# **Moisture Monitoring System**

Contents

1. **Project description**

* Goal
* Target Group
* Effect and benefit

1. **Components used**

* Hardware
* Esp32
* Soil moisture sensor
* cables
* Software
* IDE
* language
* Cloud
* Azure
* IOT Hub
* Azure Function
* Storage
* Twilio
* Externa API
* Dashboard

1. **Implementation**

* IOT Architecture design

1. **Snap shots**
2. **References**
3. **Project description**

The project aims at helping gardeners maintain their plants in good health by measuring the water content in the soil. When the moisture levels are low or when the plants are not supplied with required amount of water, the soil moisture measuring system will remind the gardener to water the plants. Along with the moisture measurement the gardener is also provided with the weather update about possibilities of rain or drizzle.

* **Goal**:

Soil moisture is a measure of soil health, the water content present in a certain area of the ground. All plants need to be in a specific moisture range – most plants thrive in soil with a moisture level that range 20% to 60%. So, by reminding the gardener to water plants when needed, keeps the plants nourished. Additionally with the update of predicted rain, the gardener would not require watering his outdoor garden.

* **Target Group**:

People who love plants and grow them in their balconies and gardens, are the people who would benefit from this soil monitoring system.

* **Effect and benefit**:

By maintaining the level of moisture in the soil we avoid plants from dying, the plants remain healthy who in turn contribute to healthy eco system.

By providing the rain update we not only save the consumption of water but also save plants from being destroyed because of over watering.

1. **Components used**

* **Hardware**
* **Capacitive Soil Moisture Sensor v2.0:**

Shape

Description automatically generated with low confidence

The electronic component known as capacitor consist of three pieces. A positive plate, a negative plate, and the space in between the plates, known as the dielectric.

A capacitive moisture sensor works by measuring the changes in the capacitance caused by the changes in the dielectric. Capacitive measuring basically measures the dielectric that is formed by the soil and the water is the most important factor that affects the dielectric.

The capacitance of the sensor is measured by the means of a 555 based circuit that produces a voltage proportional to the capacitor inserted in the soil. We then measure this voltage by use of an Analog to Digital converter which produces a number that we can interpret as soil moisture.

The sensor comes with analog sensor cable connected to 3pin Jconnector.

Why Capacitive Soil Moisture Sensor v2.0?

It is made of corrosion-resistance material which gives long service life.

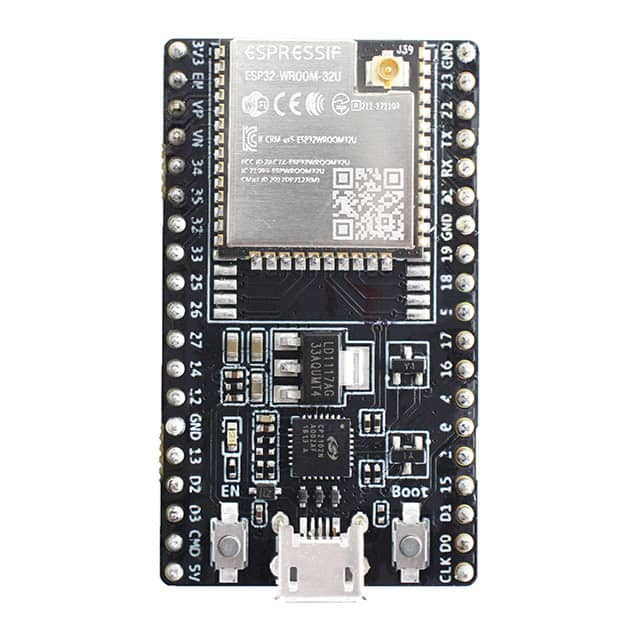
Features:

1. Supports 3-pin Gravity sensor interface
2. Analog output
3. Includes an onboard voltage regulator which gives it an operating voltage range of 3.3 ~ 5.5v.

* **Esp32 -Wroom -EXPRESSIF**:

I selected the ESP 32 as microcontroller, because of its multiple 12bits ADC inputs and the capability to generate PWM signals at higher frequencies than e.g., an Arduino Uno can do.

ESP32 has 32-bit processor. TheEsp32 is a dual-core 160MHz to 240MHz CPU, whereas the ESP8266 is a single-core processor that runs at 80MHz.



The ESP32 comes with more GPIOs with multiple functions. The ESP32 supports analog measurements on 18 channels (analog-enabled pins) versus just one 10-bit ADC pin on the ESP8266.

Reading an analog value with the ESP32 means you can measure varying voltage levels between 0 V and 3.3 V.

The voltage measured is then assigned to a value between 0 and 4095, in which 0V corresponds to 0, and 3.3 V corresponds to 4095. Any voltage between 0 V and 3.3 V will be given the corresponding value in between.

* **Cables**:
* **USB cable**:

USB cable is used to supply power to Esp32.  To power the ESP32 we use the USB port. The MCU includes a micro USB port through which you can both supply power to the board, and implement serial communication with the host computer for uploading a sketch.

* **Jumper wires**:

Shape

Description automatically generated

To connect the sensor to the ESP32.

* **Software**
* **IDE and Languages**:

Arduino IDE is used to write the code in C language to drive the MCU along with sensor.

Visual studio 19 IDE is used to write the code in C# for AZURE functions.

* Along with .NET Core Framework which helps in building application and many  .NET Core library which is provided as a NuGet package.
* **Cloud Service:**

**Azure:**

Using Azure as Paas [ platform as a service]-

Microsoft Azure can provide everything required to build an IoT solution without the need to invest in costly and fault-prone infrastructure that’s difficult to scale.

In order to implement our IOT solution of soil moisture measurement system.

**Azure IoT Hub**:

IOT Hub is the Cloud Gateway that stands between devices and the backend process. It acts as the interface point for the connected ‘things’.

It is a managed connectivity service that gives the ability to create a network of our IoT devices and providing the functionality to communicate with, manage, and collect data from all of the devices.

It gives devices that are capable of communicating over internet protocol a highly available cloud endpoint to connect to in order to interact with the back-end solution. This could be the devices sending data to the backend, or the backend sending instructions back to the device – known as bidirectional messaging.

IoT Hub implements cloud-scale bidirectional messaging which allows messages to be queued up until they can be processed. It also supports at-least-once messaging.

It supports the top three protocols for IoT, which are HTTP, AMQP and MQTT.

MQTT:

The Esp32 MCU (the device) communicates with IOH hub using MQTT protocol using libraries in the Azure IOT SDKs.

Why MQTT?

* 1. Uses less battery power to publish and subscribe data.
  2. Connection is secured using TLS /SSL where traffic is encrypted on the Web.

### Device identity:

IoT hub has an identity registry that stores information about the devices and modules permitted to connect to it. Before a device or module can connect, there must be an entry for that device or module in the IoT hub's identity registry, in our project our esp32 is the device being connected and registered on the IOT Hub. The ESP32(device) authenticates with the IoT hub based on credentials stored in the identity registry.

The *SAS-based token* method provides authentication for each call made by the device to IoT Hub by associating the symmetric key to each call methods of authentication between the device and the IoT Hub.

**Azure Functions**:

 Azure functions is a piece of code that is written to do a specific tasks, hence it is lightweight by design. Azure functions are serverless and do not require any Web servers or virtual machines to deploy and run.

Azure functions’ execution is triggered when an event is fired. Azure functions setup provides dozens of triggers that can be configured when an azure functions is executed.

In this project one of the Azure function’s tasks is to Collect and process data received by the IOT hub. The function is triggered when a new data arrives at the IOT Hub. The processed data is sent to various database with the help of other azure functions

**Storage**:

**Azure Cosmos DB** is a fully managed NoSQL database for modern app development.

Azure Cosmos DB supports strict JSON items only. The type system and expressions are restricted to deal only with JSON types.

Why Cosmos DB?

1. Schemas are not required to be defined at design time unlike relational database.

2. No need to maintaining complex parent-child entity relationships as in a relational database.

3. Easy to create and maintain.

4. Since no schema design or table creation is required any changes in the structure of data can easily be enforced across databases.

5. Deeply integrated with key Azure services used in modern (cloud-native) app development including Azure Functions.

Security:

With Azure Cosmos DB, your data can be replicated globally by adding Azure regions with just one click.

**BLOB STORAGE**:

Used as cold storage database.

Azure Blob storage is Microsoft's object storage solution for the cloud.

Blob storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data.

A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs.

**Block blobs** type of storage is used to store all data in this project.

Block blobs store text and binary data. Block blobs are made up of blocks of data that can be managed individually. Block blobs can store up to about 190.7 TiB.

* **Twilio:**

**Twilio** is an American cloud communication platform as a service (CPaas) company based in California. Twilio allows software developers to programmatically make and receive phone calls, send and receive text messages, and perform other communication functions using its web services API.

In this project we send send text messages by using Twilio bindings in Azure Functions. Azure Functions supports output bindings for Twilio.

Twilio REST API helper library was added.

Azure function written in c#,  sends an SMS by using the Twilio REST API. An account was needed to sign up for a Twilio trial account and this gave a Twilio Account SID and an authorisation token. A phone number was added to receive the text SMS.

Why Twilio**?**

Azure Communication Services enabling to send and receive SMS text messages using the Communication Service SMS SDKs ,was not available for our region.

* **External API:**

**SMHI Open Data API**:

An open API which provides weather data in a Json document format.

Used an azure function to collect the weather data by including the package

*SmhiWeather V0.9915-*

A .NET Standard API to allow .NET developers convenient access to SMHI's weather

forecast service.

The data obtained from the API are temperature, humidity and precipitation. Temperature and humidity values are in the decimal format whereas precipitation in integers from 0 to 6. Values # and 4 depicting rain or drizzle.

**NTP SERVER**:

NTP is a Network Time Protocol .

NTP allows networked devices, such as clocks, phones and computers, to request and receive time from a server that, in turn, receives precise time from a definitive time source, like an atomic clock.

Our device(ESP32) on a network needs to show the correct time, it's also important that all network devices are synchronized to GPS for legally traceable timestamping.

Accurate timestamping is key to root-cause analysis, determining when problems occurred and finding correlations.

In this project the date and time is fetched from pool.ntp.org- NTP server.

1. **Implementation**

* Architecture design

Diagram

Description automatically generated

The Capacitive Soil Moisture Sensor v2.0measures the moisture content of the soil and gives an analog output, which is converted corresponding to digital value by the Esp32 microcontroller (MCU) connected to it. The measurement along with the current date and time values taken from the NTP sever is serialised in Json format with the help of program written on Arduino IDE.

The MCU Esp32 with WIFI connection capabilities sends the data over azure cloud to IOT hub.

The data from the IOT hub is read, processed by the azure function. The azure function is triggered when the data arrives at IOT Hub.

The function gets the weather data from SMHI - The open API, in Json format, which is deserialized and necessary data is obtained, in this case the temperature , humidity and the precipitation data is extracted.

The moisture data in the json format is deserialized . The weather data and the moisture data are repackaged in to a new json payload.

The new Json payload is further posted to different Azure functions using their URLs.

There are Three Azure functions which are triggered on *HTTP trigger* , which are responsible to send the received data(data from the main azure function) to corresponding Cosmos DB , Blob storage and SMS to mobile(using Twilio).

Chart

Description automatically generated with low confidence

The Azure function, which is responsible for sending SMS to the client , send the message only when the moisture content is lower than the desired measurement, along with that it sends additional information about possibilities of rain and drizzle, so the client might not have to water outdoor plants in case of rain or drizzle.

Further the data from the cosmos DB is Visualised by using Power BI. The report generated can be made accessible to client if the client requests for it.

* **Snaps Shots**:

Data collected in the Cosmos DB

Graphical user interface, application, Teams

Description automatically generated

Data collected in the Blob Container:

Graphical user interface, text, application, email

Description automatically generated

View of the blob file

Graphical user interface, text, application

Description automatically generated

SMS via Twilio

Graphical user interface, text, application, chat or text message

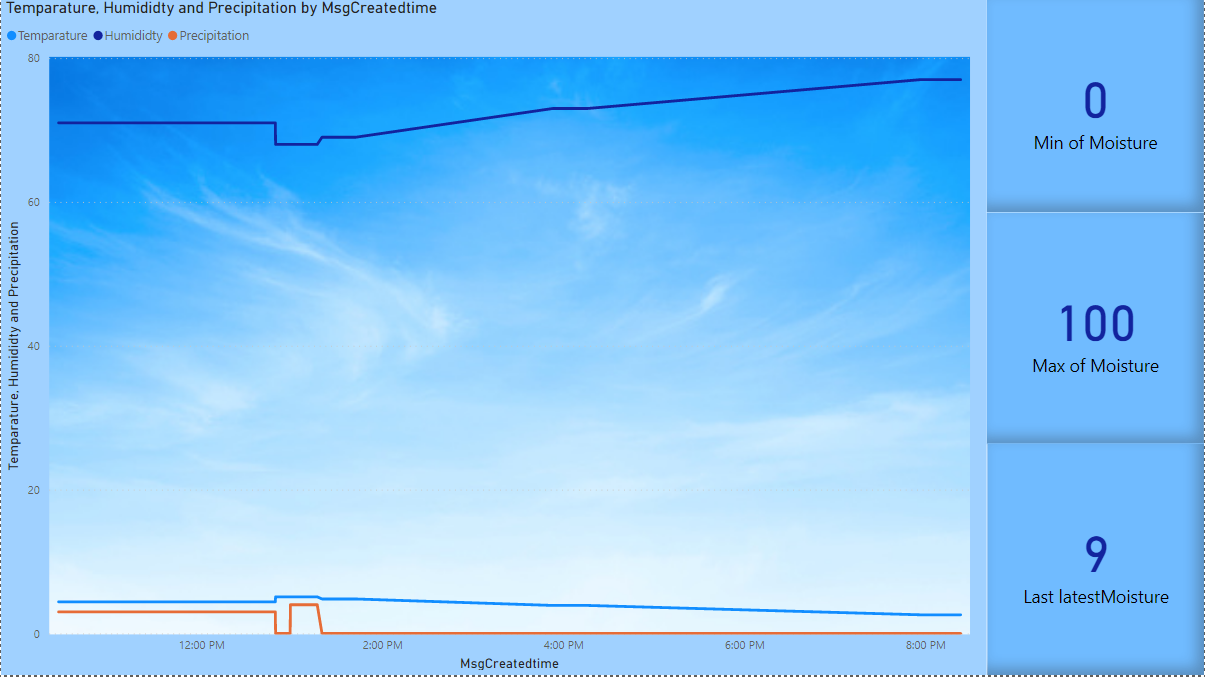
Description automatically generated

SMS 2:

Graphical user interface, text, application, chat or text message

Description automatically generated

Power BI Report:



Graphical user interface, application, PowerPoint

Description automatically generated

Resources:

<https://www.greenwaybiotech.com/blogs/gardening-articles/how-soil-moisture-affects-your-plants-growth>

<https://makeradvisor.com/>

<https://cloudacademy.com/course/introduction-to-azure-iot-hub/introductio-to-iot-hub-1/>

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-tls-support>

<https://medium.com/@nareshgongulur/sending-sms-using-azure-functions-and-twilio-fd667f8603d6>

<https://docs.microsoft.com/en-us/power-bi/collaborate-share/service-request-access>