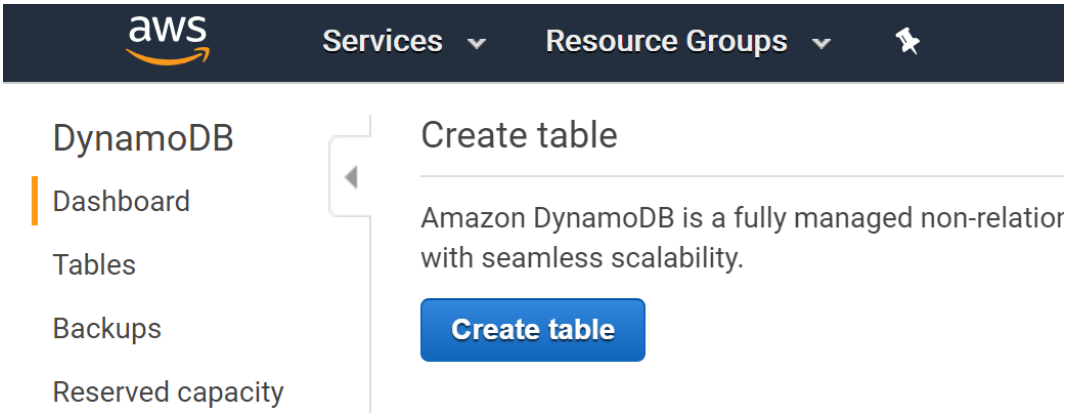
Task

a) Go back to the AWS Management Console and search for “DynamoDB”

Task



b) Click on the "Create table" button to create a new table.

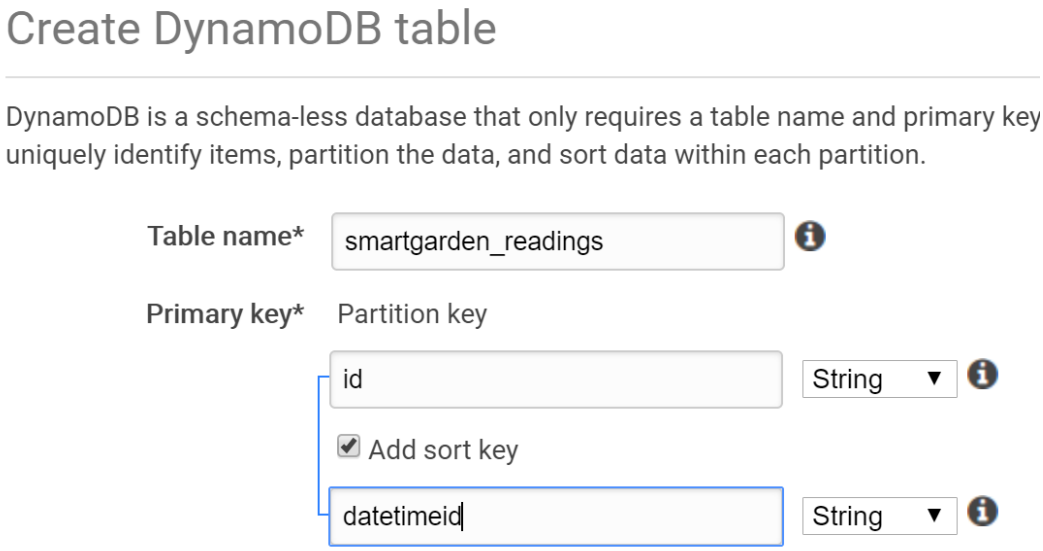


c) Create 3 tables using the attributes as shown below:

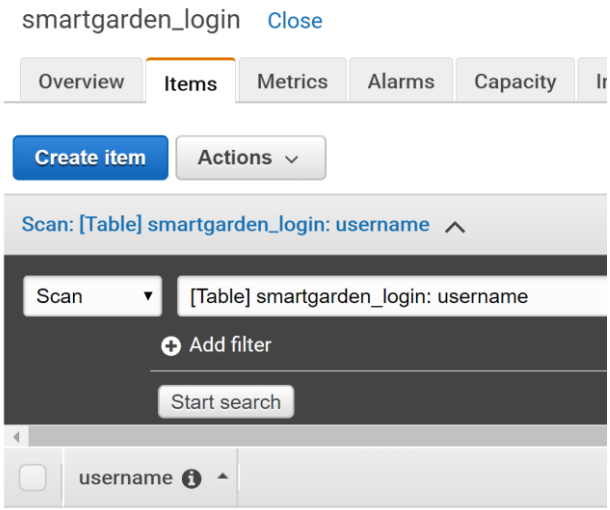
Table name	Partition key	Sort key
smartgarden_readings	id	datetimeid
smartgarden_login	username	
smartgarden_status	id	datetimeid

It will look like this:

Task

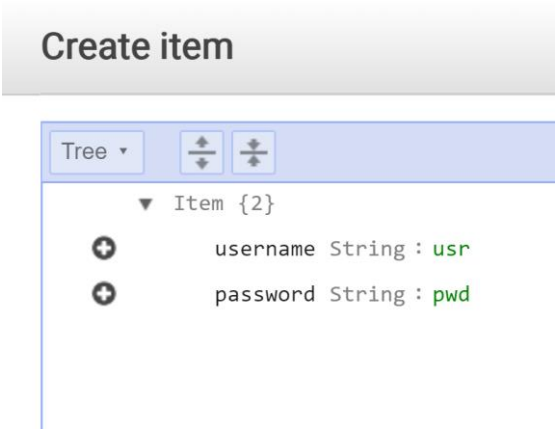


d) After creating the smartgarden_login table, click on "Create item" to create a new user for the web page.



e) Create a new user by typing in your desired username and password. For this example we will create a user with the following credentials:

username: usr
password: pwd



Task

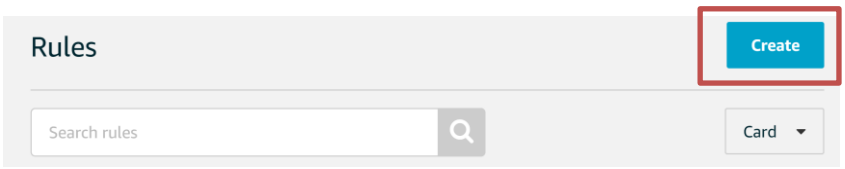
f) The item is then created and will be shown in the table:

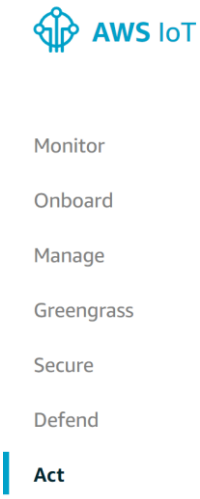
C. Create rule to publish MQTT message to DB

In this step, you will create and configure a rule to send the data received from a device to the AWS DynamoDB table you created in Step A of this section

Task

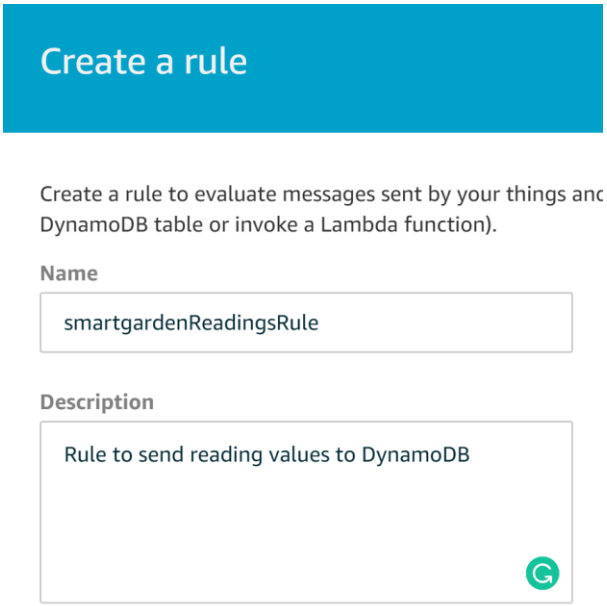
a) Going back to the AWS IoT console, in the left navigation bar, click on “Act”. Then click “Create”.



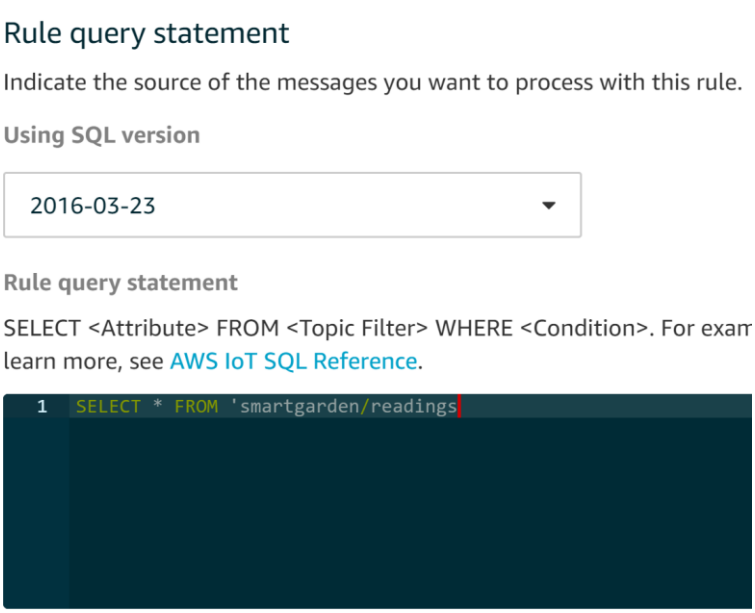


Task

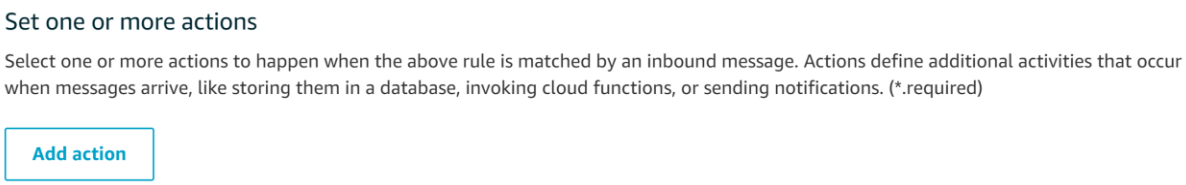
b) Type in a name and short description for your rule. Over here, we named it “smartgardenReadingsRule”, this will be for the “smartgarden_readings” database.



c) For the Rule query statement, you should select the latest SQL version and type in “SELECT * FROM ‘smartgarden/readings’” in the query statement box.



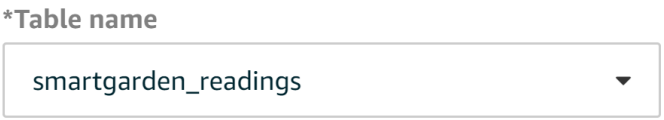
d) In Set one or more actions, choose Add action.



Task

e) Select the action to “Split message into multiple columns of a DynamoDB table (DynamoDBv2)” and click “Configure action”.

f) On the Configure action page, choose the DynamoDB table you created earlier.



g) If you are using a **AWS Paid account**, click “Create a new role”. Then select the newly created role and click “Update role”.

If you are using a **AWS Educate account**, you will not be able to create a new role. Instead, just choose the one you created in Section 8 Step A (iotlab11role) from the drop-down list and click “Update Role”

For example, we will be using the “iotrule_mon1_group_01” role.

Then click “Add action”.

h) This brings you back to the Create a Rule page. Click “Create rule” at the bottom right hand side of the screen to create the rule.

i) Create rules for the other tables as well, with the following fields:

Name	Description	SQL Query Statement	For table
smartgarden_login	Rule to login	SELECT * FROM ‘smartgarden/login’	smartgarden_login

Task

smartgarden_status	Rule to send motor status from DynamoDB	SELECT * FROM 'smartgarden/status	smartgarden_staus

Section 5.3
Configure AWS CLI

A. Configure AWS CLI

Make sure you have your AWS Access Key and Secret Access Key.

Task

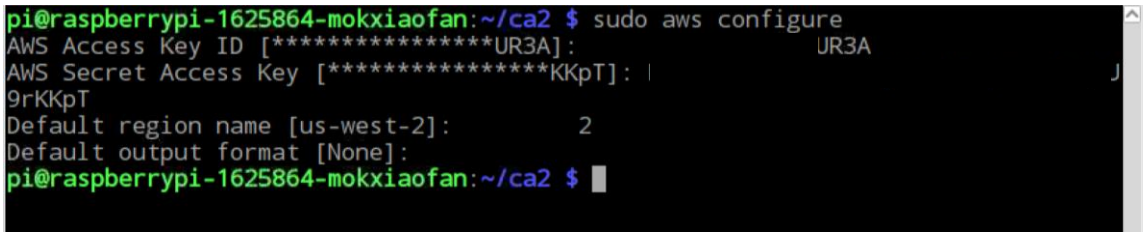
a) On your Raspberry Pi, navigate to the directory where your Python code will be stored

```
cd ~/ca2/smartgarden
```

b) Type the following command in your Raspberry Pi terminal so that you can use the AWS CLI to configure your credentials file:

```
aws configure
```

c) Enter your Access Key ID and Secret Access Key as well as region name (us-west-2 is Oregon, which was what we set at the start)



Section 6.1
Coding the Application – Smart Garden

The highlighted parts of the code are the ones you have to change according to what you have created.

A. Certifications

	Task
a)	Transfer the certifications you saved earlier (Section 5.1 C) to the folder ¬/ca2 in your RPi by using Filezilla.

B. smartgarden.ino

First, we will create an Arduino program that reads the values of the DHT11 sensor (temperature and humidity), LDR sensor (light values), and the soil moisture sensor.

Also the program will light up the red LED if the soil moisture level is too high (the higher it is , the drier the soil) and the yellow LED if the room is too dark.. Finally, the program will control the motor and automate the watering system.

It will send the values read to the RPi through serial communication to store it in the database, and receive back data from the RPi that will be used to control the motor.

	Task
b)	Open the **Arduino IDE** on the RPi and save the new file as **smartgarden.ino**. The file will be saved in the ~/sketchbook/smartgarden folder of your RPi.
c)	Copy and paste the code below to the newly created file.

```
#include <dht.h>    // dht lib

dht DHT;    // initialise dht sensor

#define DHT11_PIN 7

int soilValue = 0;    // set soil moisture value to 0
int soilPin = A0;        // set soil sensor to A0
int chk;
```

Task

```
float temp;
float hum;
int ldrValue;
int redLEDPin = 13;     // set red led to pin 13 (water)
int yellowLEDPin = 12;  // set yellow led to pin 12 (ldr)
int ldrPin = A1;        // set ldr to A1
int motorPin = 3;       // set motor to pin 3
/* 'A': auto
   'M': manual
   'O': on
   'F': off
*/
char status;
int lightLevel;

void setup() {
  Serial.begin(9600);
  //Serial.println("Soil Moisture Sensor start reading");
  pinMode(redLEDPin, OUTPUT);
  pinMode(yellowLEDPin, OUTPUT);
  pinMode(ldrPin, INPUT);
  pinMode(motorPin, OUTPUT);

  delay (2000);
  }

void loop() {
  // Receive data from server
  if (Serial.available() ) {
    status = Serial.read();
  }

  chk = DHT.read11(DHT11_PIN);
  temp = DHT.temperature;
  hum = DHT.humidity;
  soilValue = analogRead(soilPin);
  ldrValue = analogRead(ldrPin);

  Serial.println(temp);
  Serial.println(hum);
  Serial.println(soilValue);
  Serial.println(ldrValue);

  if (status == 'A') {
    if (soilValue > 500) {
      analogWrite(motorPin, 200);
      digitalWrite(redLEDPin, HIGH);
    } else {
      digitalWrite(redLEDPin, LOW);
      analogWrite(motorPin, LOW);
    }
  } else if (status == 'M' || status == 'F') {
    if (soilValue > 500) {
      analogWrite(motorPin, LOW);
      digitalWrite(redLEDPin, HIGH);
    } else {
      digitalWrite(redLEDPin, LOW);
      analogWrite(motorPin, LOW);
    }
```

Task

```
  } else if (status == 'O') {
    if (soilValue > 500) {
      digitalWrite(redLEDPin, HIGH);
    } else {
      digitalWrite(redLEDPin, LOW);
    }
    analogWrite(motorPin, 200);
  } else {
    if (soilValue > 500) {
      digitalWrite(redLEDPin, HIGH);
    } else {
      digitalWrite(redLEDPin, LOW);
    }
    analogWrite(motorPin, LOW);
  }

  if (ldrValue>=300) {
    digitalWrite(yellowLEDPin, HIGH);
  } else {
    digitalWrite(yellowLEDPin, LOW);
  }
  delay(4000);
}
```

C. aws_pubsub scripts

Next, we will create aws_pubsub scripts (aws_pubsub_readings.py, aws_pubsub_status.py) that will be used to send the readings from the sensors to the database, and receive the status of the motor controlled by the web server from the database.

Task

a) Create a **aws_pubsub_readings.py** file and copy the code below.

```
# Import SDK packages
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import serial
from rpi_lcd import LCD
from time import sleep

# Get serial to fetch data from arduino
ser = serial.Serial('/dev/ttyUSB0', 9600)
lcd = LCD()

def customCallback(client, userdata, message):
  print("Received a new message: ")
  print(message.payload)
  print("from topic: ")
  print(message.topic)
```

Task

```
  print("--------------\n\n")

host = "YOUR REST API ENDPOINT"
rootCAPath = "rootca.pem"
certificatePath = "certificate.pem.crt"
privateKeyPath = "private.pem.key"

my_rpi = AWSIoTMQTTClient("basicPubSub")
my_rpi.configureEndpoint(host, 8883)
my_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)

my_rpi.configureOfflinePublishQueueing(-1)  # Infinite offline Publish
queueing
my_rpi.configureDrainingFrequency(2)  # Draining: 2 Hz
my_rpi.configureConnectDisconnectTimeout(10)  # 10 sec
my_rpi.configureMQTTOperationTimeout(5)  # 5 sec

# Connect and subscribe to AWS IoT
my_rpi.connect()
my_rpi.subscribe("smartgarden/readings", 1, customCallback)
lcd.text("  SMART GARDEN  ", 1)
lcd.text("* Welcome back *", 2)
sleep(2)
lcd.clear()

# Publish to the same topic in a loop forever
loopCount = 0
while True:
  temp = float(ser.readline())
  hum = float(ser.readline())
  soil = int(ser.readline())
  light = int(ser.readline())

  lcd.text('Humidity: {:.2f}%'.format(hum), 1)
  lcd.text('Temp: {:.2f} C'.format(temp), 2)
  sleep(2)
  lcd.clear()

  lcd.text('Moisture: {:d}'.format(soil), 1)
  lcd.text('Light Level: {:d} C'.format(light), 2)
  sleep(2)
  lcd.clear()

  loopCount = loopCount+1
  message = {}
  message["id"] = "id_smartgarden"
  import datetime as datetime
  now = datetime.datetime.now()
  message["datetimeid"] = now.isoformat()
  message["temperature"] = temp
  message["humidity"] = hum
  message["moisture"] = soil
  message["light"] = light
  import json
  my_rpi.publish("smartgarden/readings", json.dumps(message), 1)
```

b) Create a **aws_pubsub_status.py** file and copy the code below.

Task

```
# Import SDK packages
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import boto3
from boto3.dynamodb.conditions import Key, Attr
import jsonconverter as jsonc
import serial
from time import sleep

# Get serial to fetch data from arduino
ser = serial.Serial('/dev/ttyUSB0', 9600)

def customCallback(client, userdata, message):
  print("Received a new message: ")
  print(message.payload)
  print("from topic: ")
  print(message.topic)
  print("--------------\n\n")

host = "YOUR REST API ENDPOINT"
rootCAPath = "rootca.pem"
certificatePath = "certificate.pem.crt"
privateKeyPath = "private.pem.key"

my_rpi = AWSIoTMQTTClient("basicPubSub")
my_rpi.configureEndpoint(host, 8883)
my_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)

my_rpi.configureOfflinePublishQueueing(-1)  # Infinite offline Publish
queueing
my_rpi.configureDrainingFrequency(2)  # Draining: 2 Hz
my_rpi.configureConnectDisconnectTimeout(10)  # 10 sec
my_rpi.configureMQTTOperationTimeout(5)  # 5 sec

# Connect and subscribe to AWS IoT
my_rpi.connect()
my_rpi.subscribe("smartgarden/status", 1, customCallback)
sleep(2)

# Publish to the same topic in a loop forever
loopCount = 0
while True:
  dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
  table = dynamodb.Table('YOUR SMARTGARDEN_STATUS TABLE NAME')

  response = table.query(KeyConditionExpression=Key('id').eq('id_status'),
      ScanIndexForward=False
  )

  items = response['Items']

  n=1
  data = items[:n]
  uStatus = data[0]['status']
  status = uStatus.encode('latin-1')
  print(status)
  ser.write(status)
  sleep(4)
```

Task

c) Create a **scripts.py** file and copy the code below. This script will allow you to run the other two scripts at the same time in one script.

```
from multiprocessing import Process

def script1():
  while True:
    import aws_pubsub_readings

def script2():
  while True:
    import aws_pubsub_status

if __name__ == '__main__':
  print ('Running scripts...')
  proc1 = Process(target = script1)
  proc1.start()
  print ('Reading script running...')

  proc2 = Process(target = script2)
  proc2.start()
  print ('Status script running...')

  print ('Scripts running')
```

d) Transfer the fles into the **~/smartgarden** folder in the RPi using FileZilla.

D. dynamodb.py

Next, we will create dynamodb.py where functions are defined to fetch and send data to and from the DynamoDB to the web app.

Task

a) Create a **dynamodb.py** file and copy the code below.

```
import boto3
from boto3.dynamodb.conditions import Key, Attr
import datetime as dt
from datetime import date

def login():
  try:
    dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
    table = dynamodb.Table('YOUR SMARTGARDEN_LOGIN TABLE NAME')
    response = table.scan()
```

Task

```
    items = response['Items']

    return items
  except:
    import sys
    print(sys.exc_info()[0])
    print(sys.exc_info()[1])

def get_data():
  try:
    dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
    table = dynamodb.Table('YOUR SMARTGARDEN_READINGS TABLE NAME')

    startdate = date.today().isoformat()
    response =
table.query(KeyConditionExpression=Key('id').eq('id_smartgarden') &
Key('datetimeid').begins_with(startdate),
        ScanIndexForward=False
    )

    items = response['Items']

    n=1 # get latest data
    data = items[:n]
    print(data)
    return data
  except:
    import sys
    print(sys.exc_info()[0])
    print(sys.exc_info()[1])

def get_chart_data():
  try:

    dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
    table = dynamodb.Table('YOUR SMARTGARDEN_READINGS TABLE NAME')

    startdate = date.today().isoformat()
    response =
table.query(KeyConditionExpression=Key('id').eq('id_smartgarden') &
Key('datetimeid').begins_with(startdate),
        ScanIndexForward=False
    )

    items = response['Items']

    n=15 # limit to last 15 items
    data = items[:n]
    data_reversed = data[::-1]
    return data_reversed
  except:
    import sys
    print(sys.exc_info()[0])
    print(sys.exc_info()[1])

def get_status():
  try:
    dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
    table = dynamodb.Table('YOUR SMARTGARDEN_STATUS TABLE NAME')
```

Task

```
    startdate = date.today().isoformat()
    response =
table.query(KeyConditionExpression=Key('id').eq('id_status') &
Key('datetimeid').begins_with(startdate),
        ScanIndexForward=False
    )

    items = response['Items']

    n=1
    data = items[:n]
    return data
  except:
    import sys
    print(sys.exc_info()[0])
    print(sys.exc_info()[1])

def send_status(status):
  try:
    # print("status", status)
    dynamodb = boto3.resource('dynamodb', region_name='us-west-2')
    table = dynamodb.Table('YOUR SMARTGARDEN_STATUS TABLE NAME')

    now = dt.datetime.now()
    new_item = {
      "id": "id_status",
      'datetimeid': now.isoformat(),
      'status': status
    }
    table.put_item(Item = new_item)

  except:
    import sys
    print(sys.exc_info()[0])
    print(sys.exc_info()[1])

if __name__ == "__main__":
  query_data_from_dynamodb()
```

b) Transfer the file into the **~/smartgarden** folder in the RPi using FileZilla.

E. jsonconverter.py

Next, we will create jsonconverter.py where functions are defined to convert data to json.

Task

c) Create a **jsonconverter.py** file and copy the code below.

```
from decimal import Decimal
import json
import datetime
import numpy

class GenericEncoder(json.JSONEncoder):

    def default(self, obj):
        if isinstance(obj, numpy.generic):
            return numpy.asscalar(obj)
        elif isinstance(obj, Decimal):
            return str(obj)
        elif isinstance(obj, datetime.datetime):
            return obj.strftime('%Y-%m-%d %H:%M:%S')
        elif isinstance(obj, Decimal):
            return float(obj)
        else:
            return json.JSONEncoder.default(self, obj)

def data_to_json(data):
    json_data = json.dumps(data,cls=GenericEncoder)
    # print(json_data)
    return json_data
```

d) Transfer the file into the **~/smartgarden** folder in the RPi using FileZilla.

F. Download Bootstrap Template

Task

a) For our web interface, I used the Paper Dashboard Bootstrap Template by Creative Tim and it can be downloaded from:
https://www.creative-tim.com/product/paper-dashboard

- Create 2 new folders called **templates** and **static** in a folder called **flaskapp** in your laptop inside the **~/smartgarden** folder. Copy the required files in the assets folder from the downloaded template and paste it in the static folder, this includes the css, js, img and fonts folder.
- Create the following html pages in the templates folder.
  - **dashboard.html**
  - **graph.html**
  - **login.html**

Task

- navbar.html
- template.html

- Create a **main.css** file in the static/css folder.
- Create a **main.js** file in the static/js folder.
- Delete any unnecessary files

The final folder tree directory should look like the following:

```
├──static
│   ├──css
│   │       animate.min.css
│   │       bootstrap.min.css
│   │       main.css
│   │       paper-dashboard.css
│   │       themify-icons.css
│   │
│   ├──fonts
│   │       themify.eot
│   │       themify.svg
│   │       themify.ttf
│   │       themify.woff
│   │
│   ├──img
│   │       logo.png
│   │
│   └──js
│           bootstrap-checkbox-radio.js
│           bootstrap.min.js
│           chartist.min.js
│           jquery.min.js
│           main.js
│           paper-dashboard.js
│
└──templates
       dashboard.html
       graph.html
       login.html
       navbar.html
       template.html
```

b) Copy and paste the code below into **dashboard.html**.

```
{% extends "navbar.html" %}
{% block content %}
<div class="content">
  <div class="container-fluid">
    <div class="row">
      <div class="col-lg-4 col-sm-6">
        <div class="card">
          <div class="content">
            <div class="row">
              <div class="col-xs-4">
                <div class="icon-big icon-danger text-center">
```

Task

```
                <i class="fas fa-temperature-high"></i>
              </div>
            </div>
            <div class="col-xs-8">
              <div class="numbers">
                <p>Temperature</p>
                <span id="tempValue"></span>&#176;C
              </div>
            </div>
          </div>
        </div>
      </div>
      <div class="col-lg-4 col-sm-6">
        <div class="card">
          <div class="content">
            <div class="row">
              <div class="col-xs-3">
                <div class="icon-big icon-info text-center">
                  <i class="fas fa-tint"></i>
                </div>
              </div>
              <div class="col-xs-9">
                <div class="numbers">
                  <p>Humidity</p>
                  <span id="humValue"></span>%
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
      <div class="col-lg-4 col-sm-6">
        <div class="card">
          <div class="content">
            <div class="row">
              <div class="col-xs-3">
                <div class="icon-big icon-success text-center">
                  <i class="fas fa-seedling"></i>
                </div>
              </div>
              <div class="col-xs-9">
                <div class="numbers">
                  <p>Soil Moisture</p>
                  <span id="soilValue">%</span>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
    <div class="row">
      <div class="col-lg-4 col-sm-6">
        <div class="card">
          <div class="content">
            <div class="row">
              <div class="col-xs-3">
                <div class="icon-big icon-warning text-center">
```

Task

```
                  <i class="fas fa-lightbulb"></i>
                </div>
              </div>
              <div class="col-xs-9">
                <div class="numbers">
                  <p>Brightness Level</p>
                  <span id="lightValue"></span>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>

      <div class="row">
        <div class="col-md-4">
          <div class="card">
            <div class="header">
              <h4 class="title">Automated Watering</h4>
            </div>
            <div class="content">
              <center>
                <label class="toggleBtn">
                  <input class="switch-input" id="autoSwitch"
type="checkbox" onclick="auto()" />
                  <span class="switch-label" data-on="on" data-
off="off"></span>
                  <span class="switch-handle"></span>
                </label>
              </center>
            </div>
          </div>
        </div>
        <div class="col-md-4">
          <div class="card">
            <div class="header">
              <h4 class="title">Manual Watering</h4>
            </div>
            <div class="content">
              <center>
                <label class="toggleBtn">
                  <input class="switch-input switch2-input"
id="manualSwitch" type="checkbox" onclick="manual()" />
                  <span class="switch-label switch2-label" data-on="on"
data-off="off"></span>
                  <span class="switch-handle switch2-handle"></span>
                </label>
              </center>
            </div>
          </div>
        </div>
      </div>

    </div>
  </div>
{% endblock content %}
```

Task

c) Copy and paste the code below into **graph.html**.

```
{% extends "navbar.html" %}
{% block content %}
<div class="content">
  <div class="container-fluid">

    <div class="row">
      <div class="col-md-12">
        <div class="card">
          <div class="header">
            <h4 class="title">Temperature</h4>
            <p class="category">Realtime temperature reading</p>
          </div>
          <div class="content">
            <div id="tempChart" class="ct-chart ct-major-twelfth"></div>
          </div>
        </div>
      </div>
      <div class="col-md-12">
        <div class="card">
          <div class="header">
            <h4 class="title">Humidity</h4>
            <p class="category">Realtime humidity reading</p>
          </div>
          <div class="content">
            <div id="humChart" class="ct-chart ct-major-twelfth"></div>
          </div>
        </div>
      </div>
      <div class="col-md-12">
        <div class="card">
          <div class="header">
            <h4 class="title">Soil Moisture Level</h4>
            <p class="category">Realtime soil moisture level reading</p>
          </div>
          <div class="content">
            <div id="soilChart" class="ct-chart ct-major-twelfth"></div>
          </div>
        </div>
      </div>
      <div class="col-md-12">
        <div class="card">
          <div class="header">
            <h4 class="title">Light Level</h4>
            <p class="category">Realtime light level reading</p>
          </div>
          <div class="content">
            <div id="lightChart" class="ct-chart ct-major-twelfth"></div>
          </div>
        </div>
      </div>
    </div>

  </div>
</div>
{% endblock content %}
```

Task

d) Copy and paste the code below into **login.html**.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <link rel="icon" type="image/png" sizes="96x96" href="{{
url_for('static', filename='img/logo.png') }}">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />

  {% if title %}
  <title>Smart Garden - {{ title }}</title>
  {% else %}
  <title>Smart Garden</title>
  {% endif %}

  <meta content='width=device-width, initial-scale=1.0, maximum-scale=1.0,
user-scalable=0' name='viewport' />
  <meta name="viewport" content="width=device-width" />

  <!-- Main CSS -->
  <link href="{{ url_for('static', filename='css/main.css') }}"
rel="stylesheet" />

  <!-- Bootstrap core CSS     -->
  <link href="{{ url_for('static', filename='css/bootstrap.min.css') }}"
rel="stylesheet" />

  <!-- Animation library for notifications   -->
  <link href="{{ url_for('static', filename='css/animate.min.css') }}"
rel="stylesheet" />

  <!--  Paper Dashboard core CSS    -->
  <link href="{{ url_for('static', filename='css/paper-dashboard.css') }}"
rel="stylesheet" />

  <!--  Fonts and icons     -->
  <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.6.1/css/all.css"
    integrity="sha384-
gfdkjb5BdAXd+lj+gudLWI+BXq4IuLW5IT+brZEZsLFm++aCMlF1V92rMkPaX4PP"
crossorigin="anonymous">
  <link href="https://maxcdn.bootstrapcdn.com/font-
awesome/latest/css/font-awesome.min.css" rel="stylesheet">
  <link href='https://fonts.googleapis.com/css?family=Muli:400,300'
rel='stylesheet' type='text/css'>
  <link href="{{ url_for('static', filename='css/themify-icons.css') }}"
rel="stylesheet">

  <script src="{{ url_for('static', filename='js/jquery.min.js') }}"
type="text/javascript"></script>

</head>

<body>

  <div class="wrapper">

    <div class="login">
```

Task

```
      <form class="form-signin" method="POST" action="">
        {{ form.hidden_tag() }}
        <fieldset class="form-group">
          <legend class="border-bottom mb-4">Log In</legend>

          <div class="form-group">
            {{ form.username.label(class="form-control-label") }}
            {% if form.username.errors %}
            {{ form.username(class="form-control form-control-lg is-
invalid") }}
            <div class="invalid-feedback">
              {% for error in form.username.errors %}
              <span>{{ error }}</span>
              {% endfor %}
            </div>
            {% else %}
            {{ form.username(class="form-control form-control-lg") }}
            {% endif %}
          </div>

          <div class="form-group">
            {{ form.password.label(class="form-control-label") }}
            {% if form.password.errors %}
            {{ form.password(class="form-control form-control-lg is-
invalid") }}
            <div class="invalid-feedback">
              {% for error in form.password.errors %}
              <span>{{ error }}</span>
              {% endfor %}
            </div>
            {% else %}
            {{ form.password(class="form-control form-control-lg") }}
            {% endif %}
          </div>
        </fieldset>

        <div class="form-group">
          {{ form.submit(class="btn btn-outline-info") }}
        </div>
      </form>
    </div>

    <footer class="footer-login">
      <div class="container-fluid">
        <div class="copyright pull-right">
          IOT CA2 Project by <span class="text-info">I Owe Tea</span>
        </div>
      </div>
    </footer>

  </div>

  </div>

</body>

<!--   Core JS Files   -->
<script src="{{ url_for('static', filename='js/bootstrap.min.js') }}"
type="text/javascript"></script>
```

Task

```
<!--  Checkbox, Radio & Switch Plugins -->
<script src="{{ url_for('static', filename='js/bootstrap-checkbox-
radio.js') }}"></script>

<!--  Charts Plugin -->
<script src="{{ url_for('static', filename='js/chartist.min.js')
}}"></script>

<!-- Paper Dashboard Core javascript and methods for Demo purpose -->
<script src="{{ url_for('static', filename='js/paper-dashboard.js')
}}"></script>

<!--   Main JS File   -->
<script src="{{ url_for('static', filename='js/main.js') }}"
type="text/javascript"></script>

</html>
```

e) Copy and paste the code below into **navbar.html**.

```
{% extends "template.html" %}
{% block navbar %}
<div class="sidebar" data-background-color="white" data-active-
color="warning">

  <div class="sidebar-wrapper">
    <div class="logo">
      <a href="#" class="simple-text">SMART GARDEN</a>
    </div>

    <ul class="nav">
      {% if active == 'dashboard' %}
      <li class="active">
        {% else %}
      <li>
        {% endif %}
        <a href="{{ url_for('dashboard') }}">
          <i class="fas fa-seedling"></i>
          <p>Dashboard</p>
        </a>
      </li>
      {% if active == 'graph' %}
      <li class="active">
        {% else %}
      <li>
        {% endif %}
        <a href="{{ url_for('graph') }}">
          <i class="ti-pie-chart"></i>
          <p>Graphs</p>
        </a>
      </li>

    </ul>
  </div>
</div>
```

Task

```
<div class="main-panel">
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar bar1"></span>
          <span class="icon-bar bar2"></span>
          <span class="icon-bar bar3"></span>
        </button>
        {% if active == 'dashboard' %}
        <a class="navbar-brand" href="{{ url_for('dashboard')
}}">Dashboard</a>
        {% elif active == 'graph' %}
        <a class="navbar-brand" href="{{ url_for('graph') }}">Graphs</a>
        {% endif %}
      </div>
      <div class="collapse navbar-collapse">
        <ul class="nav navbar-nav navbar-right">
          <li>
            <a href="{{ url_for('logout') }}">
              <i class="fas fa-sign-out-alt"></i>
              <p>Logout</p>
            </a>
          </li>
        </ul>

      </div>
    </div>
  </nav>

  {% block content %}{% endblock content %}

  <footer class="footer">
    <div class="container-fluid">
      <div class="copyright pull-right">
        IOT CA2 Project by <span class="text-info">I Owe Tea</span>
      </div>
    </div>
  </footer>

</div>

{% endblock navbar %}
```

f) Copy and paste the code below into **template.html**.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <link rel="icon" type="image/png" sizes="96x96" href="{{
url_for('static', filename='img/logo.png') }}">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
```

Task

```
  {% if title %}
  <title>Smart Garden - {{ title }}</title>
  {% else %}
  <title>Smart Garden</title>
  {% endif %}

  <meta content='width=device-width, initial-scale=1.0, maximum-scale=1.0,
user-scalable=0' name='viewport' />
  <meta name="viewport" content="width=device-width" />

  <!-- Main CSS -->
  <link href="{{ url_for('static', filename='css/main.css') }}"
rel="stylesheet" />

  <!-- Bootstrap core CSS     -->
  <link href="{{ url_for('static', filename='css/bootstrap.min.css') }}"
rel="stylesheet" />

  <!-- Animation library for notifications   -->
  <link href="{{ url_for('static', filename='css/animate.min.css') }}"
rel="stylesheet" />

  <!--  Paper Dashboard core CSS    -->
  <link href="{{ url_for('static', filename='css/paper-dashboard.css') }}"
rel="stylesheet" />

  <!--  Fonts and icons     -->
  <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.6.1/css/all.css"
integrity="sha384-
gfdkjb5BdAXd+lj+gudLWI+BXq4IuLW5IT+brZEZsLFm++aCMlF1V92rMkPaX4PP"
    crossorigin="anonymous">
  <link href="https://maxcdn.bootstrapcdn.com/font-
awesome/latest/css/font-awesome.min.css" rel="stylesheet">
  <link href='https://fonts.googleapis.com/css?family=Muli:400,300'
rel='stylesheet' type='text/css'>
  <link href="{{ url_for('static', filename='css/themify-icons.css') }}"
rel="stylesheet">

  <script src="{{ url_for('static', filename='js/jquery.min.js') }}"
type="text/javascript"></script>

</head>

<body>

  <div class="wrapper">

    {% block navbar %}{% endblock navbar %}

  </div>

</body>

<!--   Core JS Files   -->
<script src="{{ url_for('static', filename='js/bootstrap.min.js') }}"
type="text/javascript"></script>
```

Task

```
<!--  Checkbox, Radio & Switch Plugins -->
<script src="{{ url_for('static', filename='js/bootstrap-checkbox-
radio.js') }}"></script>

<!--  Charts Plugin -->
<script src="{{ url_for('static', filename='js/chartist.min.js')
}}"></script>

<!-- Paper Dashboard Core javascript and methods for Demo purpose -->
<script src="{{ url_for('static', filename='js/paper-dashboard.js')
}}"></script>

<!--   Main JS File   -->
<script src="{{ url_for('static', filename='js/main.js') }}"
type="text/javascript"></script>

</html>
```

g) Copy and paste the code below into **main.css**.

```
/* Dashboard CSS */
/* login */
#login {
  display: -ms-flexbox;
  -ms-flex-align: center;
  align-items: center;
  padding-top: 5px;
  padding-bottom: 40px;
  background-color: #ffffff;
}

.form-signin {
  width: 100%;
  max-width: 500px;
  padding: 50px 15px;
  margin: auto;
  font-weight: 400;
}
.form-signin .form-control {
  position: relative;
  box-sizing: border-box;
  height: auto;
  padding: 10px;
  font-size: 16px;
}

.footer-login {
  position: fixed!important;
  bottom: 15px;
  right: 15px;
}

/* Toggle buttons */
.toggleBtn {
  position: relative;
  display: block;
  width: 90px;
```

Task

```
  height: 40px;
  padding: 3px;
  bottom: 5px;
  border-radius: 30px;
}
.switch-input, .icons {
  position: absolute;
  width: 0;
  height: 0;
  opacity: 0;
}
.switch-label {
  position: relative;
  display: block;
  height: inherit;
  font-size: 12px;
  text-transform: uppercase;
  background: #bdc1c8;
  border-radius: inherit;
  cursor: pointer;
}
.switch-label:before, .switch-label:after {
  position: absolute;
  top: 30%;
  -webkit-transition: inherit;
  -moz-transition: inherit;
  -o-transition: inherit;
  transition: inherit;
}
.switch-label:before {
  content: attr(data-off);
  right: 9px;
  color: #6b7381;
}
.switch-label:after {
  content: attr(data-on);
  left: 9px;
  color: #FFFFFF;
}
.switch-input:checked ~ .switch-label {
  background: #29b5a8;
}
.switch2-input:checked ~ .switch2-label {
  background: #ff8300;
}
.switch-input:checked ~ .switch-label:before {
  opacity: 0;
}
.switch-input:checked ~ .switch-label:after {
  opacity: 1;
}
.switch-handle {
  position: absolute;
  top: 8px;
  left: 9px;
  width: 29px;
  height: 29px;
  background: white;
  background-image: -webkit-linear-gradient(top, #FFFFFF 40%, #f0f0f0);
```

Task

```
  border-radius: 100%;
}
.switch-input:checked ~ .switch-handle {
  left: 52px;
  background-color: #29b5a8;
  box-shadow: 0 0 1px #29b5a8;
}

.switch2-input:checked ~ .switch2-handle {
  background-color: #ff8300;
  box-shadow: 0 0 1px #ff8300;
}

input[type="checkbox"]:disabled + .switch-label,
input[type="checkbox"]:disabled + .switch-handle
{
    filter: contrast(70%);
  cursor: not-allowed;
}

/* Transition
========================== */
.switch-label, .switch-handle {
  transition: All 0.4s ease;
  -webkit-transition: All 0.4s ease;
  -moz-transition: All 0.4s ease;
  -o-transition: All 0.4s ease;
}

/* Slider */
*, *:before, *:after {
  box-sizing: border-box;
}
.slidecontainer {
  margin: 20px 20px 11px;
}
.slidecontainer {
  width: 100%;
}
.slider {
  -webkit-appearance: none;
  width: 70%!important;
  height: 10px;
  border-radius: 5px;
  background: #d7dcdf;
  outline: none;
  padding: 0;
  margin: 0;
  display: inline-block!important;
}
.slider::-webkit-slider-thumb {
  -webkit-appearance: none;
  appearance: none;
  width: 20px;
  height: 20px;
  border-radius: 50%;
  background: #2c3e50;
  cursor: pointer;
  transition: background 0.15s ease-in-out;
```

Task

```
}
.slider::-webkit-slider-thumb:hover {
  background: #1abc9c;
}
.slider:active::-webkit-slider-thumb {
  background: #1abc9c;
}
.slider::-moz-range-thumb {
  width: 20px;
  height: 20px;
  border: 0;
  border-radius: 50%;
  background: #2c3e50;
  cursor: pointer;
  transition: background 0.15s ease-in-out;
}
.slider::-moz-range-thumb:hover {
  background: #1abc9c;
}
.slider:active::-moz-range-thumb {
  background: #1abc9c;
}
.slider:focus::-webkit-slider-thumb {
  box-shadow: 0 0 0 3px #fff, 0 0 0 6px #1abc9c;
}
.sliderValue {
  display: inline-block;
  position: relative;
  /* width: 60px; */
  color: #fff;
  line-height: 20px;
  text-align: center;
  border-radius: 3px;
  background: #2c3e50;
  padding: 5px 10px;
  margin-left: 8px;
  font-size: 15px;
}
.sliderValue:after {
  position: absolute;
  top: 8px;
  left: -7px;
  width: 0;
  height: 0;
  border-top: 7px solid transparent;
  border-right: 7px solid #2c3e50;
  border-bottom: 7px solid transparent;
  content: '';
}
::-moz-range-track {
  background: #d7dcdf;
  border: 0;
}
input::-moz-focus-inner, input::-moz-focus-outer {
  border: 0;
}
```

h) Copy and paste the code below into **main.js**.

Task

```
////////////////////////// Automated Watering //////////////////////////
const autoSwitch = document.getElementById("autoSwitch");
const manualSwitch = document.getElementById("manualSwitch");

function getStatus() {
  jQuery.ajax({
    url: "/api/status",
    type: "POST",
    success: function (ndata) {
      // console.log(ndata[0].status);
      status = ndata[0].status;
      if (status == "A") {
        autoSwitch.checked = true;
        manualSwitch.disabled = true;
        manualSwitch.checked = false;
      } else if (status == "M" || status == "F") {
        autoSwitch.checked = false;
        manualSwitch.checked = false;
      } else if (status == "O") {
        autoSwitch.checked = false;
        manualSwitch.checked = true;
      } else {
        autoSwitch.checked = true;
        manualSwitch.disabled = true;
        manualSwitch.checked = false;
      }
    }
  })
}

function auto() {
  let autoStatus;
  if (autoSwitch.checked) {
    autoStatus = "A";
    manualSwitch.disabled = true;
    manualSwitch.checked = false;
  } else {
    autoStatus = "M";
    manualSwitch.disabled = false;
  }
  // console.log(autoStatus);

  $.ajax({
    url: "changeStatus/" + autoStatus
  })

}

function manual() {
  let manualStatus;
  if (manualSwitch.checked) {
    manualStatus = "O";
  } else {
    manualStatus = "F";
  }
  // console.log(manualStatus);
  $.ajax({
    url: "changeStatus/" + manualStatus
  })
```

Task

```
}

////////////////////////// Get readings ////////////////////////
function getData() {
  jQuery.ajax({
    url: "/api/getData",
    type: "POST",
    success: function (ndata) {
      console.log(ndata);
      tempValue = ndata[0].temperature;
      humValue = ndata[0].humidity;
      soilValue = ndata[0].moisture;
      lightValue = ndata[0].light;

      $('#tempValue').html(tempValue);
      $('#humValue').html(humValue);
      $('#soilValue').html(soilValue);
      $('#lightValue').html(lightValue);
    }
  })
}

////////////////////// Get Chart data //////////////////////
function getChartData() {
  jQuery.ajax({
    url: "/api/getChartData",
    type: "POST",
    success: function (ndata) {
      // console.log(ndata)
      const chartData = ndata;
      // console.log("Getting Chart data")

      let tempArr = [];
      let humArr = [];
      let soilArr = [];
      let lightArr = [];
      let timeArr = [];

      chartData.forEach((e) => {
        tempArr.push(e.temperature);
        humArr.push(e.humidity);
        soilArr.push(e.moisture);
        lightArr.push(e.light);

        let datetime = e.datetimeid;
        // console.log(datetime);
        jsdatetime = new Date(Date.parse(datetime));
        jstime = jsdatetime.toLocaleTimeString();
        timeArr.push(jstime);
      })

      createGraph(tempArr, timeArr, '#tempChart');
      createGraph(humArr, timeArr, '#humChart');
      createGraph(soilArr, timeArr, '#soilChart');
      createGraph(lightArr, timeArr, '#lightChart');

    }
  })
}
```

Task

```
// Charts
function createGraph(data, newTime, newChart) {

  let chartData = {
    labels: newTime,
    series: [data]
  };
  // console.log(chartData);

  let options = {
    axisY: {
      onlyInteger: true
    },
    fullWidth: true,
    width: '100%',
    height: '100%',
    lineSmooth: true,
    chartPadding: {
      right: 50
    }
  };

  new Chartist.Line(newChart, chartData, options);

}

////////////////////// run functions //////////////////////
$(document).ready(function () {
  getData();
  getStatus();
  getChartData();

  setInterval(function () {
    getData();
    getChartData();
  }, 5000);
})
```

i) Transfer the **flaskapp** folder into the **~/smartgarden** folder in the RPi using FileZilla.

G. Server files

Lastly, we will create the files that will act as a server for the Flask app. We will be structuring the files in packages to make it more organised.

Task

a) Create a **server.py** file and copy the code below.

```
from sg import app

if __name__ == '__main__':
  app.run(debug=True, host='0.0.0.0')
```

b) Create a **__init__.py** file and copy the code below.

```
from flask import Flask
import os

app = Flask(__name__)
app.secret_key = os.urandom(12)

from sg import routes
```

c) Create a **form.py** file and copy the code below.

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')
```

d) Create a **routes.py** file and copy the code below.

```
from flask import render_template, url_for, redirect, request, Response,
jsonify, session, flash
import dynamodb
import jsonconverter as jsonc
import sys

from sg.forms import LoginForm
from sg import app

# login
@app.route("/login", methods=['GET', 'POST'])
def login():
  if session.get('logged_in'):
    return redirect(url_for('dashboard'))
  else:
    form = LoginForm()
    if form.validate_on_submit():
      data = dynamodb.login()
      for d in data:
        if form.username.data == d['username'] and form.password.data ==
d['password']:
          session['logged_in'] = True
```

Task

```
          return redirect(url_for('dashboard'))
        else:
          flash('Login Unsuccessful. Please check username and password',
'danger')
  return render_template('login.html', title='Login', form=form)

# logout
@app.route("/logout")
def logout():
  session.pop('logged_in', None)
  return redirect(url_for('login'))

# pages
@app.route("/")
@app.route("/dashboard")
def dashboard():
  if not session.get('logged_in'):
    return redirect(url_for('login'))
  else:
    return render_template('dashboard.html', title='Dashboard',
active='dashboard')

@app.route("/graph")
def graph():
  if not session.get('logged_in'):
    return redirect(url_for('login'))
  else:
    return render_template('graph.html', title='Graph', active='graph')

# api routes
@app.route("/api/getData", methods=['POST', 'GET'])
def api_getData():
  if request.method == 'POST':
    try:
      data = jsonc.data_to_json(dynamodb.get_data())
      loaded_data = jsonc.json.loads(data)
      # print(loaded_data)
      return jsonify(loaded_data)
    except:
      print(sys.exc_info()[0])
      print(sys.exc_info()[1])
      return None

@app.route("/api/getChartData", methods=['POST', 'GET'])
def api_getChartData():
  if request.method == 'POST':
    try:
      data = jsonc.data_to_json(dynamodb.get_chart_data())
      loaded_data = jsonc.json.loads(data)
      # print(loaded_data)
      return jsonify(loaded_data)
    except:
      print(sys.exc_info()[0])
      print(sys.exc_info()[1])
      return None

@app.route("/api/status", methods=['GET', 'POST'])
def status():
  try:
```

Task

```
    data = jsonc.data_to_json(dynamodb.get_status())
    loaded_data = jsonc.json.loads(data)
    # print(loaded_data)
    return jsonify(loaded_data)

    status = loaded_data[0].status

    return status
  except:
    print(sys.exc_info()[0])
    print(sys.exc_info()[1])
    return None

@app.route("/changeStatus/<status>")
def changeStatus(status):
  try:
    dynamodb.send_status(status)

    return status
  except:
    print(sys.exc_info()[0])
    print(sys.exc_info()[1])
    return None
```

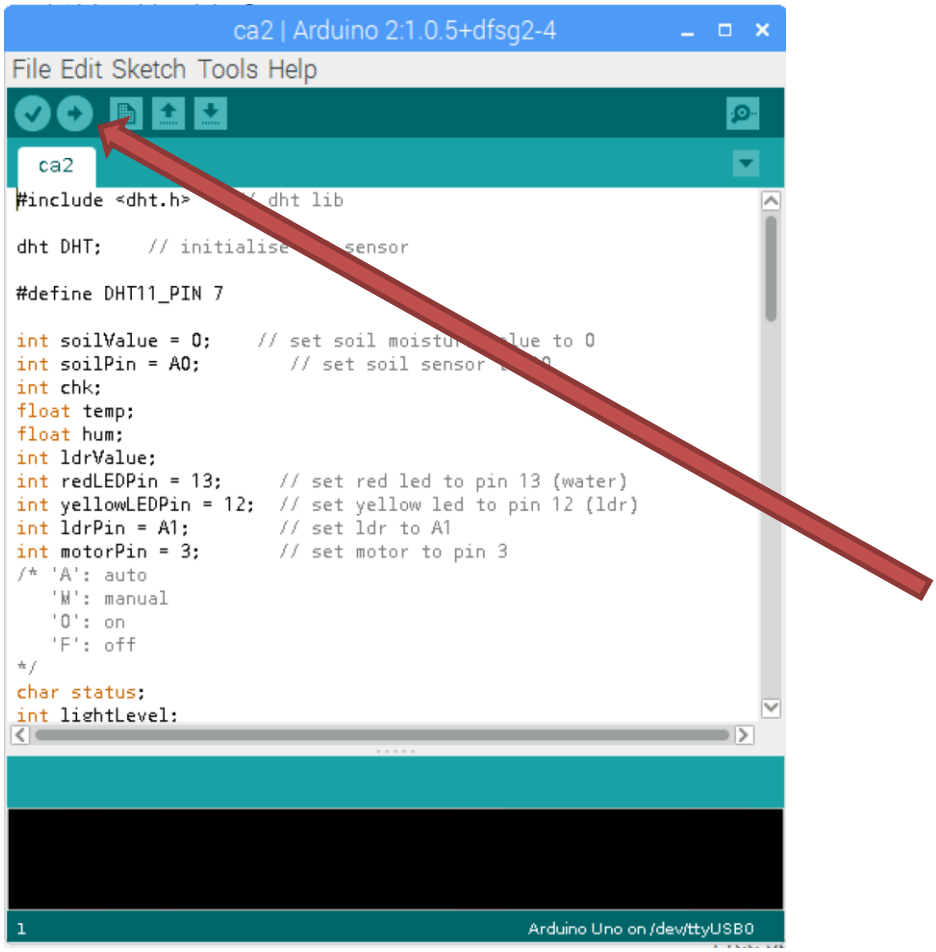
e) From your laptop, transfer the **server.py** file into the **~/smartgarden folder**, and the **__init__.py**, **routes.py** and **forms.py** files into the **~/smartgarden/flaskapp** folder in the RPi using FileZilla.

Section 6.1
Running the Application – Smart Garden

A. Run Arduino code

Task

a) From the Arduino IDE, upload the smartgarden.ino code by pressing the upload button. Once it has been uploaded your hardware should start working.



B. Run scripts.py

Task

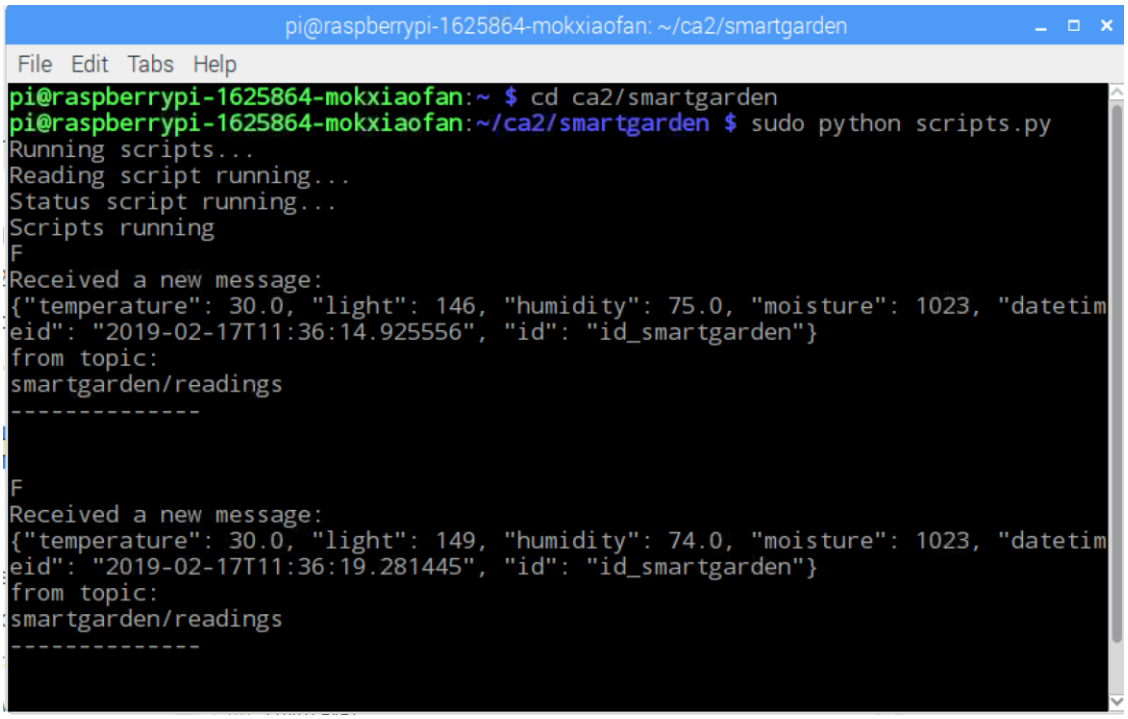
a) Open a new Terminal window and change directory to the ~/smartgarden folder:

```
cd ~/smartgarden
```

b) Once you have uploaded the Arduino code, run the scripts.py file immediately as it has to be in sync with the Arduino:

```
sudo python scripts.py
```

c) You should see the following output:



C. Run server.py

Task

a) On your RPi, open another Terminal window and change directory to the ~/smartgarden folder:

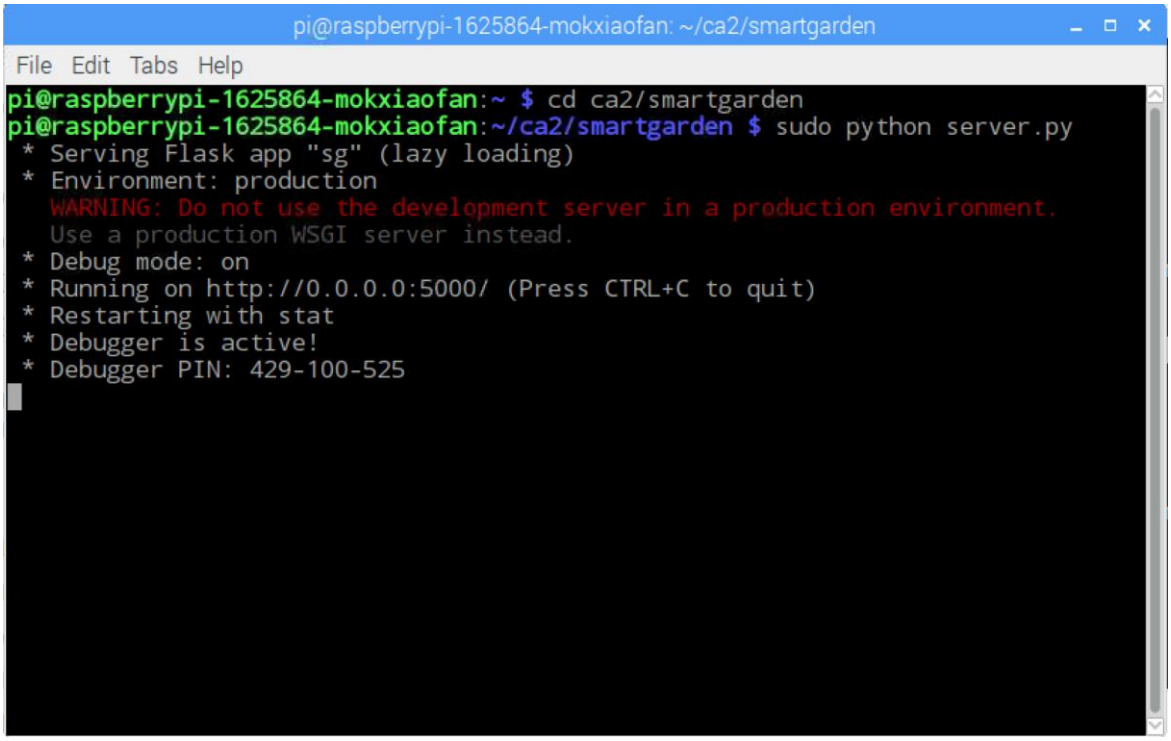
```
cd ~/smartgarden
```

b) Run the server.py file

```
sudo python server.py
```

Task

c) You should see the following output:



D. View Webpage

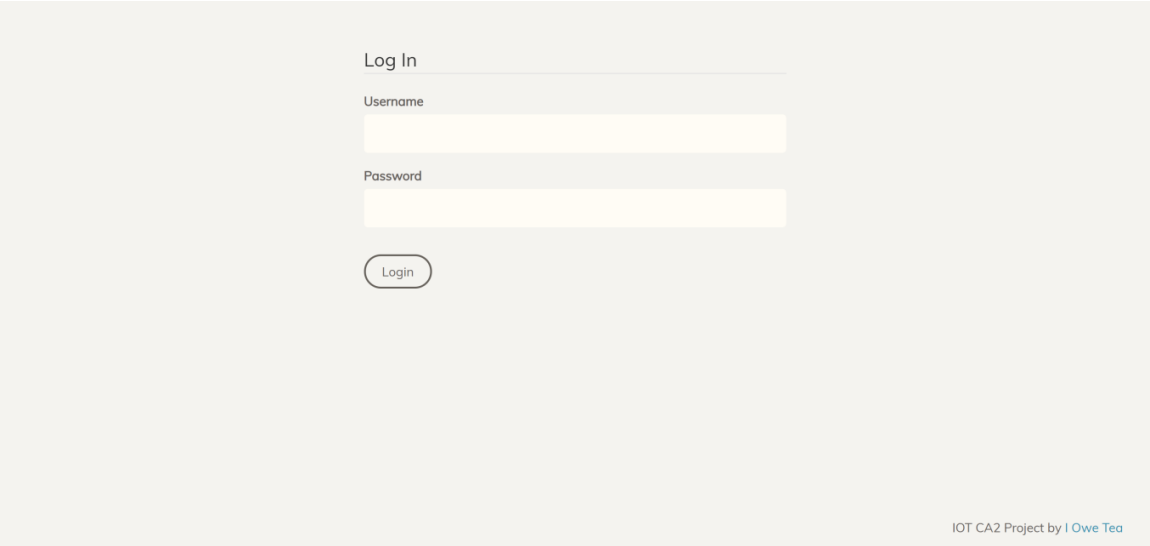
Task

a) On your laptop, open your browser and enter your RPi's IP address along with :5000 as shown (x.x.x.x is your RPi's IP address):

```
http://x.x.x.x:5000
```

b) If everything went well, you should see a similar output where you are asked to login:

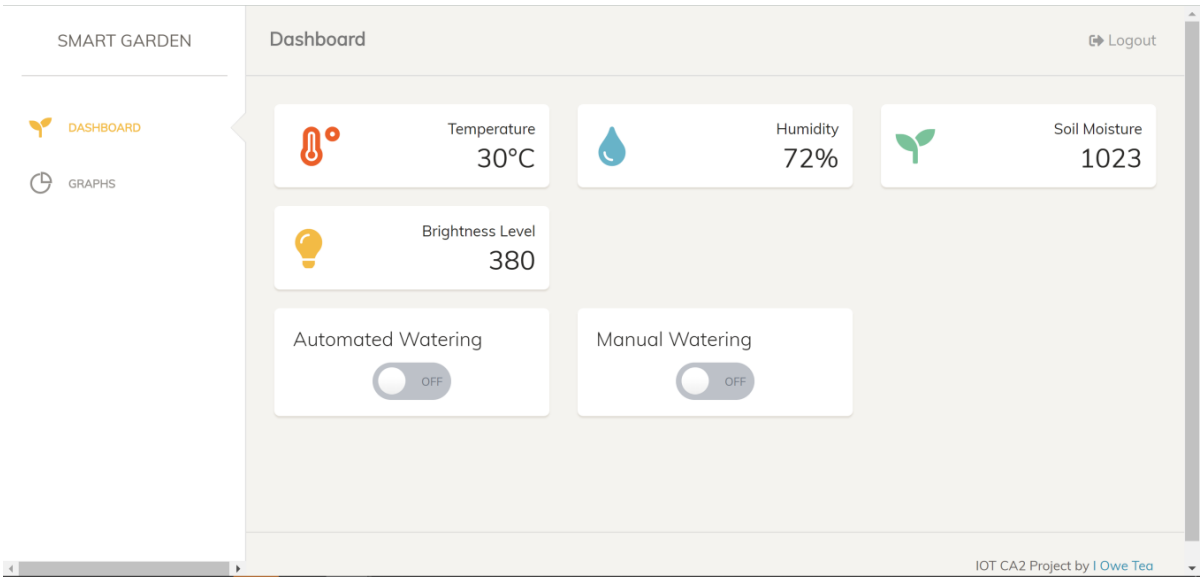
Task



Key in the with the credentials created in the smartgarden_login table.

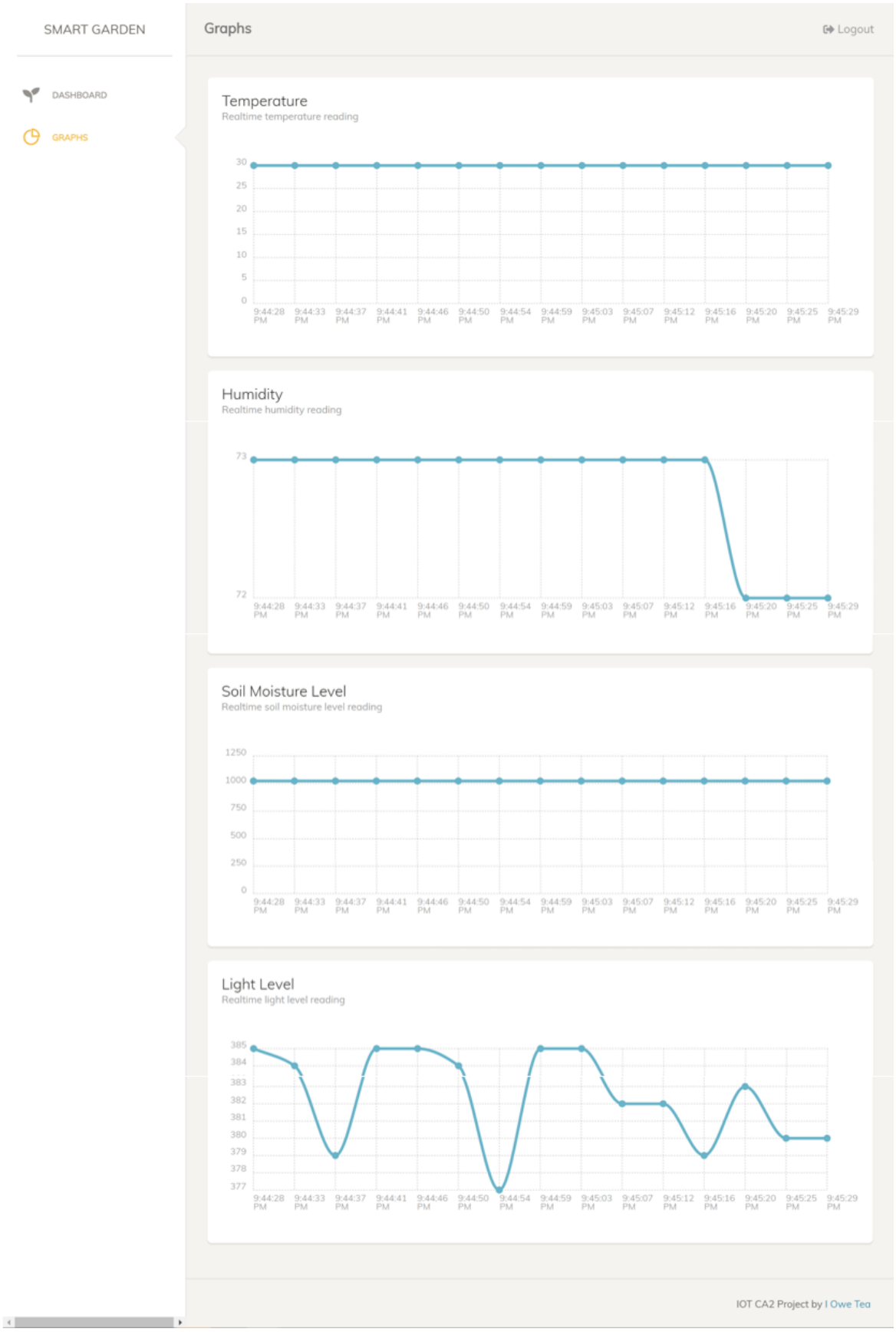
c) After logging in, you will be brought to the dashboard page where it shows the realtime values of the smart garden environment.

It is then followed by the two switches. By turning on the Automated Watering switch, the system will water the plant when the moisture level is higher than 400 (the higher then drier). And by turning it off, users can choose to water the plants manually by using the Manual Watering switch instead.



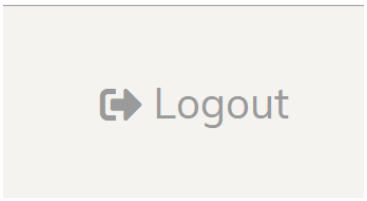
Task

d) You can toggle to the Graphs page by clicking on “Graphs” in the navigation bar at the left side of the page. The graphs page shows three graphs that displays the historical data of the latest 15 records of the temperature, humidity and soil moisture level.


SMART GARDEN
Graphs
Logout
DASHBOARD
GRAPHS
Temperature
Realtime temperature reading
30
25
20
15
10
5
0
Humidity
Realtime humidity reading
73
72
Soil Moisture Level
Realtime soil moisture level reading
1250
1000
750
500
250
0
Light Level
Realtime light level reading
385
384
383
382
381
380
379
378
377
9:44:28 PM
9:44:33 PM
9:44:37 PM
9:44:41 PM
9:44:46 PM
9:44:50 PM
9:44:54 PM
9:44:59 PM
9:45:03 PM
9:45:07 PM
9:45:12 PM
9:45:16 PM
9:45:20 PM
9:45:25 PM
9:45:29 PM
IOT CA2 Project by I Owe Tea

Task

e) Finally, you can choose to logout of the web page by clicking on the logout button located at the top right of the web page.



-- End of Step-by-step tutorial --