



UNIVERSITÀ DEGLI STUDI DI GENOVA

**DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E
DELLE TELECOMUNICAZIONI**

**CORSO DI STUDIO IN INGEGNERIA ELETTRONICA E TECNOLOGIE
DELL'INFORMAZIONE**

Tesi di Laurea Triennale

Febbraio 2024

**Progetto e implementazione di un sistema embedded
per il monitoraggio dello stato di salute di una pianta**

**Design and development of an embedded system
for plant health monitoring**

Candidato: Giacomo Tartara

Relatore: Prof. Riccardo Berta

Correlatore: Dott. Matteo Fresta

Sommario

Il presente elaborato ha come intento quello di illustrare la progettazione e la realizzazione di un sistema embedded volto all'acquisizione di dati relativi alle condizioni di salute e ambientali di una pianta. Tale sistema si occupa inoltre del mantenimento di un livello di umidità ottimale del terreno in cui giace la pianta.

Per la realizzazione è stato utilizzato il microcontrollore Arduino UNO WiFi Rev2 su cui è stata montata una Weather Shield opportunamente modificata per permettere l'acquisizione e l'elaborazione dei valori di umidità del suolo e dell'aria, della temperatura e della luminosità ambientale, oltre a controllare l'irrigazione tramite una pompa ad immersione. Tramite comunicazione Wi-Fi, i dati raccolti vengono inviati a Measurify, un API framework con un database, utilizzato per salvare tutti i valori ricevuti e per interfacciarsi successivamente con un'applicazione per smartphone realizzata con Flutter. Quest'ultima permette all'utente finale di visualizzare lo stato attuale dei valori e un grafico raffigurante le misurazioni effettuate in precedenza.

Indice

1. Introduzione	4
2. Metodi e strumenti utilizzati	6
2.1. Arduino UNO WiFi Rev2	6
2.2. SparkFun Weather Shield [DEV-13956]	7
2.2.1. Sensore di luminosità ALS-PT19	7
2.2.2. Sensore di umidità/temperatura Si7021	8
2.3. Sensore di umidità del suolo.	8
2.4. Kit irrigazione	9
2.5. Arduino IDE e librerie	11
2.6. Codice Arduino.	12
2.7. Measurify.	15
2.8. Flutter	16
3. Sperimentazione e risultati	17
4. Contributo personale e considerazioni conclusive	21
5. Riferimenti bibliografici	22
6. Ringraziamenti	23

1. Introduzione

L'avvento dell'Internet of Things (IoT) [1] ha introdotto soluzioni innovative per il monitoraggio e la gestione di una vasta gamma di scenari, tra cui l'agricoltura. In questo contesto, la presente tesi si propone di esplorare la realizzazione di un sistema embedded (un sistema progettato per eseguire una funzione specifica e dedicata) orientato all'IoT e finalizzato al monitoraggio di parametri ambientali cruciali per lo stato di salute di una pianta, quali umidità del suolo, umidità dell'aria, temperatura e luminosità.

Per Internet of Things (IoT) si intende la rete interconnessa di oggetti fisici, incorporati con elettronica, software, sensori e attuatori, che permette a questi dispositivi, detti things, di scambiare dati fra loro attraverso Internet. Nel corso degli ultimi anni questa tecnologia è diventata sempre più importante poiché ha permesso l'interazione tra il mondo fisico e quello digitale, sia in ambito privato che industriale, senza alcun confine applicativo. È molto facile, infatti, immaginare un contesto in cui un ecosistema IoT possa agevolare la vita di tutti i giorni o addirittura migliorare il processo produttivo e ridurre le risorse utilizzate e l'impatto ambientale di un'azienda. Basti pensare a una smart home in cui i vari elettrodomestici e impianti domestici sono connessi alla rete e possono quindi essere gestiti in automatico o da remoto, con l'obiettivo di ridurre i consumi e incrementare il comfort di chi vi abita. Un'ulteriore applicazione delle tecnologie IoT potrebbe verificarsi utile nel mondo dell'agricoltura, dove grazie al monitoraggio di parametri climatici si può, per esempio, ottimizzare l'utilizzo dell'acqua o di qualsiasi altro prodotto, in modo da garantire la qualità del raccolto promuovendo allo stesso tempo una coltivazione più sostenibile.

Il numero di oggetti IoT connessi a Internet in grado di raccogliere, elaborare e condividere dati e informazioni sull'ambiente circostante è in rapida e costante crescita, come di conseguenza le tecnologie che permettono tutto ciò. Ogni sistema IoT segue un'architettura di riferimento composta da alcuni elementi chiave:

- **Things:** i dispositivi e i sensori integrati nel sistema IoT;
- **Network:** la tecnologia di rete che permette l'interconnessione tra le things;
- **Cloud:** lo spazio che conserva ed elabora i dati generati dall'intero sistema.

A questi può essere aggiunto un livello di interfaccia utente che permetta di visionare i dati raccolti e interagire con il sistema.

Nel nostro contesto specifico le things sono rappresentate dal microcontrollore Arduino e dai vari sensori e attuatori gestiti da esso. Tale microcontrollore, connesso a una rete WiFi, è in grado di inviare i valori raccolti al cloud per mezzo di un API Framework chiamato Measurify [2], utilizzando semplici richieste POST HTTPS. Qui i dati vengono elaborati, memorizzati e resi disponibili per l'applicazione smartphone dedicata alla loro esposizione verso l'utente. L'app è stata sviluppata con Flutter [3], un framework open-source di Google che permette la creazione di interfacce utente (UI) multiplatforma, ed è basata su due pagine:

- Una schermata "Home", in cui vengono visualizzati gli ultimi valori memorizzati sul cloud di umidità del suolo, umidità dell'aria, temperatura e luminosità ambientale. La pagina viene aggiornata a ogni sua apertura, o con un tasto "Aggiorna", mediante una richiesta GET HTTP all'opportuno indirizzo del server contenente Measurify;
- Una schermata chiamata "Chart", in cui vengono rappresentate in un grafico le quattro serie temporali, relative ai rispettivi valori, delle ultime 20 misurazioni salvate, dando la possibilità all'utente di selezionare quali di esse visualizzare.

Nella figura seguente viene rappresentato uno schema riassuntivo dell'intero sistema, grazie al quale si può notare la divisione in due sottosistemi che si interfacciano l'uno con l'altro attraverso l'API Framework Measurify:

- Un primo sottosistema che comprende la scheda Arduino UNO WiFi Rev2 [4], che deve essere connessa a Internet mediante Modem Router WiFi o Hotspot da telefono cellulare per poter mandare i dati acquisiti a Measurify; la SparkFun Weather Shield [DEV-13956] [5] che va montata sopra la board Arduino e in cui sono integrati il sensore di luminosità ALS-PT19 [6] e quello di umidità dell'aria e temperatura Si702 [7]; il sensore di umidità del suolo e il kit d'irrigazione (relè + pompa + tubo per l'acqua) opportunamente collegati;
- Un secondo sottosistema composto dall'applicazione per smartphone che comunica con Measurify da cui ricavano i valori da mostrare all'utente.

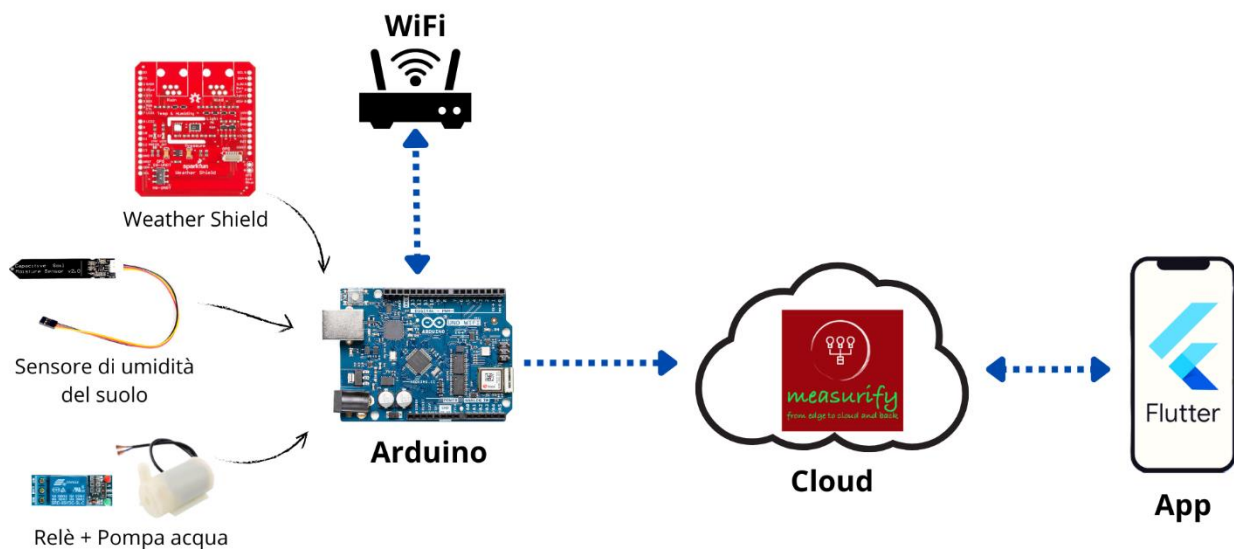


Figura 1: Schema riassuntivo

2. Metodi e strumenti utilizzati

2.1. Arduino UNO WiFi Rev2

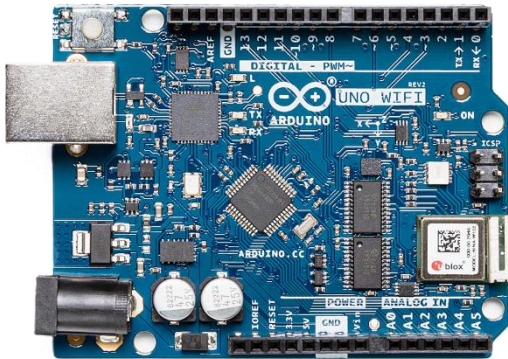


Figura 2: Arduino UNO WiFi Rev2



Figura 3: Alimentatore 9V

Per la gestione dei sensori e degli attuatori, l'elaborazione e l'invio dei dati raccolti in questo progetto viene usato un Arduino UNO WiFi Rev2 [4]. Il microcontrollore su cui si basa è l'ATmega4809 a 8-bit, prodotto da Microchip Technology e facente parte della famiglia AVR. Questa scheda costituisce una versione avanzata della classica Arduino UNO integrando il modulo WiFi e Bluetooth NINA-W102 di fondamentale importanza per stabilire connessioni alle reti wireless operanti nella banda a 2,4 GHz. Tale caratteristica la rende retrocompatibile con molti progetti sviluppati per Arduino UNO, in particolare rende possibile l'uso della Weather Shield la cui disposizione dei pin è stata creata proprio per combaciare con quella di quest'ultima board. Arduino UNO WiFi Rev2 è infatti dotata di 14 pin digitali I/O, 6 di input analogici, un connettore USB di tipo-B per poter essere collegata al computer e alimentata a 5V e uno di tipo jack dedicato a un'alimentazione esterna con una tensione da 7V a 12V. Un aspetto molto rilevante per il progetto descritto nella presente tesi è la necessità di usare un alimentatore da 9V, come quello in Figura 3. È essenziale seguire questo accorgimento poiché permette di avere in uscita dal pin Vin di Arduino la tensione che serve alla Weather Shield per funzionare correttamente.

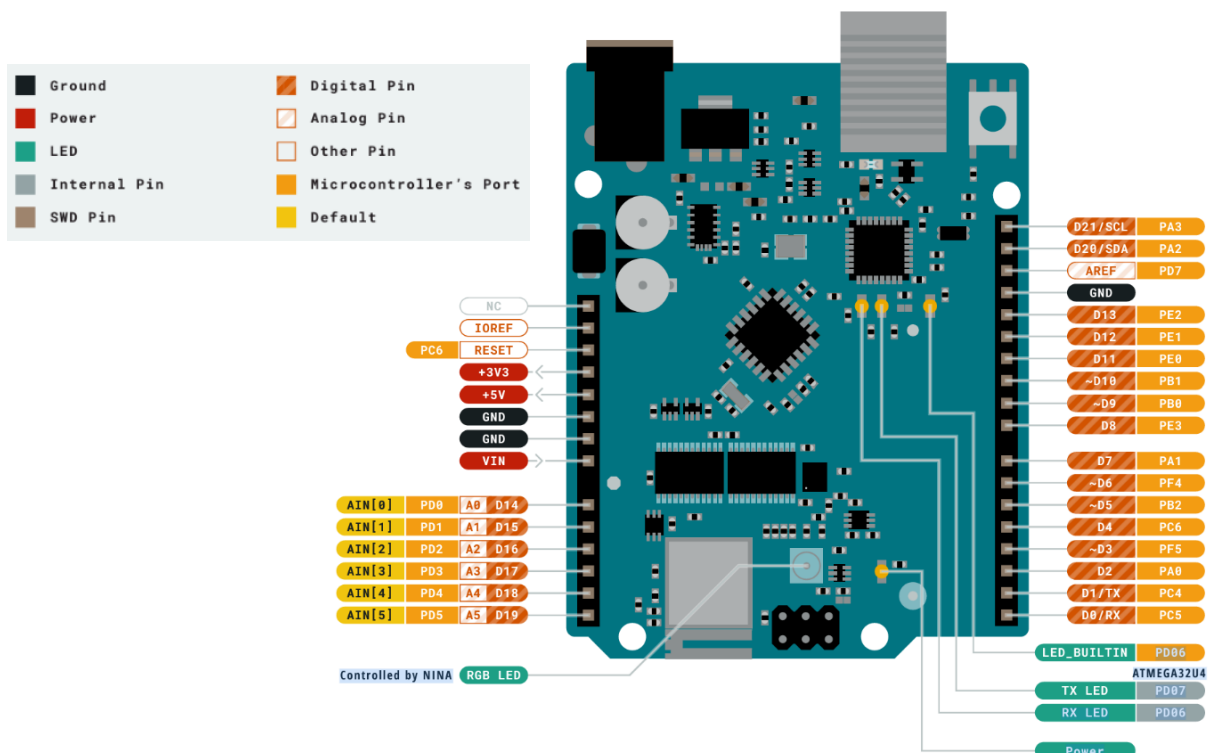


Figura 4: Pinout della scheda Arduino UNO WiFi Rev2

2.2. SparkFun Weather Shield [DEV-13956]

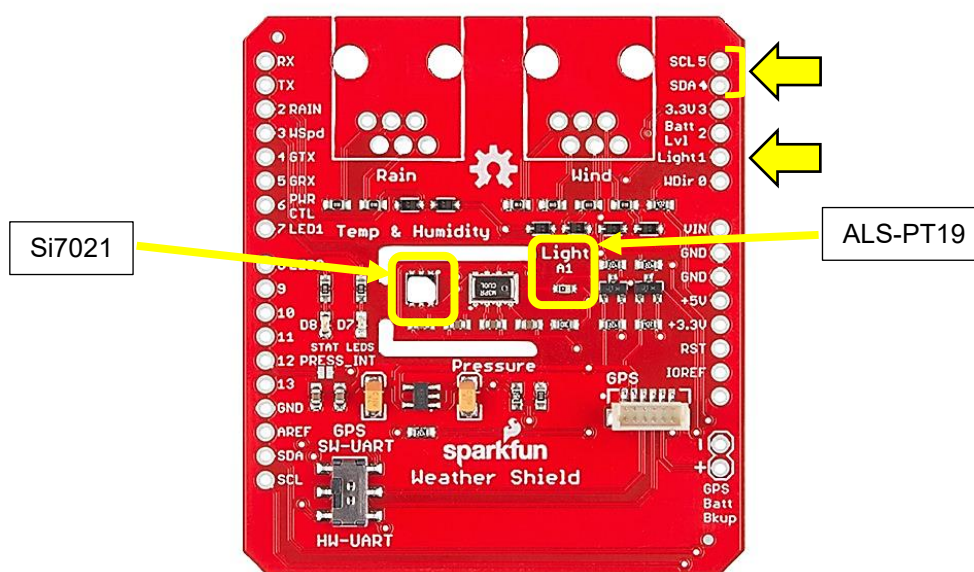


Figura 5: Sparkfun Weather Shield

La Sparkfun Weather Shield [5] è una scheda facilmente utilizzabile con Arduino UNO e qualsiasi sua versione che abbia uguali dimensioni, stesso pinout e una tensione operativa di 5V. Infatti, questa Shield viene inserita direttamente sopra Arduino e garantisce l'accesso a diversi sensori integrati su di essa. Tra questi, di particolare interesse per lo scopo di questo progetto sono il sensore di luminosità ALS-PT19 e quello di umidità e temperatura Si7021, che verranno approfonditi nei paragrafi successivi. Inoltre, sono presenti anche un sensore di pressione barometrica MPL3115A2, un connettore per un eventuale modulo GPS e lo spazio per connettere i sensori aggiuntivi di pioggia e vento. La Weather Shield è dotata di regolatori di tensione incorporati e convertitori di segnale che le permettono di operare in un range di tensione tra i 3.3V e i 16V.

2.2.1. Sensore di luminosità ALS-PT19



Figura 6: ALS-PT19

Per misurare la luminosità ambientale di cui può godere la pianta, in questo contesto viene usato il sensore ALS-PT19 [6], prodotto da Everlight e integrato nella Sparkfun Weather Shield (Figura 5). È composto da un phototransistor di esigue dimensioni progettato per il montaggio superficiale su scheda di circuito stampato (SMD). È caratterizzato da una risposta spettrale simile a quella dell'occhio umano, una buona linearità dell'uscita su un'alta gamma d'illuminazione e una tensione di alimentazione compresa tra i 2.5V e i 5.5V.

L'uscita di questo sensore è analogica, per cui sarà possibile leggerla attraverso uno dei pin di input analogici di Arduino. In particolare, il pin dedicato alla lettura del valore di luminosità sulla Weather Shield è quello denominato con "Light 1", evidenziato in Figura 5 e corrispondente ad "A1" della scheda Arduino.

2.2.2. Sensore di umidità/temperatura Si7021



Figura 7: Si7021

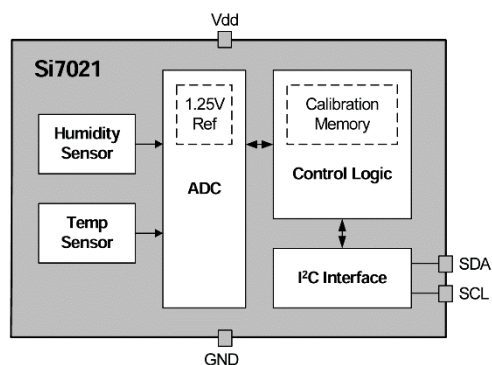


Figura 8: Diagramma blocchi funzionali

Il sensore Si7021 [7], prodotto dalla Silicon Laboratories, è un dispositivo progettato per misurare sia l'umidità che la temperatura ambientale. Si tratta di un circuito integrato (IC) CMOS monolitico che comprende, oltre agli elementi del sensore di temperatura e umidità, un convertitore analogico-digitale, l'elaborazione del segnale, alcuni dati necessari alla calibrazione, salvati sulla memoria non volatile del chip direttamente in fabbrica, e un'interfaccia I²C.

Quest'ultima componente utilizza il noto protocollo di comunicazione seriale I²C (Inter Integrated Circuit) per la trasmissione dei dati raccolti alla scheda Arduino sfruttando solo due linee di segnale: SDA (Serial Data Line) e SCL (Serial Clock Line), da cui derivano gli omonimi pin della Weather Shield (Figura 5) corrispondenti agli ingressi analogici "A4" e "A5" di Arduino.

2.3. Sensore di umidità del suolo



Figura 9: Capacitive Soil Moisture Sensor v1.2

Per acquisire le misurazioni dell'umidità del terreno viene usato il sensore capacitivo versione 1.2 [8] in Figura 9. È realizzato di un materiale resistente alla corrosione e non ha parti metalliche esposte, caratteristica che ne garantisce una lunga durata e la possibilità di essere inserito direttamente nel suolo fino alla linea limite bianca, oltre la quale ci sono componenti elettroniche che si danneggerebbero a contatto con l'acqua. Tra queste è presente un regolatore di tensione che definisce una tensione operativa da 3.3V a 5.5V.

Un sensore capacitivo, rispetto a uno resistivo, offre inoltre una più accurata lettura dell'umidità del suolo. Esso, infatti, forma un condensatore con il terreno, in cui la capacità varia in base alla quantità d'acqua presente. La capacità viene poi convertita in un valore di tensione analogico.

Questo modulo è fornito di un connettore a tre pin da cui escono tre cavi da collegare opportunamente alla Weather Shield. Più precisamente, ai pin VCC e GND deve essere collegata l'alimentazione a 3.3V rispettando la polarizzazione, mentre ad AOUT un ingresso analogico. Nel nostro caso specifico, non avendo più accesso ad alcun ingresso analogico di Arduino in quanto tutti occupati dalla Shield della Sparkfun, sono state necessarie delle piccole modifiche a quest'ultima. Analizzando lo schematico della scheda, è stato deciso di sfruttare uno dei pin dedicati al connettore per il sensore del vento (Figura 10), non utilizzato in questo progetto e la cui lettura è disponibile sul pin denominato "WDir 0", attraverso l'ingresso A0 di Arduino. Il cavo corrispondente ad AOUT è stato quindi saldato sulla Weather Shield nel pin 2 del connettore per il sensore del vento e sono state modificate due resistenze presenti sul percorso per far passare il segnale analogico senza subire perturbazioni. In particolare, si è dovuto cortocircuitare la resistenza R13 e sostituire la R17 con un circuito aperto (Figura 11).

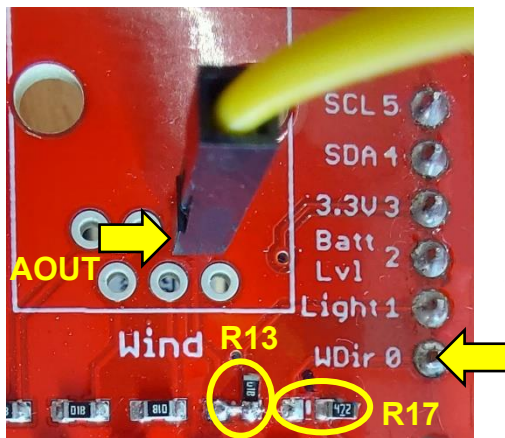


Figura 10: Weather Shield modificata

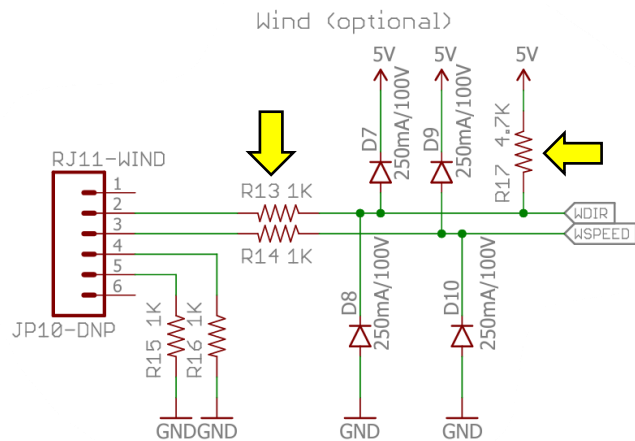


Figura 11: Schematic Weather Shield sensore vento

Per poter utilizzare correttamente il sensore capacitivo di umidità del suolo è necessario svolgere ancora un passaggio: bisogna infatti effettuare una calibrazione tramite dei test. Dopo averlo correttamente collegato ad Arduino, si devono salvare all'interno del codice del microcontrollore due valori misurati: il primo deve essere memorizzato quando il sensore è completamente asciutto, mentre il secondo quando è immerso in un bicchiere d'acqua fino alla linea bianca. Essi corrisponderanno rispettivamente al minimo e al massimo livello di umidità misurabile.

Infine per esprimere l'umidità del terreno in valore percentuale da 0% a 100% è stata utilizzata la funzione `map()` appartenente alla libreria base di Arduino, come mostrato in Figura 12.

```
const int DryValue = 579; //valore del sensore asciutto
const int WetValue = 277; //valore del sensore immerso nell'acqua

float soilMoistureAnalog = analogRead(A0);
int soilMoistureValue = map(soilMoistureAnalog, WetValue, DryValue, 100, 0);
```

Figura 12: Codice gestione e calibrazione sensore umidità del suolo

2.4. Kit irrigazione



Figura 13: Relè

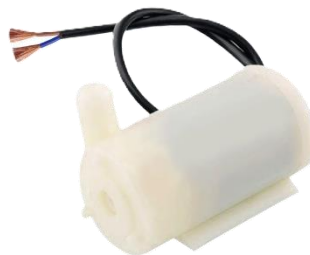


Figura 14: Pompa



Figura 15: Tubo in PVC

Per la gestione dell'irrigazione della pianta sono stati utilizzati un relè, una pompa d'acqua sommersibile e un tubo trasparente in PVC.

Il relè è un dispositivo elettronico che consente di pilotare circuiti ad alta tensione utilizzando un segnale di controllo a bassa tensione. Nel caso preso in considerazione permette ad Arduino attraverso il segnale digitale PWM del pin 9 di aprire o chiudere il circuito di alimentazione della pompa dell'acqua, ossia di disattivare o attivare, rispettivamente, l'impianto d'irrigazione. Un relè, infatti, funziona come un classico interruttore, ma è attivabile da un impulso elettrico, invece che manualmente da una persona.

Il relè preso in considerazione (Figura 13) viene alimentato a 3.3V tramite i pin VCC e GND, mentre il pin IN rappresenta l'ingresso per il segnale digitale di controllo. Sulla sinistra di questo modulo sono invece presenti tre connettori:

- **NO (Normally Open - Normalmente Aperto):** quando il relè non è attivato (o non riceve tensione), questo contatto è aperto, interrompendo il flusso di corrente attraverso il circuito collegato. Quando invece viene attivato, il contatto NO si chiude permettendo al carico collegato ad esso di ricevere alimentazione;
- **COM (Common - Comune):** questo è il punto di connessione comune per entrambi i contatti (NO e NC);
- **NC (Normally Closed - Normalmente Chiuso):** quando il relè non è attivato, questo contatto è chiuso, consentendo il flusso di corrente attraverso il circuito collegato. Se invece il relè viene attivato, il contatto NC si apre.

Ne derivano due modalità di utilizzo di un relè a seconda che si voglia un circuito aperto a relè disattivato (Modalità NO) o, al contrario, un circuito chiuso a relè sempre disattivato (Modalità NC). Per questo progetto è più adatta la prima modalità: infatti verranno collegati il positivo della pompa al connettore NO, il negativo al GND di Arduino e il connettore COM all'uscita di alimentazione 3.3V sempre di Arduino. In questo modo a relè disattivato la pompa non sarà in funzione, mentre inizierà a irrigare solo all'attivazione del relè.

Per gestire tutto ciò nel codice del microprocessore si è utilizzato un if statement per tenere l'umidità del suolo in un range compreso tra 40% e 70%, come mostrato di seguito.

```
int relayPumpPin = 9;

if(soilMoistureValue < 40) {
    Serial.println("Pompa acqua attivata");
    digitalWrite(relayPumpPin, LOW); //attivo pompa acqua
    delay(5000); //attendo 5 secondi

    Serial.println("Pompa acqua disattivata");
    digitalWrite(relayPumpPin, HIGH); //disattivo pompa acqua
}
```

Figura 16: Codice gestione irrigazione

La pompa d'acqua (Figura 14) può essere alimentata da 2.5V a 5V e ha una portata di 80-120 l/h. Funziona a immersione, preleva quindi l'acqua da un foro presente in uno dei suoi lati e la spinge attraverso il buco d'uscita nel tubo flessibile in PVC (Figura 15) destinato all'irrigazione della pianta.

2.5. Arduino IDE e librerie

L'Arduino IDE (Integrated Development Environment) è l'ambiente di sviluppo integrato ideato per programmare, realizzare e caricare codice C++ sulle schede dei microcontrollori Arduino. Si tratta di un software open-source che fornisce vari strumenti per rendere accessibile la creazione di progetti con Arduino a programmatori di qualsiasi livello di esperienza. Tra questi, infatti, mette a disposizione un editor di testo con alcune agevolazioni riguardanti la sintassi, un compilatore con il compito di tradurre il linguaggio C++ in linguaggio macchina eseguibile dai microcontrollori Arduino e un monitor seriale tramite il quale comunicare con le schede e visualizzarne i messaggi in uscita.

Un programma elaborato per Arduino viene più comunemente chiamato “sketch” ed è caratterizzato da due funzioni principali:

- **setup():** una funzione che viene eseguita una sola volta all'avvio e in cui vengono svolte le operazioni di inizializzazione della porta seriale, dei vari sensori, dei pin digitali o di qualsiasi operazione non si voglia svolgere più di una volta;
- **loop():** una funzione che, come si può intuire dal nome, viene eseguita ciclicamente e normalmente rappresenta il cuore del programma e ne definisce il comportamento principale. Qui il programmatore può decidere ogni quanto far ripetere questo loop semplicemente prolungandone la durata, ad esempio con la funzione **delay(n)** che fa attendere al microcontrollore un tempo di n millisecondi.

Nell'IDE sono incorporate già una vasta raccolta di librerie che aiutano la programmazione grazie a funzioni di uso comune come quelle dedicate alla comunicazione seriale o al controllo e alla lettura dei pin digitali e analogici. In diversi casi però sono necessarie funzioni più specifiche contenute in librerie esterne facilmente scaricabili dal Library Manager integrato nel software.

Per quanto riguarda la presente tesi, si è dovuto far uso delle seguenti librerie:

- **<WiFiNINA.h>** [9]: per gestire l'uso del modulo NINA-W102 e quindi la connessione a una rete internet tramite WiFi;
- **<SparkFunWeatherShield.h>** [10]: per l'acquisizione dei valori di umidità dell'aria, temperatura e luminosità dalla Weather Shield. Questa libreria in particolare è stata creata appositamente per essere utilizzata su un Arduino Uno WiFi Rev2 e non è presente nel Library Manager dell'IDE, ma è necessario scaricarla dal web e, successivamente, includerla nell'ambiente di sviluppo attraverso i passaggi: *Sketch > Include Library > Add .ZIP Library...*;
- **<ArduinoHttpClient.h>** [11]: per effettuare le richieste POST del protocollo http necessarie per pubblicare i dati raccolti su Measurify;
- **<ArduinoJson.h>** [12]: per fare il parsing del formato JSON del body delle richieste http;
- **<WiFiUdp.h>** [13]: facente parte della libreria WiFi di Arduino e necessaria per la gestione della connessione UDP (User Datagram Protocol) tramite cui nel presente progetto viene ricavata data e ora corrente provenienti da un server NTP (Network Time Protocol) [14] in rete;
- **<TimeLib.h>** [15]: per la gestione e la manipolazione del formato di data e ora desiderato.

2.6. Codice Arduino

La programmazione del microcontrollore usato è stata fatta, come già anticipato, con l'ambiente di sviluppo Arduino IDE utilizzando il linguaggio C++.

Il codice comincia includendo tutte le librerie utilizzate e un file chiamato "arduino_secrets.h" in cui sono salvati SSID e password della rete WiFi alla quale ci si vuole connettere e il token necessario all'autenticazione delle request POST a Measurify. Vengono poi definite alcune variabili globali che saranno utilizzate nel resto del programma. Tra le più significative ci sono l'indirizzo e la porta del server di Measurify e di quello NTP, le costanti per la calibrazione del sensore d'umidità, già mostrate in Figura 12, e l'array in cui vengono salvati i valori raccolti. In questa porzione di codice vengono anche create le istanze delle classi WiFiSSLClient, HttpClient, WiFiUDP con cui verranno gestite le richieste http; della classe DynamicJsonDocument con cui si crea un documento JSON dove scrivere il body delle suddette richieste; e di quella WeatherShield per poter interrogare i sensori della Shield.

```
char serverAddress[] = "students.measurify.org"; // indirizzo dell'API
int port = 443; // porta del server https
// define IPAddress object that will contain the NTP server IP address
// We will use an NTP server from https://tf.nist.gov/tf-cgi/servers.cgi
IPAddress timeServer(129, 6, 15, 28);
WiFiUDP wifi_UDP;
int localPort = 8888;
// array to hold incoming/outgoing NTP messages
// NTP time message is 48 bytes long
byte messageBuffer[48];

float arrayDate[4]; //array in cui salvare i dati raccolti dai sensori
WeatherShield weather;

WiFiSSLClient wifiClient;
HttpClient client = HttpClient(wifiClient, serverAddress, port);

const size_t capacity = JSON_OBJECT_SIZE(2) + JSON_ARRAY_SIZE(4); // Dimensione massima
del documento JSON
DynamicJsonDocument jsonDocument(capacity); // creo un oggetto JSON
```

Figura 17: Parte di codice iniziale

Il codice poi prosegue con la funzione di **setup()**, in cui si stabilisce una connessione con la rete WiFi, si inizializza la porta seriale a 9600 baud e tutti gli altri oggetti, si setta la data e l'ora corrente con la funzione getTime e si stabilisce che il pin digitale 9, che pilota il relè, deve funzionare come un output.

```
void setup() {
  Serial.begin(9600);
  pinMode(relayPumpPin, OUTPUT);
  wifi_UDP.begin(localPort); // start UDP

  weather.startPins(); // set on-board's pin mode
  weather.begin(); // enable I2C & sensors
```

```

// attempt to connect to WiFi network:
while (status != WL_CONNECTED) {
  Serial.println("Attempting to connect to Wifi..");
  // Connect to WPA/WPA2 network:
  status = WiFi.begin(ssid, pass);
}
// pass function getTime() to Time Library to update current time
setSyncProvider(getTime);
delay(10000); // wait 10 seconds for connection
}

```

Figura 18: Parte di codice del setup

Il cuore del programma è però rappresentato dalla funzione **loop()**, in cui vengono rilevati i dati dai sensori tramite le chiamate alle funzioni della libreria SparkFunWeatherShield e la lettura dell'ingresso analogico A0 e viene anche gestito il controllo dell'irrigazione con l'if statement già esposto in Figura 16. I valori vengono quindi stampati sul monitor seriale, per averne un riscontro immediato, e salvati nell'array inizializzato a inizio codice, che successivamente viene passato come argomento alla funzione `postTimesamples()`, definita in seguito, che si occuperà di elaborarlo e inviarlo a Measurify. Infine, viene atteso un tempo di dieci minuti prima che il ciclo della funzione `loop` ricominci, in modo da distanziare nel tempo una misurazione dalla successiva.

```

void loop() {
  float soilMoistureAnalog = 0;

  soilMoistureAnalog = analogRead(A0);
  arrayDate[0] = map(soilMoistureAnalog, WetValue, DryValue, 100, 0);

  //Contollo sensore di umidità
  arrayDate[1] = weather.readHumidity();

  //Contollo sensore di temperatura
  arrayDate[2] = weather.readTemperature();

  //Contollo sensore di luminosità con conversione da Volt a Lux
  arrayDate[3] = (weather.getLightLevel()*1000000) / (3650*0.5);

  postTimesamples(arrayDate);

  delay(600000); // attendo 10 minuti
}

```

Figura 19: Parte di codice del loop

All'interno della funzione **postTimesamples()** viene creata la struttura della request POST per l'invio dei valori della misurazione effettuata. Inizialmente viene scritto un documento JSON, destinato a essere il body della richiesta http, che conterrà un array "values" con gli elementi di `arrayDate` e il campo "timestamp" a cui è assegnata la data e l'ora corrente, ricavate con la funzione `printCurrent()`. Vengono in seguito aggiunti gli header, in particolare quello "Authorization" contenente il token per l'autenticazione per l'accesso a Measurify e infine, dopo aver finito la costruzione della richiesta, si attende la risposta da parte del server per stamparla sul Serial Monitor.

```

void postTimesamples(float date[]) {
    String cDate = printCurrentDate(); // save the current date and time
    //Scrivo in formato JSON il body della request
    jsonDocument["timestamp"] = cDate ;
    JSONArray jsonArray = jsonDocument.createNestedArray("values");
    for (int i = 0; i < 4; i++) {
        jsonArray.add(date[i]);
    }
    // Serializzo l'oggetto JSON in una stringa
    String jsonString;
    serializeJson(jsonDocument, jsonString);
    client.beginRequest();
    client.post("/v1/measurements/measurement-vine1/timeserie");

    //aggiungo gli header necessari alla request
    client.setHeader("Content-Type", "application/json");
    client.setHeader("Content-Length", jsonString.length());
    client.setHeader("Authorization", String(token));
    //aggiungo il body alla request
    client.beginBody();
    client.print(jsonString);
    client.endRequest();

    // Attendo la risposta
    int statusCode = client.responseStatusCode();
    String response = client.responseBody();
}

```

Figura 20: Parte di codice della funzione `postTimesamples()`

La funzione **`printCurrentDate()`** si occupa di creare una stringa nel formato YYYY-MM-DDThh:mm:ss, grazie anche all'utilizzo di un'altra funzione: **`addZero()`**, con lo scopo di rappresentare i numeri di una singola cifra preceduti da uno zero ("0n" invece che "n").

```

String printCurrentDate() {
    Serial.print("Current date: ");
    String currentDate = String(year()) + "-" + String(addZero(month())) + "-" +
        String(addZero(day())) + "T" + String(addZero(hour())) + ":" +
        String(addZero(minute())) + ":" + String(addZero(second()));
    return currentDate;
}

String addZero(int digits) {
    if (digits < 10)
        return "0" + String(digits);
    return String(digits);
}

```

Figura 21: Parte delle funzioni `printCurrentDate` e `addZero`

Per avere data e ora costantemente aggiornate, Arduino deve appoggiarsi a un server NTP a cui fare richieste tramite il protocollo UDP per ricevere in risposta i secondi trascorsi dalla data simbolica del 1° gennaio 1900. La richiesta viene creata ed effettuata con la funzione **sendRequest()** che riceve come argomento l'indirizzo IP del server. Tale funzione viene chiamata all'interno di un'ulteriore funzione di tipo `time_t` chiamata **getTime()** e ha il compito di elaborare il numero di secondi ricevuti in modo da ricavare la data e l'ora corrente, tenendo anche conto del fuso orario. La costruzione di queste specifiche funzioni è stata effettuata seguendo le informazioni ricavate da una guida online [16].

2.7. Measurify

Per avere accesso ai valori misurati dal sistema embedded, situato in prossimità della pianta, da un'applicazione su un qualsiasi smartphone non necessariamente nelle sue vicinanze è fondamentale l'utilizzo del cloud, dove memorizzare le informazioni ottenute. Perciò ci si è affidati al API framework Measurify [2].

Measurify, sviluppato da Elios Lab dell'Università di Genova, è un API (Application Programming Interface) framework basata sul cloud per la raccolta e la gestione di dati provenienti da varie sorgenti compresi sensori appartenenti a ecosistemi IoT. La sua struttura è fondata sulla base di alcuni concetti chiave:

- **THING**: si tratta di una "cosa" (una persona, un ambiente, un oggetto, ecc.) per la quale si sta facendo una misurazione. Nel presente caso è la pianta l'ambiente circostante;
- **FEATURE**: ciò che si sta misurando. Per questo progetto si tratta di umidità dell'aria, del suolo, temperatura e luminosità;
- **DEVICE**: un dispositivo (hardware e/o software) in grado di misurare una certa feature su una determinata thing. Il sistema embedded composto da Arduino e i vari sensori;
- **MEASUREMENT**: si tratta di una misura effettuata da un device per una certa feature su una determinata thing. Ad esempio, il fatto che il sensore di temperatura rilevi uno specifico valore in un preciso istante di tempo.

Per poter far uso di Measurify ai fini dello sviluppo del presente progetto è necessario impostare l'API per ricevere i dati con le caratteristiche desiderate, creando una nuova measurement attraverso una request POST, da effettuare una singola volta. Quest'ultima dovrà essere rivolta all'indirizzo <https://students.measurify.org/v1/measurements> e dovrà contenere un campo header chiamato Authorization a cui assegnare un token, una stringa alfanumerica necessaria per accedere al servizio senza effettuare un login con user e password. Il body della richiesta, espresso in JSON, sarà:

```
{
  "_id": "measurement-vine1",
  "thing": "vine1",
  "feature": "monitoring",
  "device": "edge-vine1",
  "startDate": "2024-01-05T00:00:00.000Z",
  "endDate": "2024-01-05T00:00:00.000Z"
}
```

Successivamente ogni singola misurazione effettuata da Arduino verrà da esso inviata tramite una POST a <https://students.measurify.org/v1/measurements/measurement-vine1/timeserie>, sempre con lo stesso campo header Authorization, ma con un body del seguente formato, in cui sostituire i vari campi con i dati correnti:

```
{
  "timestamp": "YYYY-MM-DDThh:mm:ss",
  "values": [{umidità suolo}, {umidità aria}, {temperatura}, {luminosità}]
}
```

2.8. Flutter

Per lo sviluppo dell'app smartphone è stato usato Flutter [3], un framework open source di Google per la creazione di applicazioni multiplatforma a partire da un unico codice base. Il linguaggio di programmazione orientato agli oggetti utilizzato si chiama Dart ed è stato introdotto sempre da Google. Flutter nasce appunto con l'intento di semplificare agli sviluppatori il compito di realizzare e mantenere nel tempo applicazioni native per diversi sistemi operativi e piattaforme, quali Android, iOS o Web. I componenti su cui si basa l'architettura di Flutter, oltre al linguaggio Dart, sono:

- **Flutter Engine:** scritto principalmente in C++, rappresenta il motore di Flutter. Fornisce supporto per il rendering a basso livello, si interfaccia con SDK specifiche come quelle fornite da Android o iOS e implementa le librerie core;
- **Foundation Library:** scritta in Dart, fornisce le classi e le funzioni base;
- **Widget:** un widget rappresenta una descrizione immutabile dell'interfaccia utente, ed elementi come grafici, testo e forme sono realizzati proprio mediante l'uso di questi widget. Per progettare un'interfaccia utente (UI) in Flutter si deve creare, assemblare e combinare diversi widget, con la possibilità di esplorare una vasta gamma disponibile online nella documentazione;
- **Design-specific widget:** Flutter contiene due set di widget ciascuno ispirato dai principali linguaggi di progettazione. I widget in stile Material Design implementano il design di Google con lo stesso nome, mentre i widget di Cupertino imitano il design iOS di Apple.

Il codice in linguaggio Dart con cui è stata sviluppata l'applicazione per il presente progetto è contenuto in un file chiamato *main.dart* e si può suddividere in tre parti logiche fondamentali. La prima è quella che si occupa del layout, di come esporre un titolo, uno sfondo e un menù di selezione per le due schermate dell'app. Essa gestisce inoltre la logica che permette all'utente di spostarsi da una pagina all'altra, selezionando la corrispondente icona sul fondo dello schermo. La seconda parte è quella invece che crea ed effettua le richieste GET http verso il server di Measurify, avendo sempre cura di usare il token di autorizzazione per potervi accedere. Una volta ricevuta risposta, ne viene elaborato il body, che è in formato JSON, per ricavare i valori delle varie misurazioni e delle relative timestamps che saranno quindi salvati per essere disponibili al resto del codice. La terza e ultima parte comprende la logica che si occupa di esporre graficamente queste informazioni, sia per quanto riguarda la Home sia per la schermata Chart.

Per la realizzazione delle ultime due parti è stato necessario utilizzare delle librerie esterne. Esse, oltre a essere state importate all'interno del file *main.dart*, sono state inserite nel file *pubspec.yaml* tra le altre dipendenze del progetto. In particolare, ci si è appoggiati alla libreria **package:http/http.dart** per la gestione delle richieste http e a quella **'package:syncfusion_flutter_charts/charts.dart'** per la costruzione dei grafici della schermata Chart.

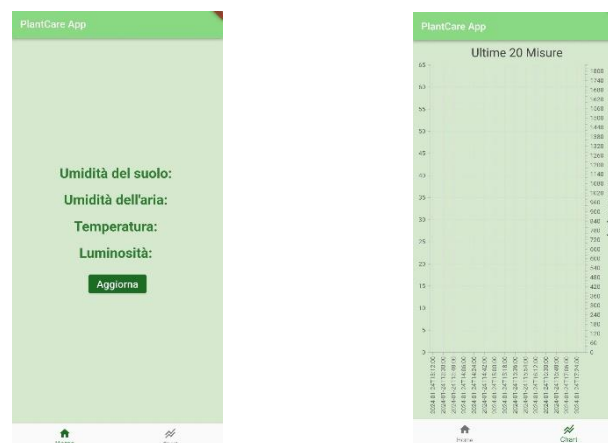


Figura 22: Schermate app vuote

3. Sperimentazione e Risultati

Come già accennato nel capitolo precedente, il sistema embedded è composto da una scheda Arduino UNO WiFi Rev2 su cui è stata collegata una Sparkfun Weather Shield [DEV-13956] tramite la saldatura, negli appositi fori, di connettori pin maschio del tipo in Figura 23. In questo modo però tutti i connettori femmina della board Arduino sono occupati e non più direttamente accessibili. Per collegare al sistema i cavi esterni necessari all'alimentazione e alla gestione del sensore capacitivo di umidità del suolo e del kit d'irrigazione, si è dovuto quindi saldare una delle estremità dei fili nella parte superiore della Shield, in corrispondenza dei rispettivi pin.

Tutte le connessioni effettuate sono state rappresentate nello schema di Figura 24.

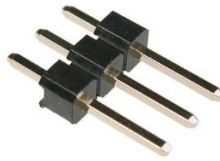


Figura 23: Connettori pin maschio

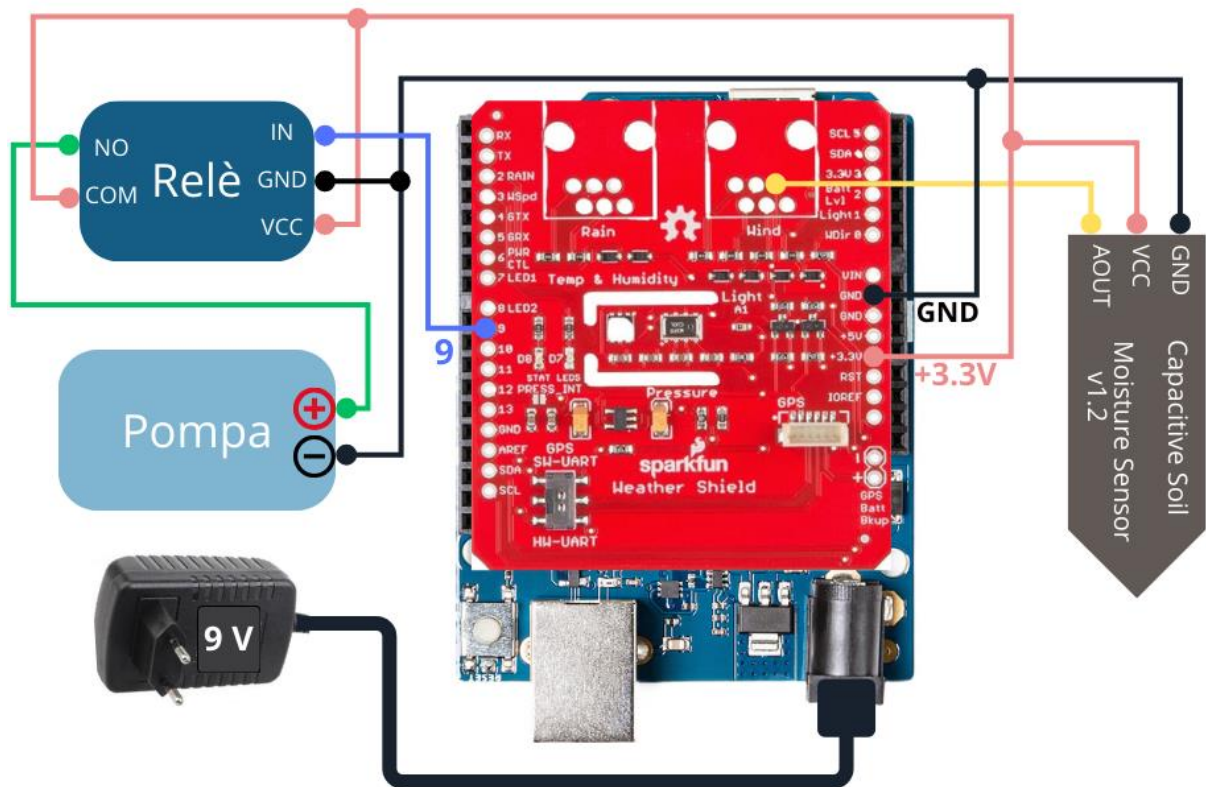


Figura 24: Collegamenti hardware del sistema

Una volta effettuate tutte le connessioni hardware, è stato caricato il codice descritto nel capitolo precedente sul microcontrollore Arduino, collegandolo con un cavo USB al PC. L'intero sistema è stato quindi posizionato vicino a una pianta, il sensore di umidità del suolo inserito nel terreno all'interno del vaso e la pompa immersa in un serbatoio con acqua. Di seguito due foto che mostrano l'effettiva realizzazione di quanto illustrato.

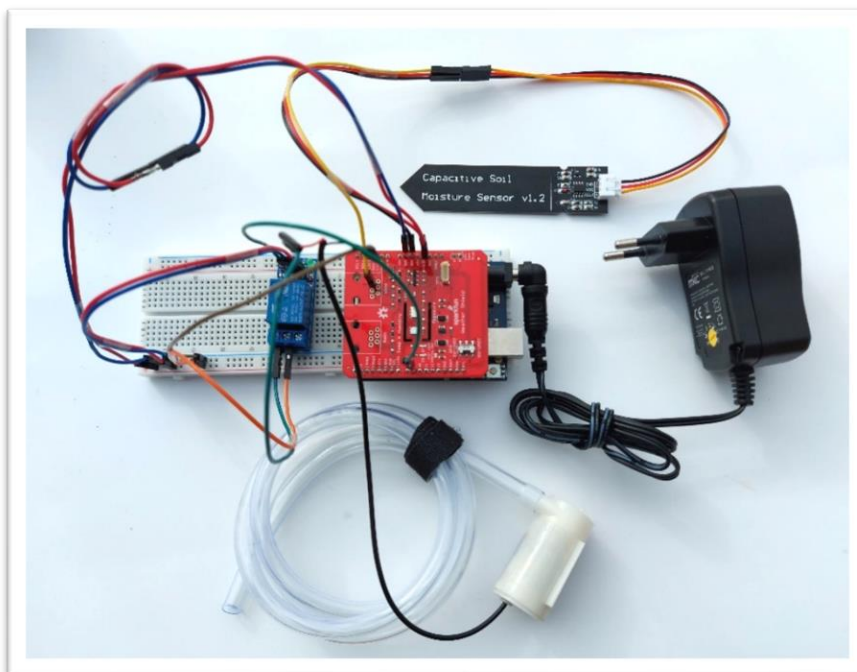
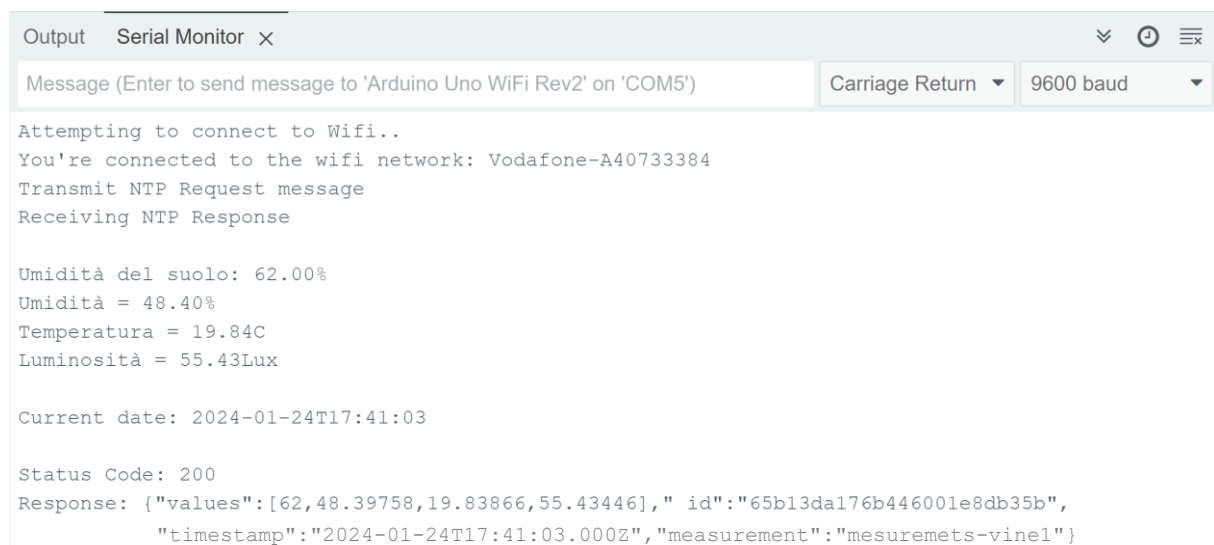


Figura 25: Foto reali del sistema

È stato quindi possibile effettuare dei test sul funzionamento di quanto realizzato che hanno permesso la modifica e il miglioramento delle parti di codice che non operavano correttamente.

Il primo riscontro del fatto che le misurazioni, la connessione alla rete WiFi e le interazioni con i server sono andate a buon fine si ha tramite il Serial Monitor dell'IDE Arduino (Figura 26), che stampa i valori ricavati e le risposte ricevute. In particolare, dopo aver atteso il tempo necessario alla connessione WiFi, viene confermato il collegamento col server NTP e visualizzati i valori letti dai sensori nel preciso istante indicato nella riga sottostante ad essi. Infine, per verificare l'effettivo invio dei dati a Measurify, viene mostrato lo status code della request http che nel caso abbia avuto un esito positivo sarà 200 e verrà anche stampata la risposta.



```
Attempting to connect to Wifi..
You're connected to the wifi network: Vodafone-A40733384
Transmit NTP Request message
Receiving NTP Response

Umidità del suolo: 62.00%
Umidità = 48.40%
Temperatura = 19.84C
Luminosità = 55.43Lux

Current date: 2024-01-24T17:41:03

Status Code: 200
Response: {"values": [62, 48.39758, 19.83866, 55.43446], "id": "65b13da176b446001e8db35b",
"timestamp": "2024-01-24T17:41:03.000Z", "measurement": "mesurements-vine1"}
```

Figura 26: Serial Monitor

Come ultima parte del progetto, di cui tratta la presente tesi, è stata realizzata un'applicazione mobile, compatibile sia con sistemi iOS che Android, per rendere i dati facilmente accessibili all'utente. Per il suo sviluppo è stato utilizzato il già citato framework Flutter e Visual Studio Code, l'editor di codice sorgente di Microsoft.

Come già descritto nell'introduzione, la struttura finale dell'app si compone di due pagine, che si aggiornano autonomamente a ogni loro apertura:

- una **Home**, dove vengono mostrati i valori dell'ultima misurazione effettuata dal sistema embedded controllato da Arduino e un tasto "Aggiorna" che verifica se ne sia stata aggiunta una più recente su Measurify e, eventualmente, la sostituisce a quella visualizzata;
- una **Chart** che invece inserisce gli ultimi 20 valori in un grafico a linee formato da un asse orizzontale relativo al tempo e due assi verticali, uno dedicato ai Lux per la luminosità e l'altro ai restanti dati. In questa schermata l'utente può facilmente conoscere i valori precisi di ogni misurazione facendo click sul punto a cui è interessato e può inoltre decidere se visualizzare le quattro linee, corrispondenti a umidità del suolo, umidità dell'aria, temperatura e luminosità, o solo alcune di esse con un conseguente adattamento degli assi.

Durante la sua creazione sono stati svolti diversi test per ogni passaggio che hanno permesso di trovare il giusto layout e soprattutto di verificare il corretto inserimento dei dati prelevati da Measurify, con le

apposite richieste GET http, all'interno del grafico rappresentato. Queste prove sono state eseguite grazie alle funzioni di Debug e di Hot Reload di Flutter e Visual Studio Code, che hanno consentito di visualizzare in tempo reale le modifiche apportate al codice direttamente su un telefono collegato via cavo al computer.



Figura 27: Schermate applicazione

4. Contributo personale e considerazioni conclusive

Il progetto descritto nella presente tesi nasce dall'idea di monitorare e gestire le condizioni ambientali del luogo in cui una pianta è situata, permettendo l'intervento dell'utente qualora fosse necessario per mantenerne il benessere e la sopravvivenza. Il sistema embedded creato è infatti in grado di controllare il livello di umidità del suolo, attraverso il kit d'irrigazione, in modo che la pianta abbia sempre a disposizione una quantità d'acqua ottimale. Oltre a questo, però, è stata anche sviluppata un'applicazione per smartphone che garantisce a chi la utilizza di essere costantemente aggiornato sui valori misurati dal sistema e studiarne l'andamento grazie al grafico che rappresenta quelle effettuate in precedente. In questo modo l'utente può decidere se fare qualcosa per migliorare le condizioni di vita e di benessere e capire se sta agendo correttamente verificando la variazione dei valori visualizzati sull'applicazione.

Il passaggio dei dati dal sistema controllato da Arduino all'applicazione è possibile grazie a Measurify, un API Framework basato sul cloud che li elabora e li memorizza e il cui utilizzo mi è stato fornito da Elios Lab.

Il mio contributo personale alla realizzazione della presente tesi è stato quello di interfacciare correttamente i vari componenti hardware che mi sono stati assegnati, programmare il microcontrollore Arduino per raccogliere e inviare i dati e infine sviluppare un'applicazione per visualizzarli.

Le possibili applicazioni di questo progetto sono attualmente ristrette a un uso indoor, dal momento che il sistema necessita di una presa di corrente e di una connessione WiFi, ma soprattutto poiché non è resistente né alle intemperie né alla sporcizia. Un passo successivo potrebbe quindi essere quello di costruire un case impermeabile per l'hardware e alimentare Arduino con delle batterie per permetterne l'utilizzo all'aperto, modifiche volte a un possibile impiego da parte di un'azienda agricola che possa trarne beneficio sotto svariati punti di vista.

Ulteriori lavori futuri potrebbero riguardare la creazione di un'unica board custom in cui integrare tutti i componenti hardware utilizzati e le rispettive connessioni. Oppure implementare nuove funzionalità per l'applicazione smartphone, per esempio dando la possibilità all'utente di selezionare un preciso arco temporale di cui si vogliano visualizzare le misurazioni all'interno del grafico, o aggiungere una schermata per la programmazione e la gestione manuale dell'irrigazione.

5. Riferimenti bibliografici

- [1] IoT: <https://www.saep-ict.it/magazine/internet-delle-cose-iot-significato-esempi-applicazioni/>
- [2] Measurify: <https://measurify.org/>
- [3] Flutter: <https://flutter.dev/>
- [4] Arduino UNO WiFi Rev2: <https://docs.arduino.cc/hardware/uno-wifi-rev2/>
- [5] SparkFun Weather Shield: <https://www.sparkfun.com/products/13956>
- [6] ALS-PT19: https://cdn.sparkfun.com/assets/b/e/c/3/d/ALS-PT19_DS.pdf
- [7] Si7021:
https://cdn.sparkfun.com/datasheets/Sensors/Weather/Si7021.pdf?_gl=1*w2v74t*_ga*MjM5NTM2Mjl4LjE3MDI0NjE1MDE.*_ga_T369JS7J9N*MTcwNjE5NjUxNy4yMi4xLjE3MDYxOTY3NTEuNjAuMC4w
- [8] Capacitive soil moisture sensor v1.2:
<https://how2electronics.com/interface-capacitive-soil-moisture-sensor-arduino/>
- [9] <https://www.arduino.cc/reference/en/libraries/wifinina/>
- [10] <https://create.arduino.cc/editor/rogermiranda1000/810942a0-9637-46ac-ae9f-feedf00e11b4/preview>
- [11] <https://www.arduino.cc/reference/en/libraries/httpclient/>
- [12] <https://www.arduino.cc/reference/en/libraries/arduinojson/>
- [13] <https://www.arduino.cc/reference/en/libraries/wifi/wifiudp>
- [14] <https://tf.nist.gov/tf-cgi/servers.cgi>
- [15] <https://playground.arduino.cc/Code/Time/>
- [16] <https://www.circuitbasics.com/using-an-arduino-ethernet-shield-for-timekeeping/>

6. Ringraziamenti

Un ringraziamento al professore Riccardo Berta e al dottore Matteo Fresta che mi hanno seguito e con cui mi sono potuto confrontare durante lo svolgimento della tesi, e a Flavio Ansovini che mi ha aiutato a effettuare le modifiche sull'hardware necessarie al sistema.

Ringrazio inoltre la mia famiglia e i miei amici che mi hanno supportato e hanno creduto in me lungo tutto il mio percorso di studio.