Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

PLANT ANALYZER: IMPLEMENTAZIONE DI UNA PIATTAFORMA IOT PER IL MONITORAGGIO E LA CONDIVISIONE DI DATI PER LA SMART AGRICULTURE

Relatore:
Dott.
Federico Montori

Presentata da:
Federico De Giorgio

II sessione - secondo appello
Anno Accademico 2020/2021

Sommario

L'utilizzo dell'internet of things ha permesso il monitoraggio costante di molti dati in real-time.
Scopo di questo progetto di tesi è offrire un'implementazione di una piattaforma che gestisca completamente il processo di *monitoring IoT*, dalla configurazione della scheda alla resa dei valori.
Oltre ad occuparsi di questo viene implementato un sistema di *accounting* e di *sharing* dei dati, in modo da fornire funzioni collaborative.
Si è deciso di svolgere le analisi su una pianta, vista la scarsa quantità di studi in questo ambito. Il codice è comunque facilmente modificabile per effettuare monitoraggi su altri settori.
La piattaforma acquisisce diversi dati importanti e può essere vista come un punto di partenza per servizi più complessi, ad esempio progetti che fanno uso di tecniche come *AI* o *Machine Learning*.
Data la mole di dati necessaria per questo tipo di integrazione, è stato scelto un approccio di tipo low-cost per la scelta della scheda *Iot* (Esp32) e delle componenti ad essa collegate.

Introduzione

Negli ultimi anni la tecnologia del cosiddetto "internet delle cose", sta diventando sempre piú predominante nello sviluppo di vari servizi : *Smart Cities* , *Connected Cars*, *Industrial IoT*, ma anche dispositivi per il fitness, gadget personali e tanto altro.
Questo progetto ha lo scopo di fornire un metodo per semplificare il monitoring di dati provenienti da dispositivi *IoT*, visualizzandoli tramite un app Android e sfruttando i servizi Firebase per lo storing di dati e per l'implementazione di un servizio di autenticazione in modo da avere funzioni collaborative tra gli utilizzatori.
Bisogna precisare che i dati degli utenti sono presi in modo anonimo e che l'unico modo di accedere ad essi è tramite la posizione su una mappa (solo per dati resi pubblici dagli utenti in fase di configurazione).
Non c'è quindi un utilizzo di dati personali degli utenti ma solo di quelli riguardanti i sensori ambientali ed anch'essi possono essere tenuti privati.
Questa piattaforma si occupa di monitorare i fattori di crescita di una pianta e di scattarle delle foto per verificarne effettivamente lo stato di salute.
Il tipo di lavoro fatto potrebbe essere applicato a molti altri ambiti in cui è richiesto un monitoraggio di dati real-time.
La scelta del settore in cui ricade questo lavoro è stata fatta poiché, viste le peculiarità del periodo in cui viviamo, la *Smart Agriculture* diventerà necessaria se non vitale per sopravvivere al cambiamento climatico, compensando la carenza di cibo che ne sarà conseguente. Si è cercato quindi di offrire un servizio che aiuti il monitoraggio delle piante da remoto, migliorando così la

produttività a fronte di un migliore controllo dei dati.
La combinazione tra rilevamenti dei sensori e foto scattate (dati inviati dal dispositivo *Iot*), assieme alla posizione in cui si trova la pianta, permette l'integrazione con molti progetti già esistenti per creare una piattaforma che possa risolvere problematiche più complesse.
Vediamo ora come è stato strutturato questo lavoro facendo una breve introduzione sulle tecnologie che fungono da nucleo del progetto, seguite da una breve panoramica e dai dettagli implementativi.
Infine verranno trattati i casi d'uso per l'applicativo allo stato attuale, e come potrebbe essere esteso in futuro.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Stato dell'arte

In questo primo capitolo viene presentato un resoconto sulle tecnologie riguardanti questo progetto, in particolare il *monitoring* di dati *IoT* e la *Smart Agriculture*.
In seguito verranno presentati dei progetti simili a questo, che sono stati presi come riferimento nello sviluppo.

1.1 Internet Of Things

In un articolo italiano si parla di *Iot* descrivendolo così: "L'internet delle cose (Internet of Things) è il nome dato alla crescente tendenza di aggiungere sensing e communication capabilities agli oggetti di casa/industria per renderne possibile monitoraggio e gestione in remoto."[1]
Giá da questa definizione si evince l'importanza del monitoraggio dei dati nell'utilizzo di dispositivi *Iot*.
Il *monitoring* é la parte su cui si concentra questo progetto, che gestisce e semplifica il processo all'utente, consentendo la visualizzazione dei dati trasmessi direttamente su dispositivo Android.

Estendere ulteriormente il servizio di monitoraggio *Iot* tramite Android permette :

1. Una facile visualizzazione dei dati
2. Possibilitá di associare un'account al dispositivo *Iot*
3. Espandere la rilevazione dati con sensori dello smartphone (posizione, giroscopio ecc..)
4. Facilitare il setup del dispositivo *Iot* direttamente da app

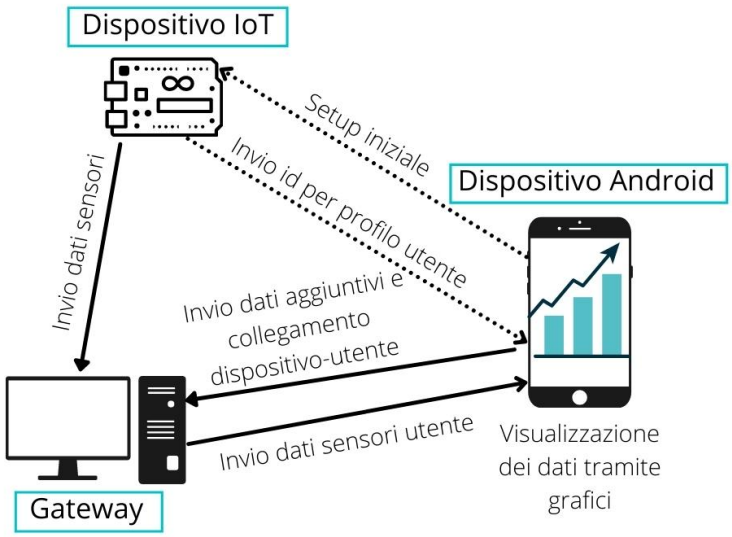

Figura 1.1: Esempio di espansione android di *Iot*

Il maggior vantaggio dell'introdurre questo tipo di approccio consiste nello svoglere le operazioni piú pesanti su dispositivo android (o dividere la mole di lavoro tra smartphone e gateway) consentendo cosí di sviluppare servizi abbastanza complessi con hardware estremamente limitati come quelli dei dispositivi *Iot*.
Inoltre la modularitá di questo approccio permette di creare diversi progetti Android o Desktop a partire dallo stesso codice di monitoring, e/o di estendere progetti esistenti aggiungendo nuove funzionalitá.

1.1.1 Casi d'uso

L' *internet of things* é una tecnologia abbastanza recente con molteplici ed eterogenei campi di applicabilitá.

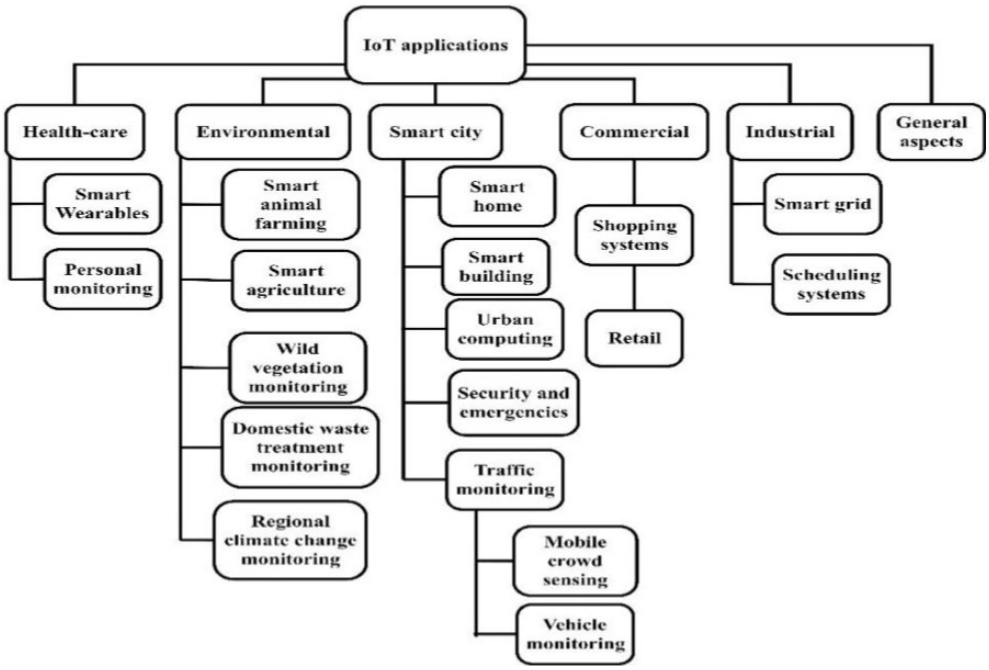

Figura 1.2: Ambiti di utilizzo *Iot* [2]

Vediamo ora in dettaglio in quali ambiti si é concentrato lo sviluppo e la ricerca negli ultimi anni e quali sono gli elementi piú rilevanti da valutare nello sviluppo di un'applicazione *Iot*.

Distribuzione casi d'uso

Come si nota dalla figura, gli ambiti in cui la tecnologia dell'internet of things viene utilizzato maggiormente sono *Smart Cities* ed *Health-care*, mentre quelli in cui é meno applicato sono *Industry* ed *Enviroment*.
Il settore in cui rientra questo progetto, la *Smart Agriculture*, fa parte dell'ambito *Enviroment* (Sviluppo) come viene mostrato in fig 1.2.
Lo scarso volume di applicazioni presenti su questo tema e la necessitá di

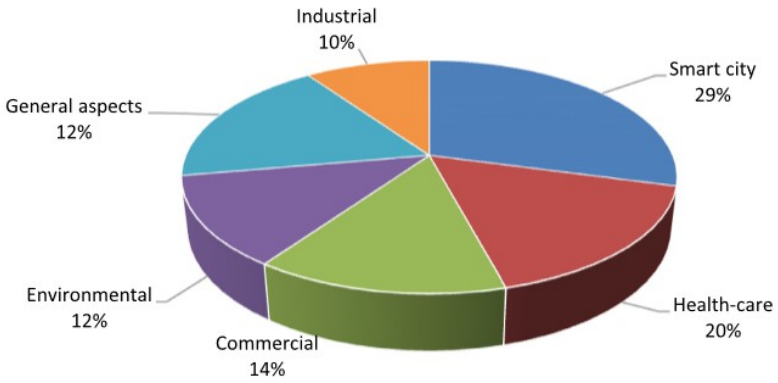

Figura 1.3: Percentuale di applicazioni *Iot* per ambito [2]

sistemi per fronteggiare gli sbalzi climatici imminenti, preservando l'agricoltura, rendono importanti la nascita di progetti come questo e la volontà di approfondire temi di ricerca riguardanti l'argomento.

Fattori da valutare per lo sviluppo di un applicazione Iot

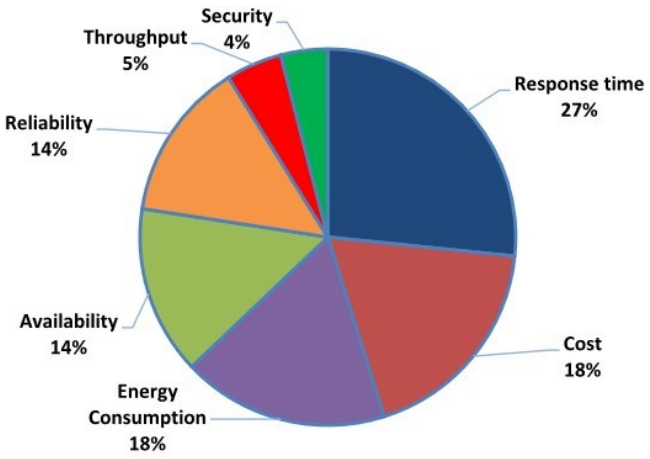

Figura 1.4: Fattori piú valutati per sviluppo applicazioni *Iot* [2]

Si nota che i fattori da tenere in maggior considerazione sono il tempo di risposta, il costo ed il consumo energetico.

Questo progetto si concentra su questi tre elementi :

- sensori e scheda basso **costo**;
- **consumo energetico** basso grazie a protocolli appositi come BTLE e MQTT;
- i dati sono trasmessi real-time al dispositivo android minimizzando il **response time**.

1.1.2 Distribuzione Tecnologia IoT nel mondo

Gli apparecchi *IoT* attualmente connessi nel mondo sono all'incirca 10 miliardi, e si pensa che entro il 2030 potrebbero piú che raddoppiare.

Anno	*Dispositivi connessi* (*miliardi*)
2019	7.74
2020	8.74
2021	10.07
2025	16.44
2030	25.44

Tabella 1.1: Numero di dispositivi *Iot* connessi al mondo [3]

In questa sezione viene esposta la diffusione di dispositivi *Iot* nel mondo e quali Paesi investono maggiormente in questa tecnologia.

La figura 1.5 mostra come nel 2014 i Paesi che maggiormente utilizzavano la tecnologia erano quasi solo europei e nord americani, (ad esclusione della Corea).
L'Italia si piazzava non troppo in alto tra le Nazioni europee, mentre alcuni tra i Paesi più all'avanguardia non rientravano in questa classifica.
Negli ultimi anni peró l'investimento da parte delle Nazioni che non facevano parte della lista degli Stati in cui questa tecnologia é maggiormente utilizzata

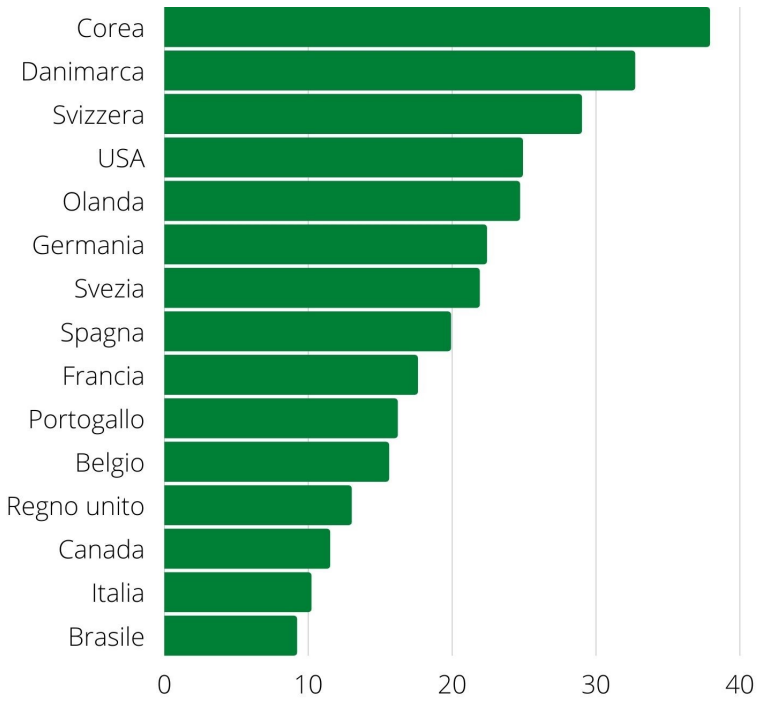

Figura 1.5: Dispositivi *Iot* connessi ogni 100 persone (2014)

é stato cospicuo.
Uno dei Paesi che piú ha investito in questo settore negli ultimi anni é stata la Cina, mente l'Italia ha superato alcuni Stati europei diventando il quarto Paese per mole d'investimento nell'*Iot* in Europa nel 2019.
Questi dati rendono chiaro il fatto che ogni Nazione sviluppata sta investendo nell' Internet of Things, inoltre mentre dove si é scelta questa direzione fin da subito i costi legati allo sviluppo si sono stabilizzati.
Gli Stati che sono rimasti indietro stanno facendo un ulteriore sforzo in questa direzione per mettersi al pari degli altri.

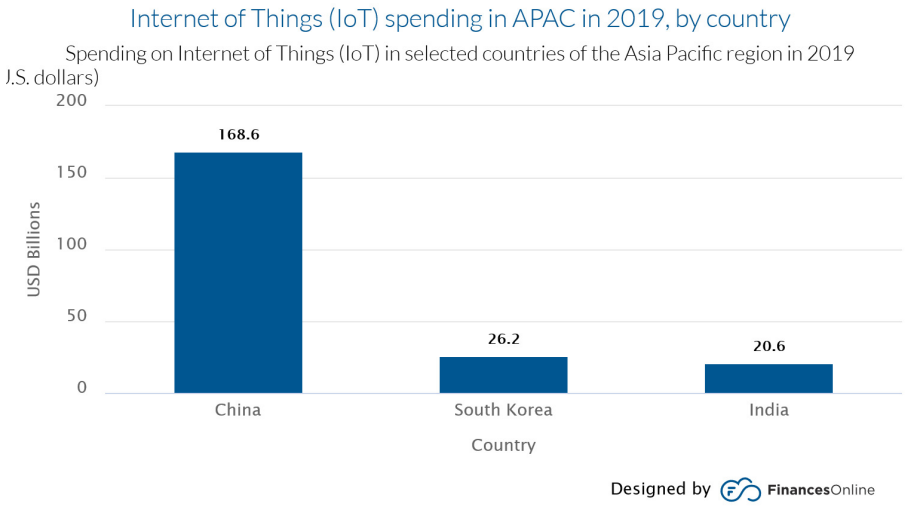

Figura 1.6: Paesi asiatici che hanno investito maggiormente nell'*Iot* (2019) [4]

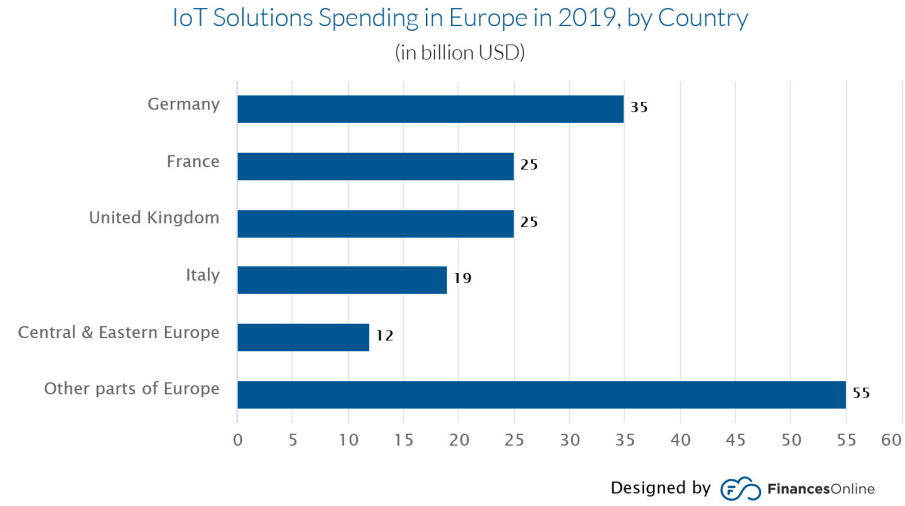

Figura 1.7: Paesi europei che hanno investito maggiormente nell'*Iot* (2019) [4]

1.2 Smart Agriculture

L'agricoltura é considerata come la base per la vita umana, poichè rappresenta il metodo principale per produrre cibo ed altri beni primari.
Negli ultimi anni il fabbisogno mondiale é aumentato e con l'inizio della crisi climatica, anche il settore agricolo é in crisi e non riesce a produrre le stesse quantitá di prodotti degli anni precedenti.
Come si puó aumentare la produttivitá, come richiesto, a fronte di un cosí difficile scenario quale la crisi ambientale?
La *Smart Agriculture* si propone di semplificare l'intero processo produttivo e di sfruttare al massimo le risorse, permettendo di ottenere risultati migliori e quindi piú produzione anche in situazione non ottimali.
L'*IoT* si integra alla perfezione con la *Smart Agriculture*, permettendo il monitoraggio dei dati delle piante ma anche il controllo di macchinari (ad esempio l'irrigazione) al verificarsi di determinati eventi (ad esempio una scarsa umiditá del terreno).
In molti articoli di ricerca contemporanei, questa tecnologia viene vista come una possibile soluzione alle problematiche sopra citate: "The implementation of *Iot* technologies in agriculture can certainly help to secure sufficient food demands and increase the efficiency of agricultural production processes in general."[5]
In un approccio tradizionale il contadino deve raccogliere manualmente i valori di ogni pianta e compiere delle azioni in conseguenza a ció che ha rilevato, in caso sia necessario.
Questo obbliga l'agricoltore a compiere ripetute analisi sul campo, anche quando non servirebbe compiere nessuna operazione.
Grazie al monitoraggio IoT, invece, l'operatore riceve i dati delle sue piante *real-time*, avendo l'opportunità di recarsi sul luogo di lavoro solo nei momenti indispensabili, ma con un controllo maggiore delle colture e quindi presumibilmente otterrà una resa finale migliore.
Inoltre, automatizzando alcune procedure (come l'irrigazione), un unico individuo riuscirebbe a gestire un numero molto più grande di piante coltivate

senza comprometterne la qualità, in quanto il processo di crescita verrebbe controllato seguendo i dati delle piante stesse.

1.2.1 Monitoring nella Smart Agriculture

Il monitoraggio *IoT* nell'ambito dell'agricoltura smart si concentra sui seguenti fattori:

- Umiditá
- Temperatura
- Luce
- Analisi del suolo (Umidità del terreno, Ph)
- Analisi dell'aria (Vento, Analisi meteo)
- Immagini pianta

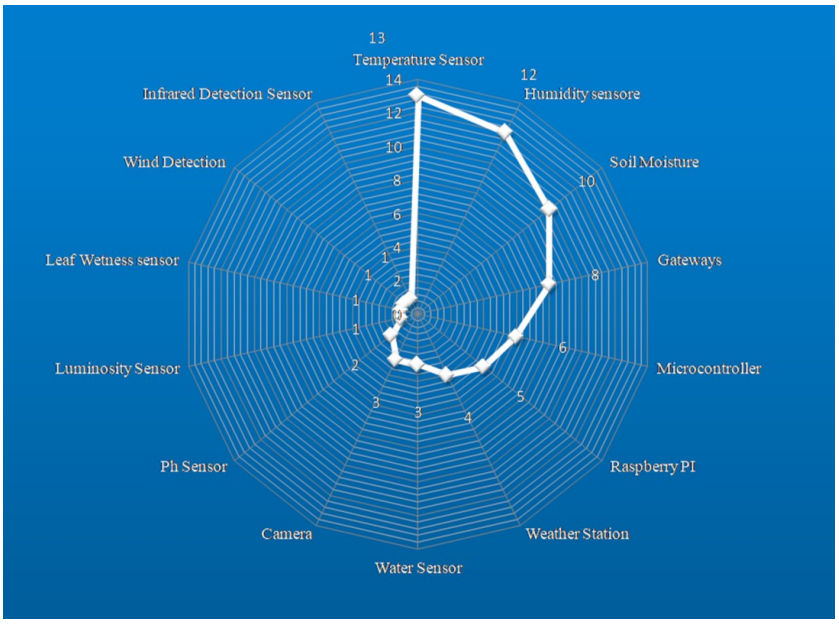

Figura 1.8: Distribuzione sensori e device Smart Agriculture [6]

Questo progetto rileva i valori di Moisture, Luminsoitá, Temperatura, Umiditá e li mostra all'utente sommandovi le foto della pianta.
Oltre a ciò rende fruibili i dati di altri utenti, rendendo possibili confronti con i propri.
Questo consente di migliorare il processo di crescita delle piante "imparando" dai metodi degli altri agricoltori.

1.2.2 Protocolli IoT nella Smart Agriculture

I dispositivi *IoT* necessitano di protocolli di comunicazione per notificare i rilevamenti fatti.
Viste le caratteristiche dei luoghi che ospitano le piante, i protocolli di comunicazione a basso raggio, come ad esempio il Bluetooth, sono poco utilizzati in quest'ambito.
La maggior parte dei dispositivi *Iot* si connette ad Internet tramite Wifi.
Oltre che per l'accesso alla rete, serve un protocollo per l'invio dei dati. Per questo compito il piú famoso é sicuramente l'*HTTP*, ma in quest'ambito ne esistono diversi. Ció é dovuto alla necessitá di un estremo risparmio energetico ed alla scarsa capacitá di calcolo di molti dispositivi *IoT*.
Per questo progetto, come protocollo di invio dei dati é stato scelto, *MQTT* (Message Queue Telemetry Transport Protocol), mentre l'accesso ad internet avviene tramite Wifi, anche se ssid e password della rete sono settati tramite Bluethooth dal dispositivo Android.
La scelta di *MQTT* serve proprio per sopperire al scarsa capacitá di calcolo della scheda utilizzata; infatti l'invio di mesaggi MQTT presenta meno overhead rispetto a quello HTTP: "Like Hypertext Transfer Protocol (HTTP), MQTT relies on Transmission Control Protocol (TCP) and IP as its underlying layers. However, compared to HTTP, MQTT is designed to have a lower protocol overhead."[7]

Proprietà	MQTT	HTTP
Architettura	Client/Broker	Client/Server
Astrazione	Publish/Subscribe	Request/Response
Dimensione Messaggio	Piccolo e non definito (fino a 256 MB)	Grande e non definito
Affidabilità/ Qualità del servizio (QoS)	QoS 0 - At most once (Fire-and-Forget), QoS 1 - At least once, QoS 2 - Exactly once	Limitata (dal protocollo di trasferimento - TCP)
Protocollo di trasferimento	TCP/UDP	TCP
Formato di codifica	Binario	Testo

Tabella 1.2: Differenza tra MQTT e HTTP

Come si può notare dalla tabella, il protocollo *HTTP* può inviare messaggi più grandi ed in aggiunta il formato testuale appesantisce il pacchetto, rispetto al formato binario dei pacchetti *MQTT*.

1.3 Progetti simili

1.3.1 Enhancement of Plant Monitoring Using IoT

Questo progetto [8] si pone come unico obiettivo il monitoraggio dei principali dati vitali di una pianta e la notifica su dispositivo Android.
Nel suddetto lavoro, come in questa tesi, vengono usati sensori low-cost, mentre la connessione ad internet avviene tramite modulo GSM.
Viene ipotizzato anche il collegamento con una pompa d'acqua che andrebbe ad irrigare la pianta quando la Moisture scende sotto una certa soglia.
Questo articolo é stato d'ispirazione per la realizzazione di questo elaborato,

anche se é un assunto teorico e non tratta di alcuna implementazione.
Non é chiaro infatti come avvenga lo scambio di informazioni tra il dispositivo *IoT* (Arduino) e quello Android.
Per di più la mancanza di immagini della pianta non consente di avere una maggiore sicurezza sul suo stato di salute.

1.3.2 An IoT-Based Smart Plant Monitoring System

Anche questo lavoro [9] si pone come obbiettivo quello di sviluppare un sistema low-cost per innaffiare una pianta quando viene rilevata una bassa umiditá del terreno.
La visualizzazione dei dati é ancora implementata tramite app Android ed il trasferimento dei dati avviene tramite MQTT e cloud service.

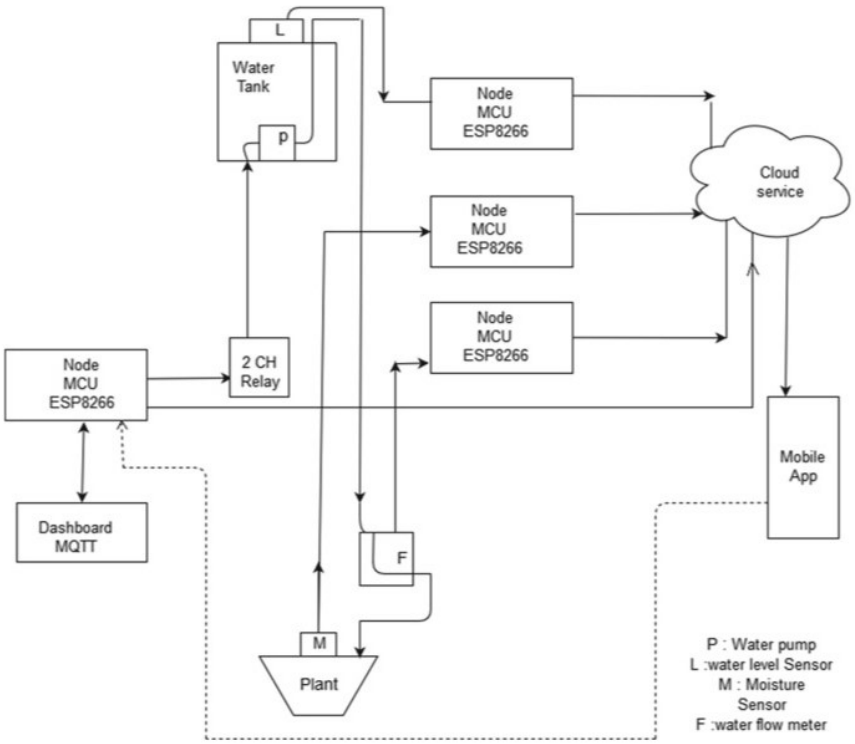

Figura 1.9: An *Iot*-Based Smart Plant Monitoring System [9]

Per quanto i servizi offerti da "An *Iot*-Based Smart Plant Monitoring System" e quelli offerti da questa tesi é possibile trovare molti elementi affini per quanto siano due progetti abbastanza diversi :

- Monitoring di un valore della pianta
- Visualizzazione su dispositivo Android
- MQTT per invio dati da dispositivo *Iot*
- Cloud Service per scambio dati con Android

Lo scheletro utilizzato per lo scambio di dati é servito da riferimento per la progettazione di questo elaborato.
Si é cercato di focalizzarsi su un monitoring piú dettagliato piuttosto che sul controllo di azioni in determinati eventi (sistema di irrigazione).

1.3.3 A Novel Approach to *Iot* Based Plant Health Monitoring System

Questo studio [10] é quello che piú si avvicina a questa tesi, in termini di servizi offerti. Infatti si propone di offrire una dashboard con i dati di vari sensori collegati alla pianta, sempre basandosi su un approccio low-cost.

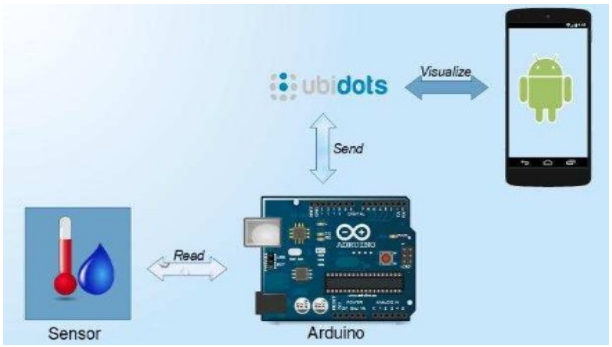

Figura 1.10: A Novel Approach to Iot Based Plant Health Monitoring System[10]

La visualizzazione dei dati é sempre gestita su dispositivo Android, mentre l'invio dei dati é gestito da Ubidots.
La connessione alla rete avviene tramite Ethernet, quindi non vi é la necessitá di configurare ssid e password della rete.
La visualizzazione dei dati ed il loro scambio sono estremamente facilitati da Ubidots; manca peró una qualsiasi visualizzazione di foto.

1.4 Punti di forza di questo progetto

Dopo aver analizzato altri tre progetti, che sono stati fonte d'ispirazione, vengono qui esposti gli aspetti che dovrebbero conferire originalitá e valore aggiunto a questo elaborato, rispetto ai lavori precedenti.
Innanzitutto la presenza di foto associata ai valori di moisture, luminositá, umiditá e temperatura aggiunge un controllo effettivo sulla salute della pianta e offre un ampio spettro di monitoraggio, consentendo una precisa analisi da smartphone.
Bisogna anche considerare che avere delle immagini della pianta offre vari spunti di estensione del sistema, come ad esempio l'utilizzo di machine learning per individuare la specie e capire i valori medi raccomandati per quel preciso vegetale.
Rispetto ai progetti che ottengono accesso alla rete tramite Ethernet o Gsm, l'utilizzo del Wifi richiede un setup iniziale dove vengono specificati il nome della rete a cui connettersi e la sua password.
Questo viene gestito completamente dall'app in fase di configurazione del dispositivo IoT tramite BLE.
Viene anche data la possibilitá all'utente di collegare piú dispositivi all'app, offrendo cosí una soluzione compatta per il monitoraggio di piú piante dallo stesso dispositivo.
La cosa che peró rende piú originale questo progetto é la possibilitá di condividere i propri dati e visualizzare quelli degli altri utenti del sistema, su una mappa.

Anche questo fattore si presta allo sviluppo di molteplici servizi, quali la ricerca del miglior sito per la coltivazione di determinate specie di piante, o la ricerca dei migliori fattori ambientali facendo una media tra i valori degli utenti che hanno ottenuto ottimi risultati.
La combinazione tra sistema di autenticazione e mappa si presta anche allo sviluppo di una sorta di social network, implementando la possibilitá per gli utenti di interagire tra loro tramite messaggistica o condivisione piú accurata dei dati.
Tutte queste opzioni future sono esplicitate nel capitolo 4.

Capitolo 2

Panoramica del progetto

Dopo aver fatto un'introduzione sulle tecnologie utilizzate in questo progetto e il loro sviluppo allo stato attuale, viene ora presentata una panoramica di questo lavoro.
Innanzitutto viene esposto il flusso del funzionamento della piattaforma, spiegandone brevemente le varie parti.
Vengono poi trattate le fasi iniziali di progettazione quali l'assemblaggio della scheda Esp32 e l'integrazione dei servizi Firebase (*Authentication*, *Realtime Databse* e *CloudStorage*).
I dettagli implementativi delle varie componenti che compongono la piattaforma sono riportati nel capitolo successivo.

2.1 Flusso della piattaforma

Il servizio viene implementato mediante tre sotto-progetti che interagiscono tra di loro :

- Sketch su Esp32 (dispositivo *Iot*);
- Applicazione Android;
- Script nodeJs per scambio dati;

L'interazione tra queste componenti realizza una piattaforma che gestisce l'intero ciclo di vita del *monitoring* della pianta, dalla configurazione della scheda, allo scambio dati, fino alla visualizzazione degli stessi tramite grafici. In aggiunta offre un meccanismo di autenticazione e la possibilità di condividere i propri dati e visualizzare quelli di altri utenti su una mappa.

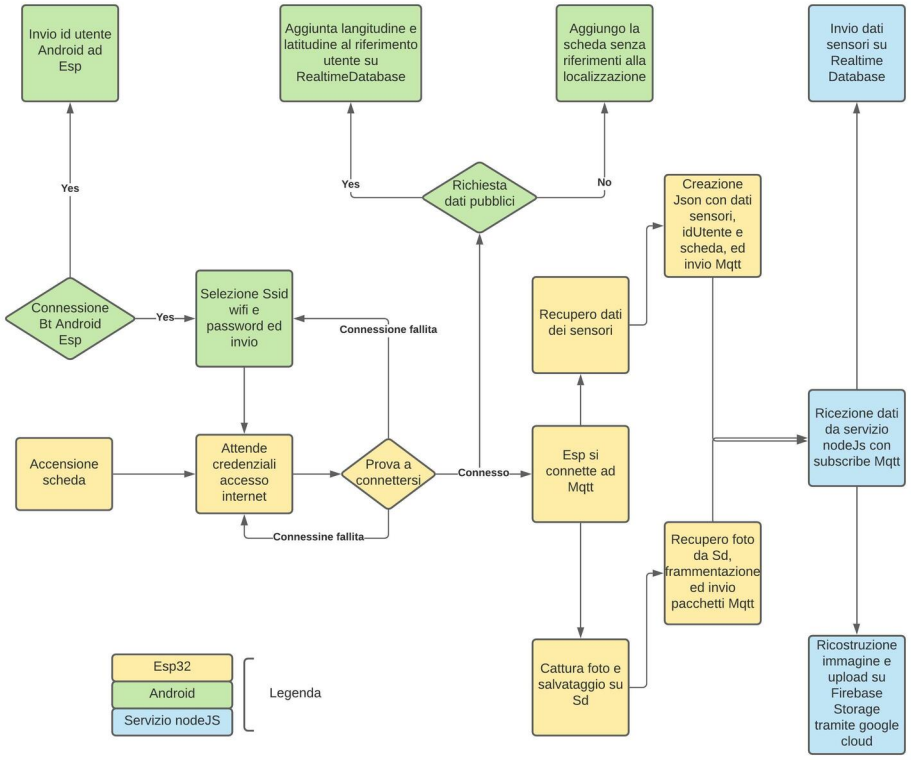

Figura 2.1: Diagramma di flusso generale del progetto

2.1.1 Configurazione iniziale

Come si può notare dalla figura 2.1, quando Esp viene accesa, resta in attesa delle credenziali per la connessione alla rete Wifi.
Questo avviene tramite dispositivo Android utilizzando il protocollo *Bluetooth Low Energy*; al momento della connessione l'applicazione invia l'id

utente con cui si è eseguito l'accesso ad essa, aggiungendo in seguito anche il nome della rete e la password Wifi, selezionate tramite un'apposita schermata che esegue lo scan delle reti disponibili.
Se la scheda si connette ad internet, l'utente viene notificato e gli viene chiesto come chiamare la scheda appena connessa (il nome con il quale verrà identificata nell'app) e se rendere pubblici i suoi dati pubblicando quindi la posizione dell'utente su db in modo da essere identificato.
In caso di connessione non riuscita viene rilanciata la selezione della rete ed il processo riparte.

2.1.2 Invio dati

Successivamente alla riuscita connessione ad internet, la scheda si collega al broker MQTT, inizia a prendere i dati dei sensori e, se necessario, catturare l'immagine (la foto non viene presa ad ogni rilevamento).
Dopo di che pubblica un Json con i dati dei sensori. In caso di rilevamento dell'immagine la segmenta in chunk ed invia ogni segmento sempre in un Json.

2.1.3 Gestione dati

I dati, come già detto, vengono pubblicati tramite il protocollo MQTT, su un canale riservato per i sensori ed un altro per le immagini.
Un servizio nodeJs, in esecuzione su una macchina esterna, intercetta entrambi i tipi di dato (in formato Json) e li inserisce nel database Firebase al quale è collegata l'applicazione Android la quale recupera così i dati.
Il servizio stabilisce il giusto percorso dove inserire i dati dal Json stesso, tramite *idScheda*e *idUtente*; l'immagine, invece, viene ricostruita ed aggiunta allo Storage Firebase.
In entrambi i casi il dato è arricchito dal timestamp, acquisito dal servizio prima di inviare i dati.

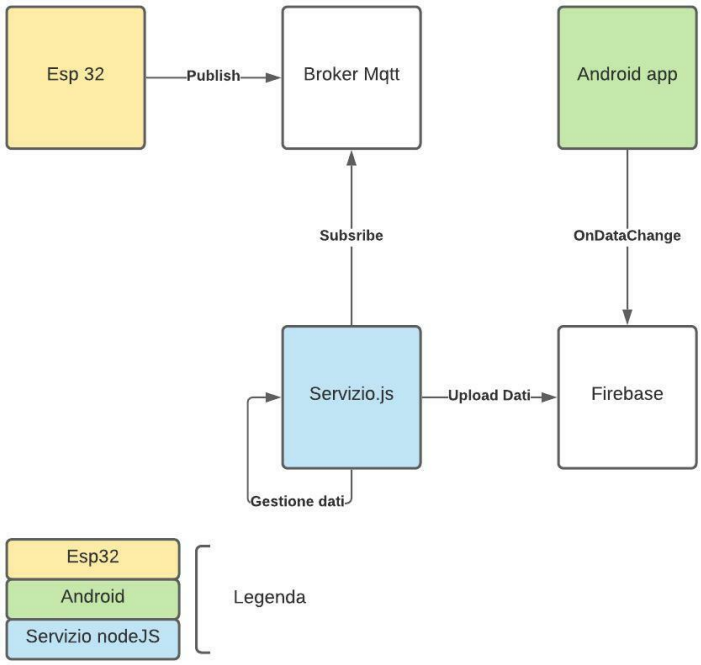

Figura 2.2: Diagramma di flusso dello scambio di dati

2.1.4 Visualizzazione dati

Con i riferimenti di nome utente ed identificatore scheda, vengono richiesti i dati dell'utente da database Firebase (vengono visualizzate tutte le Esp32 associate a quello smartphone).
A questo punto i dati sono suddivisi per tipo di sensore e visualizzati tramite grafici. Inoltre è presente una lista di foto visualizzabili, ordinate per data di acquisizione.
Infine una funzione dell'app permette di caricare una mappa di Google Maps, centrata sulla posizione dell'utente, che visualizza come marker tutte le schede che hanno i propri dati in condivisione.
Toccando gli indicatori corrispondenti, è possibile visualizzare gli ultimi rilevamenti di dati e l'ultima foto della pianta.

2.2 Assemblaggio Esp32

Come anticipato in precedenza la scheda utilizzata per questo progetto è Esp32, una board low-cost dotata sia di modulo Wifi che Bluetooth (BTLE).

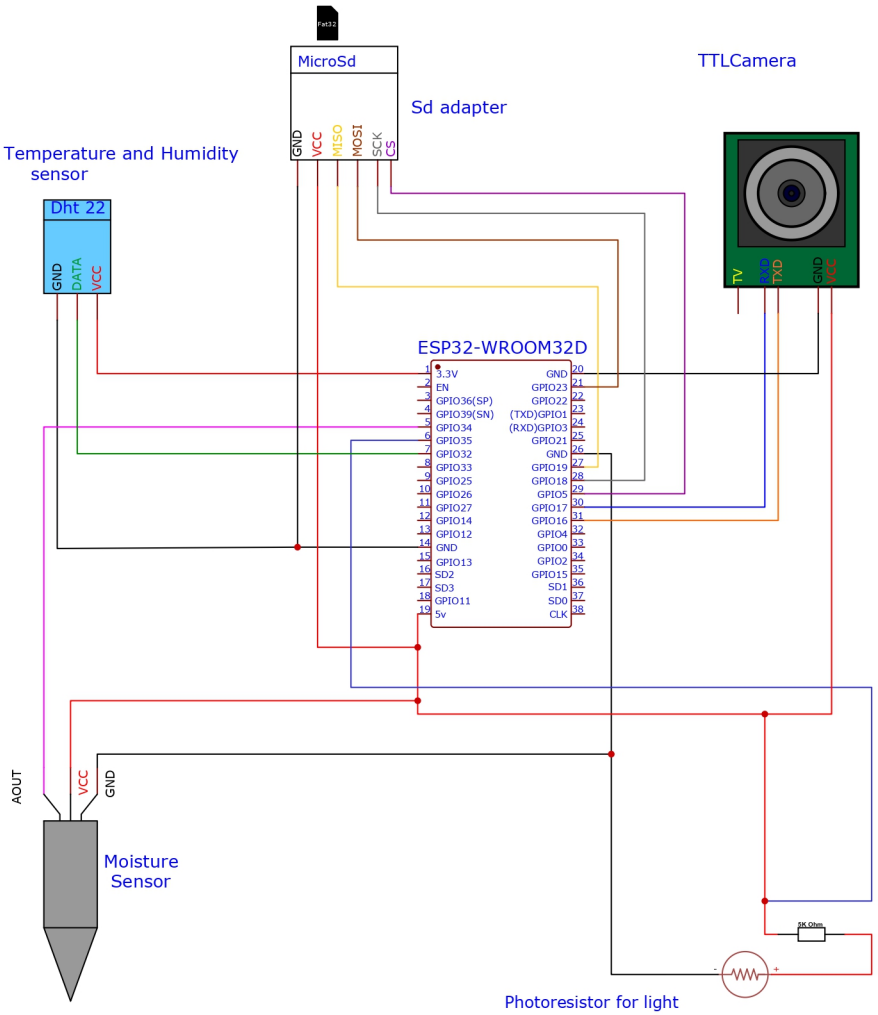

Figura 2.3: Struttura Esp32

Per il monitoring dei dati la scheda è stata assemblata con sensori sia analogici che digitali, oltre ad una camera seriale ed un adattatore per la scheda Sd dove salvare le immagini della Cam, in quanto non è possibile salvare dei

file .jpg in Esp32 (eccedono la capacità di memoria della scheda).
La rilevazione della luce è presa tramite un fotoresistore; la conversione in luminosità, così come il dettaglio dello sketch caricato su Esp, saranno trattati nel prossimo capitolo.
Essendo un lavoro sperimentale, servono ancora alcune piccole modifiche per essere usato, soprattutto in outdoor.
La scheda al momento è alimentata tramite cavo; andrebbe invece usato un modulo con batteria in modo da poter utilizzare la scheda anche dove non è presente una presa elettrica.
Anche la camera andrebbe cambiata a favore di una impermeabile, sempre per poter usare il dispositivo all'aperto.
Infine, mentre ora viene utilizzata semplicemente una breadboard per gestire il wiring delle componenti, servirà una struttura sulla quale rendere compatto il macchinario con tutti i sensori, che permetta di averli nel posto giusto (ad esempio luce e camera sopra la pianta e moisture nel terreno).

2.2.1 Utilizzo di Esp32

É utile osservare come sono configurati i pin di Esp32 per capire le scelte progettuali prese nella fase di assemblaggio della scheda.

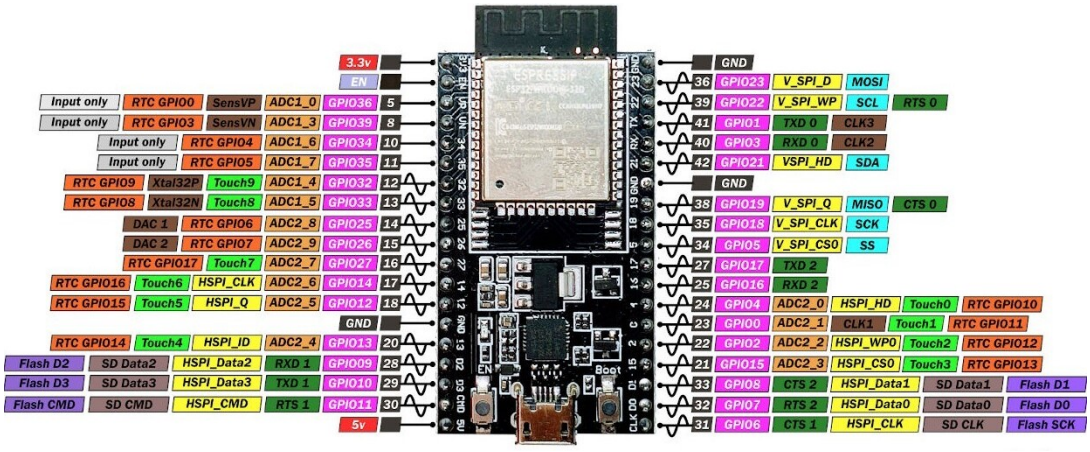

Figura 2.4: Esp32 Pinout [11]

In figura 2.4 si può notare come i pin di input-output (quelli con l'etichetta arancione ADC) siano limitati e divisi in due canali.
I pin con ADC2 appartengono al secondo canale e non possono essere usati in contemporanea all'utilizzo del Wifi (possono avere delle interferenze e restituire valori sbagliati); quindi per i risultati dei sensori sono stati utilizzati solo quelli appartenenti al canale ADC1.
La scheda è sfruttata in modo massimale:

- non si possono aggiungere altri sensori poiché i pin sono praticamente tutti occupati (ADC1);
- lo sketch occupa tutta la memoria della scheda, pur avendo utilizzato un partizionamento che massimizza lo spazio per il programma;
- il consumo energetico è al limite, infatti aggiungendo altre componenti che necessitano alimentazione si avrebbe bisogno di un power supply.

2.3 Firebase

Un ottima definizione di cosa rappresenta servizio da un studio che ne fa uso: "Firebase is considered as web application platform. It helps developers builds high-quality apps. It stores the data in JavaScript Object Notation (JSON) format which doesn't use query for inserting, updating, deleting or adding data to it. It is the backend of a system that is used as a database for storing data."[12]
In questa piattaforma, Firebase si occupa della gestione degli account degli utenti dell'app e dello storing dei dati.
Vediamo ora come vengono utilizzati i servizi Firebase e con quale struttura vengono immagazzinati i dati in modo da poter essere correttamente gestiti dalle varie componenti della piattaforma.

2.3.1 Firebase Authentication

Un aspetto interessante di Firebase, molto utile per Android, è l'offerta di un servizio per l'autenticazione in modo semplice e veloce che permette agli sviluppatori di loggare gli utenti tramite diversi metodi come ad esempio Google, Facebook o Twitter.
Inoltre anche gli aspetti di sicurezza ed alcune funzioni, come ad esempio il recupero o la modifica della password, sono gestiti dal servizio stesso.
In particolare per questo lavoro è stato usato come metodo di log-in quello tramite email e password.
Il log-in consente di avere un identificativo univoco per ogni utente (generato da Firebase) in modo da poter associare ad esso i dati della scheda che ha configurato. Questo id, infatti, assieme a quello della scheda stessa, costituiscono la coppia di valori per poter accedere al giusto *path* del Realtime Database e dello Storage, come vedremo a breve.

2.3.2 Firebase Real-Time Database

Proprietà	**Firebase**	**SQL**
Data Storage	Albero JSON	Modello relazionale (tabelle)
Flessibilità dello schema	Schema dinamico, i dati possono essere modificati in ogni momento	Schema fisso. Alterarlo porta ad essere temporaneamente offline
Specialità	Dati con tipo e struttura non definiti	Dati di cui la struttura è conosciuta a priori
Tecnica di accesso ai dati	Sincronizzazione dei dati	Query

Tabella 2.1: Differenza tra Db SQL e RealtimeDatabase

Realtime Database è stato il primo prodotto ad essere sviluppato da Firebase; dunque è il servizio più stabile e consolidato di tutta la Piattaforma. Il servizio è un Cloud Storage NoSQL che può essere collegato con vari tipi di progetto, per fornire accesso in tempo reale ai dati.
Il Database, come visto in tabella, è strutturato come un'enorme albero Json. Analizziamo ora come è strutturato quello associato a questo progetto:

```
"Root":{
    "$userId" : {
        "$boardId" : {
            "$timestamp" : {
            "boardId" : 246f28969298,
            "humidity" : 55.8,
            "lux" : 74,
            "moisture" : 53,
            "temperature" : 17.8,
            "timestamp" : 1636110567232,
            "userId" : KSHwKdsgiNe5kgh1eKTXJZHGxxq1
            },
            "$timestamp" : {
            ..........
            Values of another detection for the same
                board/user
            }
        },
        "$boardId" : {
        .........
        Values of another board of the same user
        }
    },
    "$userId" : {
```

```
        .........
        Values of another user
        },
        "publicLocation" : {
            "$boardId" : {
                "boardId" : 246f28969298,
                "latitude" : 44.4900701,
                "longitude" : 11.2972809,
                "userId" : KSHwKdsgiNe5kgh1eKTXJZHGxxq1
            },
            "$boardId" : {
            ..........
            Location of another board
            }
        }
}
```

Innanzitutto è presente un elenco di *userId* (presi dall'authentication), i cui figli (le *boardId*) rappresentano le schede connesse a quell'utente.
All'interno di esse, per ogni momento in cui viene fatta una rilevazione si aggiunge un figlio timestamp (sempre univoco per quella scheda) al cui interno sono contenuti i dati dei sensori e di nuovo gli stessi identificatori di scheda e utente.
Oltre ai *path* degli utilizzatori contenenti i valori dei rispettivi dispositivi, è presente un campo *publicLocation* dove vengono salvati i riferimenti e le posizioni delle schede i cui utenti hanno acconsentito di pubblicare i dati.

2.3.3 Firebase Cloud Storage

Come scritto sul sito di Firebase: "Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads

and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content. On the server, you can use Google Cloud Storage APIs to access the same files."[13]
Cloud Storage è un servizio che permette la gestione di file di grandi dimensioni come immagini e video; gli stessi file sono poi accessibili mediante Google Cloud Storage.
Le foto prese da Esp32 sono caricate dal servizio Nodejs accedendo ad un *bucket Google*, sempre usando *idUtente*, *idScheda* e timestamp come identificatori univoci, con cui viene creato il path d'inserimento ed il nome del file.
Le immagini sono poi riprese dal dispositivo Android mediante Firebase Storage.

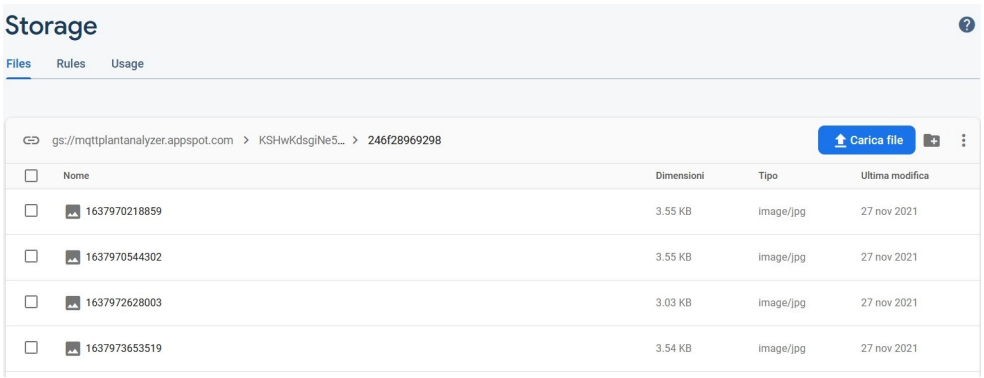

Figura 2.5: Cloud Storage del progetto (cartella della scheda "test1")

Capitolo 3

Implementazione

In questo capitolo andremo ad analizzare i dettagli implementativi delle diverse parti del sistema.
Facendo riferimento al flusso totale della piattaforma, verranno analizzate le varie fasi per ognuna delle tre componenti :

- Sketch Esp32
- Script NodeJs
- Applicazione Android

I vari punti del flusso verranno esposti per ogni applicativo che ne fa parte, senza soffermarsi troppo sull'interazione tra di essi.
Questa modalità di esposizione rende meno chiaro il comportamento generale del progetto.
Per quest'aspetto fare riferimento alla sezione "Flusso della piattaforma".

3.1 Sketch Esp32

Dopo aver discusso della fase di assemblaggio di Esp32 con i vari sensori utili, andiamo ora ad analizzare il programma (sketch) che viene eseguito al suo interno.

Innanzitutto verrà trattata la fase di configurazione che si conclude con la fine della funzione "setup()" del codice.
Verrà poi descritto il funzionamento del "loop()" (la parte di sketch eseguita ad ogni intervallo di tempo).
É bene precisare che, per la compilazione dello sketch sulla scheda, è stata usata come opzione di partizionamento "minimal spiff" cioè la configurazione che conserva meno memoria per lo storing di dati.
Questo perchè alcune tra le librerie incluse sono molto grandi (ad esempio quelle per BLE) e perchè questo svantaggio viene compenasto dalla presenza della scheda Sd.

3.1.1 Configurazione iniziale

Inizialmente vengono settati alcuni valori costanti come ad esempio il pin di output del sensore Dht22, le porte seriali per la connessione alla camera, Service-Uuid e Characteristic-Uuid per BLE, il WifiClient ed il PubSubClient per la connessione al broker MQTT (senza le credenziali per la connessione ad internet).
Vengono poi definite le *callback* per il servizio BLE. In particolare il dispositivo funge da server e client; vengono quindi implementate le funzioni nel caso in cui un dispositivo si connette ad Esp (server Bluetooth Low Energy), e nel caso sia connesso, per l'invio e la ricezione delle *characteristics* sul suo codice *Uuid* (il codice Uuid serve per la connessione ed identificazione tra dispositivi BLE, è lo stesso sul quale si connetterà l'applicazione Android o qualsiasi dispositivo che voglia comunicare con Esp32 tramite Bluetooth).

```
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
```

```
};
class MyCallbacks: public BLECharacteristicCallbacks {
    //Quando arriva un update
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();
      String valuestr= "";
      if (rxValue.length() > 0) {
        if(conn){
        //Se esp ha accesso ad internet
        //lo notifica al dispositivo (android)
          String s = "Connected";
          pCharacteristic->setValue(s.c_str());
          pCharacteristic->notify();
        }
        else{
        // riceve ssid, pwd per wifi e l'id dell'utente
        //che lo sta configurando,selezionando il
        //valore giusto dal primo carattere
          for (int i = 1; i < rxValue.length(); i++){
            valuestr=valuestr+rxValue[i];
          }
          if(rxValue[0]=='$'){
            namecheck=true;
            nameid=valuestr;
          }

          if(rxValue[0]=='!'){
            ssidBool=true;
            ssid=valuestr;
          }
          if(rxValue[0]=='?'){
            pwdBool=true;
            pwd=valuestr;
```

```
            updateDataConnection=true;
          }
        }
      }
    };
    //Quando esp invia un update
    void onRead(BLECharacteristic* pCharacteristic) {
        // Non deve fare nulla
    }
};
```

Dopo aver settato i valori costanti e le callback, parte la funzione setup() in cui viene inizializzato il server BLE, gli vengono associate le funzioni definite sopra e da esso, viene creato un servizio con l'identificativo unico Service-Uuid.
Si creano poi le *characteristics*, ed anche a queste vengono associate le corrispettive callback.

```
//Inizializzazione Device BLE
BLEDevice::init("MyESP32");
// Creazione Server BLE
pServer = BLEDevice::createServer();
pServer->setCallbacks(new MyServerCallbacks());
// Creazione Servizio
BLEService *pService = pServer->createService(
   SERVICE_UUID);
//Definizione Characteristic
pCharacteristic = pService->createCharacteristic(
                  CHARACTERISTIC_UUID,
                  BLECharacteristic::PROPERTY_READ |
                  BLECharacteristic::PROPERTY_WRITE);
pCharacteristic->setCallbacks(new MyCallbacks());
// BLE Descriptor
pCharacteristic->addDescriptor(new BLE2902());
```

```
// Inizializza il servizio
pService->start();
// Si rende visibile
pServer->getAdvertising()->start();
```

A questo punto la scheda è visibile ed è possibile connettersi ad essa ed inviarle dei messaggi.
Dopo aver ricevuto le credenziali per accedere ad internet ed averne verificato la connessione, Esp inizializza il sensore Dht 22 (l'unico con output digitale, un po' lento a ricevere i dati), verifica la presenza del modulo SD ed esce dalla funzione setup().

3.1.2 Inizio loop() e dati dei sensori

Dopo essersi connessa ad internet, la scheda entra nella funzione loop() e notifica al dispositivo Bluetooth a lei accoppiato che è riuscita a compiere l'accesso; dopo di che si connette al broker MQTT.
A questo punto viene creato un file Json a cui vengono aggiunti l'id dell'utente che ha configurato la scheda (preso in fase di setup), un identificativo unico di Esp32 ricavato dal codice MAC della scheda ed i dati dei sensori che vengono presi durante lo svolgimento del programma.
Il dato di umidità del terreno (moisture) è ricevuto in modo analogico con valori da 0 a 4095 ed è opportunamente mappato per averlo in percentuale.
Per quanto riguarda invece umidità e temperatura, il sensore digitale Dht 22 restituisce direttamente i valori in gradi Celsius (temperatura) e percentuale (umidità).

Lux da fotoresistore LDR

Partendo dal valore analogico del fotoresistore collegato ad una resistenza di 5k ohm, come si può notare in figura 2.3, il valore di luminosità invece è ricavato nel seguente modo:

```
double light_intensity (int raw) {
  float resistorVoltage = (float)raw / 4095 * VIN; //
     VIN=5v
  float ldrVoltage = VIN - resistorVoltage;
  // resistenza che LDR ha da questo voltaggio (R=ohm
     resistenza)
  float ldrResistance = ldrVoltage/resistorVoltage * R;
  int ldrLux = 12518931 * pow(ldrResistance, -1.405);
  return ldrLux;
}
```

Questo codice è stato sviluppato seguendo la formula [14]:

$$lux = (1.25 \times 10^7) \times R^{-1.4059}$$

I rilevamenti sembrano abbastanza precisi e usare un fotoresistore LDR al posto del sensore per la luminosità rende la configurazione più economica. Una volta raccolti dati dei sensori, il Json viene pubblicato sul topic MQTT, e la scheda passa alla fase di acquisizione dell'immagine.

3.1.3 Acquisizione della foto ed invio

Dopo aver inviato il messaggio con il valore dei sensori, la scheda chiede alla camera di acquisire l'immagine. Questa non viene scattata ad ogni rilevamento per non riempire troppo il Cloud Storage.
Viene quindi settata la risoluzione (è stata scelta quella minima visto il consumo energetico ed il tempo di acquisizione) e viene creato il file 32 byte alla volta (l'opzione 64 rende la maggior parte delle immagini file non validi anche se è ovviamente più veloce). Il file viene così creato sulla scheda Sd.
Al termine di questa operazione, la scheda recupera l'immagine appena scattata e la ridivide in frammenti, ognuno dei quali viene codificato in Base64. Per ognuna di queste stringhe viene creato un Json dove sono specificati, come sempre, gli id di utente e scheda. Oltre a ciò viene aggiunto un campo in cui viene specificato se il frammento inviato è quello di inizio, corpo o fine,

per far capire al servizio che lo riceverà quando creare un nuovo file e quando chiuderne uno che è stato inviato per intero e salvarlo.
Anche questi Json sono pubblicati via MQTT sullo stesso topic degli altri dati.

3.1.4 Fine dello sketch

Dopo aver inviato tutto ciò che doveva, il dispositivo elimina dalla scheda Sd l'immagine appena acquisita (per non riempire la scheda sd anche se le foto sono davvero minime in termini di occupazione di memoria).
Infine si blocca per circa mezz'ora e ricomincia la funzione di loop() facendo una nuova rilevazione.

3.2 Script NodeJs

Lo script NodeJs si occupa di prendere i dati trasmessi da Esp32 e trasferire sul *RealtimeDatabase* i valori acquisiti dai sensori e su *Cloud Storage* le immagini scattate dalla camera.
Il codice:

- supporta l'invio simultaneo da parte di diverse schede/utenti,
- è eseguito su una macchina esterna in locale,
- deve essere sempre attivo per non perdere i dati delle schede.

3.2.1 Configurazione Firebase

Innanzitutto vengono settate le configurazioni per permettere al servizio l'accesso ai db della piattaforma.

```
const firebaseConfig = {  //Prese da console Firebase
  apiKey: "app-Api-Key",
  authDomain: "nome-progetto.firebaseapp.com",
  databaseURL: "url-Realtime-db",
```

```
  projectId: "project-id",
  storageBucket: "bucket-collegato-a-cloud-storage",
  messagingSenderId: "id",
  appId: "id",
  measurementId: "id"
};
```

Dopo di che il servizio Firebase viene inizializzato, così come la connessione al database ed allo storage :

```
const app = initializeApp(firebaseConfig);
const db = getDatabase(app);    //Real-Time db
const storage= new Storage();  //google cloud Storage
let bucketName = "gs://mqttplantanalyzer.appspot.com";
   //per storage
```

3.2.2 Connessione MQTT ed invio

Il servizio, dopo aver preso i riferimenti Firebase, si connette al broker MQTT e fa una *subscribe* al topic della piattaforma, sul quale arrivano i messaggi delle schede Esp32.

```
var mqtt=require('mqtt');
var client = mqtt.connect("broker",{clientId:"id"});
client.subscribe("topic",{qos:2})
```

Dopo di che viene definita la funzione da eseguire quando ci sono messaggi in arrivo su questo topic.
In particolare :

- Se il messaggio è una rilevazione di sensori, prende i dati inviati dalla scheda e tramite *idScheda* ed *idUtente* stabilisce il giusto path di riferimento dove inserirli nel realtime database; quindi aggiunge il timestamp e li inserisce nel modo seguente (ricordiamo che il db è un grande albero JSON) : Root-IdUtente-IdScheda-Timestamp-ValoriRilevazione.

- Nel caso dell'immagine, invece, visto che non arriva un singolo pacchetto, il servizio crea una lista di file a cui ne viene aggiunto uno ogni volta che viene pubblicato un pacchetto con etichetta "init". Ogni volta che un nuovo frammento di foto viene pubblicato su MQTT, il servizio verifica, tramite *idScheda* e *idUtente*, a quale file della lista deve aggiungere i byte in arrivo (codificati in base64). Quando viene ricevuto un frammento con il segmento "eof" (end of), lo script crea il file, lo salva in locale, e fa l'upload tramite Google Cloud al bucket di Firebase Storage collegato all'applicazione. Inserisce poi il file nella cartella della scheda contenuta in quella dell'utente, usando il timestamp come nome file, per poter avere un accesso semplice alla data di upload.

Con questo si conclude l'invio dei dati ed il lavoro del servizio NodeJs, per vedere la struttura dei database Firebase si rimanda alle sezioni Firebase Real-Time Database e Firebase Cloud Storage, mentre nella sezione successiva verrà trattato come essi sono gestiti lato Android.

3.3 Applicazione Android

La parte più corposa dell'implementazione della piattaforma ricade sull'applicazione Android che, come detto in precedenza, deve gestire:

- Sistema di autenticazione
- Connessione iniziale Esp32
- Visualizzazione dati
- Mappa condivisa con dati degli utenti

Quest'applicazione richiede il permesso di utilizzo del Bluetooth e della posizione e che essi siano accesi. In caso non sia così, viene richiesta l'autorizzazione e/o l'attivazione dei due servizi.
Per poter utilizzare gli strumenti offerti da Firebase bisogna aggiungerlo al

progetto Android, collegando le configurazioni in modo simile a come è stato fatto in Nodejs [15].
Vediamo ora come sono implementati i servizi offerti dall'applicazione, facendo riferimento a screenshot della stessa.

3.3.1 Sistema di autenticazione

All'apertura dell'app, se non si è già autenticati, viene richiesto il log-in dell'utente tramite e-mail. Se esiste già un utente associato a quella mail viene richiesta la password e si effettua l'accesso, altrimenti viene chiesto di registrarsi inserendo nome e password da associare al nuovo utente.

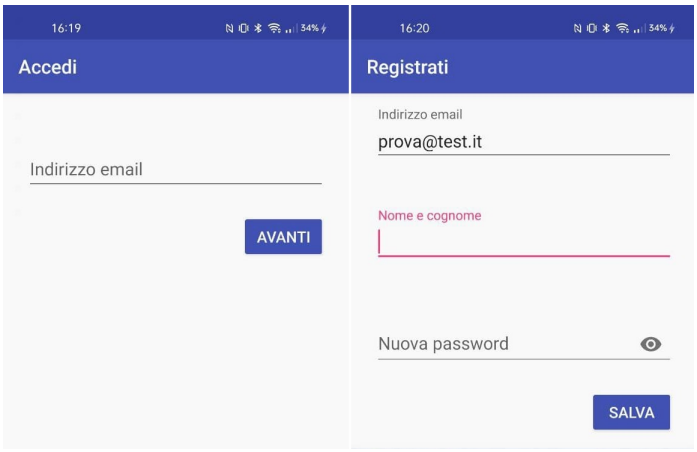

Figura 3.1: Accesso/Registrazione

Come già descritto, tutto ciò è gestito da Firebase, che associa ad ogni utente un identificativo unico che chiameremo *UserId*.
Questo campo serve per cercare i valori nei database e offre una chiave univoca, assieme all'id della scheda, per identificare il dispositivo IoT associato ad un utente.

3.3.2 Connessione iniziale Esp32

Una volta effettuato l'accesso, l'utente visualizza una schermata in cui sono presenti le schede a lui associate (in caso non ce ne siano la lista è vuota). Per connettere una scheda Esp32 all'utente è sufficiente selezionare

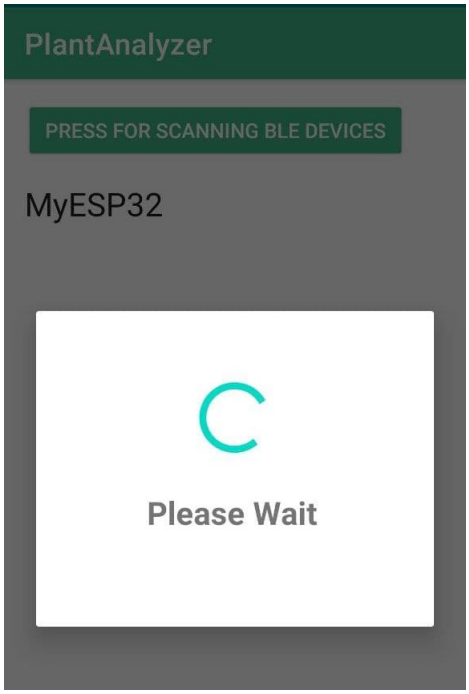

Figura 3.2: Scan BLE

il campo "Bt Connection" dal menù in alto a destra.
Selezionandolo, l'app aprirà una schermata in cui è possibile effettuare una scansione dei dispositivi BLE disponibili.
Selezionando la scheda con il nome *MyEsp32* il telefono si connette ad essa ed invia l'*idUtente* con il quale si è loggati.
A questo punto, effettuando una scansione Wifi, l'utente seleziona a quale rete la scheda deve collegarsi e ne setta la password. Dopo una schermata di caricamento (per dare il tempo alla scheda di ricevere i dati e tentare l'accesso ad internet), l'app notifica lo stato di connessione di Esp32:

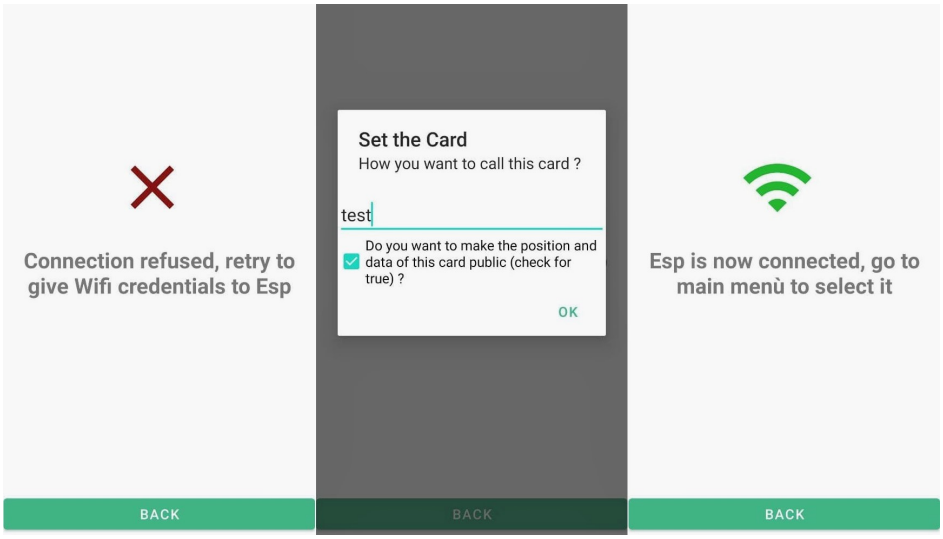

Figura 3.3: Risultato connessione

- In caso di mancata connessione, verrà chiesto di riprovare e si torna nella schermata di scan BLE;
- Se il dispositivo viene connesso, verrà chiesto all'utente con che nome vuole identificare la scheda all'interno dell'app (il campo non può essere vuoto per visualizzarlo nella lista).
  Sempre nella stessa scheda viene invitato a rendere pubblici i dati per quella scheda.
  Se si desidera condividere i propri dati l'app prende la posizione attuale del dispositivo e lo inserisce nel ramo public location come visto in Firebase Real-Time Database.

A questo punto tornando, alla scherma di visualizzazione dei dispositivi associati, sarà presente la nuova scheda con il nome usato in fase di configurazione.

Lo scambio di dati con Esp32 è implementato tramite la libreria Blessed [16], che permette una gestione più semplice del modulo BLE.
Viene creato un oggetto **BluetoothCentralManager** (definito da Blessed) che consente la scansione e la connessione ad un dispositivo **BluetoothPeri-**

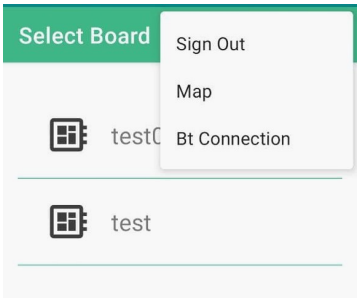

Figura 3.4: Lista dispositivi connessi e menù

pheral. Quando un dispositivo viene associato, si attiva una *callback* che gli invia, come *characteristics* Ble, l'*idUtente* Firebase con il quale si è eseguito l'accesso.
Allo stesso modo avviene l'invio delle credenziali d'accesso alla rete Wifi.
Viene quindi inizializzato un timeout allo scadere del quale:

- Se Esp32 ha inviato un *ack* di avvenuta connessione si accede alla schermata di *success*
- Altrimenti si andrà ad una vista di errore che indicherà all'utente di ripetere il processo.

Nel caso di *success*, se si accetta di rendere pubblici i proprio dati tramite il **Dialog**, l'applicazione recupera la posizione attuale (quindi quella di Esp vista la poca distanza per la connessione Bluetooth) e pubblica i dati sul Database.
La posizione è recuperata nel seguente modo (dopo aver verificato di avere il permesso per la posizione e che sia attiva):

```
mFusedLocationClient.getLastLocation()
.addOnCompleteListener
(new OnCompleteListener<Location>(){
    @Override
    public void onComplete(@NonNull Task<Location> task)
    {
```

```
            Location location = task.getResult();
            if (location == null){
                requestNewLocationData();
            }
            else{
                latitude=location.getLatitude();
                longitude=location.getLongitude();
            }
        }
});
```

3.3.3 Visualizzazione dati

Scegliendo una delle schede presenti nella lista iniziale, verrà caricata una nuova vista per selezionare quale valore si intende monitorare.

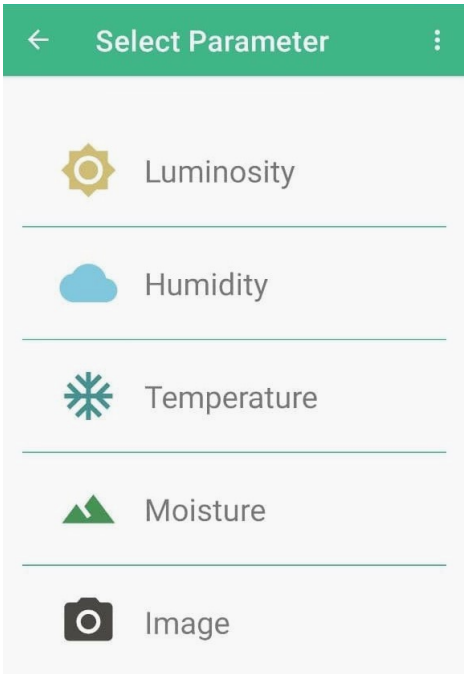

Figura 3.5: Lista parametri

I valori di luminosità, temperatura, umidità ed umidità del terreno (moisture) sono resi mediante l'uso di grafici; per quanto riguarda le foto viene visualizzata una lista con le date in cui sono state scattate: selezionandone una comparirà l'immagine desiderata.
I grafici sono stati realizzati grazie alla libreria MPAndroidChart [17] e mostrano i dati reperiti dal Realtime Database in ordine di tempo dal più recente, tramite un grafico a linee con diverse colorazioni in base al valore selezionato.

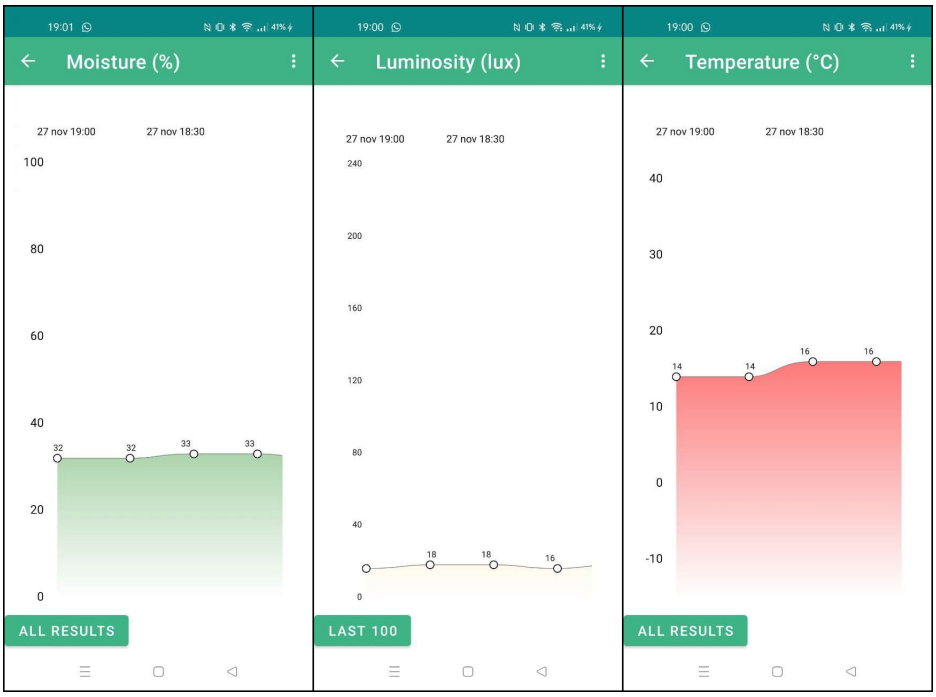

Figura 3.6: Grafici dei rilevamenti (All e Limited)

Quando viene richiesta la visualizzazione dei valori, l'app si connette alla regione di database corrispondente alla scheda/utente e, ogni qual volta che viene aggiunto un valore in quella posizione (dal servizio Nodejs), aggiorna i grafici con i nuovi dati.
I risultati, di default, sono limitati agli ultimi cento definendo una *query* al database con la proprietà *limitToLast(100)*, questo per non far attendere

troppo per la visualizzazione.
Tramite apposito pulsante si possono ricevere tutti i dati, bloccando gli aggiornamenti della *query* precedente e creandone un'altra generica (senza limitazioni sul numero di risultati).
Questo è implementato spostando l'**Event listener** di Firebase (funzione che viene richiamata ogni volta che c'è un aggiornamento sul db), da una *query* all'altra.

```
DatabaseReference userRef = mDatabase.child(boardId);
Query recentPostsQuery = userRef
    .limitToLast(100);
ValueEventListener listener=new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot)
       {
    /*Qui arrivano i dati dal db che vengono usati per
       aggiornare i grafici */
    }
Button querybtn = view.findViewById(R.id.querybtn);
querybtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if(all==false) {
            recentPostsQuery.removeEventListener(
               listener);
            allPostsQuery.addValueEventListener(listener
               );
            querybtn.setText("Last 100");
            all=true;
        }
        else{
            allPostsQuery.removeEventListener(listener);
            recentPostsQuery.addValueEventListener(
               listener);
```

```
            querybtn.setText("All results");
            all=false;
        }
    }
});
```

Questa sezione dell'applicazione è implementata tramite **Fragment** che condividono lo stesso layout (quelli dei grafici), un **Fragment** per la visualizzazione della lista di selezione del parametro ed uno per la lista delle immagini. La navigazione tra i diversi **Fragment** è sviluppata tramite la classe **NavHostFragment**, tramite il file *navgraph.xml* vengono definite le azioni che spostano l'utente da un **Fragment** ad un altro; nella lista di selezione, quando viene toccato un parametro, il **NavHostFragment** esegue l'azione definita per quell'id, ad esempio:

- Nella classe della lista

```
NavHostFragment.findNavController(ChartListFragment.
    this)
     .navigate(R.id.
        action_ListFragment_to_LuminosityFragment);
```

- Nel file navgraph.xml

```
<action
     android:id="@+id/
        action_ListFragment_to_LuminosityFragment"
     app:destination="@id/LuminosityFragment" />
```

Per quanto riguarda le immagini, invece, non hanno limitatazioni nella *request* in quanto nella prima *query* allo Storage vengono richiesti solamente i nomi dei file nella cartella della scheda.
Dopo di che, selezionando uno di essi, viene lanciata un'**Activity** che prende la foto vera e propria e la rende visibile nell'app.

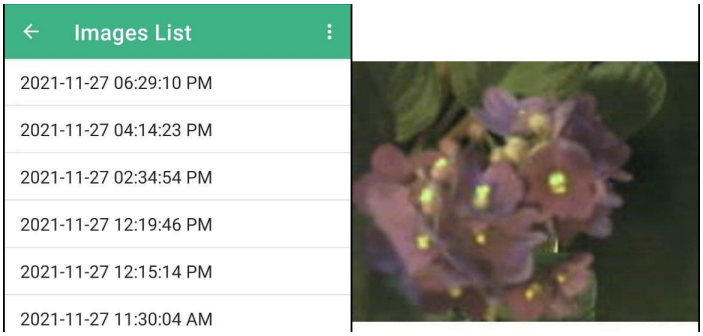

Figura 3.7: Lista immagini e foto d'esempio

3.3.4 Mappa condivisa con dati degli utenti

Scegliendo la voce *Map* nel menù, si accede alla parte di applicazione dove sono visualizzabili i dati degli altri utenti che hanno acconsentito alla condivisione.

Figura 3.8: Mappa

L'**Activity** che implementa questa parte è caratterizzata da una cartina Google Maps (gestita tramite **SupportMapFragment**) dove vengono visualiz-

zati dei marker rappresentanti le varie Esp32 *public*. L'utente può quindi visualizzare gli ultimi dieci rilevamenti e l'ultima foto della pianta di qualcuno vicino a lui.

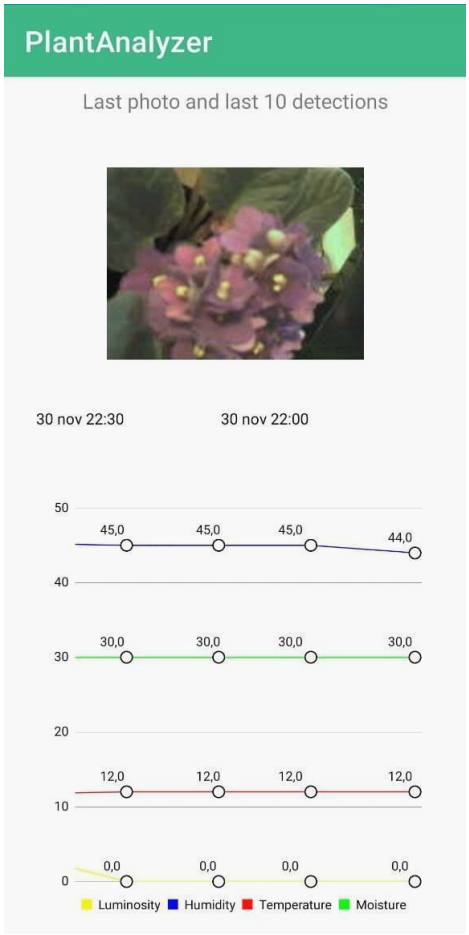

Figura 3.9: Ultime rilevazioni di un altro utente

Inizialmente viene fatta una *query* al *Realtime Database* nella posizione *publicLocation* per settare i marker per ogni scheda; ognuno di essi viene messo in una **Map** (oggetto Java, non Map di Google), dove la prima *chiave* è l'id del marker, mentre il *valore* il l'id della scheda.
In questo modo si ha la possibilità di trovare la scheda (e l'utente) associati ad un indicatore sulla mappa.

Quando l'utente seleziona un marker, l'app lancia un'**Activity** che prende in input *idUtente* e *idScheda* e recupera dal database i dati e l'immagine per costruite la vista, tramite **Intent** esplicito.

Capitolo 4

Casi d'uso e Sviluppi Futuri

Dopo aver esposto gli obbiettivi di questa tesi e come essi sono implementati, cerchiamo di capire l'utilità di questo progetto e in che contesti può essere utilizzato.

4.1 Casi d'uso

Innanzitutto la piattaforma offre la possibilità di avere sullo smartphone i dati della propria pianta *real-time*, consentendone una crescita migliore e rendendo più agevole per il proprietario prendersene cura.
Ovviamente il fatto di poter collegare più di un dispositivo, rende ancora più utile l'applicativo sotto questo punto di vista, potendo gestire più rilevamenti da luoghi anche distanti tra di loro (necessita però di essere configurata nelle vicinanze del dispositivo *Iot*).
La funzione, implementata tramite mappa, consente di visionare i dati di altri utenti per poter verificare sotto quali parametri le piante hanno un migliore aspetto visionandone le foto, al fine da avere un riferimento per chi magari non si è mai dedicato a questo tipo di attività.
Per i servizi che offre ed il basso costo dei componenti utilizzati, il progetto si presta per il momento ad un utilizzo amatoriale.

4.2 Possibili Integrazioni

Questa piattaforma è un'ottima base per lo sviluppo di progetti più complessi, vista la sua configurazione modulare e le scelta di avere un sistema di *accounting* associato ai dispositivi *IoT*.
Per alcuni di questi sviluppi si ha la necessità di avere una grande mole di dati per rendere i risultati più precisi. Per questo motivo si è deciso di affidarsi ad un approccio *low-cost* per le componenti *IoT* e l'infrastruttura è stata fatta in modo da poter gestire molte schede in contemporanea. Seguendo questa linea di sviluppo, cioè limitando la capacità computazionale della scheda in favore del basso costo, sono state escluse le possibilità di incremento del servizio che necessitino di un *upgrade* della scheda.
Ad esempio, non verrà trattata la possibilità di automatizzare alcuni aspetti del processo di crescita come già visto nella sezione Progetti simili.
Le integrazioni di cui andremo a parlare, invece, sfruttano gli stessi dati che vengono esposti al momento, ricavando da essi (nel servizio NodeJs o nell'applicazione Android) delle informazioni aggiuntive, o aggiungendo delle funzionalità per gli utenti.

Interazione tra utenti

Allo stato attuale di configurazione del database, sarebbe semplice implementare un servizio di messaggistica tra gli utenti vicini (grazie alla mappa), creando così un'interazione sociale tra gli utilizzatori, che potrebbero aiutarsi a vicenda e confrontarsi.
Potrebbero inoltre decidere di scambiarsi gli id delle proprie schede per poter supervisionare l'uno la scheda dell'altro o comunque, avere informazioni più precise su come stia crescendo la pianta collegata.

Analisi dell'immagine

Un altro dato molto interessante da poter sfruttare, è la foto della pianta. Grazie ad essa infatti si potrebbe riconoscere la specie vegetale (tramite tec-

niche di AI o Machine Learning) e trovare informazione specifiche su di essa. Esistono già vari progetti che implementano questa funzionalità, che potrebbe quindi essere aggiunta alla piattaforma. Grazie a questo, l'utente avrebbe dei dati di riferimento nei grafici che corrispondono a quelli ottimali per quella specifica pianta e potrebbe essere notificato quando uno dei valori si allontana di una certa soglia da quello perfetto.
Oltre a fornire specifiche aggiuntive per quel tipo di pianta, vari servizi riescono a ricavarne lo stato di salute, sempre attraverso analisi dell'immagine [18].
Questo tipo di informazioni, oltre ad essere molto utili, servono a sviluppare servizi più complessi.

Creazione di un database di valori ottimali

Grazie alla condivisione dei dati, si potrebbe realizzare un servizio che, analizzato lo stato di salute della pianta dalle foto, riesce a selezionare i parametri migliori, scelti dalle immagini la cui analisi ha dato risultati molto positivi.
Il servizio, crea quindi una sezione nel database per ogni specie di pianta condivisa sulla piattaforma, migliorandone i valori di riferimento man mano che arrivano dati il cui stato di salute rilevato dalla foto risulti migliore del precedente.
In questo modo grazie alla collaborazione dei vari utenti si andrebbe a costruire un database sempre più ampio ed affidabile per migliorare il processo di crescita delle varie specie.
Così facendo la piattaforma "apprenderebbe" dai dati degli utenti divenendo sempre più precisa.
Andrebbe ad utilizzare questo database per consigliare i nuovi utenti, "aspettandosi" dei risultati sempre migliori.
Questo tipo di servizio risulterebbe utile non solo all'interno del progetto stesso, ma soprattutto a creare un'API facilmente accessibile da un qualsiasi altro progetto che voglia migliorare o automatizzare il processo di crescita di

diverse specie vegetali.

Verifica dell'inquinamento di un terreno

Alcune piante reagiscono in modi visibili in caso di presenza di alcuni agenti inquinanti nel terreno o nell'aria.
Grazie alla foto ed alla mappa, questa piattaforma potrebbe segnalare facilmente se alcuni terreni presentano questo tipo di contaminanti, coltivando queste piante "speciali" e verificando:

- se avviene il cambio di colorazione tramite la camera,
- se la pianta è in salute, tramite i sensori, e dunque la reazione non è dovuta a carenze sugli aspetti monitorati.

In questo modo si avrebbe una maggior certezza che sia stato l'inquinamento a far "mutare" l'aspetto della coltivazione.
Questo perché è stato controllato che il resto dei valori vitali abbiano avuto dei valori accettabili per tutto il processo.

Luoghi migliori per la crescita

Nel caso in cui questa piattaforma venga utilizzata per coltivazioni outdoor, potrebbe capire, sempre tramite analisi della foto, in quali luoghi la crescita è facilitata da fattori ambientali e per quali tipologie vegetali.
Incrociando i valori dei sensori con quelli derivanti dall'analisi sul suo stato di salute, si potrebbe costruire un algoritmo abbastanza preciso per individuare in quali luoghi si hanno avuto i risultati migliori.
Si creerebbe così una mappa "arricchita" con le migliore zone dove coltivare determinati tipi di piante o consigliare all' utente su quale sarebbe il soggetto più adatto da far crescere nella propria zona.

Conclusioni

Quello che è stato presentato è un progetto che offre la possibilità di configurare un dispositivo IoT e visualizzarne in modo chiaro e compatto le rilevazioni, tutto da dispositivo Android.
Certamente non si prefigge come obbiettivo quello di risolvere le problematiche legate all'inquinamento o al cambiamento ambientale, ma propone un comodo strumento per l'avvicinarsi a questo mondo da parte di persone non esperte (né sul campo tecnologico, né agricolo).
Il basso costo e la facilità di utilizzo sono certamente dei fattori positivi sotto quest'aspetto e la completa gestione da Smartphone potrebbe invogliare una fascia giovane della popolazione ad interessarsi a questo settore.
In aggiunta, grazie ai servizi di accounting e mappa condivisa e ai dati che se ne ricavano, può essere un'ottima base per lo sviluppo di progetti che andrebbero seriamente a migliorare le problematiche attuali in questo ambito.

Appendice A

Foto e link del progetto

In questa appendice ho voluto inserire la foto del dispositivo collegato alla pianta ed il link di *GitHub* dove visionare il codice di tutta la piattaforma: `https://github.com/federicodegiorgio/tesi`

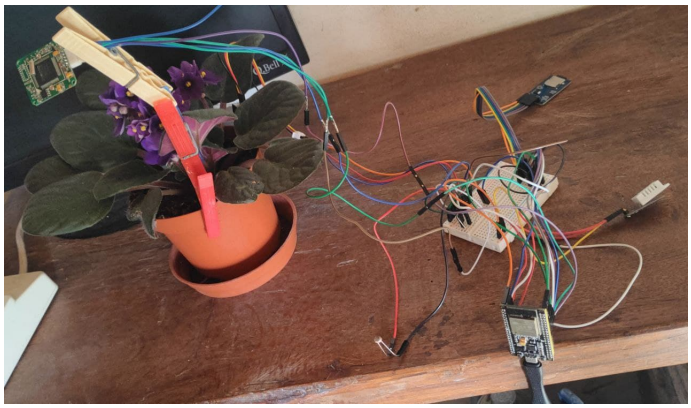

Figura A.1: Foto della scheda

Tutte le rilevazioni d'esempio sono state fatte su questa configurazione. La pianta non riceve abbastanza luce (in effetti non è una stanza molto luminosa), come risulta anche dai valori riportati.
Anche le ultime foto mostrano come i fiori stiano leggermente appassendo; sarà necessario spostarla in uno luogo più esposto al sole per preservarne lo stato di salute.

Bibliografia

[1] Luigi Coppolino, Salvatore D'Antonio, Luigi Romano, and Luigi Sgaglione. Iot: una tecnologia distruptive. *La Comunicazione NR &N.*

[2] Parvaneh Asghari, Amir Masoud Rahmani, and Hamid Haj Seyyed Javadi. Internet of things applications: A systematic review. *Computer Networks*, 148:241–261, 2019.

[3] Transformainsights. `https://transformainsights.com/`.

[4] Number of internet of things (iot) connected devices worldwide 2021/2022 : Breakdowns, growth and predictions. `https://financesonline.com/number-of-internet-of-things-connected-devices/`.

[5] Sandro Nižetić, Petar Šolić, Diego López-de-Ipiña González-de, Luigi Patrono, et al. Internet of things (iot): Opportunities, issues and challenges towards a smart and sustainable future. *Journal of Cleaner Production*, 274:122877, 2020.

[6] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Tariq Umer, and Yousaf Bin Zikria. Role of iot technology in agriculture: A systematic literature review. *Electronics*, 9(2):319, 2020.

[7] Dinesh Thangavel, Xiaoping Ma, Alvin Valera, Hwee-Xian Tan, and Colin Keng-Yan Tan. Performance evaluation of mqtt and coap via

a common middleware. In *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*, pages 1–6. IEEE, 2014.

[8] A Pravin, T Prem Jacob, and P Asha. Enhancement of plant monitoring using iot. *International Journal of Engineering and Technology (UAE)*, 7(3):53–55, 2018.

[9] SV Athawale, Mitali Solanki, Arati Sapkal, Ananya Gawande, and Sayali Chaudhari. An iot-based smart plant monitoring system. In *Smart Computing Paradigms: New Progresses and Challenges*, pages 303–310. Springer, 2020.

[10] Srinidhi Siddagangaiah. A novel approach to iot based plant health monitoring system. *Int. Res. J. Eng. Technol*, 3(11):880–886, 2016.

[11] Esp32 pinout. `https://www.mischianti.org/`.

[12] Chunnu Khawas and Pritam Shah. Application of firebase in android app development-a study. *International Journal of Computer Applications*, 179(46):49–53, 2018.

[13] Firebasecloud storage for firebase. `https://firebase.google.com/docs/storage`.

[14] Design a luxmeter using a light dependent resistor. `https://www.allaboutcircuits.com/projects/design-a-luxmeter-using-a-light-dependent-resistor/`.

[15] Add firebase to your android project. `https://firebase.google.com/docs/android/setup`.

[16] Martijn van Welie. Blessed for android - ble made easy. `https://github.com/weliem/blessed-android`.

[17] Philipp Jahoda. `https://github.com/PhilJay/MPAndroidChart`.

[18] Sachin D Khirade and AB Patil. Plant disease detection using image processing. In *2015 International conference on computing communication control and automation*, pages 768–771. IEEE, 2015.

Ringraziamenti

A conclusione di questo elaborato, desidero menzionare tutte le persone, senza le quali tutto questo non sarebbe stato possibile.

Ringrazio il mio relatore, che in questi ultimi mesi, ha saputo guidarmi nella progettazione della piattaforma, nelle ricerche e nella stesura dell'elaborato.

Ringrazio di cuore i miei genitori. Grazie per avermi sempre sostenuto e per avermi permesso di portare a termine gli studi universitari.

Un ringraziamento molto sentito va anche a tutti le persone che mi sono state vicine durante questo percorso.

Infine, vorrei dedicare questo traguardo a me stesso, affinché possa essere l'inizio di un'appassionante carriera professionale.