



Prototipizzazione con Arduino 2023

Irrigatore Automatico

Gaiga Alex VR471343

Zerman Nicolò VR471404

Indice

1	Introduzione	3
2	Tabella dei Componenti	4
3	Sviluppo	5
4	Descrizione Codice e Librerie	6
5	Schema Collegamenti	13
6	Implementazione Fisica	14
7	Interfaccia Web	16
8	Riferimenti	17

1 Introduzione

Il progetto ha lo scopo di gestire l’irrigazione di una pianta in modo automatico, evitando la disidratazione del terreno.

Per questo, sono state utilizzate due schede comunicanti: un Arduino UNO e un ESP32 Thing Plus.

L’Arduino è responsabile di determinare se la pianta ha bisogno di essere irrigata, utilizzando vari sensori. Grazie ad opportuni controlli, non si irriga eccessivamente la pianta, evitando di danneggiarla.

Attraverso un sensore di livello dell’acqua viene monitorata la quantità di acqua disponibile. Nel caso in cui il livello dell’acqua fosse basso, viene segnalato il problema mediante l’accensione di un LED.

Un display OLED 128x64, collegato all’Arduino, mostra l’ora e la data corrente (ricevuti dall’ESP32), la temperatura e l’umidità ambientale.

Infine, ogni 16 secondi, l’Arduino invia i valori dei vari sensori all’ESP32, che saranno memorizzati in cloud (ThingSpeak).

L’ESP32 ha il compito di inviare i dati all’Arduino (ora/data e valori delle threshold) quando richiesto e quando necessario, oltre a memorizzare i valori dei vari sensori nel cloud per la visualizzazione tramite un’interfaccia web.

Infatti, è stata realizzata un’interfaccia web, utilizzando Dash e Plotly, con lo scopo di:

- recuperare i dati dal cloud e mostrarli attraverso opportuni grafici;
- permettere l’aggiornamento delle soglie di umidità e illuminazione.

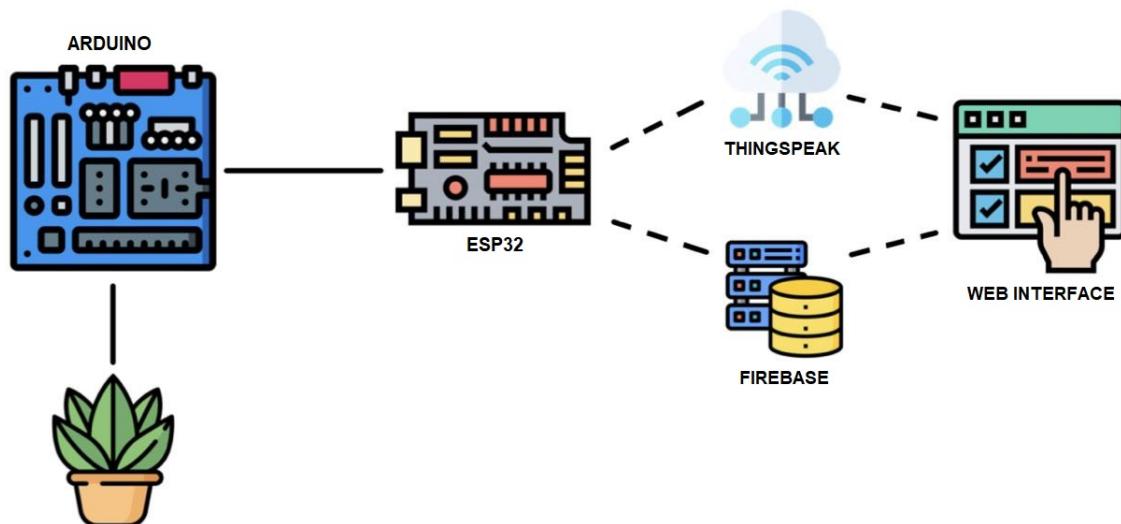


Immagine 1: Schema del progetto

2 Tabella dei Componenti

Componenti	
Lato Arduino	Lato Esp32
x1 Arduino Uno x1 Display OLED x3 LED x3 Resistenze (330 ohm) x1 Sensore di Luminosità x1 Sensore di Temperatura e Umidità Ambientale x1 Sensore di Umidità del Terreno x1 Sensore di Livello dell'Acqua x1 Sensore di Movimento PIR x1 Relay	Esp32 Thing Plus

Infine è presente una pompetta (con alimentatore) che ha lo scopo di irrigare la pianta. Il come, ovviamente, è gestito dall'Arduino.

3 Sviluppo

Durante lo sviluppo del progetto, ci siamo trovati di fronte a diverse decisioni da prendere, le quali ci hanno portato a soluzioni diverse tra loro.

Prima di tutto, abbiamo eseguito dei test sui vari sensori utilizzati.

Tra questi, il PIR Motion Sensor ci ha messo più in difficoltà a causa della sua logica di funzionamento, inizialmente poco chiara. Per quanto riguarda gli altri sensori, invece, abbiamo avuto maggiore facilità.

Poi, ci siamo focalizzati sul come comunicare cosa sta effettivamente accadendo in un preciso momento. Abbiamo utilizzato tre LED (rosso, giallo e verde) per comunicare lo stato dell'irrigazione:

- se tutti i LED sono accesi (solo in fase iniziale), significa che l'Arduino sta aspettando delle informazioni dall'ESP32;
- se il LED rosso è acceso, significa che il serbatoio d'acqua è quasi vuoto;
- se il LED verde è acceso, significa che in quel momento la pianta sta venendo irrigata (attivazione pompa).

Si noti che il LED giallo e il LED verde non possono mai essere accesi contemporaneamente (ad eccezione della fase iniziale).

Successivamente, abbiamo scelto la comunicazione seriale come metodo di comunicazione tra le due schede, poiché è risultata la soluzione più comoda e semplice. Questa ci ha permesso di scambiare tutti i dati necessari tra Arduino e ESP32 e viceversa.

Ci siamo poi concentrati sulla visualizzazione delle informazioni sul display OLED, decidendo quali dati mostrare. Grazie alle informazioni trovate su delle librerie GitHub (segnalate in Riferimenti), abbiamo affrontato questo passaggio senza troppi intoppi.

Infine, abbiamo realizzato un'interfaccia web utilizzando Dash e Plotly (script Python).

Inizialmente, l'interfaccia comunicava con l'Arduino tramite porta seriale, ma questa soluzione è stata scartata a causa delle difficoltà riscontrate nel far comunicare l'Arduino su più porte seriali.

Abbiamo quindi optato per un Realtime Database su Firebase, il quale si è rivelato molto più comodo.

Si noti che i dati presenti su Firebase sono controllati e inviati dall'ESP32 all'Arduino, poiché non è stato più possibile farlo dall'interfaccia.

4 Descrizione Codice e Librerie

Codice Arduino:

```
void setup() {  
    pinMode(redLED,OUTPUT);  
    pinMode(yellowLED,OUTPUT);  
    pinMode(greenLED,OUTPUT);  
    pinMode(PIRMotion,INPUT_PULLUP);  
    pinMode(relay,OUTPUT);  
  
    pinMode(soilMoisture,INPUT);  
    pinMode(lightSensor,INPUT);  
    pinMode(waterSensor,INPUT);  
  
    Serial.begin(9600);  
    espSerial.begin(9600);  
  
    digitalWrite(redLED,HIGH);  
    digitalWrite(yellowLED,HIGH);  
    digitalWrite(greenLED,HIGH);  
  
    u8g.setFont(u8g_font_unifont);  
  
    espSerial.print("D"); //D --> Request Data  
    while(!espSerial.available()){}  
    String reader = espSerial.readString();  
    receiveDataESP(reader);  
  
    espSerial.print("H"); //H --> Request Hour and Date  
    while(!espSerial.available()){}  
    reader = espSerial.readString();  
    receiveHoursESP(reader);  
  
    Timer1.initialize(8000000);  
    Timer1.attachInterrupt(sendDataESP);  
    attachInterrupt(digitalPinToInterruption(PIRMotion),showDisplay,CHANGE);  
}
```

Immagine 2: Funzione Setup

All'interno della funzione di Setup andiamo ad inizializzare:

- i vari pin dei sensori utilizzati nel progetto;
- la porta seriale;
- un interrupt (eseguito ogni 16 secondi) a cui sarà associato un timer.

Troviamo anche due 'attese'.

Una che ci permetterà di ottenere, attraverso la comunicazione con l'ESP, i valori delle threshold che dovremo utilizzare.

La seconda è per ottenere orario e data correnti, nel caso venisse richiesta la visione del display OLED.

```

void loop() {
    if(espSerial.available()){ //if the serial is available --> it has to read data
        String reader = espSerial.readString();
        if(reader[0]=='D'){
            receiveDataESP(reader);
        }
        else if(reader[0]=='H'){
            receiveHoursESP(reader);
        }
    }

    if(analogRead(waterSensor)<50){ //if the value of the sensor < 50 (by default), it's gonna turn on the red LED
        digitalWrite(redLED,HIGH);
    } else {
        digitalWrite(redLED,LOW);
    }

    if(analogRead(soilMoisture)<umidityThreshold){ //if the value of the sensor < of the threshold, it's gonna water the plant
        digitalWrite(greenLED,HIGH);
        digitalWrite(yellowLED,LOW);
        digitalWrite(relay, HIGH); //Active relay for 0.8 seconds
        delay(800);
        digitalWrite(relay, LOW); //Disable relay
        delay(5000);
    } else { //else it does nothing
        digitalWrite(yellowLED,HIGH);
        digitalWrite(greenLED,LOW);
    }
    delay(1000);
}

```

Immagine 3: Funzione Loop

Possiamo dire che il loop esegue tre compiti importanti.

Inizialmente controlla se l'ESP ha inviato dei dati e, nel caso essi siano presenti, andrà ad aggiornare i rispettivi dati all'interno del codice.

Poi, controlla se il valore dato dal sensore del livello dell'acqua è sotto una determinata soglia. In caso positivo accenderà il LED rosso, altrimenti rimarrà spento.

Infine controlla se il valore dell'umidità del terreno è minore rispetto alla threshold dell'umidità ricevuta dall'ESP (e modificabile attraverso l'interfaccia). Nel caso fosse minore andremo ad irrigare la pianta.

```

void receiveDataESP(String reader){
    char* readerC = (char*) reader.c_str();

    char* token;
    token = strtok(readerC,";");
    token=strtok(NULL,""); //Avoid first token

    while(token!=NULL){
        if(checkUmidity){ //if checkUmidity=True, it saves the umidityThreshold
            umidityThreshold = atoi(token); //String to int
            checkUmidity=!checkUmidity;
        }
        else{ //else checkUmidity=False, it saves the lightingThreshold
            lightingThreshold = atoi(token); //String to int
            checkUmidity=!checkUmidity;
        }
        token=strtok(NULL,""); //Next token
    }
}

```

Immagine 4: Funzione receiveDataESP

Lo scopo di questa funzione è quello di salvare i dati riguardanti threshold di umidità ed illuminazione, che sono stati inviati dall'ESP.

```
void receiveHoursESP(String reader){
    char* readerC = (char*) reader.c_str();

    char* token;
    token = strtok(readerC,";");
    token = strtok(NULL,""); //Avoid first token

    int counter = 1;
    while(token!=NULL){
        switch(counter){ //it controls which data token is
            case 1:
                strcpy(day,token);
                break;
            case 2:
                strcpy(month,token);
                break;
            case 3:
                strcpy(year,token);
                break;
            case 4:
                strcpy(hour,token);
                break;
            case 5:
                strcpy(minutes,token);
                break;
        }
        counter++;
        token=strtok(NULL,""); //Next token
    }
}
```

Immagine 5: Funzione receiveHoursESP

Lo scopo di questa funzione è quello di salvare le informazioni su orario e data corrente, inviati sempre dall'ESP. In questo modo, se fosse richiesta la visione del display, sia l'orario che la data saranno sempre aggiornati.

```

void showDisplay(){
    String hours = (String) hour + ":" + minutes;
    String date = (String) day + "/" + month + "/" + year;

    sht.readSensor(); //read temperatura and humidity sensor

    char tempStr[5];
    float temp = sht.getTemperature_C();
    dtostrf(temp, 2, 1, tempStr);
    String tempF = (String)tempStr + " " + "°C";

    char umidStr[5];
    float umid = sht.getHumidity();
    dtostrf(umid, 2, 1, umidStr);
    String umidF = (String)umidStr + " %";

    if(digitalRead(PIRMotion)==HIGH){
        u8g.firstPage();
        do [
            u8g.setFont(u8g_font_7x13r);
            u8g.setFontPosBaseline();
            u8g.setPrintPos(20,27);
            u8g.print(hours);

            u8g.setFont(u8g_font_6x10r);
            u8g.setFontPosTop();
            u8g.setPrintPos(14,37);
            u8g.print(date);

            u8g.setFont(u8g_font_6x13r);
            u8g.setFontPosBaseline();
            u8g.setPrintPos(84,29);
            u8g.print(tempF);

            u8g.setFontPosTop();
            u8g.setPrintPos(84,35);
            u8g.print(umidF);

            u8g.drawFrame(0,0,128,64);
            u8g.drawLine(10,32,65,32);
            u8g.drawLine(75,7,75,57);

            u8g.setContrast(200);
        ] while (u8g.nextPage());
    } else {
        u8g.firstPage();
        do [
        ] while (u8g.nextPage());
    }
}

```

Immagine 6: Funzione showDisplay

La funzione ha lo scopo di impostare il display e mostrare attraverso esso: orario, data, temperatura e umidità ambientale.

```

void sendDataESP(){
    if(checkInterrupt){ //if i receive the second interrupt (because the first does nothing)
        sht.readSensor(); //read sensor
        espSerial.println((String)analogRead(A0) + ";" + sht.getHumidity() + ";" + sht.getTemperature_C() + ";" + analogRead(A1));
        checkInterrupt=!checkInterrupt;
    } else {
        checkInterrupt=!checkInterrupt;
    }
}

```

Immagine 7: funzione sendDataESP

La funzione invia i valori ricevuti dai sensori all'ESP (ogni 16 secondi).

Codice Esp32:

```
void setup() {
    Firebase.begin(FIREBASE_HOST,FIREBASE_AUTH);
    Firebase.reconnectWiFi(true);

    Serial.begin(115200);
    Serial1.begin(9600,SERIAL_8N1,RX,TX);

    WiFi.mode(WIFI_STA);
    ThingSpeak.begin(client);

    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
}
```

Immagine 8: Funzione Setup

Viene inizializzata la connessione al Database Firebase, la porta seriale, la connessione a ThingSpeak e viene configurato l'orario che sarà inviato, ad ogni cambiamento, all'Arduino.

```
void loop() {
    if(WiFi.status() != WL_CONNECTED){
        Serial.print("Connection attempt...");
        while(WiFi.status() != WL_CONNECTED){
            WiFi.begin(ssid,password);
            delay(5000);
        }
        Serial.println("\Connected.");
    }

    String reader;
    if(Serial1.available()){
        reader = Serial1.readString();
    }

    if(reader.equals("D")){
        sendDataArduino();
    }
    else if(reader.equals("H")){
        sendHoursArduino();
    }
    else {
        char* readerC = (char*) reader.c_str();

        int counter=1;
        char* token;
        token = strtok(readerC,";");

        while(token!=NULL){
            ThingSpeak.setField(counter,token);
            token=strtok(NULL,";");
            counter++;
        }

        ThingSpeak.writeFields(thingspeakChannel,APIKey);
    }
}
```

Immagine 9: Funzione Loop

Inizialmente viene controllato lo stato di connessione dell'ESP.

Nel caso l'Arduino abbia inviato qualcosa sulla seriale, si cerca di capire se è una richiesta o se sono i valori dei sensori, che verranno poi inseriti su ThingSpeak (ogni 16 secondi saranno presenti questi dati nella seriale).

```

if(Firebase.ready() && (millis()-getDataPrevMillis > 5000 || getDataPrevMillis == 0)){
    getDataPrevMillis = millis();
    int umidity;
    int lighting;
    if(Firebase.RTDB.getInt(&fData,"/illuminazione")){
        if(fData.dataType()=="int"){
            lighting = fData.intData();
        }
    }
    if(Firebase.RTDB.getInt(&fData,"/umidita")){
        if(fData.dataType()=="int"){
            umidity = fData.intData();
        }
    }
    if(umidity!=umidityBK || lighting!=lightingBK){
        umidityBK = umidity;
        lightingBK = lighting;
        sendDataArduino();
    }
}

if(millis()-getHourPrevMillis > 5000 || getHourPrevMillis==0){
    getHourPrevMillis = millis();

    if(getLocalTime(&timeinfo)){
        strftime(year,3,"%y",&timeinfo);

        strftime(month,3,"%m",&timeinfo);

        strftime(day,3,"%d",&timeinfo);

        char timeHour[3];
        strftime(timeHour,3,"%H",&timeinfo);

        char timeMinutes[3];
        strftime(timeMinutes,3,"%M",&timeinfo);

        if(strcmp(hour,timeHour)!=0 || strcmp(minutes,timeMinutes)!=0){
            strcpy(hour,timeHour);
            strcpy(minutes,timeMinutes);
            sendHoursArduino();
        }
    }
}

```

Immagine 10: Funzione Loop (cont.)

Ogni 5 secondi si controlla se i dati delle threshold su Firebase sono cambiati e/o l'orario si è aggiornato. In caso positivo, vengono aggiornati i valori che saranno poi inviati all'Arduino attraverso delle funzioni ausiliarie.

```

void sendDataArduino(){
    Serial1.println((String) "D;" + umidityBK + ";" + lightingBK);
}

void sendHoursArduino(){
    Serial1.print((String) "H;" + day + ";" + month + ";" + year + ";" + hour + ";" + minutes);
}

```

Immagine 11: Funzioni sendDataArduino - sendHourArduino

La prima avrà lo scopo di inviare i dati riguardo le threshold, mentre la seconda di inviare i dati riguardo orario e data.

Ovviamente queste informazioni sono inviate all'Arduino.

Librerie utilizzate:

- **TimerOne** : per il timer dell'Arduino (associato all'interrupt)
- **u8g** : per la gestione del display OLED
- **SoftwareSerial** : per aprire una comunicazione seriale con l'ESP su dei pin diversi dai pin 0 ed 1 dell'Arduino
- **Wifi** : per permettere la connessione dell'ESP ad internet
- **cactus io SHT15** : per poter permette la lettura del sensore di temperatura e umidità ambientale
- **ThingSpeak** : per permettere all'ESP la scrittura dei dati su ThingSpeak
- **FirebaseESP32** : per permette all'ESP la lettura dei dati presenti sul Realtime Database di Firebase

5 Schema Collegamenti

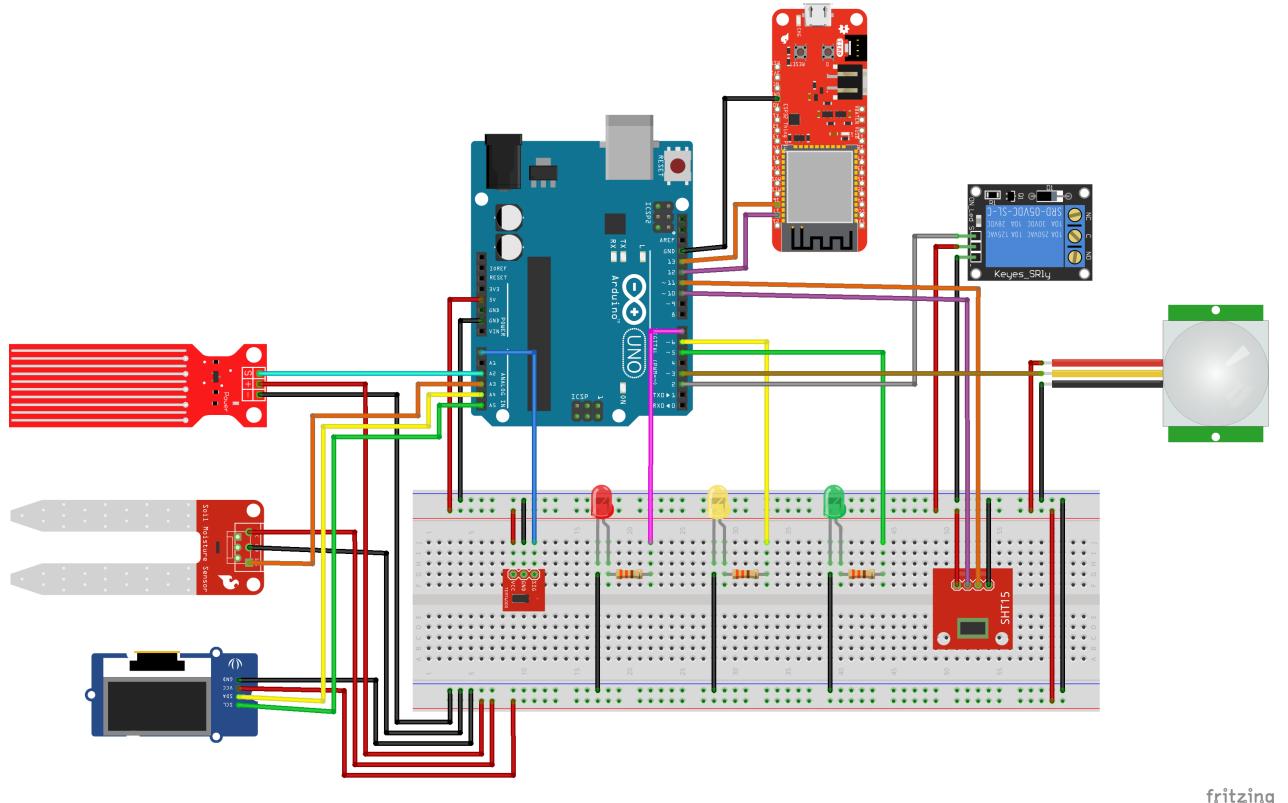


Immagine 12: Schema dei collegamenti tra i vari componenti utilizzati

6 Implementazione Fisica

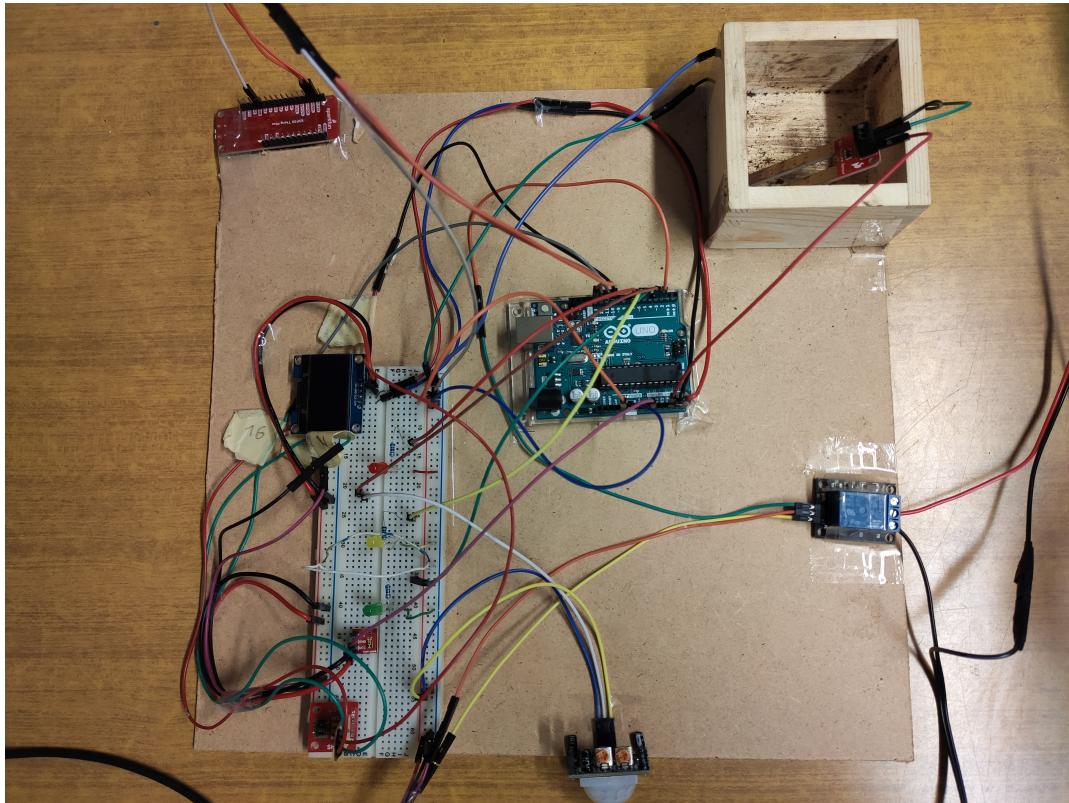


Immagine 13: Foto progetto

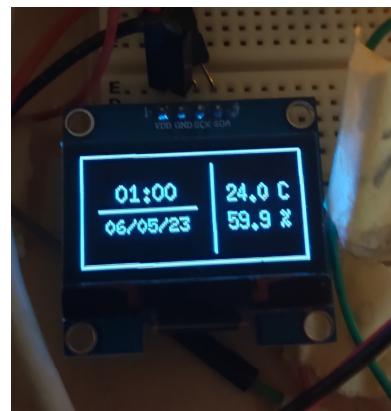


Immagine 14: Display che viene mostrato quando il PIR Motion Sensor nota un movimento nel suo range di azione

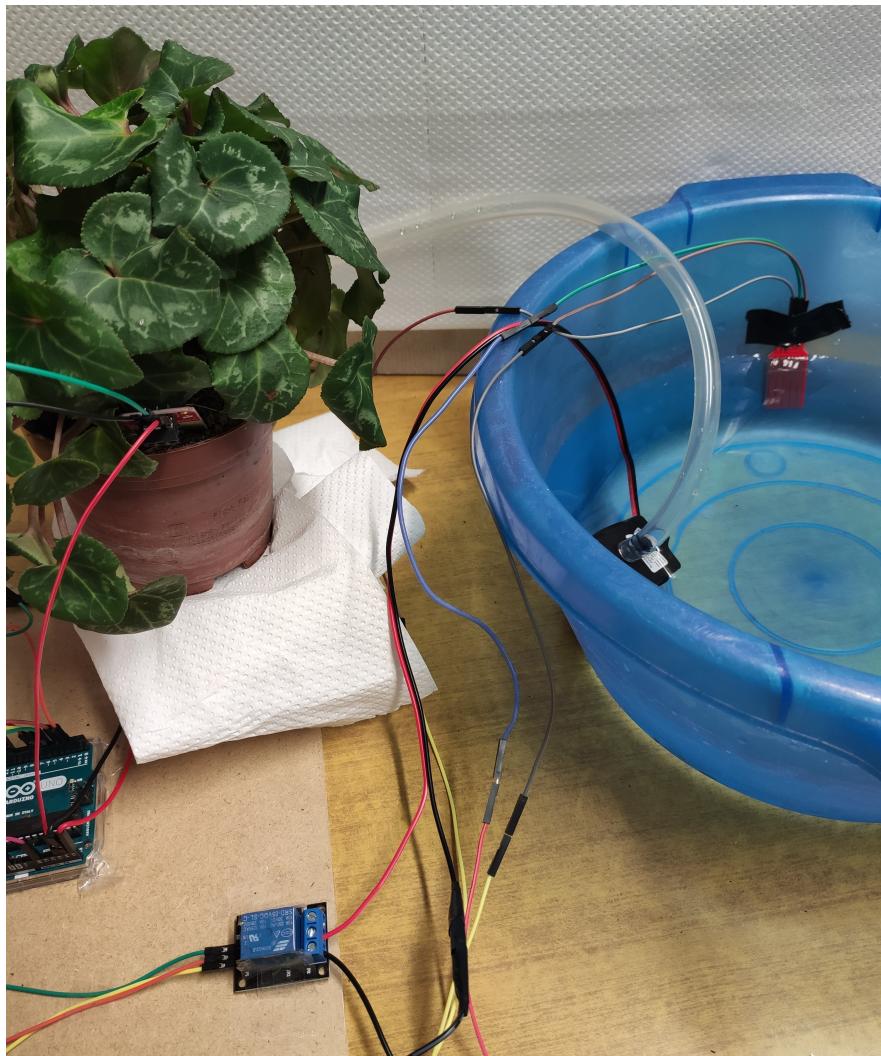


Immagine 15: Foto progetto (cont.)

7 Interfaccia Web

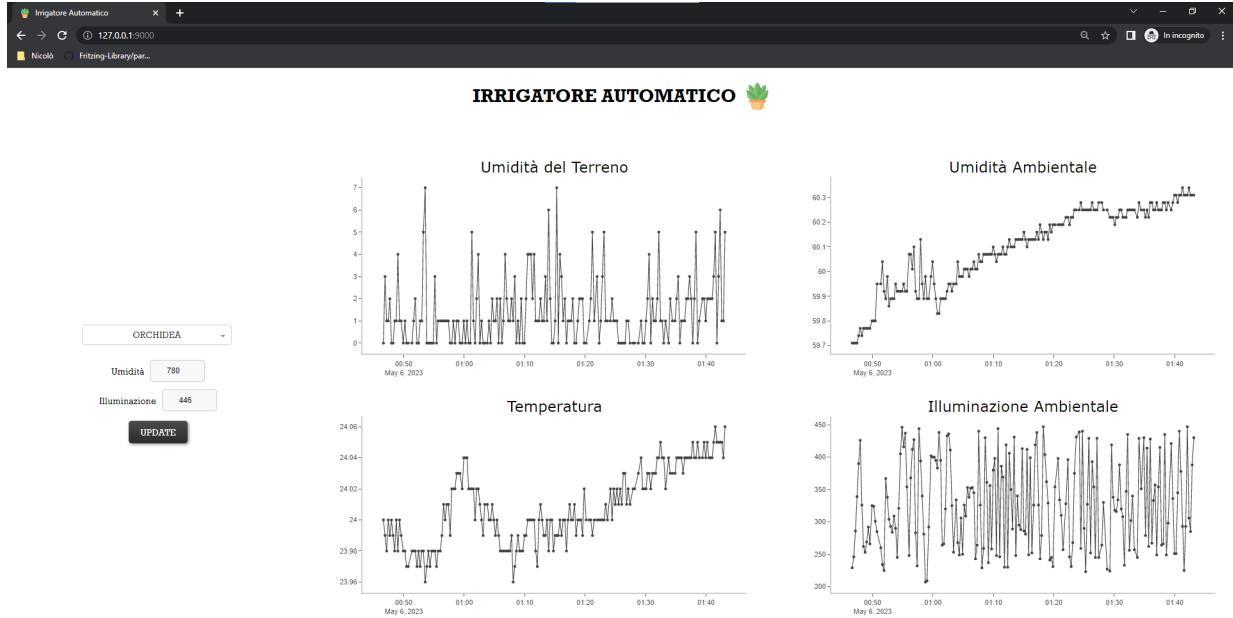


Immagine 16: Interfaccia Web

Permette di gestire le threshold di umidità ed illuminazione, che verranno aggiornate su Firebase nel caso vengano modificate. Inoltre è presente anche un file Excel, utilizzato proprio dall'interfaccia, in cui sono presenti i vari valori di soglia, aggiornati all'ultima modifica, delle varie piante.
Poi, sulla destra, possiamo vedere i grafici nel tempo dei vari sensori.

```
app.title= "Irrigatore Automatico"
app.layout = html.Div([
    html.Div([
        html.H1("IRRIGATORE AUTOMATICO", style={'text-align': 'center', 'font-family': 'Rockwell'}),
        html.Img(src="assets/fattoni.ico", style={'position: left; width: 3em; margin-left: 0.5em; height: 3em;'}),
        ].style='display: flex; align-items: center; justify-content: center;'),
    html.Div([
        children=[
            doc.Dropdown(plantsArray,defaultPlant,id="plants",clearable=False,style={'margin-top': '704px', 'width': '250px', 'padding-left': '24px', 'text-align': 'center', 'border-radius': '5px', 'font-family': 'Rockwell'}),
            html.Br(),
            html.Div([
                html.P("Umidità", style={'float: center', 'font-family': 'Rockwell'}),
                doc.Input(umidDefault,id="umid", type="number", min=0,max=1024,style={
                    'padding': '1px',
                    'margin-left': '28px',
                    'width': '70px',
                    'text-align': 'center',
                    'border': '1px solid black',
                    'border-radius': '5px',
                    'background-color': '#f0f0f0'
                })
            ],style='display: flex; align-items: center; justify-content: center;'),
            html.Div([
                html.P("Illuminazione", style={'float: center', 'font-family': 'Rockwell'}),
                doc.Input(luminDefault,id="lumin", type="number", min=0,max=1024,style={
                    'padding': '1px',
                    'margin-left': '28px',
                    'width': '70px',
                    'text-align': 'center',
                    'border': '1px solid black',
                    'border-radius': '5px',
                    'background-color': '#f0f0f0'
                })
            ],style='display: flex; align-items: center; justify-content: center;'),
            html.Button("Aggiorna", id="update", style="background-color: #4CAF50; color: white; border: none; padding: 5px; border-radius: 5px; font-size: 14px; margin-left: 10px;"),
            html.Div([
                doc.Graph(figureG1,conting="displayNoneBar", False,"staticPlot":True),
                html.Div([
                    doc.Graph(figureG1,conting="displayNoneBar", False,"staticPlot":True),
                    style={'flex: 1; width: 76px; height: 50px; display: inline-block'},
                ], style={'width: 100%; display: inline-block; height: 50px;'})
            ],style='display: flex; vertical-align: middle; text-align: center; width: 24%; height: 70px; display: inline-block')
        ],
        style='display: flex; align-items: center; justify-content: center; margin-top: 20px')
    ],
    style='display: flex; align-items: center; justify-content: center; margin-top: 20px')
], style='display: flex; align-items: center; justify-content: center; margin-top: 20px')
```

Immagine 17: Layout

Ecco il layout utilizzato per l'interfaccia web.

Non sono mostrate tutte le varie funzioni di callback e di gestione presenti all'interno del file .py

8 Riferimenti

Per la creazione dell'interfaccia web, e per la risoluzione dei vari problemi affrontati ci siamo affidati a:
<https://dash.plotly.com/>

Per l'utilizzo e il funzionamento della libreria u8g abbiamo utilizzato la documentazione presente su
github:

<https://github.com/olikraus/u8glib/wiki/userreference>

Per l'utilizzo dei vari sensori abbiamo trovato utili le seguenti guide:

<https://randomnerdtutorials.com/guide-for-oled-display-with-arduino/>

<https://lastminuteengineers.com/water-level-sensor-arduino-tutorial/>

<https://lastminuteengineers.com/soil-moisture-sensor-arduino-tutorial/>

<https://arduinogetstarted.com/tutorials/arduino-relay>

<https://randomnerdtutorials.com/arduino-with-pir-motion-sensor/>

Per la soluzione ad eventuali problemi, sia hardware che software, ci siamo appoggiati ai vari forum
riguardanti Arduino.