



Gajalakshan Chandrasegaran, Minh Do, Andrey Verbovskiy, Danil Sisov

## Internet of Things Project - Mizu-ten

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Internet of Things Project

9 May 2023

## Abstract

Authors:	Gajalakshan Chandrasegaran, Minh Do, Andrey Verbovskiy, Danil Sisov
Title:	Mizu-ten
Number of Pages:	16 pages
Date:	9 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information technology
Professional Major:	Smart IoT Systems
Supervisor:	Joseph Hotchkiss

---

## **Contents**

### List of Abbreviations

1 Introduction	1
2 Webpage	1
2.1 Login and registration pages	1
2.3 Front-End	8
2.4 MQTT Broker	9
3 Embedded	11
4 Conclusion	16

## **List of Abbreviations**

HTML: Hyper Text Markup Language

CSS: Cascading Style Sheets

JS: JavaScript

EJS: Embedded JavaScript

IoT: Internet of Things

MQTT: Message Queuing Telemetry Transport

MongoDB: Mongo Database

# 1 Introduction

We decided to design a system that would resolve the excess watering by monitoring the soil moisture level and temperature. Our system will have multiple sensors across the entire farm, so that all regions would be monitored. Based on moisture level, the system will decide whether the specific region should be watered. Then it checks the temperature, as if it is low, watering plants can be harmful. If the soil is dry and temperature is decent, the system automatically activates sprays through relays. The farmer can monitor the process through the device. This solution is not only financially beneficial, it is also eco-friendly, which can be really important to some companies.

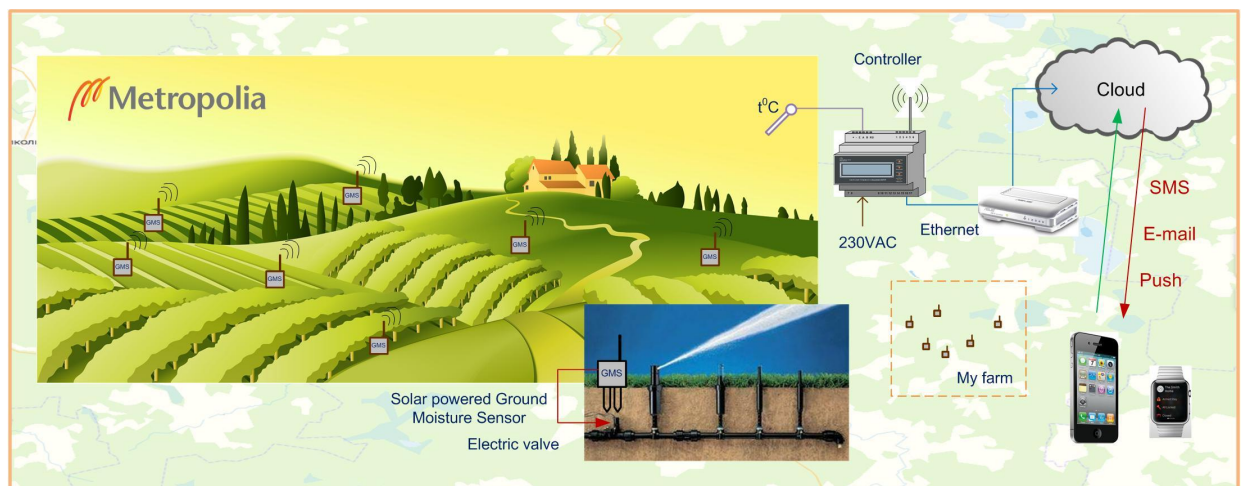


Figure 1. Project interpretation

## 2 Webpage

### 2.1 Login and registration pages

The web application starts with a login page where the design is made according to the theme of the project and the required elements are in a convenient place for an easy conception and quick use. The login page is shown in Figure 1.

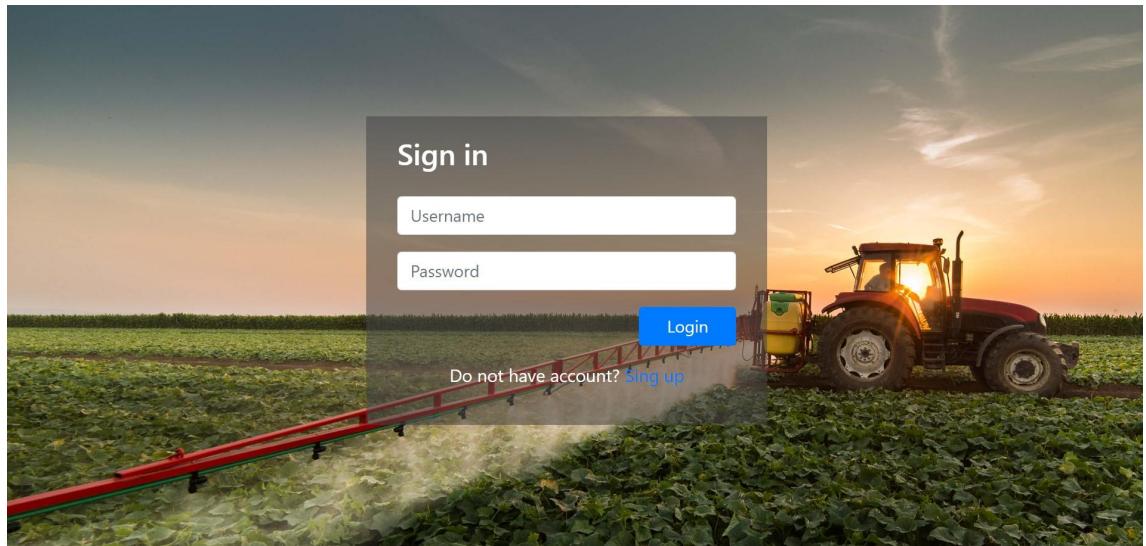


Figure 2. Login page

At this stage, the user can log into his account if they are already registered or register a new user. If the user does not have an account, they would have to click on the sign-up button, after which they will be redirected to the registration page. The registration page shown in Figure 2 is made in a similarly simple design as the login page and also makes it possible for a new user to register easily and quickly

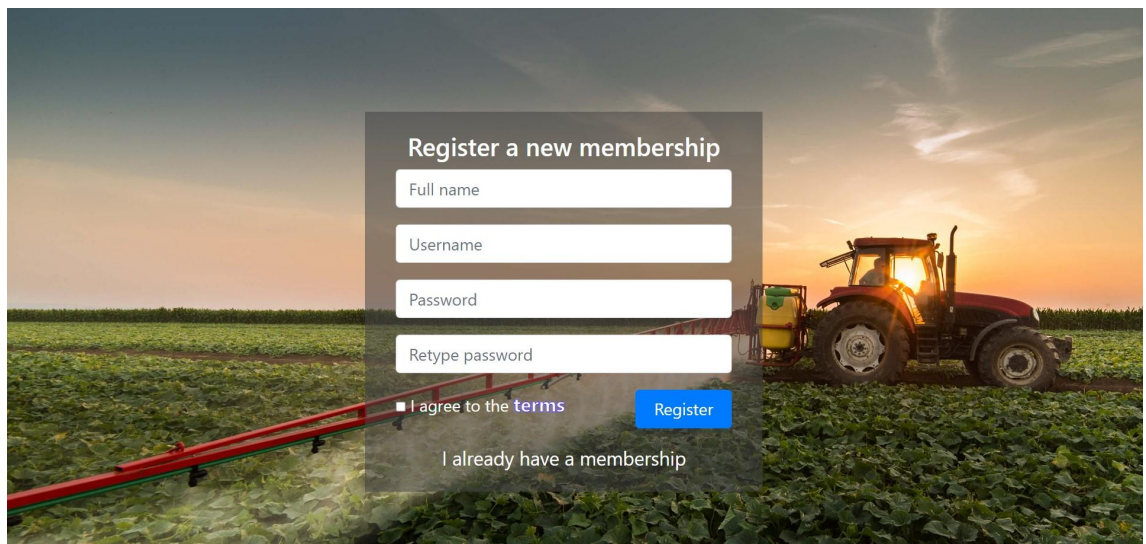


Figure 3. Registration page

In order to register a new user, only a few pieces of information will be required:

- Full name
- Username
- User Password
- Accept application terms and conditions

From this point, the user will be asked for a username and password when logging in for further access to the capabilities of the application and device. If the filled-in data is incorrect, does not match or the terms were not accepted, there will be a notification clarifying that missing information should be corrected or confirmed. Examples of such notifications are shown in Figures 3 and 4.

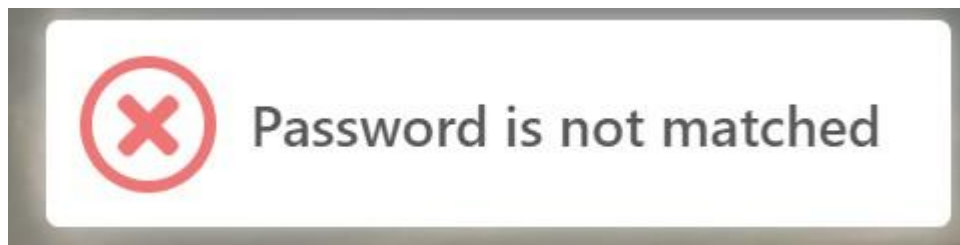


Figure 4: Incorrect password notification

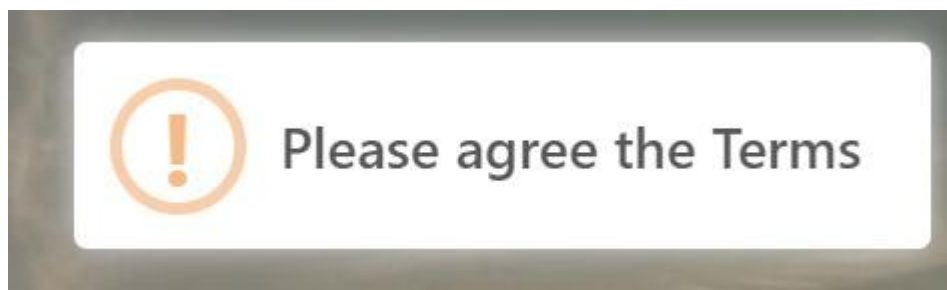


Figure 5: Terms and conditions are not accepted by the user

After the new user has filled in all the necessary parts and agreed with the terms, the user will automatically be transferred to the main page in which he will already be able to log into the application itself.

## 2.2 Login and registration pages code overview

The design and basic functions as well as script initialization are performed in register.ejs and login.ejs files.

Auth.js is responsible for the login and registration functionality.js and user\_register.js files. Auth.js displayed in Listening 1.

```
const url = '.';

const auth = document.querySelector('#auth');
const username = document.querySelector('#username');
const password = document.querySelector('#password');
var Toast = Swal.mixin({
  toast: true,
  position: 'top-end',
  showConfirmButton: false,
  timer: 3000
});

auth.addEventListener('submit', async(evt) => {
  evt.preventDefault();
  const fetchOptions = {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      "username": username.value,
      "password": password.value
    })
  };
  const response = await fetch(url + '/login', fetchOptions);
  const json = await response.json();
  if (json.success == 1) {
    Toast.fire({
      icon: 'success',
      title: json.message
    })
    window.location.href = "http://localhost:3000/index";
  }
  else {
    Toast.fire({
      icon: 'error',
      title: json.message
    })
  }
});
```



## Listening 1. Auth.js, user authorization

The code shown in Listening 1 handles the submission of a login form on a web page. It does so by sending a POST request with the user's credentials to an API endpoint using the `fetch()` method.

The code also uses the `Swal` package to create a notification box that appears at the top-right corner of the screen and disappears after 3 seconds if a login attempt is successful. If the login attempt failed, an error message is displayed instead.

In the registration case, `user_register.js` is a file that handles new user registration.

```
const url = '.';
const register = document.querySelector('#register_user');
const fullname = document.querySelector('#fullname');
const username = document.querySelector('#username');
const pw = document.querySelector('#pw');
const cpw = document.querySelector('#cpw');
const agreeTerms = document.querySelector('#agreeTerms');

var Toast = Swal.mixin({
  toast: true,
  position: 'top-end',
  showConfirmButton: false,
  timer: 3000
});

const form_clear = () => {
  fullname.value = '';
  username.value = '';
  pw.value = '';
  cpw.value = '';
  agreeTerms.removeAttribute('checked');
}

function wait(ms) {
  var start = new Date().getTime();
  var end = start;
```

```

while(end < start + ms) {
  end = new Date().getTime();
}
}

register.addEventListener('submit', async (evt) => {
  evt.preventDefault();
  if (pw.value == cpw.value) {
    if (agreeTerms.checked) {
      const fetchOptions = {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          "username": username.value,
          "fullname": fullname.value,
          "password": pw.value
        })
      };
      const response = await fetch(url + '/register', fetchOptions);
      const json = await response.json();

      if (json.success == 1) {
        Toast.fire({
          icon: 'success',
          title: json.message,
        })
        location.replace("http://localhost:3000/login")
      }
      else {
        Toast.fire({
          icon: 'error',
          title: json.message
        })
      }
      form_clear();
    }
    else {
      Toast.fire({
        icon: 'warning',
        title: 'Please agree the Terms'
      })
    }
  }
  else {
    Toast.fire({
      icon: 'error',
      title: 'Password is not matched'
    })
  }
}

```

```

    })
  }
});

```

## Listening 2. user\_register.js, user registration

The code that is shown in Listening 2 defines a function that registers a user by sending a POST request to a server's "/register" endpoint. The function listens to the submit event of a form element with an ID of "register\_user". If the user agrees to the terms and the entered passwords match, the function sends a POST request to the server's "/register" endpoint with the entered username, full name, and password as JSON data.

Afterward, if the server returns a success response, a success message is displayed using the SweetAlert library, and the user is redirected to the login page. Otherwise, an error message is displayed. If the user does not agree to the terms, or if the entered password does not match, a warning or error message is displayed accordingly. The mongoDB database is used to store user data and later to log out. The connection to the database is made in the index file.js, and the connection to the database is shown in Listening 3.

```

//MongoDb connection
const mongoose = require('mongoose'); //mongoose
const mongoURL = 'mongodb://danils:metropolial23@cluster0-shard-00-00.bcxje.mongodb.net:27017,cluster0-shard-00-01.bcxje.mongodb.net:27017,cluster0-shard-00-02.bcxje.mongodb.net:27017/test?replicaSet=atlas-cl2wx4-shard-0&ssl=true&authSource=admin';
mongoose.connect(mongoURL, { useNewUrlParser: true });

const db = mongoose.connection;
//Error
db.on('error', console.error.bind(console, 'MongoDB connection error:'));

```

## Listening 3. Database connection in index.js file

## 2.3 Front-End

The front-end is an essential part of a webpage that focuses on providing an interface for the user to interact with. It includes primarily visual elements such as the design, layout and interactive functionality.

The technologies used to develop the front-end consists of HTML, CSS and EJS. HTML and CSS are standard markup languages used for developing the visual representation of a webpage, whereas EJS allows the generation of HTML code via JavaScript in order to create templates that can be displayed dynamically and reused with ease. Below is Figure 5 which shows the index page of the website, which uses EJS to render its meters.

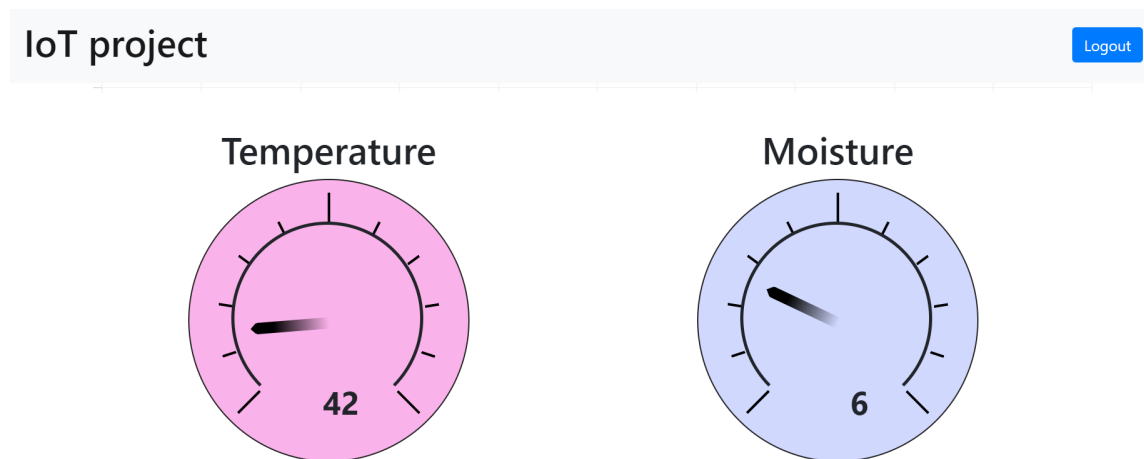


Figure 6: Index page with temperature and moisture content

The temperature and the moisture content use EJS to dynamically display the measurements on the gauge upon refreshing the page. They would be retrieving the data from the backend before displaying the measured output, however as of now they are only displaying dummy values which are randomly generated upon refreshing the page. In reality, the proper values would be provided by the MQTT. Below is Listing 4 which shows how the gauges receive the data in order to display them.

```
<div id="demoGauge" class="gauge" style="
  --gauge-bg: #FAB2EA;
  --gauge-value:<%= Math.floor(Math.random() * 60) %>;
  --gauge-display-value:<%= Math.floor(Math.random() * 60) %>;
  width:200px;
  height:200px;">
```

Listening 4: HTML & EJS code used to display specific data

In the current implementation, it only displays randomly generated values between 0 and 60. Furthermore both the value and displayed value do not correspond to each other, although it must be reiterated that they are only placeholder values. If required to use real data, it suffices to replace the contents with a value called `dummyData` that was declared in the backend and sent to the frontend via EJS. Below is Listening 5, an example of such code that performs the task.

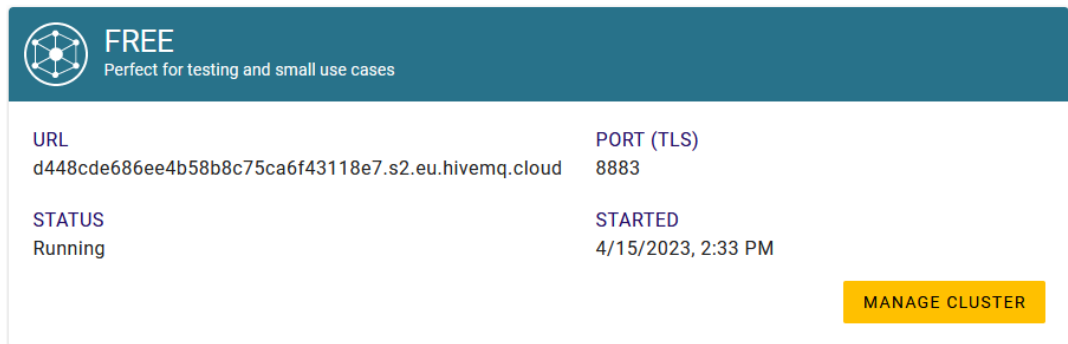
```
router.get("/index", (req, res, next) => {
  if (!req.session.user) {
    res.status(401).render(path.resolve(__dirname + '/views/login'));
  } else {
    res.status(200).render(path.resolve(__dirname + '/views/index'), { dummyData, dummyData2 });
  }
});
```

Listening 5: Sending dummyData to the front-end

The figure above shows the code from `routes.js`, which retrieves two values named `dummyData` and `dummyData2` that were previously declared. These two values can then be used in the front-end side.

## 2.4 MQTT Broker

As the main communication protocol, the MQTT was chosen. The choice of platform came upon the HiveMQ, which provides free MQTT broker. After some registrations, the cluster was formed:



**FREE**  
Perfect for testing and small use cases

**URL**  
d448cde686ee4b58b8c75ca6f43118e7.s2.eu.hivemq.cloud

**PORT (TLS)**  
8883

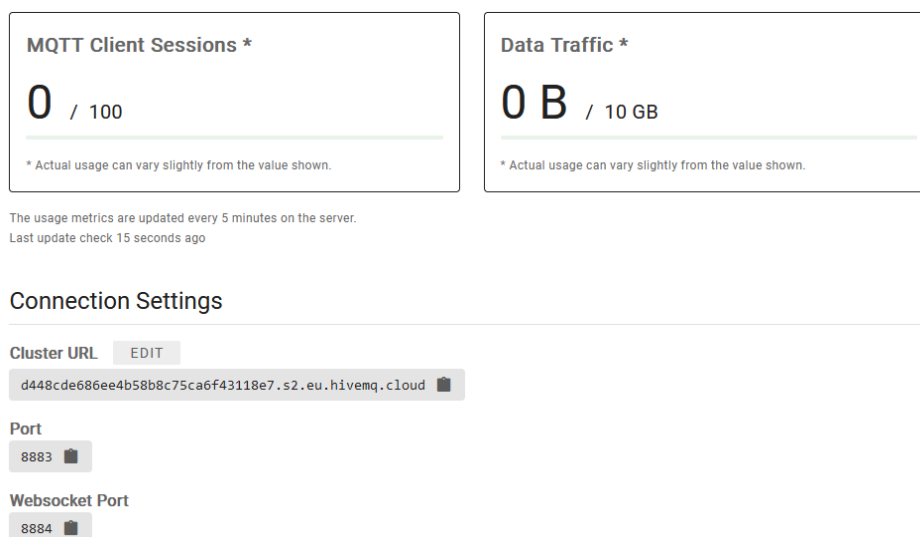
**STATUS**  
Running

**STARTED**  
4/15/2023, 2:33 PM

**MANAGE CLUSTER**

Figure 7. HiveMQ cluster.

Inside the client, the cluster can be managed, along with displaying number of current client sessions and data traffic. Also it provides an URL link with the port values:



**MQTT Client Sessions \***  
0 / 100  
\* Actual usage can vary slightly from the value shown.

**Data Traffic \***  
0 B / 10 GB  
\* Actual usage can vary slightly from the value shown.

The usage metrics are updated every 5 minutes on the server.  
Last update check 15 seconds ago

**Connection Settings**

**Cluster URL** **EDIT**  
d448cde686ee4b58b8c75ca6f43118e7.s2.eu.hivemq.cloud

**Port**  
8883

**Websocket Port**  
8884

Figure 8. MQTT cluster manager.

After setting up the MQTT cluster, the next step would be connecting it to the website. Firstly, with the help of terminal, MQTT needs to be installed:

```
PS C:\Users\averb\mizu-ten> npm install mqtt --save
```

Listening 6: Installing mqtt

Then, inside the **routes.js** file, a few lines need to be added, that of MQTT constant and the broker URL:

```
const mqtt = require('mqtt');  
const addr = 'd448cde686ee4b58b8c75ca6f43118e7.s2.eu.hivemq.cloud';
```

Listening 7: Inputting broker url

Third step is to add a few more constants of broker port, client id, topic, username and password:

```
const brokerPort = 8883;  
const clientId = 'web-client';  
const username = 'metropolia';  
const password = 'P4ss12345!';  
const topic = 'my/topic';
```

Listening 8: Storing broker parameters in variables

Finally, the client constant can be initialised, using the previously listed values:

```
const client = mqtt.connect(`${addr}:${brokerPort}`, {  
  clientId: clientId,  
  username: username,  
  password: password,  
});
```

Listening 9: Connecting to MQTT

### 3 Embedded

This section describes the components used to build the IoT device.

RemoteXY is a software development platform that enables the creation of mobile and web applications capable of remotely controlling microcontrollers, such as the Arduino board or ESP8266 module, through the utilization of Wi-Fi or Bluetooth connectivity. The platform provides a straightforward approach to

designing personalized interfaces that incorporate various widgets such as buttons, sliders, and text fields, among others, which enable users to remotely manage and supervise their devices.

The ESP8266 is a cost-effective microcontroller module that is equipped with Wi-Fi capabilities and can be programmed using the Arduino Integrated Development Environment (IDE). The device possesses the capability to establish connections with Wi-Fi networks and engage in communication with other devices via the internet, rendering it a suitable choice for projects pertaining to the Internet of Things (IoT). The device possesses an integrated Wi-Fi stack and TCP/IP protocol suite, and can be programmed via the Arduino IDE or other programming languages.

The Arduino board is a programmable microcontroller board that employs diverse microcontrollers and can be programmed through the utilization of the Arduino Integrated Development Environment (IDE). This technology has the capability to regulate various hardware components such as sensors, motors, and lights. The platform boasts a substantial user and developer community, thereby facilitating the accessibility of support and resources for diverse projects.

The integration of ESP8266 and Arduino board can yield a robust framework for constructing remote control applications. RemoteXY is a software tool that enables the development of personalized user interfaces that can be conveniently accessed through a mobile device or web browser. The ESP8266 module has the capability to establish a wireless connection with a Wi-Fi network and facilitate communication with the Arduino microcontroller board. The utilization of the Arduino board enables the manipulation of hardware components, such as motors or lights, contingent upon the directives obtained from RemoteXY.



The present script has been developed in the Arduino Integrated Development Environment (IDE) for a project that encompasses the utilization of a moisture sensor linked to an Arduino board, an OLED display, and a control interface created through the RemoteXY library. The program facilitates bidirectional data transfer between the control interface and the system, enabling the presentation of moisture readings and other relevant parameters on the OLED display.

The source code incorporates the subsequent libraries:

The SPI.h library offers a set of functions that enable communication with peripheral devices via the Serial Peripheral Interface (SPI) bus.

The Wire.h library offers a set of functions that facilitate communication with devices utilizing the I2C protocol. On the other hand, the Adafruit\_SSD1306.h library provides a range of functions that enable control of an OLED display.

The Serial.h library offers a set of functions that facilitate the establishment of serial communication between the Arduino board and the ESP8266 WiFi module.

RemoteXY.h: provides functions for creating and communicating with a remote control interface using a WiFi or Bluetooth connection.

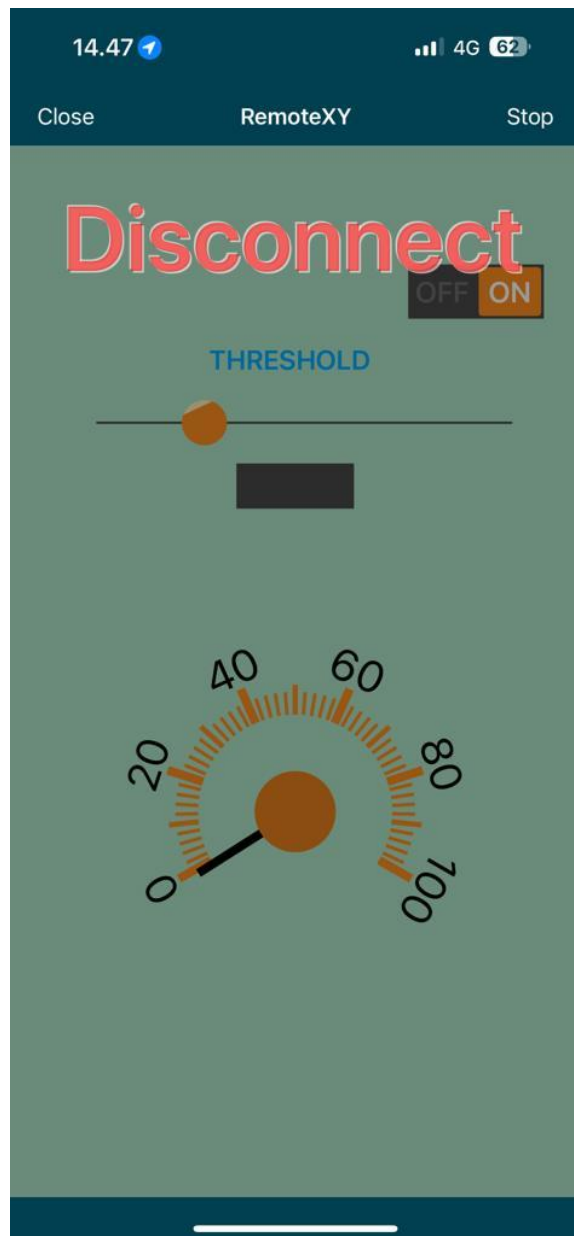


Figure 9. Mobile app interface

The RemoteXY library utilizes several connection settings that are defined by the user, including REMOTEXY\_SERIAL\_RX, REMOTEXY\_SERIAL\_TX, REMOTEXY\_SERIAL\_SPEED, REMOTEXY\_WIFI\_SSID, REMOTEXY\_WIFI\_PASSWORD, and REMOTEXY\_SERVER\_PORT. These settings are integral to establishing a connection between the RemoteXY library and the user's device.

The variables PIN\_SWITCH\_1, PIN\_BUTTON\_1, and AOUT\_PIN are being utilized. The objective is to establish the pins and thresholds of diverse components.

OLED\_ The reset pin for the OLED display is a designated input pin that is responsible for resetting the display module to its default state.

RemoteXY\_ The RemoteXY library utilizes a configuration structure known as CONF.

The RemoteXY library utilizes a struct that specifies the input, output, and additional variables employed by the library.

The initialization of the RemoteXY library and the OLED display is carried out through the setup() function. The DrawTitles() function is invoked to exhibit the project title on the OLED display.

The loop() function is responsible for managing the primary program logic. The initial step involves invoking the RemoteXY\_Handler() function, which facilitates the modification of the input and output variables specified in the RemoteXY structure, relying on the data obtained from the remote control interface.

Subsequently, the system retrieves the analog data from the moisture sensor, transforms it into a percentage value, and designates it to the RemoteXY.instrument\_1 variable, denoting a bar graph on the remote control interface. The digital output pin PIN\_SWITCH\_1 is established in accordance with the state of the RemoteXY.switch\_1 variable.

Subsequently, the aforementioned function proceeds to refresh the OLED display by incorporating the moisture readings and a corresponding bar graph. The moisture reading value is formatted using the Format() function prior to being presented on the OLED display.

The DrawTitles() function is invoked by the setup() function and solely exhibits the project title on the OLED display.

The Format() function is a bespoke function that enables the formatting of a numerical value to a designated number of digits and decimal places. The loop() function utilizes the moisture reading value formatting process to exhibit it on the OLED display.

The ESP8266 board is connected to a MQTT broker hosted on the HiveMQ cloud utilizing MQTT protocol. The configuration of the client involves establishing a secure WiFi connection to the broker, followed by the publication of sensor data acquired from a DHT11 sensor and a moisture sensor to a designated topic on the broker. The software application utilized by the user is programmed to establish a connection with a central messaging broker and register for updates on a specific subject. These updates are subsequently utilized to regulate the operational status of a Light Emitting Diode (LED) that is physically linked to the circuit board. The RemoteXY library is employed for the purpose of designing the user interface that enables the manipulation of the LED and the presentation of sensor data.

## **4 Conclusion**

In the end, the product turned out to be a success. It was tested for the time period of over a month and fully completed its main assignments. In addition, an MQTT cluster, Web application and a mobile application are all available for users to interact with.

