smart garden

IOT SENSOR AND IRRIGATION SYSTEM INTEGRATED INTO AWS AND AN ANDROID APP

Sivan Oddes

NSSL Technion

Guided By : Oren Kalinsky

Supervised By : Roy Mitrani

TABLE OF CONTENT

INTRODUCTION

In this project, I created an automatic, Wi-Fi based, irrigation system controlled and monitored by a cellphone application.

The project integrates between a NodeMCU, the AWS services and an Android app.
This infrastructure enables:

- To control the "on/off" state of a garden tap.
- To Monitor the temperature, light and soil moisture collected by sensors in the garden.

The motivation for this project came from the will to combine my love of gardening with my technological studies. I have a vegetable garden that gets watered by a timer which I set manually two times a year, winter and summer. The ability to water the garden more accurately and easily, based on weather climate, from my cellphone, can be significant in saving water and improving the plants' health.

NodeMCU is a type of a single board microcontroller that is designated as an open source IoT platform.

Amazon Web Services (AWS) is a subsidiary of Amazon.com that provides on-demand cloud computing platforms.

Internet of Things (IoT) is the network of physical devices embedded with electronics, software, sensors, actuators and connectivity which enables these things to connect and exchange data.

In this project book, I'll give a technical overview of the system, both physical and logical. I'll review the key functions of the system and finally discuss some of the Issues and resolutions I encountered along the way.

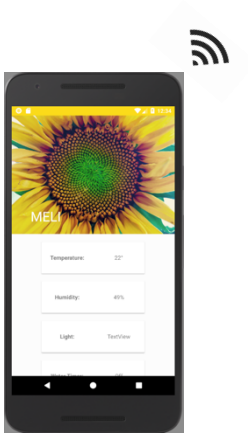
PROJECT SPECIFICATIONS

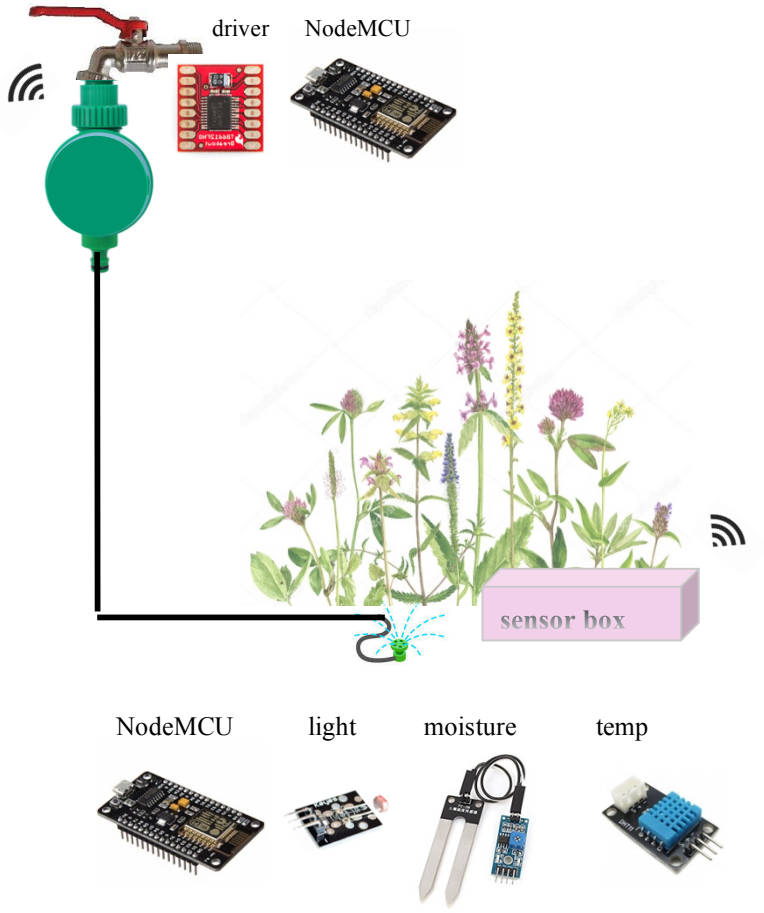
System properties:

- Control a water tap through a cellphone application.
  - Define a timer for the water tap ranging from one minute to an hour.
  - Pause, Play or Stop the timer.
  - Control and view the timer from the notification drawer.
- Measure the temperature, soil moisture and light over time and present it to the app user.

Hardware components:

- NodeMCU x2
- Motor Driver 1A Dual TB6612FNG
- DC motor mounted to a solenoid valve that sits in a plastic pipe connector
- DHT11 temp sensor
- Light sensor
- Soil moisture sensor
- 3D printed sensor case





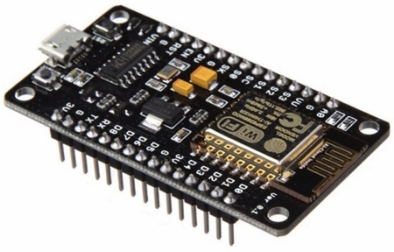
SYSTEM BACKGROUND

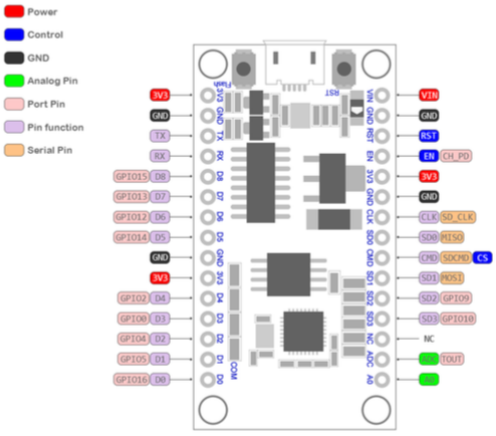
NodeMCU:

The NodeMCU is a single board microcontroller designed as an open source IoT platform. NodeMCU started on Oct. 2014. It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.

Specification:

- CPU - ESP8266MOD, 80 MHz (default) 32-bit microcontroller
- Memory - 128kBytes
- Storage - 4MBytes
- Power - USB (5-9v)
- 13 GPIO pins
- VCC - 3.3v output
- Wi-Fi 2.4 GHz, support WPA/WPA2
- Cost ~ $ 3.5





Garden Sensors:

DHT11 Temperature Sensor:

The DHT11 is a basic, ultra-low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin.

Specifications:

- 3 to 5V power and I/O
- 2.5mA max current
- Good for 0-50°C temperature readings ±2°C accuracy
- Cost ~ $1

FC-28 Soil Moisture Sensor:

The soil moisture sensor consists of two probes which are used to measure the volumetric content of water. The two probes allow the current to pass through the soil and then it gets the resistance value to measure the moisture value.

When there is more water, the soil will conduct more electricity which means that there will be less resistance. Therefore, the moisture level will be higher. Dry soil conducts electricity poorly, so when there is less water, the soil conducts less electricity which means that there is more resistance. Therefore, the moisture level is lower.

Specifications:

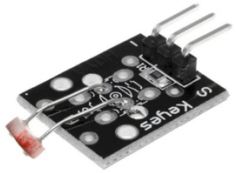
- Input Voltage 3.3 - 5V
- Output Voltage 0 - 4.2 V
- Input Current 35mA
- Cost ~ $0.5

LDR Light Sensor:

LDR sensor module is used to detect the intensity of light. When there is light, the resistance of LDR will become low according to the intensity of light. The greater the intensity of light, the lower the resistance of LDR. The sensor has a potentiometer knob that can be adjusted to change the sensitivity of LDR towards light.

Specification:

- Input Voltage: DC 3.3V to 5V
- Output: Analog and Digital
- Cost ~ $1

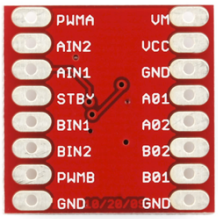


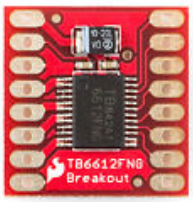
TB6612FNG Motor Driver:

The TB6612FNG Motor Driver can control up to two DC motors at a constant current of 1.2A (3.2A peak). Two input signals (IN1 and IN2) can be used to control the motor in one of four function modes: CW, CCW, short-brake and stop. The two motor outputs (A and B) can be separately controlled, and the speed of each motor is controlled via a PWM input signal with a frequency up to 100kHz. The STBY pin should be pulled up to take the motor out of standby mode.

Specifications:

- Output Current 1.2A (3.2 peak)
- Logic Supply Voltage (VCC) 2.7-5.5V
- Motor Supply Voltage (VM) 0-15V
- Cost ~ $1





Pin Label	Function	Power/Input/Output	Notes
VM	Motor Voltage	Power	This is where you provide power for the motors (2.2V to 13.5V)
VCC	Logic Voltage	Power	This is the voltage to power the chip and talk to the microcontroller (2.7V to 5.5V)
GND	Ground	Power	Common Ground for both motor voltage and logic voltage (all GND pins are connected)
STBY	Standby	Input	Allows the H-bridges to work when high (has a pulldown resistor so it must actively pulled high)
AIN1/BIN1	Input 1 for channels A/B	Input	One of the two inputs that determines the direction.
AIN2/BIN2	Input 2 for channels A/B	Input	One of the two inputs that determines the direction.
PWMA/PWMB	PWM input for channels A/B	Input	PWM input that controls the speed
A01/B01	Output 1 for channels A/B	Output	One of the two outputs to connect the motor
A02/B02	Output 2 for channels A/B	Output	One of the two outputs to connect the motor

	In2	PWM	Out1	Out2	Mode
	H	H/L	L	L	Short brake
	H	H	L	H	CCW
	H	L	L	L	Short brake
	L	H	H	L	CW
	L	L	L	L	Short brake
	L	H	OFF	OFF	Stop

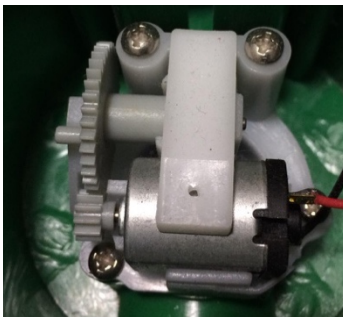
Water Tap Component:

I took apart a water tap timer and revealed the mechanism, discovering it had a DC motor moving a solenoid valve thus controlling the water flow. The direction of the motor rotation depends on the current direction which is controlled through a driver.

Specification:

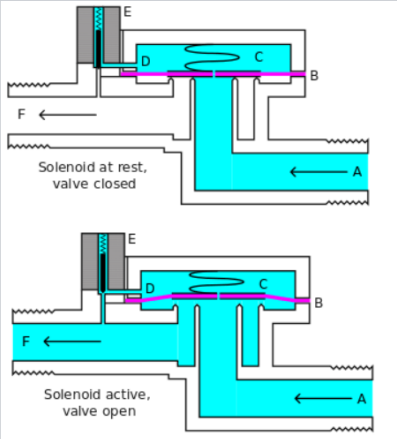
- 3v input
- ~1.5 A
- Cost ~$13

water timer



The motor inside

Solenoid mechanism:



The diagram to the right shows the design of a basic valve, controlling the flow of water in this example. At the top figure is the valve in its closed state. The water under pressure enters at **A**. **B** is an elastic diaphragm and above it is a weak spring pushing it down. The diaphragm has a pinhole through its center which allows a very small amount of water to flow through it. This water fills the cavity **C** on the other side of the diaphragm so that pressure is equal on both sides of the diaphragm, however the compressed spring supplies a net downward force. The spring is weak and is only able to close the inlet because water pressure is equalized on both sides of the diaphragm.

Once the diaphragm closes the valve, the pressure on the outlet side of its bottom is reduced, and the greater pressure above holds it even more firmly closed. Thus, the spring is irrelevant to holding the valve closed.

The above all works because the small drain passage **D** was blocked by a pin which is the armature of the solenoid **E** and which is pushed down by a spring. If current is passed through the solenoid, the pin is withdrawn via magnetic force, and the water in chamber *C* drains out the passage *D* faster than the pinhole can refill it. The pressure in chamber **C** drops and the incoming pressure lifts the diaphragm, thus opening the main valve. Water now flows directly from **A** to **F**.

When the solenoid is again deactivated and the passage **D** is closed again, the spring needs very little force to push the diaphragm down again and the main valve closes. In our water tap, there is no separate spring; the elastomer diaphragm is molded so that it functions as its own spring, preferring to be in the closed shape.

Android based smartphone:

The application was implemented on Android operating system.
Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touchscreen mobile devices such as smartphones and tablets.

MongooseOS:

Mongoose OS is an IoT Firmware Development Framework I worked with on the NodeMCU.
This operating system supports low power, connected microcontrollers such as: ESP32, ESP8266 and more. Its purpose is to be a complete environment for prototyping, development and managing connected devices.
It supports programing in JavaScript and features Integration with the AWS cloud as both came in use in this project.

features:

- User friendly Over the Air (OTA) updating.
- Secure connectivity and crypto support.
- Integrated Mongoose Web Server.
- Programming in either JavaScript (integrated mJS engine) or C.
- Integration with private and public clouds (e.g. AWS IoT, Mosquitto, HiveMQ etc.).

Amazon Web Services: 

IoT core: 

AWS IoT provides secure, bi-directional communication between Internet-connected devices such as sensors, actuators, embedded micro-controllers, or smart appliances and the AWS Cloud.

Through AWS IoT I connected the two NodeMCUs and the Android App.

The NodeMCUs were represented in the cloud by a **device shadow** which is a JSON document used to store and retrieve current state information for a device. The communication between the device and the shadow is done over MQTT protocol.

> **MQTT** (MQ Telemetry Transport or Message Queuing Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker.

The IoT section also provides a rule engine which enables a connection between the different services of the cloud. I used this service to connect the between the NodeMCUs and the DynamoDB service. The rule is trigged by a MQTT message which is the processed by a **lambda function**.

DynamoDB: 

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database, so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. Also, DynamoDB offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

I used this service to store the data collected by the sensors that can later be accessed by the app.

Lambda: 

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. With AWS Lambda, you can run code for virtually any type of application or backend service - all with zero administration.

As mentioned before I used this service in order process the MQTT message from the sensors and store the data into the DynamoDB table.

Cognito: 

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps.
The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services.

In this project, the app uses the identity pool to access both the DynamoDB table and the IoT water tap shadow.

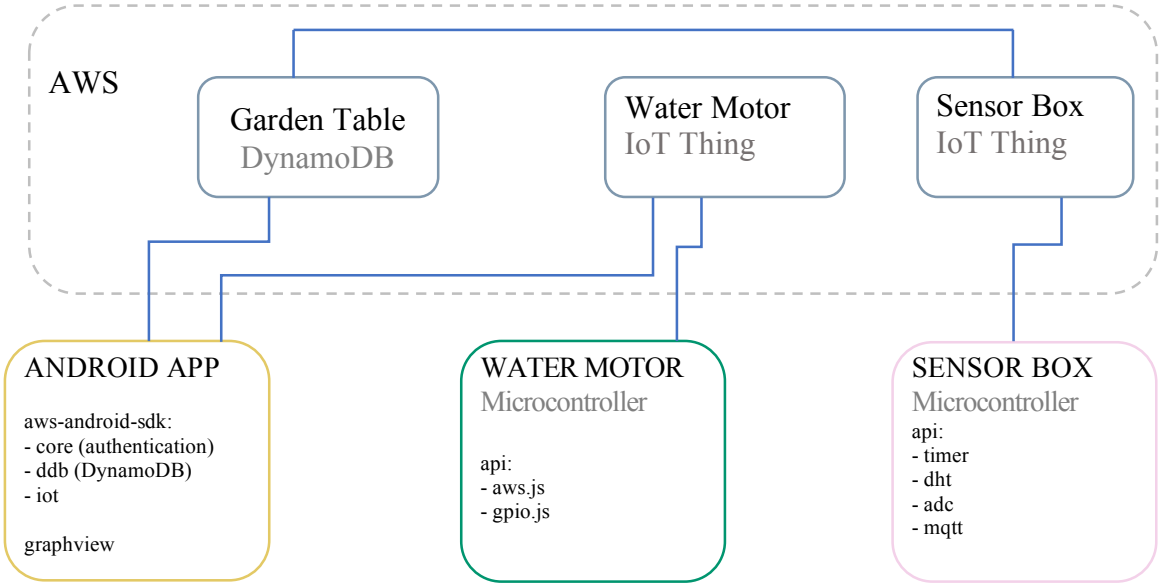
IAM: 

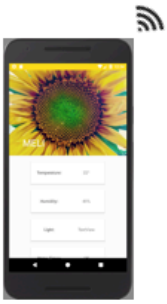
AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.
When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account.

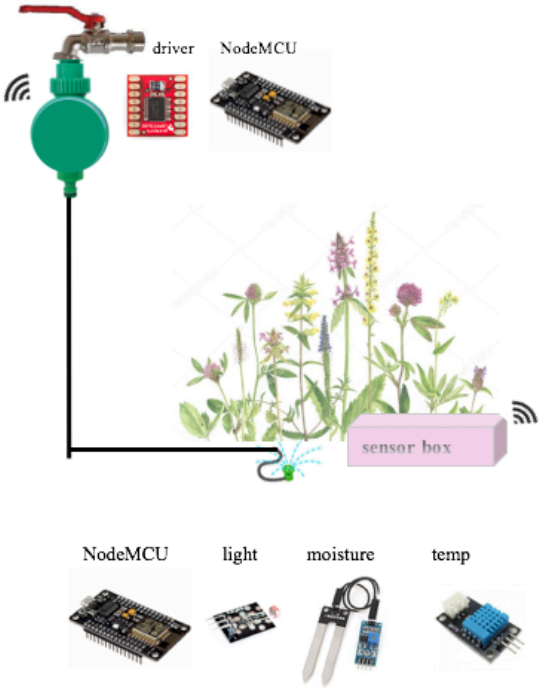
The IAM service is used to secure the AWS resources and give permissions only to specific components for specific operations.

ARCHITECTURE

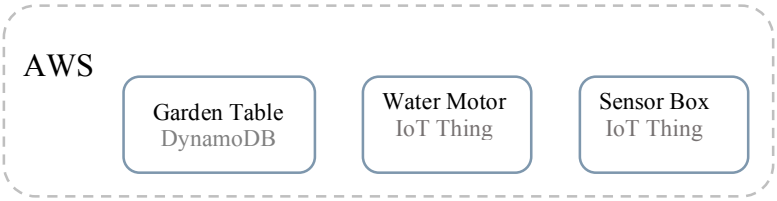
General Architecture:







AWS Architecture



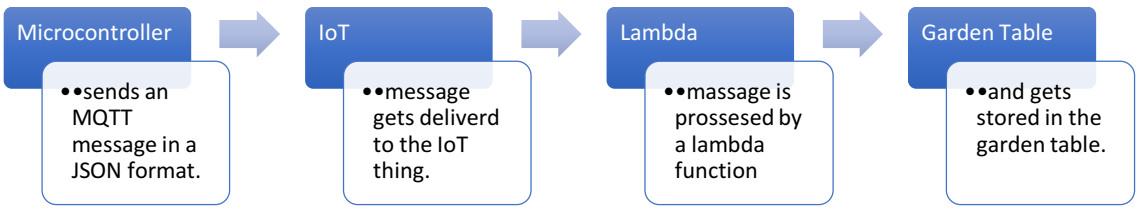
The AWS cloud acts as a broker between the two Microcontrollers and the Android app.
Its job is to manage the sensors and water motor so that the app can view the data they collected or send them commands to execute.

The cloud receives all the data from the sensors and stores them in the "Garden_Table" so that at any time the app can connect to the cloud and view that data. In addition, it enables the app to control the water motor - turn it on/off by creating a "thing shadow" that bridges over the "desired" state verses the "reported" one.

The two microcontrollers are managed under the **IoT Core** service. Each one of them is represented by a "thing" of type "garden_device". The "Garden_Table" is stored under the **DynamoDB** service which is managed by a **lambda** function.

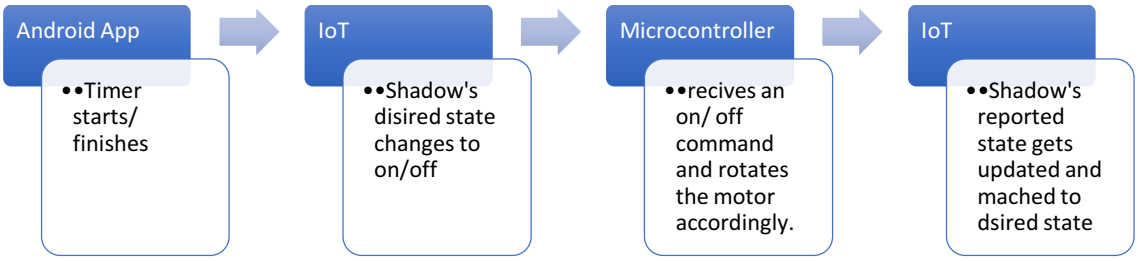
How it works:

- Sensor Box
  The sensor microcontroller sends a **MQTT** message to the cloud with the appropriate subject ('mos/subject1'). The message is sent to the cloud in a JSON format and received by the "writToGarden" rule which redirects it to be handled by a **lambda** function that inserts the data into the **DynamoDB Garden Table**.



- Water Motor
  The app user starts the water timer. Then the "on" command gets sent to the aws there it updates the **thing's "shadow"** and lets it know the desired state has changed. When the

timer is finished the "off" command gets sent to the thing's "shadow". In every change of the desired state, the shadow keeps a "delta" between the desired and reported state so that the water motor microcontroller will recognize it needs to change its state and execute accordingly.

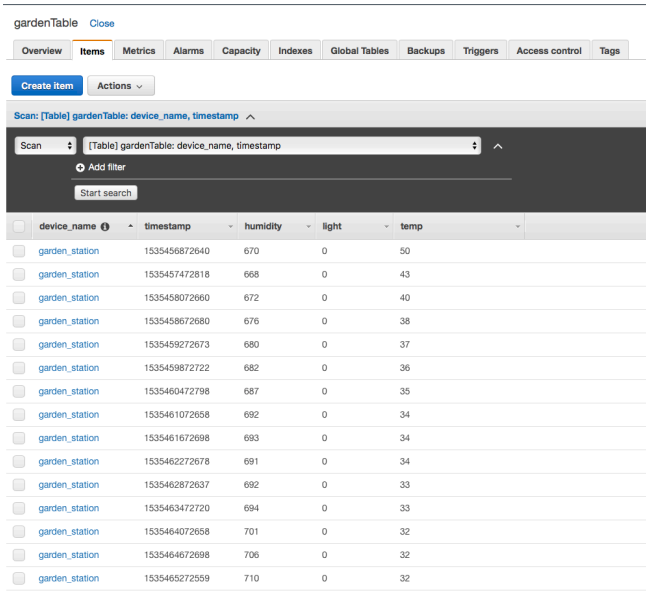


– Garden Table

In this DynamoDB table we store all the sensor's data that can later be accessed by the app user. The stored parameters are:

Device name| Timestamp| Humidity| Temperature |Light.

The Garden table:


gardenTable Close

Overview | Items | Metrics | Alarms | Capacity | Indexes | Global Tables | Backups | Triggers | Access control | Tags

Create item | Actions

Scan: [Table] gardenTable: device_name, timestamp

Scan | [Table] gardenTable: device_name, timestamp

Add filter

Start search

device_name	timestamp	humidity	light	temp
garden_station	1535456872640	670	0	50
garden_station	1535457472818	668	0	43
garden_station	1535458072660	672	0	40
garden_station	1535458672680	676	0	38
garden_station	1535459272673	680	0	37
garden_station	1535459872722	682	0	36
garden_station	1535460472798	687	0	35
garden_station	1535461072658	692	0	34
garden_station	1535461672698	693	0	34
garden_station	1535462272678	691	0	34
garden_station	1535462872637	692	0	33
garden_station	1535463472720	694	0	33
garden_station	1535464072658	701	0	32
garden_station	1535464672698	706	0	32
garden_station	1535465272559	710	0	32

Security / authentication:

androidPubSub pool- Cognito_androidpubsubUnauth_Role
attached pilcy: certificateandkeypolicy

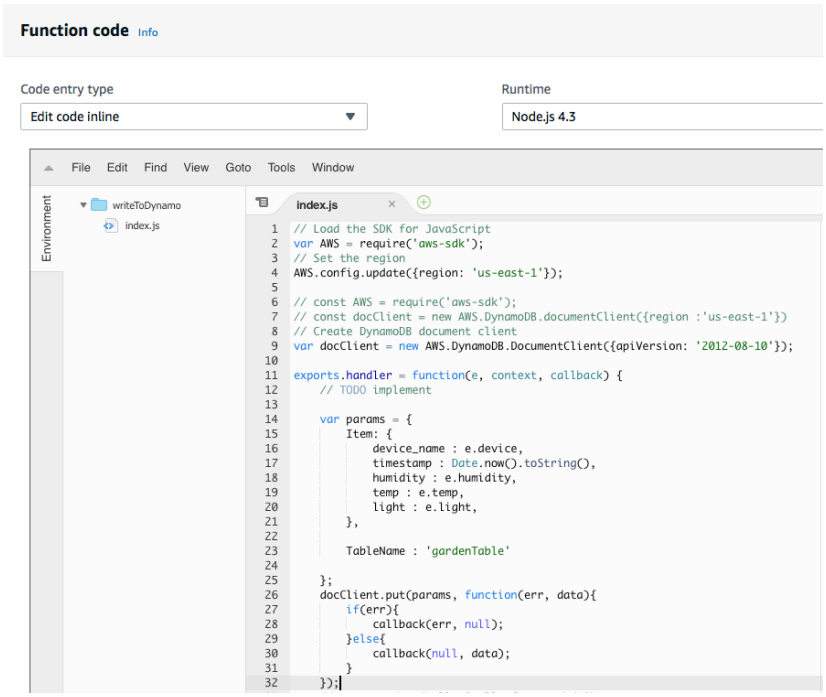
garden_pool - Cognito_garden_poolUnauth_Role
attached policy: garden_app_policy

The water motor shadow:

Shadow state:

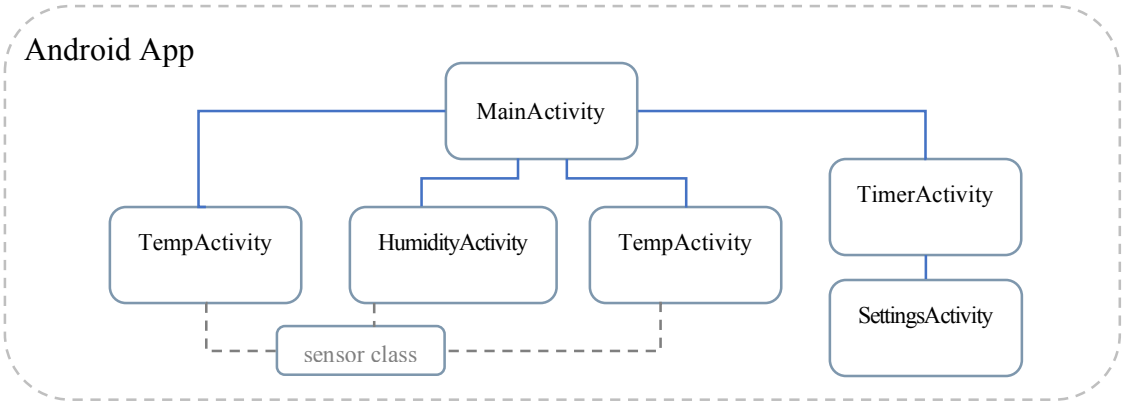

```
{
  "desired": {
    "welcome": "aws-iot",
    "on": false
  },
  "reported": {
    "welcome": "aws-iot",
    "on": true,
    "ota": {
      "message": "idle",
      "status": 0,
      "is_committed": true,
      "commit_timeout": 0,
      "partition": 0,
      "progress_percent": 0,
      "fw_version": "1.0",
      "fw_id": "20180804-065901/???",
      "mac": "620194296496",
      "device_id": "esp8266_296496",
      "app": "ota-shadow",
      "arch": "esp8266"
    }
  },
  "delta": {
    "on": false
  }
```


The Sensor Box Lambda function:



Github repository: https://github.com/sivodd/smart-garden/tree/master/aws

Application Architecture



The Android app is divided into 4 main activities:

- Temperature graph
- Humidity graph
- Light Intensity graph
- Timer

I created a "Sensor" class that consist of the above variables. In addition, the class's methods query and change the data set in the Garden Table (only the query is used in the app at this point).
In the Temp, Humidity & Light activities there are graphs that are created based on the data stored in the DynamoDB Garden Table. Every graph shows the sensor's property on a timescale. In this version, I chose to show the last 24 hour samples, is at least two hours apart (I iterate over the 180 last samples, each taken every 10 minutes, and chose only samples that are 2 hours apart).

The Timer activity controls the water motor by sending a MQTT message to the water motor's shadow to turn on/off. The timer can be configured from 1 min to 60 min, in the Settings Activity. It will also run in the background when the app is closed and can be controlled from the notification drawer. In the TimerActivity I declare the enum class "TimerState". TimerState describes the state of the timer with the values of: "Stopped", "Running" & "Paused". Every time we enter this activity the app establishes a connection to the cloud so that it is ready to receive our MQTT messages for the play/pause/stop command sent to the water motor.

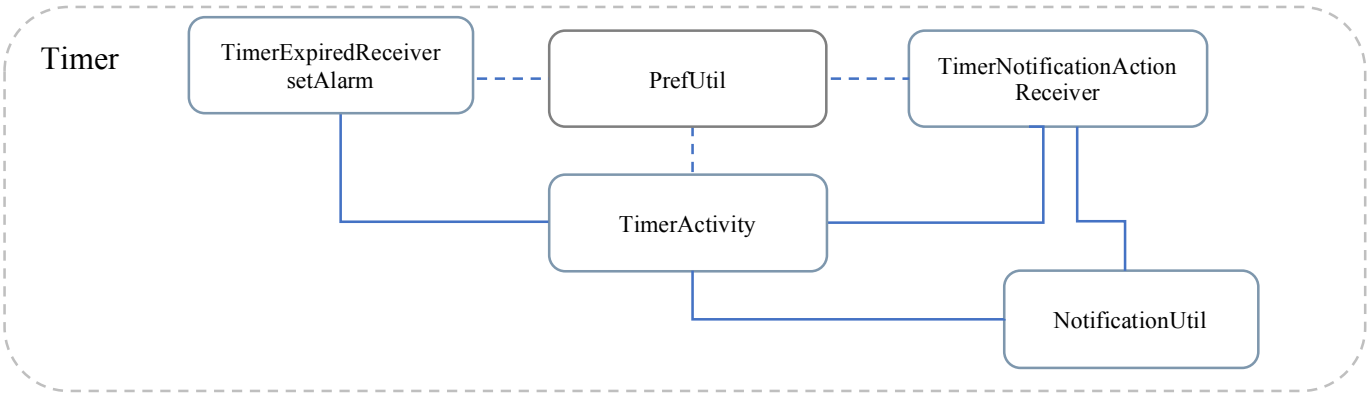
In TimerActivity I define the Alarm function. The SetAlarm function sends a broadcast message to **TimerExpiredReceiver** when the timer expires, so that TimerExpiredReceiver will wake up and send a MQTT message to the water motor to turn off.

NotificationUtil manages the Timer in the notification drawer. NotificationUtil gets updated from TimerActivity. When the status of the Timer gets changed it will update NotificationUtil which will

then call **TimerNotificationActionReceiver** which will then send a MQTT message to the cloud to update the water motor status.

PrefUtil manages all the shared Preferences between TimerActivity, TimerExpiredReceiver and TimerNotificationActionReceiver. In this class, all the "get"/"set" shared preferences methods of the Timer are declared:

- getTimerLength
- get/setPreviousTimerLengthSeconds
- get/setTimerState
- get/setSecondsRemaining
- get/setConnectionState
- get/setAlarmSetTime



SDKs in use:

- core (authentication)
- ddb/ddb - mapper (DynamoDB)
- iot (MQTT)

Libraries:

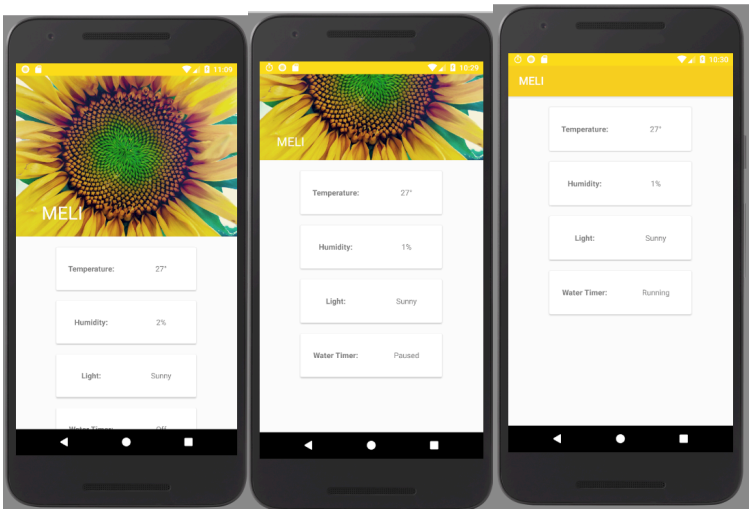
- com.amazonaws.auth.CognitoCachingCredentialsProvider
- com.amazonaws.regions.Regions
- com.amazonaws.services.dynamodbv2.*
- com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.*
- com.jjoe64.graphview.*
- android.app.AlarmManager
- android.app.PendingIntent

Third party libraries:

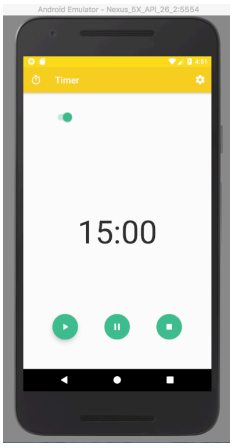
- com.pavelsikun:material-seekbar-preference:2.3.0+, for the seekbar in settings.
- me.zhanghai.android.materialprogressbar:library:1.4.2, for the round progress bar of the timer.

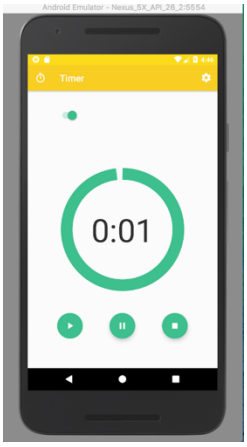
GitHub repository: https://github.com/sivodd/smart-garden/tree/master/application

MainActivity - home screen

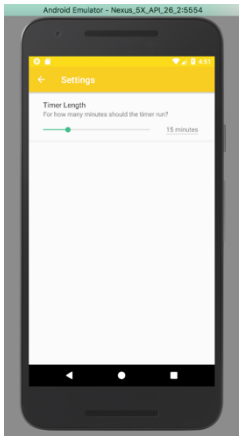


TimerActivity

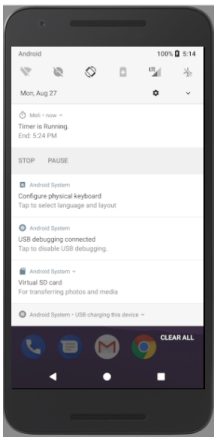




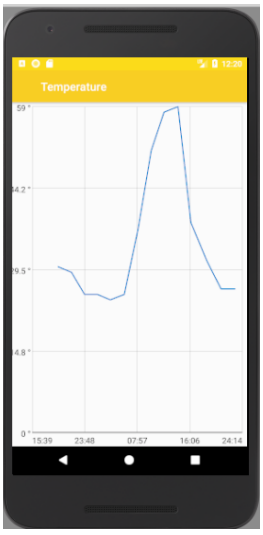
SettingsActivity



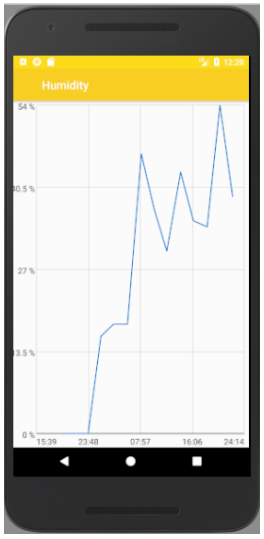
NotificationUtil



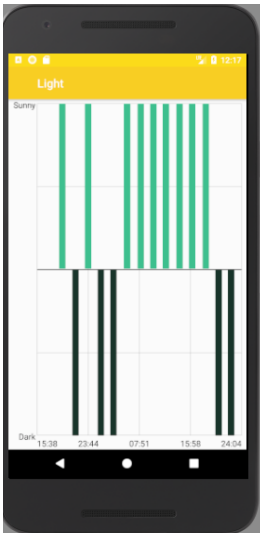
TempActivity



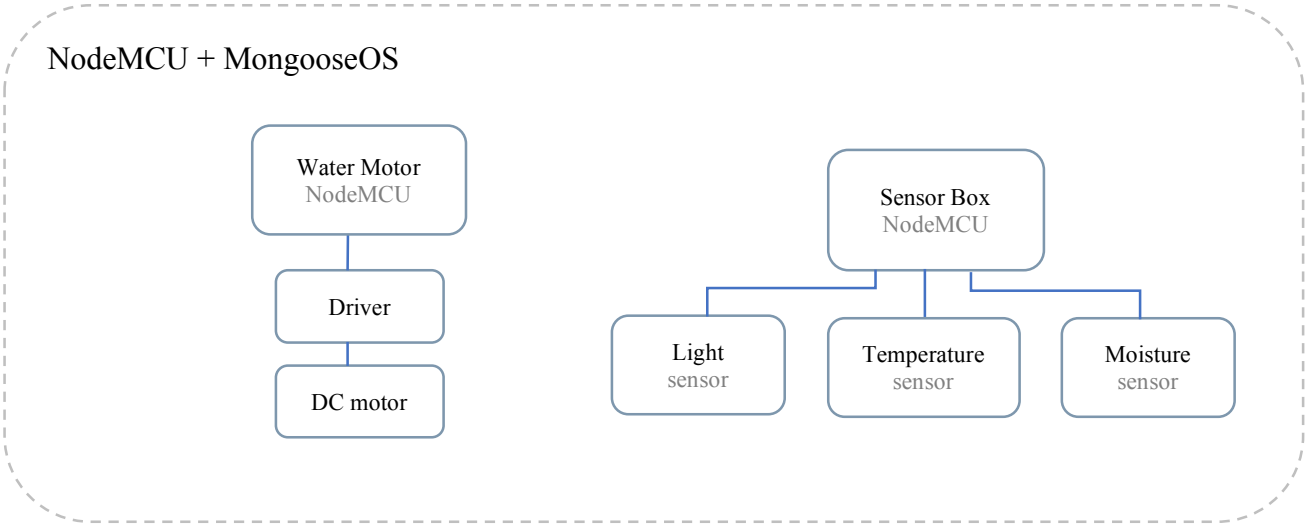
HumidityActivity



LightActivity



Hardware Architecture

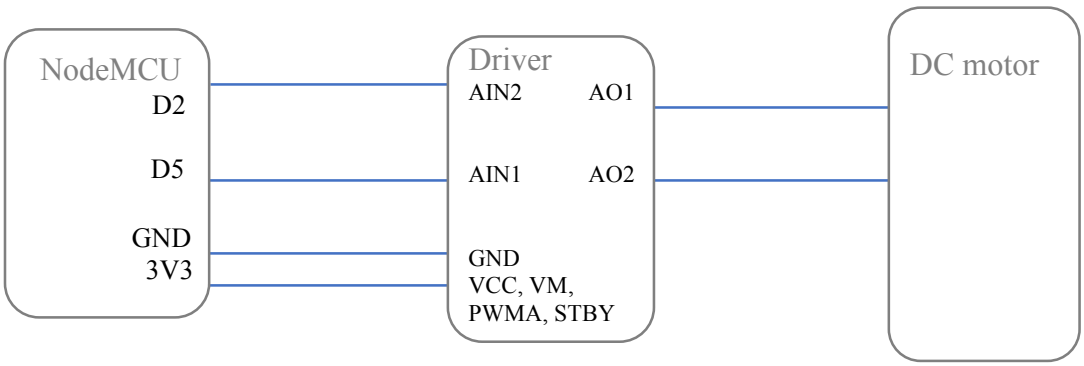


The Water Motor controller is connected to a DC motor through a driver. The DC motor is mounted to a water valve, whom moves up/down by plastic gears. The NodeMCU is connected to the Driver throw D2 & D5 which are connected to AIN1_PIN & AIN2_PIN respectively. The PWMA_PIN & STBY_PIN of the driver are both connected to the 3v3 pin of the Node, as well as the VM & VCC pins. The AO1 & AO2 pins of the driver are connected to the motor cables.
As I mentioned in the background chapter, the motor direction is controlled by the direction of the current flow. The current is controlled by the "forward"/"backward" functions in the mongoose code. These functions are triggered when the shadow of the "water_motor thing" changes the "desired" state so that it doesn't match the "reported" state. If so, my code will move the motor in the suitable direction and then update the shadow's reported state.

APIs:

- aws
- gpio

A sketch of the connected cables:

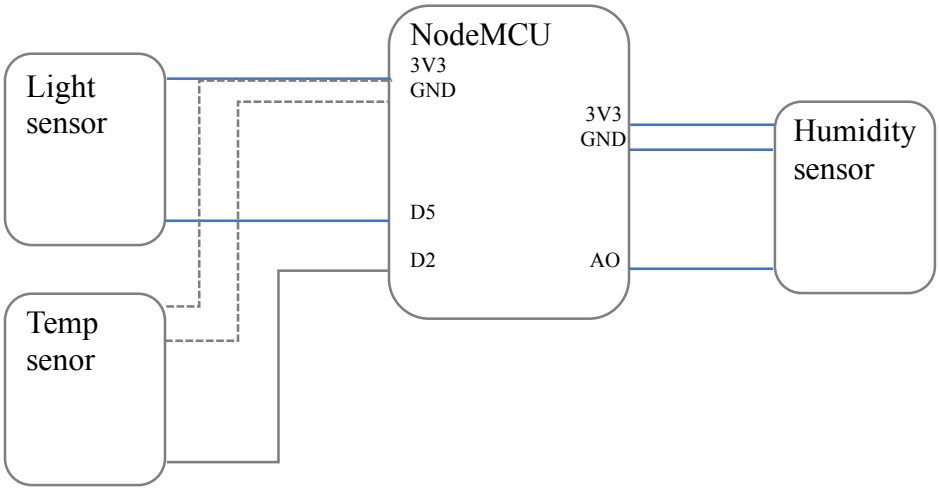


The Sensor Box controller is connected to - light, temperature and soil moist sensors.
This mongoose code samples the sensors every 10 minutes and sends the data to the AWS in a JSON format, MQTT message.

APIs:

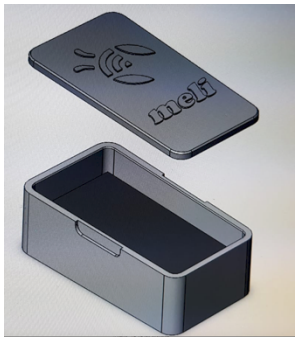
- mqtt
- timer
- dht
- adc

A sketch of the connected cables:



Github repository: https://github.com/sivodd/smart-garden/tree/master/Mongoose

3D sensor box model: https://github.com/sivodd/smart-garden/tree/master/3D_CASE



CODE EXAMPLES

All code available in GitHub: https://github.com/sivodd/smart-garden

Android app - MainActivity:

```
package com.example.sivan.meli;

import android.content.Intent;
import android.os.StrictMode;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.CardView;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.*;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.*;

import java.util.List;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private CardView tempCard, humCard, timerCard, lightCard;
    private TextView temp, hum, timer, light;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // added this if
        if (android.os.Build.VERSION.SDK_INT > 9) {
            StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
        }

        tempCard = (CardView) findViewById(R.id.temp_card);
        humCard = (CardView) findViewById(R.id.hum_card);
        lightCard = (CardView) findViewById(R.id.light_card);
        timerCard = (CardView) findViewById(R.id.timer_card);
        temp = (TextView) findViewById(R.id.textView1);
        hum = (TextView) findViewById(R.id.textView3);
        timer = (TextView) findViewById(R.id.textView7);
        light = (TextView) findViewById(R.id.textView5);

        Sensor Sensor;
        String temp_string;
        String hum_string;
        String light_string;

        CognitoCachingCredentialsProvider credentialsProvider = new
CognitoCachingCredentialsProvider(
                getApplicationContext(),
                "us-east-1:732b9a49-4529-4b84-9a6a-20a12fd838d8", // Identity pool ID
                Regions.US_EAST_1 // Region
```

```
        );

        AmazonDynamoDBClient ddbClient = new
AmazonDynamoDBClient(credentialsProvider);
        DynamoDBMapper mapper = new DynamoDBMapper(ddbClient);

        String device = "garden_station";
        Sensor sensorKey = new Sensor();
        sensorKey.setDevice(device);

        // query the last 10 results of 'garden_station'
        int limit = 10;
        DynamoDBQueryExpression<Sensor> queryExpression = new
DynamoDBQueryExpression<Sensor>()
                .withHashKeyValues(sensorKey)
                .withScanIndexForward(false)
                .withLimit(limit);

        QueryResultPage<Sensor> queryResult = mapper.queryPage(Sensor.class,
queryExpression);
        List<Sensor> result = queryResult.getResults();

        int resIndex = 0;
        Sensor = result.get(resIndex);
        temp_string = Sensor.getTemp();
        while (temp_string.equals("-1") && resIndex < limit){
            resIndex++;
            Sensor = result.get(resIndex);
            temp_string = Sensor.getTemp();
        }
        if (resIndex == limit)
            temp_string = "error";
        hum_string = Integer.toString(Sensor.getHumidity()*100/1024);
        if (Sensor.getLight()==0)
            light_string = "Sunny";
        else light_string = "Dark";


        temp_string = temp_string + "\u00b0";
        hum_string = hum_string + "%";

        temp.setText(temp_string);
        hum.setText(hum_string);
        light.setText(light_string);
        timer.setText("Off");


        tempCard.setOnClickListener(this);
        humCard.setOnClickListener(this);
        lightCard.setOnClickListener(this);
        timerCard.setOnClickListener(this);


    }

    @Override
    public void onClick(View v) {
        Intent i;

        switch (v.getId()) {
            case R.id.temp_card:
```

```
                i = new Intent(this, TempActivity.class);
                startActivity(i);
                break;
            case R.id.hum_card:
                i = new Intent(this, HumidityActivity.class);
                startActivity(i);
                break;
            case R.id.light_card:
                i = new Intent(this, LightActivity.class);
                startActivity(i);
                break;
            case R.id.timer_card:
                i = new Intent(this, TimerActivity.class);
                startActivityForResult(i, 999);
                break;
            default:
                break;
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == 999 && resultCode == RESULT_OK){
            String timer_string = data.getStringExtra("timer_state");
            timer.setText(data.getStringExtra("timer_state"));
            Log.d("MyApp",timer_string);
        }
    }
}
```

MongooseOS - waterMotor:

```
//mongoose app
load('api_events.js');
load('api_net.js');
load('api_sys.js');
load('api_gpio.js');
load('api_config.js');
load('api_timer.js');
load('api_arduino_onewire.js');
load('api_esp8266.js');
load('api_pwm.js');

// Pin number assignment for TB6612FNG
let PWMA_PIN = 16; //D0
let AIN2_PIN = 14; //D5
let AIN1_PIN = 4; //D2
let STBY_PIN = 5; //D1

//ref from https://github.com/stylixboom/lr_motor/blob/master/app_l.js
GPIO.set_mode(PWMA_PIN, GPIO.MODE_OUTPUT);
GPIO.set_mode(AIN2_PIN, GPIO.MODE_OUTPUT);
GPIO.set_mode(AIN1_PIN, GPIO.MODE_OUTPUT);
GPIO.set_mode(STBY_PIN, GPIO.MODE_OUTPUT);

function pin_init() {
    GPIO.write(STBY_PIN, 0);
    GPIO.write(AIN1_PIN, 0);
    GPIO.write(AIN2_PIN, 0);
    GPIO.write(PWMA_PIN, 0);
    print("init");
    Sys.usleep(3000000);
}

function forword() {
    // ==== A ====
    GPIO.write(STBY_PIN, 1);
    GPIO.write(PWMA_PIN, 1);
    GPIO.write(AIN1_PIN, 0);
    GPIO.write(AIN2_PIN, 1);
    print("forword");
    Sys.usleep(3000000);
    brake();
}

function brake() { //LIBRARY SUORCE
    // ==== A ====
    GPIO.write(STBY_PIN, 1);
    GPIO.write(PWMA_PIN, 0);
    GPIO.write(AIN1_PIN, 1);
    GPIO.write(AIN2_PIN, 1); // OR 0
    print("brake");
    Sys.usleep(3000000);
}
function backword() {
    // ==== A ====
    GPIO.write(STBY_PIN, 1);
    GPIO.write(PWMA_PIN, 1);
    GPIO.write(AIN1_PIN, 1);
    GPIO.write(AIN2_PIN, 0);
    print("backword");
    Sys.usleep(3000000);
    brake();
}

load('api_config.js');
load('api_gpio.js');
load('api_shadow.js');

let led = Cfg.get('pins.led');  // Built-in LED GPIO number
let state = {on: false};        // Device state - LED on/off status

// Set up Shadow handler to synchronise device state with the shadow state
Shadow.addHandler(function(event, obj) {
  if (event === 'CONNECTED') {
    // Connected to shadow - report our current state.
    Shadow.update(0, state);
  } else if (event === 'UPDATE_DELTA') {
    // Got delta. Iterate over the delta keys, handle those we know about.
    print('Got delta:', JSON.stringify(obj));
    for (let key in obj) {
      if (key === 'on') {
        // Shadow wants us to change local state - do it.
        state.on = obj.on;
        GPIO.set_mode(led, GPIO.MODE_OUTPUT);
        GPIO.write(led, state.on ? 0 : 1);
        state.on ? forword() : backword();
        print('LED on ->', state.on);
      }
    }
    // Once we've done synchronising with the shadow, report our state.
    Shadow.update(0, state);
  }
});
```

ISSUES AND RESOLUTIONS

Water Timer

The initial outline of this project was to have the sensor box as sensors hidden in a plant pot without the watering system. My biggest resolution for this project was to change the specifications so it will suit my garden today. This brought me to create two components that connect to the app, one controlling the water tap and the other placed in the garden.

The main issue was to figure out how I can include my water timer in the project. I bought this timer online and it didn't come with a brand name or specifications. I know it ran on 2 AAA batteries, hence I assumed the mechanism in it could be controlled by a NodeMCU (which has a 3.3V output). I took it apart and revealed a DC motor mount to a solenoid valve. With the help of the control lab, I found the right current it needed and figured out I had to connect it to the NodeMCU through a DC motor driver. Accordingly, I took its control system apart from the motor and connected my NodeMCU instead.

App GraphView

Initially I was querying the whole DB with "scan" method that would download the whole table and present all the data collected so far starting from the first ever sample in the DB. Of course, this was a problem because the most relevant data is the most resent one. Therefore, I had to change the scan method to "query" method which has the attribute "withScanIndexForward" that queries the most recent data first. But then the problem was that the "series.append" method didn't know how to deal with the X values that came in an ascending order. Finally, I queried only a limited number of rows (also much more efficient for the program) and made a revers loop on these results to show only the lase x samples taken.

MongooseOS water motor

The TB6612FNG Hookup Guide is compatible with Arduino and not Mongoose. Therefore, I had to understand the source library to apply the code in the mongooseOS system. In this library, they used the function "analogWrite" which doesn't exist in mongooseOS. First I figured out that I can replace it with just the "pull_up" method that does appear in the API but this function wasn't working. Not knowing the reason the motor wasn't reacting was a software problem I had to isolate the NodeMCU, then the driver and last the motor. When isolating the Node, I found out the pinout related to this function wasn't responding (using a LED light) and found there is another function called "GPIO.write" I had to use instead.

AWS IoT provision

I had a problem with the provisioning of the IoT in the mongoose web interface. It turns out that the mongoose clock didn't match my computer's time so I had to take my clock one hour back for it to work.

FUTURE TASKS

Here are some recommendations for future development of the system:

- Making the two field components self-sufficient and solar run.
- Developing the monitoring features to be more versatile or even characterized by the user himself.
- Integrating a trigger function in AWS that will send watering notifications to the app or even control the water tap automatically based on trigger rules.
- Integrate more sensors such as soil ph.
- Make the system into a machine learning robot that will decide for itself when and how much to water the plants.
- Creating an iPhone app.

SUMMARY

In this project, I implemented a system of many components. All the parts of the project (AWS, Android and hardware) were new to me and gave me a glimpse into their world. For the first time in my degree I got to express myself and had the privilege to combine my hobbies with my studies.

The IoT is a fast developing and relevant field that is already a part of our lives in vehicles, home appliances, farming and more. Here I connected my water tap and sensors in the garden to the internet and integrated the control and monitor capabilities to a cellphone.
I learned the basis of making an android app, dealing with NodeMCU and sensors and integrating them all with the AWS. In addition, I was exposed to the big variety of services the Cloud has to offer and the potential of the things one can create using a cheap component such as a NodeMCU.

I handled the streaming of data from the sensors to the cloud, from the cloud to the NodeMCU, and back and forth from the cloud to the application. In addition, I dealt with connecting hardware components such as the sensors to the NodeMCU and the Node to the DC motor. I used the new and yet developing operating system of MongooseOS and managed to fit it to my needs. In almost every step of the way I faced challenges and nothing went "smoothly".

The biggest added value to this project, especially doing it on my own, was the fact that it intensified my self-ability to take on a new subject, learn it, and deal with the difficulties along the way. It has been a fun and difficult learning experience that has given me great satisfaction.

BIBLIOGRAPHY

[1] NodeMCU - https://en.wikipedia.org/wiki/NodeMCU
[2] AWS - https://en.wikipedia.org/wiki/Amazon_Web_Services
[3] Internet of Things - https://en.wikipedia.org/wiki/Internet_of_things
[4] DHT11 - https://www.adafruit.com/product/386
[5] Soil Moisture sensor - https://www.sparkfun.com/products/13322
[6] LDR light sensor - https://www.instructables.com/id/LDR-Sensor-Module-Users-Manual-V10
[7] Motor driver - https://learn.sparkfun.com/tutorials/tb6612fng-hookup-guide
[8] Solenoid - https://en.wikipedia.org/wiki/Solenoid
[9] Water timer - https://www.aliexpress.com/item/LCD-Display-Automatic-Intelligent-Electronic-Garden-Water-Timer-Rubber-Solenoid-Valve-Irrigation-Sprinkler-Control-Gasket-Design/32805966913.html?spm=a2g0s.11045068.rcmd404.2.562556a4NZQFAs&gps-id=detail404&scm=1007.16891.96945.0&scm_id=1007.16891.96945.0&scm-url=1007.16891.96945.0&pvid=f62276e5-0063-4b4f-bf99-abbdbe40ff5d
[10] Android - https://en.wikipedia.org/wiki/Android_(operating_system)
[11] MongooseOS - https://en.wikipedia.org/wiki/Mongoose_OS
[12] MQTT - https://en.wikipedia.org/wiki/MQTT
[13] AWS documentaion - https://aws.amazon.com/documentation
[14] MongooseOS API - https://mongoose-os.com/docs/README.md