



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

Luglio 2020

Progetto e implementazione di un sistema embedded per il monitoraggio remoto di piante e fiori

Candidato: Riccardo Parosi

Relatore: Prof. Riccardo Berta

Sommario

L'obiettivo di questa tesi è la progettazione e realizzazione di un sistema embedded per il monitoraggio remoto di piante e fiori. Per il rilevamento è stato utilizzato un sensore di luminosità, SparkFun TSL2561, un sensore di umidità dell'aria e della temperatura, DHT22 Pro v1.3, e un sensore capacitivo di umidità del terreno, DFRobot SEN0193. Inoltre è stato adottato uno schermo Arduino TFT per mostrare le informazioni, in locale, dell'ultima misurazione. Tutti i componenti precedentemente citati si interfacciano con un microcontrollore, Arduino Uno Wifi Rev 2, il quale si occupa dell'acquisizione dei dati e dell'invio degli stessi su un cloud. La comunicazione con quest'ultimo viene effettuata attraverso delle API rilasciate da un framework chiamato Measurify. Esso ha il ruolo di interfaccia tra il sistema di rilevamento e l'applicazione per smartphone. Quest'ultima, che è stata sviluppata con Flutter, permette di monitorare i fiori in remoto e di inserire i parametri per impostare il sistema in base alle esigenze dell'utente. Il monitoraggio da smartphone consente, tramite dei grafici, di visualizzare tutti i dati in tre modalità: giornaliera, mensile e annuale.

1 Introduzione

1.1 Sistema Embedded

Questo progetto consiste nella realizzazione di un sistema embedded in grado di monitorare, sia in locale che da remoto, piante e fiori. Esso permette all'utente di controllare completamente lo stato della pianta e quindi coltivarla nelle condizioni ottimali. Infatti ogni tipo di pianta ha bisogno di essere curata in modo specifico per rimanere sana e vigorosa. Il sistema è formato da un microcontrollore (Arduino Uno Wifi Rev 2) che acquisisce i dati tramite tre differenti sensori:

- Sensore di luminosità che fornisce direttamente la misura in Lux, ovvero l'unità di misura, per l'illuminamento, accettata dal Sistema Internazionale. Grazie a questo l'utente ha la possibilità di valutare se il vegetale riceve abbastanza luce per continuare la sua crescita.
- Sensore di temperatura e umidità dell'aria, il quale restituisce le misurazioni in °C (Gradi Celsius) e %RH (Umidità relativa). In questo modo l'utente è in grado di conoscere nel dettaglio l'ambiente in cui si trova la coltura.
- Sensore di umidità del terreno, questo permette di capire se la pianta ha bisogno di acqua. Tuttavia esso non fornisce la misura in %RH (Umidità relativa), ma è necessario elaborare il dato fornito per ricondurre il valore all'unità di misura desiderata. Questo procedimento verrà approfondito più avanti quando sarà descritto nel dettaglio questo sensore.

Una volta che il microcontrollore ha ricevuto le misurazioni, queste vengono mostrate su uno schermo TFT, in modo da rendere semplice e intuitiva la visualizzazione dei dati in locale. Subito dopo, Arduino invia tutti i valori (tramite Wifi) al cloud, attraverso delle richieste POST HTTP, grazie a delle API rilasciate da Measurify, un framework open-source.

Il sistema complessivo è rappresentato in figura 1.

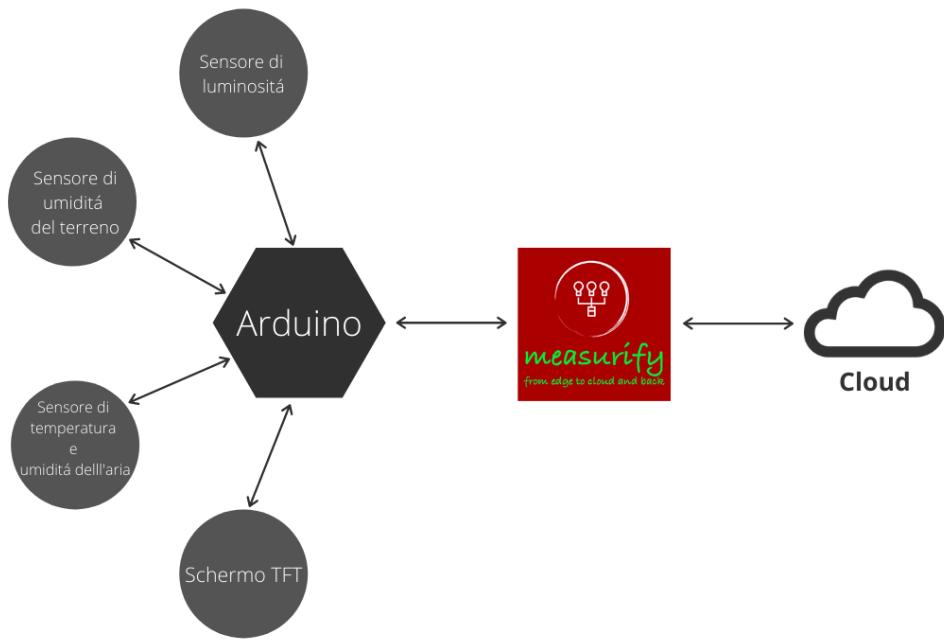


Figura 1: Schema complessivo del sistema

1.2 Applicazione Smartphone

Inoltre è stata sviluppata una applicazione per smartphone, che è stata chiamata Plant Monitor, compatibile con i sistemi operativi Ios e Android, che permette di controllare lo stato della pianta in remoto.

Questo é stato fatto utilizzando il framework open-source Flutter, ideato da Google proprio per questi scopi.

L'applicazione richiede i dati al cloud con delle richieste GET HTTP, sempre sfruttando le stesse API citate precedentemente.

Una volta ricevuti i dati, essi vengono mostrati all'utente tramite dei grafici. Questi ultimi mostrano tutte le misure raccolte in un dato periodo di tempo, che viene scelto dall'utente. In base all'arco temporale scelto vi sono tre modalità in cui vengono mostrati i dati:

- Modalità giornaliera, mostra tutti i dati raccolti nel giorno scelto in base all'ora in cui sono stati rilevati.
- Modalità mensile, mostra la media giornaliera delle misurazioni per tutti i giorni selezionati dall'utente.
- Modalità annuale, mostra la media mensile dei dati raccolti per i mesi selezionati dall'utente.

Oltre a quanto appena descritto, l'applicazione permette di modificare alcuni parametri del sistema, ad esempio username e password di accesso al cloud.

Un parametro molto importante, che é possibile personalizzare, é l'intervallo di tempo tra una misurazione e l'altra, infatti per alcune coltivazioni vi é il bisogno di effettuare rilevazioni più frequentemente mentre per altre meno. In questo modo l'utente riesce ad adattare completamente il sistema alle proprie esigenze.

Le impostazioni del sistema vengono salvate dall'applicazione sul cloud mediante una richiesta PUT HTTP, per mezzo delle API di Meaurify.

Il funzionamento dell'applicazione é rappresentato in figura 2.

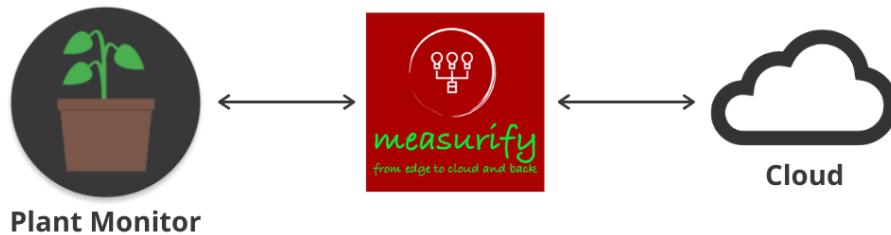


Figura 2: Schema complessivo dell'applicazione

2 Metodi e strumenti utilizzati

2.1 Measurify

Measurify (precedentemente chiamato Atmosphere) é un framework open-source che fornisce gli strumenti necessari per la gestione di "smart things" in un ecosistema IoT.

Esso modella un sistema IoT come una risorsa e permette di gestire tutti i dati raccolti, mettendoli a disposizione per qualunque elaborazione o visualizzazione, come ad esempio web application oppure applicazioni smartphone.

Una risorsa é formata da semplici elementi che sono comunemente presenti in un ecosistema IoT. Di seguito saranno descritti questi elementi, dato che sono fondamentali per capire il funzionamento di questo framework:

- **Thing:** un qualunque oggetto dal quale si vogliono ricavare informazioni, quindi é il soggetto che si vuole monitorare. In questo progetto é la pianta (plant).
- **Feature:** indica che cosa si vuole misurare dell'oggetto preso in esame (Thing). È caratterizzato da un nome e un'unità di misura, infatti fa riferimento a una dimensione fisica, che viene misurata da un dispositivo apposito (Device). In questa tesi ci sono più "feature" ovvero la temperatura, l'umidità dell'aria, l'umidità del terreno e la luce.
- **Device:** un dispositivo che é in grado di misurare una o più dimensioni fisiche (Feature) riferite a un determinato oggetto (Thing). Nel nostro caso é il sistema embedded composto dal microcontrollore e tutti i sensori ad esso collegati. Tuttavia potrebbe essere anche un attuatore (ad esempio un relè che controlla la luce di una stanza) che modifica lo stato di un determinato oggetto (Thing).
- **Measurement:** rappresenta il valore effettivo di una dimensione fisica (Feature), riferita a un oggetto (Thing) e misurata da un determinato dispositivo (Device). In questo progetto é ogni singola misurazione effettuata da ogni sensore, quindi ogni volta che il sistema effettuerà una misurazione produrrà quattro "measurement" che fanno riferimento alle quattro "feature".

Ora che sono stati introdotti i concetti fondamentali, é possibile descrivere il funzionamento di Measurify. Ogni operazione che coinvolge il cloud, quindi sia richiedere che memorizzare informazioni da parte del "device" o da parte dall'applicazione smartphone, viene effettuata attraverso delle RESTful API che vengono chiamate tramite richieste HTTP (GET, POST, PUT, DELETE), differenti in base a ciò che si vuole effettuare.

Le informazioni trasmesse o ricevute dal cloud sono in formato JSON, che é universale, molto conosciuto e semplice per lo scambio di dati.

Per poter comunicare con il cloud é necessario possedere un token. Questo si ottiene per mezzo di una chiamata POST, il cui body, in formato JSON, deve contenere username e password che permettono di identificare l'utente. Il token ha una validità di 30 minuti, dopo i quali é necessario effettuare nuovamente il login per scambiare informazioni con il cloud.

Di seguito é mostrato un esempio della chiamata di autenticazione:

```
POST http://test.atmosphere.tools/v1/login
```

Con il seguente body:

```
{  
    "username" : "plant-monitor-user-username",  
    "password" : "plant-monitor-user-password"  
}
```

In risposta a questa richiesta si ottiene un JSON contenente il token. Questo deve essere inserito come header in ogni richiesta successiva.

Per inviare le misurazioni si usa una richiesta POST, con un body JSON dove vengono indicati tutti gli elementi di Measurify descritti precedentemente.

Si prende come esempio la richiesta per salvare sul cloud una misurazione della temperatura:

```
POST http://test.atmosphere.tools/v1/measurements
```

Con il seguente header (che contiene il token di autenticazione):

```
Authorization : token
```

Con il seguente body:

```
{  
    "thing" : "plant",  
    "feature" : "temperature",  
    "device" : "arduino-wifi",  
    "samples" : [ { "values" : 22.9 } ]  
}
```

Analizzando il body si può notare quanto sia intuitivo e semplice descrivere un oggetto grazie agli elementi del modello di Measurify.

In risposta a questa richiesta si ottiene un JSON contenente la misura appena inviata, questo conferma che il dato è stato salvato correttamente.

Invece per ottenere delle misurazioni dal cloud si usa una richiesta GET, con degli eventuali filtri per ottenere i dati cercati.

Alcuni esempi di filtri sono "thing : plant" che permette di ottenere solo i dati relativi alla nostra pianta e "feature : temperature" che permette di ottenere solo le misurazioni relative alla temperatura.

Oltre a questi ci sono "limit = x" dove x sta a significare quante misurazioni si vogliono ricevere, "page = y" dove y indica in quante pagine si vogliono suddividere i risultati ricevuti.

Al fine di capire bene questi ultimi due filtri, è necessario spiegare che i dati richiesti vengono gestiti in ordine cronologico, quindi se nel filtro "limit" ci fosse 1 al posto di x, la risposta conterrà solamente l'ultima misurazione che è stata inviata al cloud.

Consideriamo, come esempio, una richiesta per ottenere le ultime tre misure relative all'umidità del terreno della pianta:

```
POST http://test.atmosphere.tools/v1/measurements?filter={  
    "thing": "plant", "feature": "moisture"}&limit=3&page=1
```

In risposta a questa richiesta si ottiene un array di oggetti JSON contenenti le tre misure desiderate.

Per semplicità, in questo esempio e in quelli successivi, viene omesso l'header contenente il token, tuttavia esso è strettamente necessario come è stato spiegato in precedenza.

Inoltre nel cloud viene salvato anche un parametro di funzionamento del sistema, l'intervallo di tempo tra una misurazione e l'altra, attraverso uno script. Quest'ultimo è semplicemente un oggetto JSON. Esso viene impostato dall'applicazione smartphone tramite una richiesta PUT, avente come body il JSON.

Come esempio viene mostrata la richiesta per impostare l'intervallo di tempo a 2, ovvero una misurazione ogni due ore:

```
PUT http://test.atmosphere.tools/v1/scripts/plant-monitor  
-script
```

Con il seguente body:

```
{  
    "code": 2  
}
```

Anche in questo caso otterremo, come conferma, una risposta contenente un JSON con il valore inviato.

Questo script viene analizzato dal sistema embedded, ogni volta che viene effettuata una nuova misurazione, tramite una richiesta GET sulla rotta usata nel precedente esempio e ottenendo come risposta il JSON.

2.2 Arduino UNO Wifi Rev 2

Arduino UNO Wifi Rev 2 é un microcontrollore pensato per applicazioni nel campo dell'IoT in ambienti in cui é disponibile una connessione Wifi. Esso utilizza come microcontrollore un ATmega4809 a 8 bit realizzato da Microchip, che viene utilizzato anche in altri prodotti della famiglia Arduino UNO, ad esempio Arduino UNO Rev 2.

Come la maggior parte dei prodotti Arduino, ha una tensione di funzionamento pari a 5V.

Inoltre é dotato di un IMU (Inertial Measurement Unit), in particolare LSM6DS3TR, formato da un accelerometro e un giroscopio. Questo componente non viene utilizzato in questo progetto, dato che la pianta é un soggetto statico, tuttavia esso si rivela molto utile in altri applicazioni.

La connettività Wifi e bluetooth é implementata dal modulo NINA-W102, realizzato da U-BLOX, che utilizza le reti con banda 2,4 GHz. Per implementare la comunicazione Wifi all'interno dello sketch é necessario utilizzare la libreria WiFiNINA di Arduino, che sarà descritta in seguito.

Mentre la comunicazione sicura é garantita attraverso il chip crittografico ECC608, fornito da Microchip, che usa una crittografia SHA-256.

Questa scheda é dotata di 14 pin I/O digitali, che permettono il collegamento di sensori e altri dispositivi. In particolare 5 pin possono essere usati come uscite PWM (pulse width modulation), ovvero pin che erogano in uscita un segnale digitale in cui il periodo dell'impulso a livello alto varia rispetto il periodo del segnale.

Oltre a questo, Arduino é dotato di sei ingressi analogici e una porta di comunicazione USB. Quest'ultima viene utilizzata sia per alimentare il microcontrollore con una tensione di 5V, sia per effettuare il caricamento dello sketch creato dallo sviluppatore e per comunicare con l'IDE in fase di debug.

Inoltre é anche presente un jack di alimentazione che ammette tensioni comprese tra 7V e 12V.

É programmabile con l'uso del popolare IDE di Arduino, tramite il linguaggio C++.

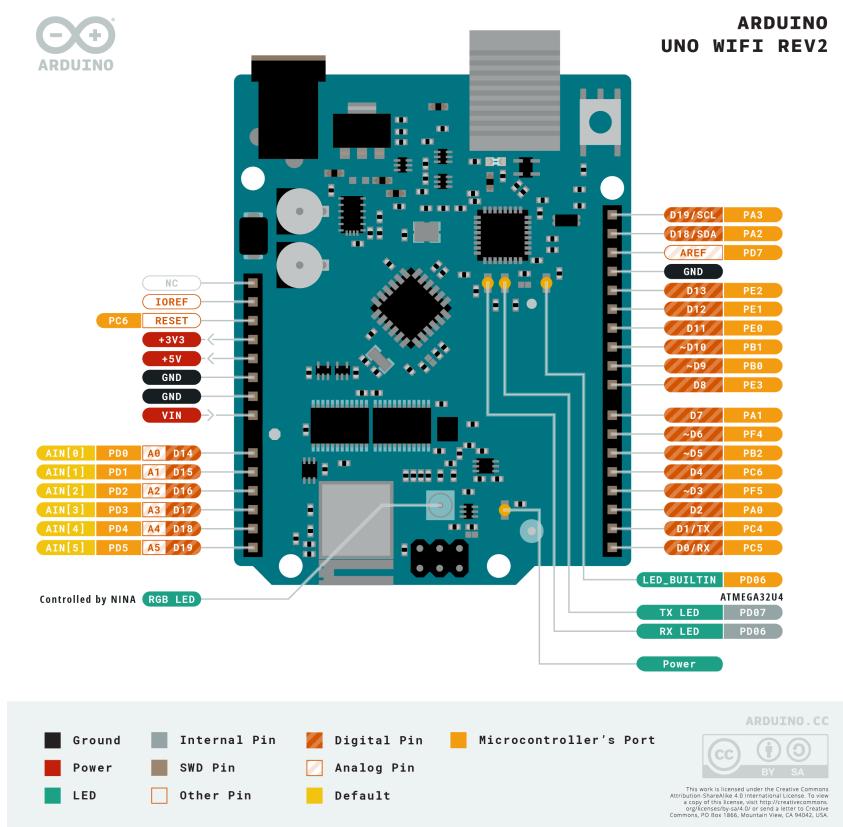


Figura 3: Diagramma I/O Arduino UNO Wifi Rev 2

2.3 Sensore di Luminosità TSL2561

Il TSL2561, realizzato da SparkFun, è un convertitore che trasforma l'intensità della luce in un segnale digitale.

È formato da un fotodiodo a banda larga (in grado di misurare lo spettro di luce visibile all'occhio umano e anche gli infrarossi) e un fotodiodo a risposta infrarossa.

Vi sono anche due convertitori analogico-digitali (ADC) che sono in grado di integrare contemporaneamente le correnti provenienti dai due fotodiodi. Proprio per la presenza di questi componenti è un sensore di luminosità molto sofisticato, perché è in grado di misurare sia piccole che grandi quantità di luce modificando il tempo di integrazione. Infatti riesce a misurare da 0.1 a più di 40k Lux.

Questo sensore è dotato anche di un interrupt programmabile ed è in grado di comunicare con il microcontrollore sfruttando il protocollo di comunicazione I2C.

Come si vede in figura 4, questa scheda è dotata di cinque pin, di seguito vengono descritti partendo da sinistra verso destra:

- **INT:** Si tratta del pin dedicato all'interrupt, esso va collegato al microcontrollore solo se necessario. Ad esempio in questo progetto è scollegato, perché le misurazioni vengono effettuate periodicamente e non ci sono valori per i quali bisogna subito avvisare l'utente.
- **3V3 e GND:** Sono i pin di alimentazione della scheda, GND è la massa mentre 3V3 è la tensione di alimentazione. Quest'ultima deve essere pari a 3V per far funzionare correttamente il sensore.
- **SCL e SDA:** Sono i pin dedicati alla comunicazione tramite il protocollo I2C, è tramite questi che il sensore riceve i comandi e trasmette le misurazioni al microcontrollore.

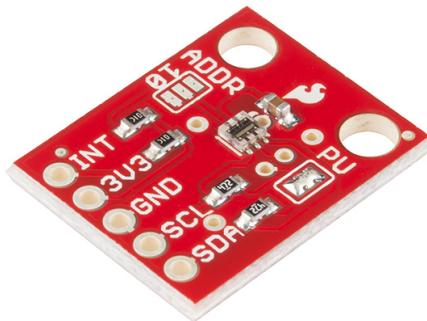


Figura 4: SparkFun TSL2561

2.3.1 Protocollo di Comunicazione I2C

I2C sta per "Inter Integrated Circuit", ovvero un protocollo studiato appositamente per la comunicazione tra circuiti integrati.

È un protocollo master-slave ovvero vi possono essere uno o più master che hanno il controllo del bus e comandano slave multipli.

A ogni scheda collegata al bus viene dato un indirizzo a 7 bit, per questo è possibile collegare 128 dispositivi sullo stesso bus. Questa è la grande potenzialità del protocollo, perché i microcontrollori hanno un numero limitato di pin I/O e quindi non è possibile collegare una grande quantità di componenti, mentre invece utilizzando I2C questo è possibile sfruttando solo due pin.

Infatti questo protocollo è composto da un bus a due fili.

Il primo filo è SDA (Serial Data Line), è una linea utilizzata esclusivamente per trasmettere informazioni in modo seriale.

Il secondo filo è SCL (Serial Clock Line) ed è usato solamente per condividere il segnale di clock. Entrambi i segnali hanno bisogno di una resistenza di pull-up collegata alla tensione VDD (che in questo

progetto é la tensione erogata dal microcontrollore), questo é il riferimento comune della tensione.

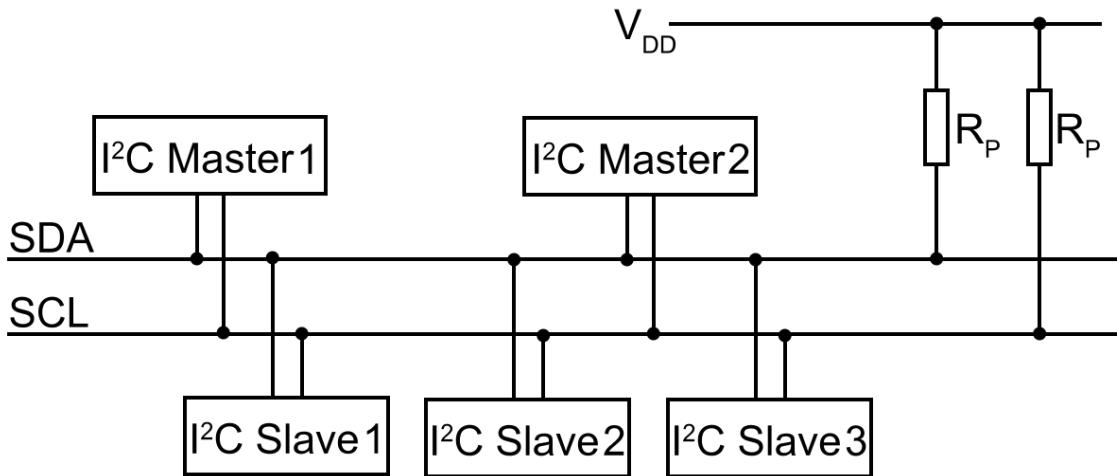


Figura 5: Bus I2C

Il dispositivo master condivide il segnale di clock e genera i segnali di START e STOP, i quali aprono e chiudono la comunicazione con lo slave scelto. Mentre i dispositivi slave attendono l'arrivo di un segnale che può indicare la presenza di un dato in lettura, inviato dal master, oppure la richiesta da parte di quest'ultimo di un particolare dato, che verrà condiviso sulla linea SDA dallo slave.

Dunque il canale é utilizzabile in quattro modi:

- **Master in trasmissione:** il master invia i dati allo slave controllando il segnale di clock.
- **Master in ricezione:** il master riceve dati dallo slave controllando il segnale di clock.
- **Slave in trasmissione:** dopo aver ricevuto la richiesta dal master, lo slave gli invia i dati.
- **Slave in ricezione:** lo slave legge i dati che sono stati trasmessi dal master sul canale.

In questo progetto il master é Arduino e lo slave é il sensore di lumisotità SparkFun.

Per usare il protocollo I2C con Arduino é necessario utilizzare la libreria Wire di Arduino.

2.3.2 Libreria Adafruit TSL2561

Per rendere più semplice la scrittura del codice, necessario al funzionamento e alla comunicazione del sensore con Arduino si é utilizzata la libreria TSL2561_U di Adafruit. É stata scelta questa libreria rispetto a quella di Sparkfun, produttore del dispositivo, perché più intuitiva e pratica nell'utilizzo.

A seguire vengono descritti i principali elementi di questa libreria:

- **Adafruit_UTSL2561_Unified:** La dichiarazione di un oggetto appartenente a questa classe tramite il costruttore, che vedremo al prossimo punto. Permette di dichiarare il sensore all'interno dello sketch e avere accesso a tutti i metodi che la classe mette a disposizione.
- **Adafruit_UTSL2561_Unified(addr, id):** Si tratta del costruttore citato sopra. Come si può notare necessita di due argomenti. Il primo é l'indirizzo I2C del dispositivo, quello di default é 0x39. Il secondo é l'id univoco che viene utilizzato con le API dei sensori di Adafruit per poter identificare il dispositivo nei data log.
- **begin():** Questo metodo inizia la connessione tra Arduino e il sensore, quindi é fondamentale per il funzionamento di quest'ultimo.
- **enableAutoRange():** Permette di abilitare la modalità autorange del sensore che consente al dispositivo di adattare automaticamente la sua sensibilità in base alla situazione. Ha come argomento un bool, se true la modalità é abilitata, se false é disabilitata. In questo progetto la modalità é attivata.

- **setIntegrationTime()**: Metodo utile a impostare il tempo di integrazione del dispositivo e quindi la risoluzione delle misurazioni. Si è deciso, per questo progetto, di utilizzare la risoluzione migliore corrispondente a un tempo di 402 ms, dato che non vi è necessità di effettuare velocemente le misurazioni.
- **getEvent()**: Effettua la misurazione tramite un evento di Adafruit. Infatti necessita come argomento quest'ultimo che deve essere stato definito precedentemente. La misura sarà contenuta nel campo light dell'evento.

2.4 Sensore di Umidità e Temperatura dell'Aria DHT22

Il DHT22 Pro v1.3 è un sensore digitale di temperatura e umidità dell'aria, realizzato da Grove. Utilizza un sensore di umidità capacitivo e un termistore (un resistore il cui valore di resistenza varia in maniera significativa con la temperatura). Questi componenti sono gestiti da un microcontrollore 8 bit ad alte prestazioni.

Per quanto riguarda l'umidità il raggio di rilevamento di questo sensore è da 0 %RH a 99.9 %RH, mentre per la temperatura da -40 °C a 80 °C.

Ha una risoluzione di 0.1 %RH per l'umidità e 0.1 °C per la temperatura.

La sua precisione è di ±2 %RH di umidità relativa e ±0.5 °C di temperatura.

Come si vede in figura 6, questo sensore è dotato di quattro pin, a seguire vengono descritti partendo dall'alto verso il basso:

- **GND e VCC**: Sono i pin di alimentazione della scheda, GND è la massa mentre VCC è la tensione di alimentazione. Quest'ultima deve essere compresa tra 3.3V e 6V per far funzionare correttamente il sensore.
- **NC**: Questo pin non è da collegare, infatti NC sta per Not Connected.
- **SIG**: Il pin dedicato alla trasmissione dei dati.

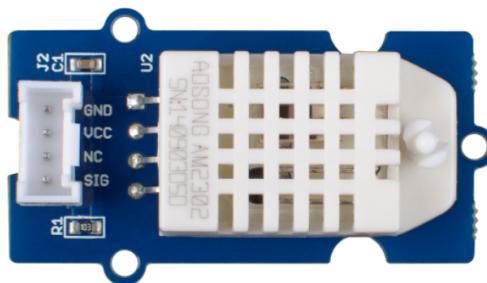


Figura 6: DHT22 Pro v1.3

2.4.1 Libreria Adafruit DHT

Per rendere più semplice la scrittura del codice, necessario al funzionamento e alla comunicazione del sensore con Arduino si è utilizzata la libreria DHT di Adafruit.

Di seguito vengono descritti i principali elementi di questa libreria:

- **DHT**: La dichiarazione di un oggetto appartenente a questa classe tramite il costruttore, che vedremo al prossimo punto. Permette di dichiarare il sensore all'interno dello sketch e avere accesso a tutti i metodi che la classe mette a disposizione.
- **dht(dhtPin, dhtType)**: Si tratta del costruttore citato sopra. Come si può notare necessita di due argomenti. Il primo è semplicemente il numero del pin di Arduino a cui è attaccato il pin SIG del sensore. Il secondo è il tipo di dispositivo che si sta usando, infatti questa libreria è utilizzabile con tutti i sensori DHT, in questo progetto ad esempio è DHT22.

- **begin()**: Questo metodo inizia la connessione tra Arduino e il sensore, quindi è fondamentale per il funzionamento di quest'ultimo.
- **readHumidity()**: È il metodo che restituisce la misura dell'umidità in %RH sul pin SIG.
- **readTemperature()**: È come il metodo citato sopra, soltanto che restituisce la misura della temperatura in °C.

2.5 Sensore di Umidita' del Terreno SEN0193

Il SEN0193 è un sensore capacitivo di umidità del terreno, prodotto da DFRobot.

È realizzato in un materiale resistente alla corrosione che gli conferisce una durata longeva.

Una cosa importante su questo sensore è che non è totalmente impermeabile. Infatti come è possibile notare in figura 7, prima dei componenti elettronici a sinistra, vi è una riga bianca perpendicolare alla lunghezza del dispositivo, questa rappresenta il limite fino al quale si può inserire il sensore nel terreno.

Come è stato accennato nell'introduzione, è l'unico sensore tra quelli scelti, che non trasmette i valori direttamente nell'unità di misura desiderata, ovvero l'umidità relativa (%RH).

Difatti questo dispositivo riporta il dato in una scala da 260 a 520. Questi valori variano leggermente da apparecchio ad apparecchio. Per conoscere quelli relativi al proprio, è necessario effettuare due test:

- **Test in acqua**: Consiste nel verificare quale valore riporta il sensore mentre è immerso in acqua. La misura sarà molto simile a 260, ad esempio in questo progetto era 261 ed è stato approssimato per difetto a 260.
- **Test in aria**: Si tratta di verificare quale valore riporta il sensore mentre è esposto all'aria. La misura sarà molto simile a 520, ad esempio in questo progetto era esattamente così.

Queste due misure ottenute, rappresentano semplicemente il 100 %RH (260) e lo 0 %RH (520).

Per portare questi valori in umidità relativa, è stata usata la seguente formula:

$$\frac{m - 520}{260 - 520} \cdot 100$$

Dove m rappresenta la misura che si vuole convertire in %RH.

Come si vede in alto a sinistra nella figura 7, questo sensore è dotato di un connettore a tre pin, a seguire vengono descritti questi ultimi partendo dall'alto verso il basso:

- **SIG**: Il pin dedicato alla trasmissione dei dati, va collegato a un pin analogico del microcontrollore che intende implementarlo.
- **VCC e GND**: Sono i pin di alimentazione della scheda, VCC è la tensione di alimentazione mentre GND è la massa. Quest'ultima deve essere compresa tra 3.3V e 5.5V per far funzionare correttamente il sensore.



Figura 7: DFRobot SEN0193

2.6 Schermo Arduino TFT

Lo schermo Arduino TFT, come suggerisce il nome stesso, é un display LCD TFT da 160 x 128 pixel. Questo apparecchio é provvisto di uno slot per scheda microSD, da quest'ultima possono essere caricate immagini da visualizzare sullo schermo.

Questo dispositivo può essere configurato in due modi:

- **Interfaccia SPI:** Si tratta di collegare l'apparecchio a questa interfaccia hardware, la quale é implementata dalla maggior parte dei prodotti della famiglia di Arduino. In seguito verrà trattata in modo dettagliato. Come in questo caso, questa interfaccia semplifica notevolmente la configurazioni di molti dispositivi, poiché permette di
- **Configurazione Manuale:** Questo metodo prevede il collegamento di tutti i pin dello schermo a pin I/O normali di arduino e la dichiarazione di tutti i pin da parte dello sviluppatore. Anche se non richiede molto tempo in più rispetto alla configurazione precedente, é sconsigliato perché meno intuitivo.

Il dispositivo funziona nello stesso modo in entrambe le configurazioni, tranne le operazioni legate alla scheda microSD. Infatti queste ultime, sono disponibili solo se lo schermo é collegato attraverso interfaccia SPI.

Come si vede in figura 8, questo sensore é dotato di 10 pin, di seguito vengono descritti partendo dall'alto verso il basso:

- **5V:** É uno dei due pin di alimentazione, questo é la tensione di alimentazione. Quest'ultima deve essere di 5V per far funzionare correttamente il dispositivo.
- **MISO, SCK, MOSI:** Sono i pin dedicati alla comunicazione tramite l'interfaccia SPI.
- **LCD CS e SD CS:** Si tratta dei di selezione dei vari chip presenti sul display. Il primo seleziona il chip SPI dello schermo. Il secondo seleziona il chip SPI della scheda microSD.
- **D/C:** Si tratta del pin per i dati SPI dello schermo ma anche il pin selettore dei comandi. Infatti D/C sta per data/command.
- **RESET:** É il pin di reset dello schermo, si attiva quando viene connesso alla massa, quindi quando riceve un livello logico basso.
- **BL:** Questo è l'ingresso PWM per il controllo della retroilluminazione. Per impostazione predefinita, è a livello alto (retroilluminazione attivata) quindi é possibile usare il PWM a qualsiasi frequenza. Per disattivarla é necessario mettere il PWM a livello basso.
- **Ground:** É il secondo dei due pin di alimentazione, in particolare é la massa.

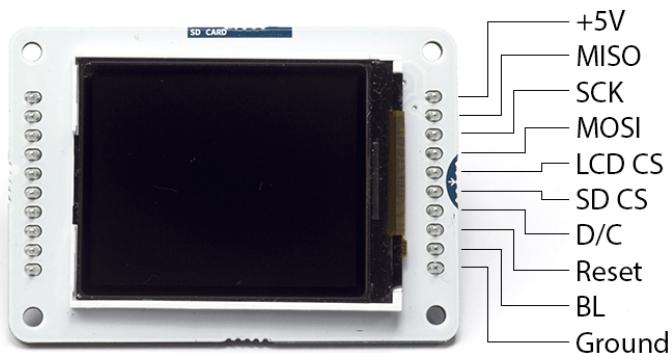


Figura 8: Schermo Arduino TFT

2.6.1 Interfaccia SPI

SPI significa Serial Peripheral Interface ovvero interfaccia periferica seriale, è uno standard basato su un bus di comunicazione sincrono, full duplex, con architettura master/slave, originariamente sviluppato da Motorola.

Si basa su due registri, generalmente da 8 bit, posti uno sul master e uno sul dispositivo slave. La comunicazione avviene attraverso lo "shift" (spostamento) dei bit di questi registri grazie a un segnale di clock. Difatti come verrà descritto tra poco, alcuni pin permettono di simulare che i due registri siano contigui. Il master può leggere e scrivere dati solo nei primi 8 bit del registro, lo slave lo può fare solo negli ultimi 8.

Questa interfaccia è composta da quattro pin:

- **SCLK:** Sta a indicare Serial Clock, ed è il pin dedicato alla condivisione del segnale di clock.
- **MISO e MOSI:** Si tratta dei pin che permettono di simulare che i due registri siano contigui, come detto in precedenza. Il primo significa Master Input Slave Output e serve per rendere il primo bit del registro master contiguo all'ultimo bit del registro slave. Il secondo significa Master Output Slave Input e ha lo scopo di rendere l'ultimo bit del registro master contiguo al primo bit del registro slave.
- **SS:** Significa Slave Select, questo pin ha il fine di selezionare lo slave desiderato.

Come è possibile notare nelle figure 9 e 10, ci sono due modalità architettonali per la connessione di più slave allo stesso master.

La prima (in figura 9) ha i canali SCLK, MISO e MOSI condivisi, ma i canali SS sono uno per ogni slave. Questo ha il vantaggio di poter selezionare e comunicare con solo uno slave e ha anche una velocità elevata per la trasmissione dei dati. Come svantaggio ha il fatto che se un microcontrollore ha pochi pin non si potranno controllare tanti slave.

Il secondo (in figura 10) è chiamato Daisy Chain ed è un collegamento a catena, ovvero gli slave sono connessi tra di loro uno dopo l'altro. In particolare come si può vedere in figura il canale MISO del primo slave è connesso al canale MOSI dello slave successivo e così via per tutti gli slave ad esclusione del MISO dell'ultimo slave che è connesso con il MISO del master. Questa configurazione ha il vantaggio di occupare solo quattro pin. Tuttavia ha molti svantaggi come il dover comunicare simultaneamente con tutti gli slave e una velocità ridotta per la trasmissione dei dati.

Lo schermo citato in questo paragrafo appartiene alla prima configurazione poiché vi sono i pin LCD CS e SD CS.

Per usare l'interfaccia SPI con Arduino è necessario utilizzare la libreria SPI di Arduino.

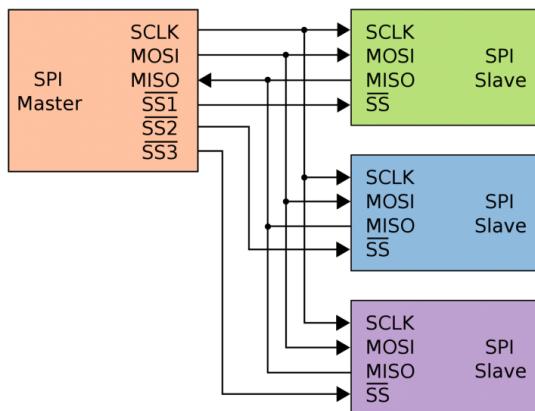


Figura 9: Interfaccia SPI con slave select multipli

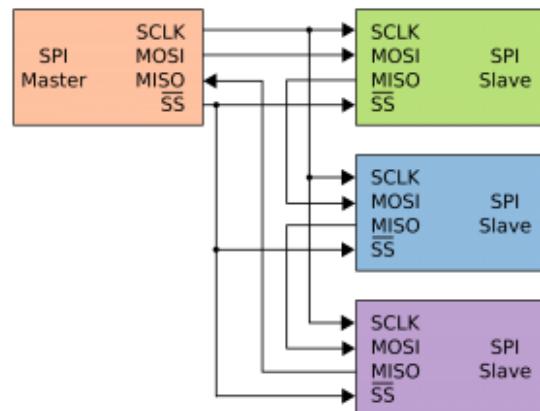


Figura 10: Interfaccia SPI Daisy Chain

2.6.2 Libreria TFT

Per rendere più semplice la scrittura del codice, necessario al funzionamento e alla comunicazione dello schermo con Arduino si è utilizzata la libreria TFT di Arduino.

A seguire vengono descritti i principali elementi di questa libreria:

- **TFT:** La dichiarazione di un oggetto appartenente a questa classe tramite il costruttore, che vedremo al prossimo punto. Permette di dichiarare il display all'interno dello sketch e avere accesso a tutti i metodi che la classe mette a disposizione.
- **TFT(tftCs, tftDc, tftReset):** Si tratta del costruttore citato sopra. Come si può notare necessita di tre argomenti. Sono semplicemente i numeri dei pin di Arduino a cui sono collegati i pin LCD CS, D/C e RESET dello schermo.
- **begin():** Questo metodo inizia la connessione tra Arduino e il display, quindi è fondamentale per il funzionamento di quest'ultimo.
- **background(r, g, b):** Permette di riempire lo schermo di un solo colore. I tre argomenti sono rispettivamente il codice RGB del colore scelto.
- **stroke(r, g, b):** Deve essere chiamato prima di iniziare a scrivere o disegnare. Difatti serve a definire il colore del testo o del bordo del disegno. Come il metodo precedente, i tre argomenti sono rispettivamente il codice RGB del colore scelto.
- **fill(r, g, b):** È esattamente come quello appena citato, solo che definisce il colore di riempimento dei disegni.
- **rect(xStart, yStart, width, height):** Disegna un rettangolo sul display. I primi due argomenti sono le coordinate dell'angolo in alto a sinistra della figura. Mentre gli altri due argomenti sono la larghezza e l'altezza del rettangolo.
- **setTextSize(s):** Imposta la grandezza del testo che segue questo metodo. L'argomento indica la dimensione, di default è 1 e ogni numero successivo aumenta il testo di 10 pixel (quindi 1 è 10 pixel, 2 è 20 pixel e così via).
- **text(text, xPos, yPos):** Scrive il testo sullo schermo nelle coordinate date. Il primo argomento indica il testo da disegnare e deve essere un char array. Il secondo e il terzo argomento sono le coordinate x e y di dove si vuole iniziare a scrivere il testo.

2.7 Flutter

Flutter è un framework open-source creato da Google, con lo scopo di sviluppare applicazioni native per IOS e Android e anche applicazione Web. Come editor è stato utilizzato Visual Studio Code, realizzato da Microsoft, il quale implementa e si interfaccia con l'SDK.

Flutter si basa su questi elementi principali:

- **Dart platform:** È un linguaggio di programmazione sviluppato da Google con lo scopo di sostituire JavaScript per lo sviluppo Web. Il compilatore di Dart permette di scrivere programmi sia per web che per desktop, smartphone e server, tramite due diverse piattaforme.
La prima è Dart Native pensata per smartphone, server, desktop e altro.
La seconda è Dart Web pensata appositamente per il web.
- **Flutter Engine:** Scritto principalmente in C++, è ciò che rende possibile il rendering di basso livello utilizzando la libreria grafica di Google, chiamata Skia Grapichs. Una delle funzionalità più apprezzate di questo motore, è nominata "hot-reload" e permette, in fase di sviluppo, di mostrare le modifiche apportate senza il bisogno di riavviare l'applicazione.
- **Foundation Library:** Si tratta di una libreria scritta in Dart, la quale fornisce tutte le classi e i metodi di base per sviluppare applicazioni tramite Flutter.
- **Widgets:** Il codice sviluppato con Flutter si basa sui widgets, i quali sono descrizioni immutabili dell'interfaccia utente, alcuni esempi possono essere testi, pulsanti, forme, grafici e animazioni. I widgets sono una delle cose che rendono lo sviluppo con Flutter semplice e intuitivo, poiché se si è interessati a creare parti più complesse, è necessario solamente combinare widgets elementari di diverso tipo.

2.8 Libreria WiFiNINA

Come detto nel paragrafo 2.1, per implementare la connettività nello sketch di Arduino è necessario usare la libreria WiFiNINA, realizzata da Arduino.

Questa libreria si divide in due classi principali: la classe WiFi e la classe Client.

Di seguito vengono descritti i principali elementi della classe WiFi, e quelli che sono stati usati in questo progetto :

- **begin(ssid, pass):** Questo metodo permette di connettere Arduino a un access point tramiti Wifi. Il primo argomento è l'ssid della rete alla quale ci si vuole collegare. Il secondo argomento è la password necessaria per connettersi alla rete.
- **status():** Permette di conoscere lo stato della connessione alla rete. Infatti restituisce una stringa, la quale è un codice specifico che identifica lo stato della connessione.
- **end():** È il metodo che spegne il modulo Wifi. Se il modulo è connesso a una rete, esso verrà disconnesso da quest'ultima.

Invece di seguito vengono descritti i principali elementi della classe Client, e quelli che sono stati usati in questo progetto:

- **WiFiClient:** La dichiarazione di un oggetto appartenente a questa classe. Permette di dichiarare il WiFiClient all'interno dello sketch e avere accesso a tutti i metodi che la classe mette a disposizione.
- **connect(url, port):** Permette di connettere Arduino a un server. Il primo argomento è l'indirizzo url del server al quale ci si vuole connettere. Il secondo argomento è la porta attraverso la quale ci si connette.
- **connected():** Consente di capire se Arduino è connesso al server. Difatti ritorna un bool, il quale è true se la connessione è ancora attiva, false nel caso contrario.
- **available():** Restituisce il numero di byte disponibili per la lettura (ovvero la quantità di dati che è stata scritta sul client dal server a cui è connesso).
- **flush():** Questo metodo elimina tutti i byte che sono stati scritti sul client, ma non ancora letti.
- **read():** Legge i dati che sono stati scritti sul client. Se non viene fornito alcun dato, verrà restituito il carattere successivo nel buffer.
- **println():** Scrive i dati sul server, a cui è connesso il client, seguiti da un ritorno a capo e una riga vuota.

2.9 Libreria ArduinoJson

Oltre alle librerie che sono già state citate precedentemente, ne è stata utilizzata un'altra. In particolare ArduinoJson, sviluppata da Benoit Blanchon. Essa ci permette di creare e realizzare oggetti JSON velocemente e intuitivamente.

Gli elementi principali di questa libreria sono i seguenti:

- **StaticJsonDocument<size>:** È un oggetto utile a salvare un documento JSON in memoria. Tuttavia è necessario conoscere la dimensione del file per poter istanziare l'oggetto correttamente, difatti essa va inserita al posto di size.
- **serializeJson(jsonDocument, text):** Permette di creare un documento JSON. Quest'ultimo verrà salvato nello StaticJsonDocument indicato nel primo argomento. Il secondo argomento è la stringa che si vuole trasformare in JSON.
- **deserializeJson(jsonDocument, stream):** Si tratta del metodo che permette di salvare un documento in formato JSON proveniente da una Stream, ad esempio un client Wifi. Il primo argomento indica lo StaticJsonDocument dove si vuole salvare il file ricevuto. Il secondo argomento indica la Stream dalla quale arriva il file.

3 Sperimentazione e risultati

3.1 Acquisizione e invio dei dati

Il sistema embebedded necessario all'acquisizione dei dati ed al loro invio a Measurify é composto da un microcontrollore, Arduino UNO Wifi Rev 2, al quale sono stati collegati tutti i dispositivi descritti precedentemente. Si é deciso, per non appesantire questa parte del testo (dato che vi sono molti collegamenti), di schematizzare le connessioni con delle frecce. A seguire verrà utilizzato questo metodo, per descrivere il cablaggio del sistema complessivo (figura 11):

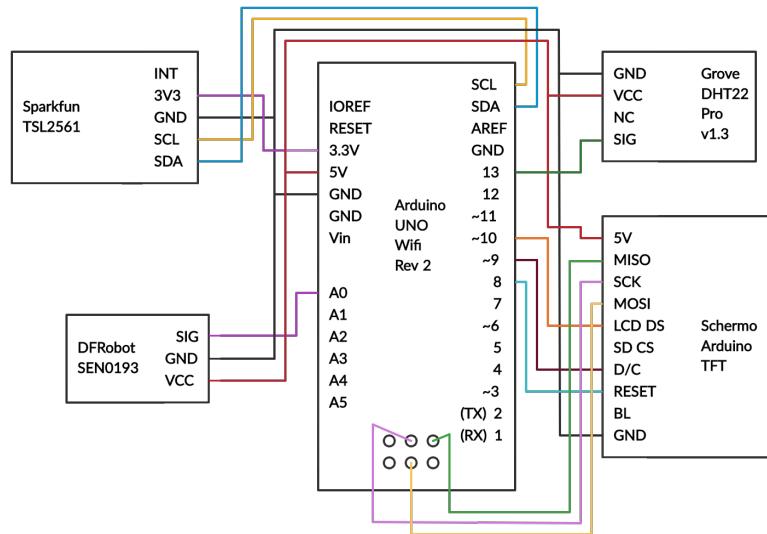


Figura 11: Cablaggio del sistema complessivo

- **Cablaggio Sensore di Luminosità Sparkfun TSL2561:**

Il lato sinistro corrisponde ai pin del sensore Sparkfun TSL2561, mentre il lato destro corrisponde ad Arduino UNO Wifi Rev 2.

TSL2561 → Arduino
SDA → SDA
SCL → SCL
3V3 → 3.3V
GND → GND

- **Cablaggio Sensore di Umidità e Temperatura dell'Aria DHT22:**

Il lato sinistro corrisponde ai pin del sensore DHT22, mentre il lato destro corrisponde ad Arduino UNO Wifi Rev 2.

DHT22 → Arduino
SIG → Pin 13
VCC → 5V
GND → GND

- **Cablaggio Sensore di Umidità del Terreno SEN0193:**

Il lato sinistro corrisponde ai pin del sensore SEN0193, mentre il lato destro corrisponde ad Arduino UNO Wifi Rev 2.

SEN0193 → Arduino
SIG → Pin A0
VCC → 5V
GND → GND

- Cablaggio Schermo Arduino TFT:**

Il lato sinistro corrisponde ai pin dello schermo TFT, mentre il lato destro corrisponde ad Arduino UNO Wifi Rev 2.

Schermo TFT → Arduino
5V → 5V
MISO → MISO
SCK → SCK
MOSI → MOSI
LCD CS → Pin 10
D/C → Pin 9
RESET → Pin 8
GND → GND

Per alimentare il microcontrollore è stato utilizzato il suo trasformatore (venduto assieme ad Arduino) collegato a una presa elettrica.

Il sistema esegue le istruzioni che sono racchiuse all'interno di uno sketch, il quale viene caricato su Arduino attraverso il suo IDE.

Lo sketch è composto da due fasi, quella di setup e quella di loop.

3.1.1 Fase di setup

La fase di setup è eseguita solamente una volta all'avvio del sistema. In questo processo vengono dichiarati gli oggetti delle librerie che sono state descritte precedentemente e avviati tutti i dispositivi collegati al microcontrollore.

3.1.2 Fase di loop

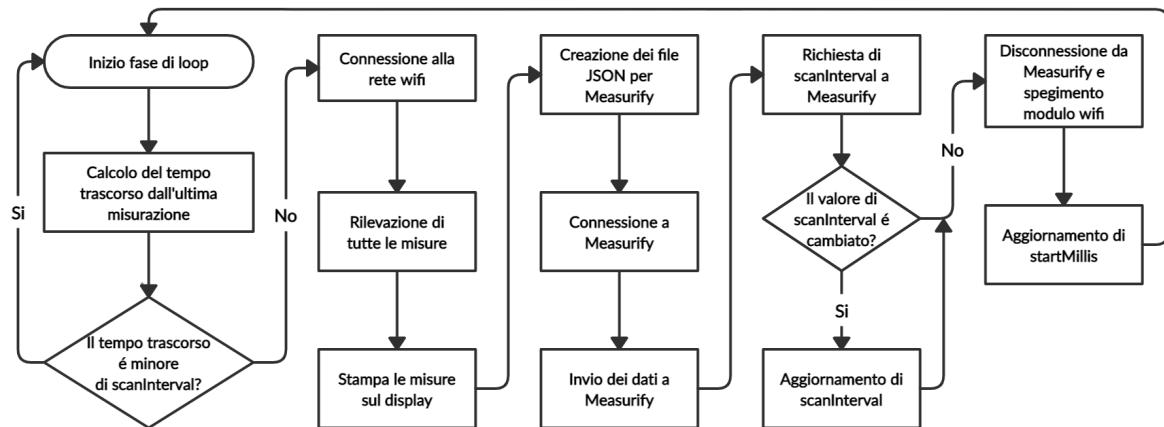


Figura 12: Diagramma di flusso della fase di loop

La fase di loop è caratterizzata da un algoritmo (descritto nel diagramma di flusso in figura 11) che viene iterato per tutto il tempo di funzionamento del sistema. Di seguito viene analizzato l'algoritmo seguendo gli step in figura 11:

- Calcolo del tempo trascorso dall'ultima misurazione:** Si tratta della prima operazione che viene eseguita nella fase di loop. Si basa sull'uso della funzione millis(). Quest'ultima ritorna il tempo trascorso dall'avvio del sistema. Questo dato viene sottratto con la variabile startMillis (rappresenta il valore di millis() calcolato alla fine della fase di loop), ottenendo il tempo trascorso dall'ultima misurazione. È importante notare che startMillis è inizializzata a zero.
- Si controlla se il tempo trascorso è minore di scanInterval:** La variabile scanInterval rappresenta l'intervallo di tempo tra una rilevazione e l'altra ed è scelta dall'utente. Essa è inizializzata a zero, questo permette che nella prima iterazione della fase di loop le misurazioni vengano eseguite subito. Infatti in quest'ultimo caso, il tempo trascorso sarà sicuramente maggiore di zero e

ciò permette di proseguire immediatamente con l'algoritmo. Invece nelle iterazioni successive alla prima, se il tempo trascorso è minore di scanInterval, verrà iniziata nuovamente la fase di loop fino a quando la condizione non sarà più verificata.

- **Connessione alla rete wifi:** In questo step viene usata la classe WiFi della libreria WiFiNINA, descritta nel capitolo 2.8, per connettere Arduino alla rete wifi.
- **Rilevazione di tutte le misure:** Tutte le rilevazioni, tranne quella della luminosità, vengono effettuate cento volte con un ritardo di 100 ms tra una e l'altra, al fine di ottenere una maggiore precisione. Successivamente verranno divise per cento in modo da ottenere la media. Inoltre per ottenere misurazioni con solo un numero decimale (altri numeri sono superflui per i fini del progetto), la media viene moltiplicata per dieci, approssimata con la funzione round() e divisa per dieci. Per quanto riguarda la rilevazione della luminosità, essa è eseguita soltanto una volta. Questo perché il dato può assumere valori compresi tra zero e oltre 40k Lux, considerando anche che il sensore utilizzato è molto preciso, una variazione di qualche Lux è trascurabile ai fini del progetto.
- **Stampa delle misure sul display:** Dopo aver eseguito le misurazioni, queste vengono rappresentate sullo schermo Arduino TFT, tramite l'apposita libreria descritta nel paragrafo 2.6.2. Ciò permette di monitorare in locale la pianta senza l'uso dello smartphone. In figura 12 si può osservare l'interfaccia del display.

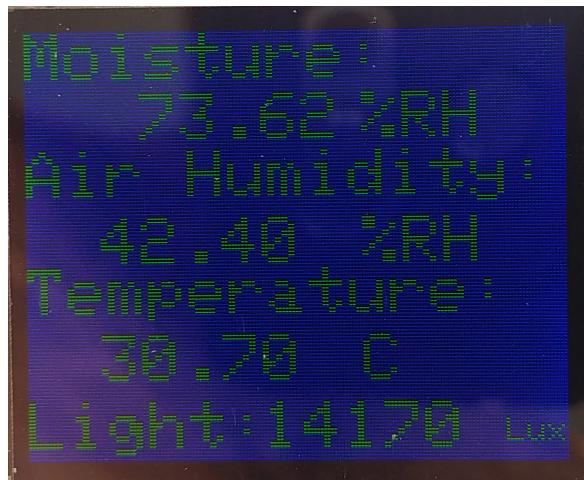


Figura 13: Interfaccia del display TFT

- **Creazione dei file JSON per Measurify:** A questo punto i dati raccolti vengono convertiti in oggetti JSON, tramite la libreria ArduinoJson descritta nel paragrafo 2.9. Questo passaggio ha lo scopo di creare i body per le richieste di salvataggio a Measurify. Difatti i file JSON sono costruiti esattamente come quello mostrato nel paragrafo 2.1.
- **Connessione a Measurify:** La connessione avviene tramite la classe WiFiClient della libreria WiFiNINA, illustrata nel paragrafo 2.8. Ciò permette di comunicare con Measurify.
- **Invio dei dati a Measurify:** Questo è il processo che permette di salvare le misurazioni sul cloud. Avviene mediante quattro richieste POST a Measurify (una per ogni misura), aventi come body i file JSON creati precedentemente. Le richieste sono uguali a quella mostrata nel paragrafo 2.1.
- **Richiesta di scanInterval a Measurify:** È il momento di verificare se l'utente ha modificato il valore di scanInterval. Per prima cosa viene eseguita una richiesta GET sullo script di Measurify (richiesta descritta nel paragrafo 2.1), per ottenere il valore attuale caricato sul server. Dopo viene confrontato il dato appena ricevuto con quello salvato sul microcontrollore. Se i due dati sono diversi, scanInterval verrà sostituito dal nuovo valore. In caso contrario non saranno eseguite modifiche. Un esempio di questo step, è la prima iterazione della fase di loop. Difatti in essa scanInterval vale zero e dunque sarà sostituito con il valore inserito dall'utente (poiché l'applicazione smartphone non permette di salvare sul cloud il valore zero).
- **Disconnessione da Measurify e spegnimento del modulo wifi:** Arrivati a questo punto le operazioni che richiedono una connessione a internet sono terminate. Dunque è possibile procedere

con la disconnessione da Measurify e con lo spegnimento del modulo wifi. Queste operazioni avvengono tramite le classi WiFi e WiFiClient della libreria WiFiINA (paragrafo 2.8).

- **Aggiornamento di startMillis:** Si è giunti alle ultime istruzioni della fase di loop. Esse sovrascrivono il valore di startMillis con quello che ritorna la funzione millis(). Ciò permette di iniziare a contare il tempo trascorso dall'ultima misurazione (avvenuta qualche secondo prima).

3.2 Applicazione smartphone

Come anticipato nell'introduzione, é stata sviluppata un'applicazione smartphone mediante l'utilizzo del framework open-source Flutter (paragrafo 2.7), realizzato da Google.

L'applicazione é stata chiamata Plant Monitor ed é disponibile per i sistemi operativi IOS e Android.

La scelta di creare questo software é stata fatta per rendere il progetto completo. Infatti grazie all'app, l'utente è in grado di monitorare le piante da remoto e modificare l'intervallo di scansione tra una misurazione e l'altra.

L'applicazione é formata da sei schermate: InitialLoadingScreen, LogInScreen, HomeScreen, PlantScreen, SettingsScreen e AccountScreen.

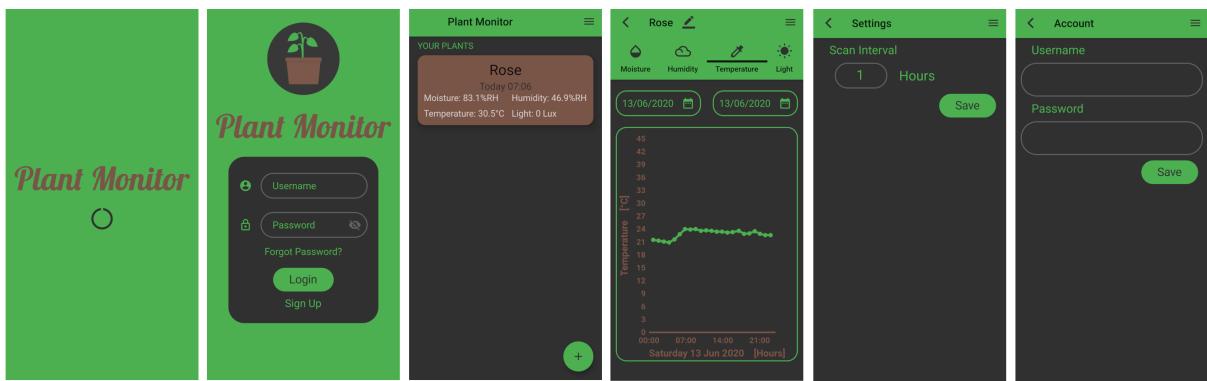


Figura 14: Partendo da sinistra: InitialLoadingScreen, LogInScreen, HomeScreen, PlantScreen, SettingsScreen, AccountScreen

3.2.1 InitialLoadingScreen

Questa pagina viene visualizzata se l'app é stata precedentemente chiusa oppure se é la prima volta che viene aperta. Si tratta di una schermata di caricamento, infatti é formata da un Text che riporta la scritta "Plant Monitor" e un CircularProgressIndicator (un cerchio che gira per indicare lo stato di caricamento). È stata creata per verificare se l'utente ha già eseguito l'accesso oppure no. Difatti se il log in non é ancora stato compiuto, verrà caricata l'apposita schermata. Mentre in caso contrario, l'utente sarà indirizzato alla pagina di home. È possibile visualizzare l'interfaccia in figura 14.

3.2.2 LogInScreen

É la pagina di log in, ovvero permette all'utente di inserire le credenziali di Measurify per poter accedere al proprio account e all'applicazione stessa.

Il primo elemento, in alto, che forma la schermata é un'immagine che riporta il logo dell'app. Sotto di questo vi é un Text con scritto "Plant Monitor". Ancora sotto si trova un Container con svariati widgets. I primi sono Due TextField dove l'utente deve inserire l'username e la password. Poi vi é Un Text che contiene la scritta "Forgot Password?". In questa versione rappresenta un elemento di design. Tuttavia può trasformarsi nel classico tasto per recuperare la password, nel caso in cui questa funzionalità sarà implementata.

A seguire un Button, che se premuto tenta il log in. Questo avviene tramite una richiesta POST a Measurify (trattata nel paragrafo 2.1). Se la risposta a questa richiesta ha il codice 200, l'utente sarà portato alla schermata di home e saranno salvate le credenziali, mentre in caso contrario sarà visualizzato un FlutterToast che riporterà la scritta "Wrong username or password". I Toast sono anche usati (con messaggi diversi) nel caso in cui l'utente clicchi il tasto senza inserire uno dei due campi o entrambi.

Come ultimo elemento abbiamo un altro Text che presenta la scritta "Sign Up". Per questo vale lo stesso discorso fatto per quello contenente la scritta "Forgot Password??".

3.2.3 HomeScreen

La pagina principale dell'applicazione. Mostra tutti i sistemi relativi alle piante che l'utente possiede. In questa fase di test il sistema era solamente uno. Oltre all'AppBar con il nome dell'app e il tasto opzioni sulla sinistra, vi è un altro widget. Si tratta di un Container cliccabile con al suo interno ha sei Text. Il primo riferito al nome che l'utente ha assegnato alla pianta, in questo esempio "Rose". Il secondo rappresenta la data e l'ora delle ultime misurazioni effettuate. Queste sono rappresentate nei restanti Text. Gli elementi riferiti alle misurazioni sono caricati tramite una richiesta GET a Measurify (molto simile a quella vista nel paragrafo 2.1). Stessa cosa accade effettuando uno scroll verso il basso, ovvero i dati vengono ricaricati e aggiornati. Mentre premendo sul tasto impostazioni in alto o facendo uno scroll verso sinistra, si apre un Drawer. Esso è presente in tutte le schermate tranne quella di caricamento e quella di log in. È caratterizzato dal logo dell'app, due tasti "Account" e "Log Out" (il quale riporta l'utente alla pagina di log in, cancellando le credenziali di accesso dalla memoria) e una BottomNavigationBar che riporta la scritta "Settings".

3.2.4 PlantScreen

Pagina relativa ai grafici della pianta. Vi è un'AppBar con ai due lati il tasto indietro e impostazioni, mentre in centro il nome della pianta di cui si stanno visualizzando i dati. Vicino al nome della pianta, vi è un tasto che permette il suo cambio per mezzo di un AlertDialog. Sotto l'AppBar vi è il TabController, infatti questa schermata si basa su quattro Tab diverse, una per ogni misura. Per passare da una all'altra basta schiacciare il simbolo della misura desiderato o scorrere verso destra o sinistra. Ancora sotto, che fanno parte della Tab, possiamo vedere due Container cliccabili, contenenti due date. Quella a sinistra corrisponde a startDate, quella a destra a endDate. Premendo su queste ultime si accede a un calendario dove è possibile modificare le due date. Infine vi è il grafico delle misurazioni, è disponibile in tre modalità:

- **Modalità giornaliera:** Questa modalità è attiva quando startDate ed endDate sono uguali. Il grafico mostra tutte le misure della giornata scelta, indicando sull'asse x l'orario.
- **Modalità mensile:** Si Attiva quando la differenza tra startDate ed endDate è maggiore di zero e minore uguale a 31 giorni. Il grafico mostra le medie giornaliere del periodo selezionato, indicando sull'asse x i giorni.
- **Modalità annuale:** Si Attiva quando la differenza tra startDate ed endDate è maggiore di 31 giorni. Il grafico mostra le medie mensili nel periodo scelto, indicando sull'asse x i mesi.

3.2.5 SettingsScreen

Pagina che si raggiunge premendo la BottomNavigationBar all'interno del Drawer (descritto nel paragrafo 3.2.3). È relativa alle impostazioni del sistema. In questa versione vi è solamente la personalizzazione di scanInterval. La schermata è composta dall'AppBar con ai due lati il tasto indietro e impostazioni, mentre in centro la scritta "Settings". Oltre a questa è presente un Text che indica il valore che è permesso modificare (scanInterval). Dopo vi è un TextField che quando si apre la pagina conterrà già un valore. Si tratta del valore salvato nel plugin di Measurify. Infatti ogni volta che la schermata è caricata, viene inviata una richiesta GET al plugin di Measurify (richiesta vista nel paragrafo 2.1) che restituisce il suddetto valore. Infine vi è il tasto "Save", che l'utente deve premere quando ha finito la modifica del TextField. Una volta premuto, se il dato inserito è corretto (il caso contrario verrà notificato con un FlutterToast), ovvero è un intero non nullo, esso verrà salvato sul plugin di Measurify tramite una richiesta PUT (paragrafo 2.1).

3.2.6 AccountScreen

Schermata che si raggiunge premendo il tasto "Account" all'interno del Drawer (descritto nel paragrafo 3.2.3). Sono presenti due Text che hanno lo scopo di descrivere cosa vi è dentro le TextField sotto di loro (username e password). Queste ultime saranno già compilate quando la pagina verrà caricata. Si tratta dei valori salvati, nella memoria, al momento del log in. Vi è anche un tasto "Save". Tuttavia in questa versione non ha associato nessun comando, come i Text nel paragrafo 3.2.2.

4 Contributo personale e considerazioni conclusive

L'obiettivo di questa tesi era realizzare un sistema embedded per il monitoraggio di piante e fiori e sviluppare un'applicazione smartphone, in modo da poter rendere consultabili i dati anche da remoto.

Il sistema è composto da un microcontrollore, Arduino UNO Wifi Rev 2, un sensore di luminosità, SparkFun TSL2561, un sensore di umidità dell'aria e della temperatura, DHT22 Pro v1.3, e un sensore capacitivo di umidità del terreno, DFRobot SEN0193. Inoltre è stato adottato uno schermo Arduino TFT per mostrare le informazioni, in locale, dell'ultima misurazione.

I dati acquisiti dal sistema dovevano essere inviati ad un cloud, attraverso delle RESTful API. Queste ultime dovevano essere messe a disposizione da un framework open-source, chiamato Measurify. Grazie a questo i dati dovevano poter essere richiesti per essere utilizzati dall'applicazione smartphone.

Tutti i sensori e i dispositivi usati si sono rivelati funzionali e adatti all'uso richiesto.

Parlando di Measurify, usarlo è stato semplice e intuitivo. Senza alcun dubbio è un framework funzionale nell'ambito dell'IoT, merito anche del modello di risorsa implementato.

Per lo sviluppo dello sketch, che in seguito è stato caricato su Arduino, è stato usato l'IDE incluso con quest'ultimo. Come visto è stato molto importante l'utilizzo di librerie. Infatti sono state usate per la maggior parte dei dispositivi, collegati al microcontrollore, compresa anche la parte di connessione alla rete e quella del client. Questo ha aiutato e semplificato notevolmente la scrittura del codice.

Invece per quanto riguarda lo sviluppo dell'applicazione smartphone, è stato utilizzato un framework open-source, nominato Flutter e realizzato da Google. Il suo utilizzo è stato decisamente interessante, soprattutto perché permette di creare app native in grande velocità. Grazie anche al fatto che la parte riguardante il layout e quella del funzionamento sono racchiuse in un unico file in formato Dart. L'editor che è stato usato in questo caso è Visual Studio Code, prodotto da Microsoft.

Inoltre il progetto sviluppato, potrebbe ricevere, in futuro, delle migliorie. Una possibilità sarebbe quella di alimentare il sistema embedded attraverso una batteria, che potrebbe essere anche collegata a un piccolo pannello solare in modo da ricaricarsi durante il giorno. Così facendo, l'apparecchio non dipenderebbe più dalle prese elettriche e sarebbe in grado di monitorare le piante in zone dove l'elettricità non arriva.

Un'altra grande aggiunta che potrebbe essere fatta, è l'implementazione di un sistema di irrigazione. Questo comandato in remoto dall'applicazione, potrebbe agire in base alle misurazioni e alle preferenze dell'utente. In questo modo le coltivazioni sarebbero in grado di crescere sempre con la giusta quantità di acqua, senza il bisogno che l'utente si adoperi per dar loro da bere.

5 Riferimenti bibliografici

- [1] Arduino UNO Wifi Rev 2, <https://store.arduino.cc/arduino-uno-wiFi-rev2>
- [2] Measurify, <https://measurify.org/>
- [3] Sparkfun TSL2561, <https://www.sparkfun.com/products/12055>
- [4] TSL2561 Library Adafruit, https://github.com/adafruit/Adafruit_TSL2561
- [5] Grove DHT22, <https://www.seeedstudio.com/Grove-Temperature-Humidity-Sensor-Pro-AM2302-DHT22.html>
- [6] DHT Library Adafruit, <https://github.com/adafruit/DHT-sensor-library>
- [7] DFRobot SEN0193, https://wiki.dfrobot.com/Capacitive_Soil_Moisture_Sensor_SKU_SEN0193
- [8] Arduino TFT Screen, <https://www.arduino.cc/en/Guide/TFT>
- [9] TFT Library Arduino, <https://www.arduino.cc/en/Reference/TFTLibrary>
- [10] ArduinoJson Library, <https://arduinojson.org/>
- [11] WiFiNINA Library Arduino, <https://www.arduino.cc/en/Reference/WiFiNINA>