UNIVERSITÀ DEGLI STUDI
DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

CORSO DI STUDIO IN INGEGNERIA ELETTRONICA E TECNOLOGIE DELL'INFORMAZIONE

Tesi di Laurea Triennale

Febbraio 2020

Progetto e realizzazione di un sistema IoT per il rilevamento di dati dal campo

Candidato: Davide Delucchi

Relatore: Prof. Riccardo Berta

Sommario

In questa tesi si è eseguita la progettazione e la realizzazione di un sistema per il rilevamento di dati dal campo, in particolare un sistema che monitora il livello di riempimento di una vasca che funge da accumulo per un acquedotto. Per la rilevazione del livello e l'invio dei dati è stato utilizzato un microcontrollore, quale Arduino MKR GSM 1400 ed il sensore a ultrasuoni HC-SR04. I dati raccolti vengono inviati ad un cloud tramite chiamate API rilasciate da un framework detto Atmosphere che si interpone tra il dispositivo di misura e l'applicazione per smartphone dedicata al monitoraggio da parte del gestore della rete idrica. Tale applicazione, sviluppata utilizzando Flutter, si compone essenzialmente di 3 schermate dove è possibile visualizzare il livello in tempo reale, un grafico che riporta l'andamento delle ultime 24 ore ed infine impostare i vari parametri di funzionamento del sistema.

1. Introduzione

Con questo progetto si vuole realizzare un sistema in grado di raccogliere dati dal campo e permetterne il monitoraggio da parte di un supervisore o un utente, in particolare si vuole controllare il livello di riempimento di un recipiente di accumulo, facente parte di un'infrastruttura di distribuzione idrica, situato in una zona boschiva, rendendo disponibile il dato in tempo reale a colui che è l'incaricato della gestione dell'acquedotto. Si è prevista inoltre la possibilità di notificare all'utente, tramite SMS, eventuali dati anomali che potrebbero comportare la presenza di un problema sull'impianto.

Progettazione e realizzazione interessano sostanzialmente due sottosistemi utili a realizzare quanto sopra citato, la cui comunicazione è resa possibile da delle API fornite da un cloud il cui nome è "Atmosphere".
Per quanto riguarda la raccolta dei dati dal campo si è impiegato un sensore ad ultrasuoni il cui compito è quello di rilevare la distanza tra punto di fissaggio del sistema e livello d'acqua all'interno della vasca, quest'ultimo è collegato ad un microcontrollore dotato di connettività GSM e GPRS. La Figura 1 mostra il principio di funzionamento del prototipo atto alla misura.

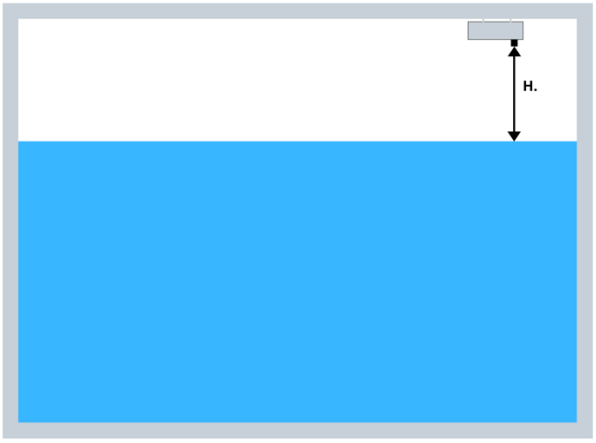

Figura 1: principio di funzionamento del prtottipo

Il monitoraggio e la visione dei dati sono possibili grazie ad una semplice applicazione per smartphone composta di 3 schermate:

- Pagina di login dove è possibile eseguire l'autenticazione con username e password.
- Homepage o schermata principale dove vengono mostrati: percentuale di riempimento in tempo reale, un grafico dell'andamento e due indicatori che riportano massima e minima percentuale raggiunta in riferimento alle ultime 24 ore. I dati inoltre vengono scaricati dal cloud tramite una richiesta GET HTTP ogni qual volta si apre l'applicazione o si effettua un aggiornamento della pagina eseguendo uno scroll verso il basso.
- Un'ultima schermata è utile all'impostazione dei parametri di funzionamento del prototipo; è possibile infatti settare l'intervallo di misurazione, cioè il tempo che trascorre tra una rilevazione del livello e l'altra e l'intervallo di invio dei dati, ovvero il tempo che scandisce con quale frequenza viene effettuato l'invio dei dati raccolti ad Atmosphere tramite una POST HTTP. Inoltre è necessario, per il corretto funzionamento del sistema, l'inserimento della distanza massima misurabile dal sensore ultrasonico, a vasca vuota, e quella minima, a vasca piena, in modo da poter calcolare la percentuale di riempimento. Infine l'utente può definire due livelli di soglia percentuali in base ai quali il sistema, in caso di superamento di tali limiti, avviserà con un SMS l'utente al numero impostato. Questi parametri sono inviati al cloud per mezzo di una PUT HTTP e vengono scaricati dal prototipo tramite una GET HTTP.

2. Metodi e strumenti utilizzati

2.1 Atmosphere

Atmosphere è un framework che soddisfa a pieno quelli che sono i servizi e le funzionalità necessari ad un sistema IoT, permette infatti la comunicazione bidirezionale tra Edge e Cloud e costituisce l'elemento che mette in comunicazione il microcontrollore, dotato di una connessione ad internet, e l'applicazione per smartphone. Ha inoltre il compito di contenere tutti i dati raccolti dai vari dispositivi detti "device" e di mantenerli fruibili ad una determinata "application".

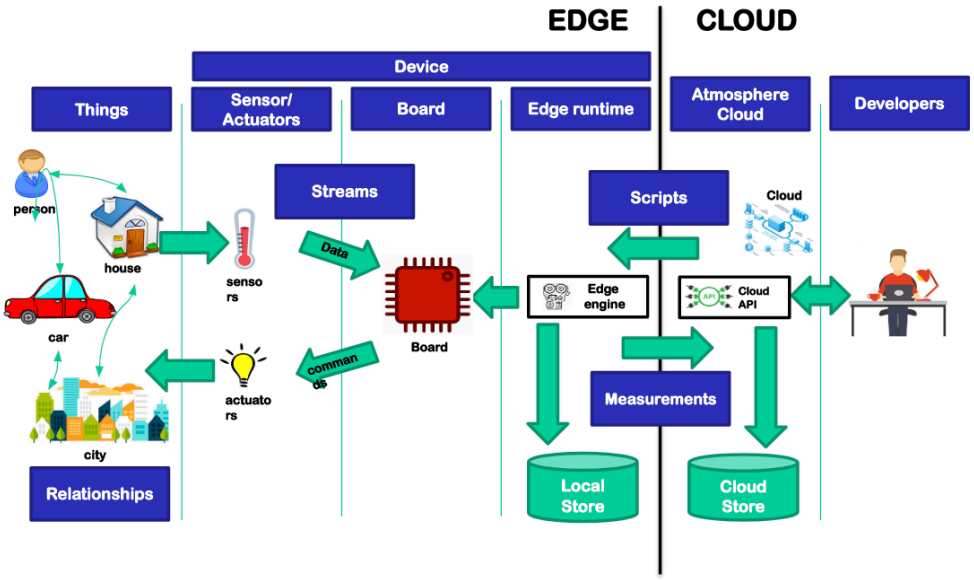

Figura 2: struttura del framework Atmosphere

Il framework si compone di diverse parti utili al suo funzionamento e alla sua manutenzione, ecco un elenco delle più importanti:

- **Developer:** entità che sviluppa applicazioni o boards che utilizzano atmosphere come ad esempio aziende, enti pubblici e sviluppatori.
- **Application:** si tratta di una particolare applicazione che va ad utilizzare quelle che sono le funzionalità offerte dal framework. In questo progetto la mobile app incalza perfettamente il concetto di "application".
- **Thing**: un generico oggetto dal quale si vogliono raccogliere e memorizzare dati per il quale si effettuano delle misure; ne sono un esempio persone, costruzioni, mezzi di trasporto e città. Nel nostro caso si tratta di un grande recipiente contenente acqua potabile.
- **Feature**: è la grandezza fisica o il dato al quale una misurazione, effettuata da un determinato device, si riferisce. Nel caso di nostro interesse vogliamo misurare il livello di riempimento della vasca presa in esame.
- **Sensor/Actuator**: sensore che fornisce misurazioni su una determinata cosa (thing) o attuatore che può agire su un elemento e cambiarne il suo stato.
- **Measurement**: rappresenta il valore di una "feature" misurato rispetto ad una specifica "thing", per noi si tratta di ogni singola rilevazione del livello di riempimento da parte del device.
- **Device**: si tratta di un dispositivo che può misurare una certa dimensione (feature) su una determinata cosa (thing). In questo particolare progetto si tratta di un microcontrollore, quale Arduino MKR GSM 1400, che si occupa di effettuare le misurazioni ed inviarle al cloud.

Chiarita la struttura ed i concetti che stanno dietro ad atmosphere passiamo ad analizzare il suo funzionamento.

L'invio e la ricezione dei dati presuppongono l'accesso al cloud da parte dei dispositivi e delle applicazioni, tale accesso è garantito per mezzo di RESTful API chiamate tramite richieste HTTP (GET, POST, PUT, DELETE) utilizzate in base all'operazione che si vuole portare a termine. Infine ricezione ed invio dell'informazione avvengono attraverso un formato per lo scambio di dati molto utilizzato e universale detto JSON.

Per poter inviare e ricevere dati è necessario possedere un token, ottenibile per mezzo di una chiamata POST HTTP, contente nel body un username ed una password che abilitano l'accesso alle informazioni permettendo inoltre di identificare l'utente, l'applicazione o il device responsabili di tale operazione. Il token ha una validità di 30 minuti, trascorsi i quali, bisognerà ripetere il login effettuando un'altra chiamata.

Vediamo un esempio pratico di autenticazione. Sarà necessario utilizzare la seguente chiamata:

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

con il seguente body formattato come JSON:

```
{
    "username" : "user-tank-server-username",
    "password" : "user-tank-server-password"
}
```

in risposta riceveremo un JSON contenente il token necessario per compiere le altre operazioni.

Per inviare una misurazione al cloud sarà necessario utilizzare la seguente chiamata:

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

avente il seguente body sempre formattato come JSON:

```
{
    "startDate": "{{current_timestamp}}",
    "endDate": "{{current_timestamp}}",
    "thing": "tank",
    "feature": "water-level",
    "device": "tank-level-sensor",
    "samples": [ { "values": [40] } ]
}
```

Inoltre tra gli headers della richiesta dovrà essere presente il token sopra citato al fine dell'autenticazione; è semplice comprendere che nel caso in cui non si possieda il token oppure quest'ultimo sia scaduto, sarà necessario eseguire nuovamente il login per ottenere un nuovo token.

Passiamo ad analizzare il processo utile a scaricare alcuni dati dal cloud, per renderli, ad esempio, fruibili all'utente attraverso l'applicazione. Dovremo utilizzare la seguente chiamata:

GET http://test.atmosphere.tools//v1/measurements?filter={"thing":"tank"}&limit=1&page=1

Analogamente a quanto visto prima, sarà richiesto l'inserimento del token tra gli headers ed è possibile inoltre filtrare i dati in base ad una serie di parametri. La risposta che si ottiene dal cloud sarà un oggetto o un array di oggetti JSON che rispettano i requisiti da noi imposti.

Infine è possibile servirsi di uno script, nient'altro che un file JSON contente una serie di parametri e valori che possono essere scaricati dal nostro device in modo che possa modificare il suo comportamento in base a ciò che il file JSON contiene. In questa tesi il device effettua una chiamata GET HTTP per ottenere i parametri di funzionamento, come ad esempio l'intervallo di invio dei dati e di rilevamento del livello. La mobile app invece si serve di una PUT HTTP per andare a modificare lo script stesso.

2.2 Arduino MKR GSM 1400

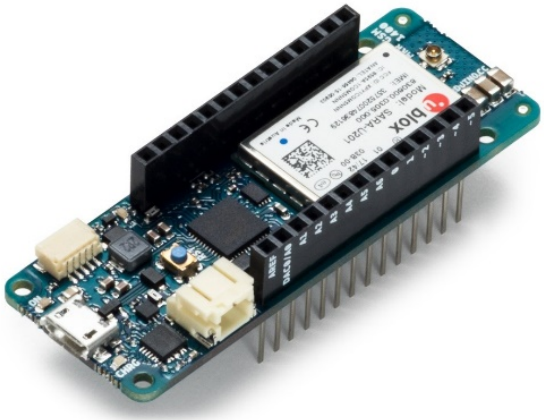

Figura 3: Arduino MKR GSM 1400

L'Arduino MKR GSM 1400 è un microcontrollore di dimensioni ridotte pensato per applicazioni IoT in ambienti dove non si dispone di una connessione Wi-Fi o Ethernet, infatti sfrutta la rete cellulare come mezzo per comunicare. La rete GSM / 3G è una di quelle che copre la percentuale più elevata della superficie terrestre, rendendo questa opzione di connettività ideale per questo progetto.
Il processore principale della scheda è un SAMD21 con architettura ARM a 32 bit a bassa potenza, come nelle altre schede della famiglia Arduino MKR.

La connettività GSM / 3G viene implementata da un modulo di U-BLOX, il SARA-U201, un chipset a bassa potenza che opera nelle diverse bande della gamma cellulare (GSM 850 MHz, E-GSM 1900 MHz, DCS 1800 MHz, PCS 1900 MHz).
Infine, la comunicazione sicura è garantita attraverso il chip crittografico Microchip® ECC508 che utilizza una crittografia SHA-256.

La scheda è dotata di numerosi pin I/O attraverso i quali è possibile collegare sensori di vario tipo e altri dispositivi, è inoltre programmabile attraverso il popolare IDE di Arduino. È presente una porta micro USB attraverso la quale è possibile effettuare il caricamento dello Sketch creato dallo sviluppatore e abilitare la comunicazione tra IDE e microcontrollore in fase di debug. La porta USB è inoltre in grado di alimentare la scheda con una tensione di 5V, alimentazione che può essere anche fornita tramite i pin I/O oppure attraverso una batteria Li-Po da 3.7V che viene ricaricata direttamente dalla scheda; il passaggio da una fonte di alimentazione all'altra avviene automaticamente.
A differenza della maggior parte delle schede Arduino, l'MKR GSM 1400 tollera un voltaggio massimo di 3.3V, anziché 5V, sui pin I/O; il superamento di tale soglia potrebbe portare al suo danneggiamento.

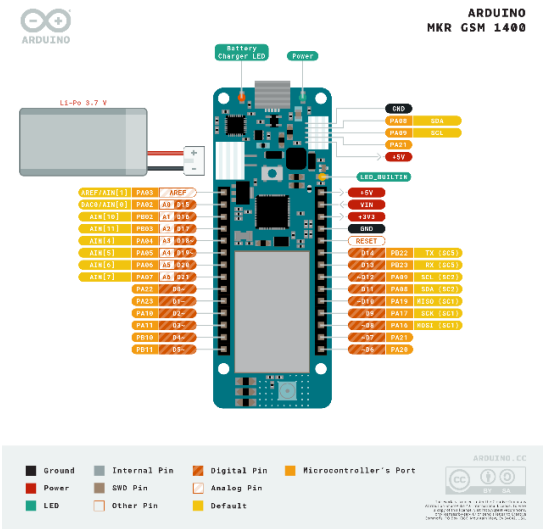

Figura 4: pinout diagram

2.3 Sensore ad ultrasuoni HC-SR04

Figura 5: sensore ad ultrasuoni HC-SR04

L'HC-SR04 è un sensore digitale che sfrutta la tecnologia ad ultrasuoni per misurare la distanza che si pone tra esso ed un ostacolo con una precisione di circa 3 mm, in un range compreso tra i 2 e i 450 cm. Il sensore è costituito nella sua parte posteriore da un circuito elettronico e nella parte anteriore da un quarzo e due trasduttori ad ultrasuoni. Uno di questi invia ultrasuoni che rimbalzano contro ad un qualunque oggetto posto di fronte ad esso, i quali, di ritorno, entrano nell'altro cilindro.

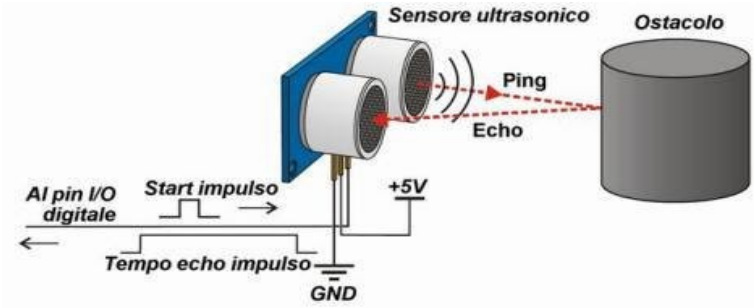

Figura 6: schema di funzionamento

Il sensore opera ad una tensione di 5V ed utilizza due pin, uno per inviare il suono (trig) ed uno per riceverlo (echo). Il processo di misura prevede l'invio di un impulso della durata di 10 µS sul pin trig dopo il quale il dispositivo invia 8 impulsi ad ultrasuoni aventi una frequenza di 40 KHz. Il sensore risponderà sul pin echo con un impulso della durata corrispondente al tempo impiegato dalle onde per ritornare al ricevitore; dopo 38 millisecondi si considera che non sia stato incontrato alcun ostacolo. Per sicurezza si aspettano in genere 50-60 millisecondi per far sì che non vi siano interferenze tra una misura e l'altra. Quanto detto in precedenza è schematizzato in Figura 7.

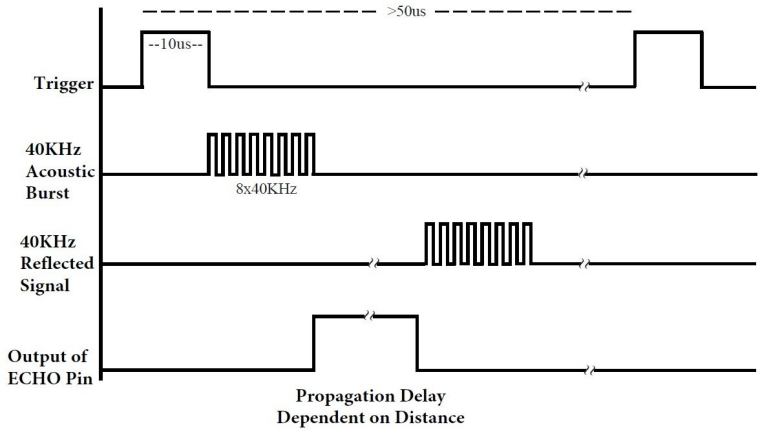

Figura 7: processo di misura

Per misurare la distanza tra sensore e ostacolo possiamo sfruttare la legge oraria della velocità: $Distanza = tempo \cdot velocità\ del\ suono$.
La velocità di propagazione di un'onda sonora dipende fondamentalmente dalla densità del mezzo che deve attraversare (gassoso, liquido o solido) e non dalle caratteristiche dell'onda stessa; in particolare velocità e temperatura sono legate da un rapporto di proporzionalità diretta dato dalla seguente equazione: $Velocità = 331{,}45 + (0{,}62 \cdot T)\ [m/S]$ dove T indica la temperatura misurata in °C.
Bisognerà tener conto inoltre che il suono percorre una distanza doppia in quanto dovrà, una volta incontrato l'ostacolo, tornare indietro fino al ricevitore; possiamo quindi dire che lo spazio percorso è doppio rispetto alla distanza che si interpone tra sensore ed ostacolo.
Ecco la funzione utilizzata per rilevare la distanza:

```
unsigned int measureDistance()
{
  float average = 0;
  for (int i = 0; i < NUMBER_OF_MEASURE; i++)
  {
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    // Detect the reflected wave.
    unsigned long time = pulseIn(ECHO_PIN, HIGH);
    // Calculate the distance using a sound speed constant.
    average += ((0.03438 * time / 2) * 10);
    delay(400);
  }
  return (average / NUMBER_OF_MEASURE);
}
```

Come possiamo notare è stata utilizzata la formula sopra indicata, con l'aggiunta di un 2 al denominatore, per il calcolo della distanza e si è assunta la velocità del suono costante al valore di 343,8 m/s, riferita ad una temperatura di 20 °C. Si è provveduto inoltre ad inserire un ciclo for grazie al quale vengono effettuate un numero prestabilito di misurazioni, intervallate da un tempo di 400 mS, in modo da compensare eventuali errori significativi.

2.4 MKR SD proto shield

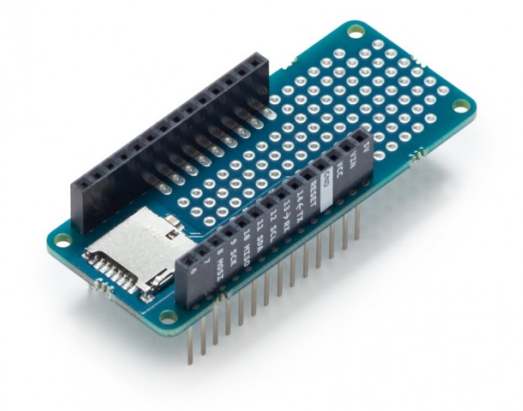

Figura 8: Arduino MKR SD Proto Shield

L' Arduino MKR SD Proto Shield consente di collegare facilmente una scheda micro SD ad una qualsiasi scheda della famiglia MKR, utile per migliorare le funzionalità IoT del sopracitato MKR GSM 1400. L'utilizzo di una scheda SD permette l'implementazione di un buffer nel quale vengono salvati i dati raccolti in attesa dell'invio al cloud e consente di tenere traccia di eventuali errori o problematiche di funzionamento. Inoltre è presente una piccola area di prototipazione sulla quale possono essere saldati alcuni componenti.

2.5 Flutter

Flutter è un framework open-source creato da Google con il quale è possibile creare interfacce native per iOS e Android; l'editor di testo utilizzato è Visual Studio Code con il quale l'SDK si interfaccia perfettamente.
I componenti principali del framework sono:

- **Dart platform:** si tratta di un linguaggio di programmazione sviluppato da Google con lo scopo di "sostituire" JavaScript in ambito di sviluppo web. Il compilatore Dart permette di scrivere programmi sia per il web che per desktop e server, attraverso due differenti piattaforme:
    - Dart Native per i dispositivi quali smartphone, desktop, server e altro;
    - Dart Web pensato per il web.
- **Flutter engine:** scritto principalmente in C++ rende possibile il rendering di basso livello ed utilizza la libreria grafica di Google (Skia Graphics). Una caratteristica molto apprezzata del Flutter engine, grazie al codice scritto in Dart, è la possibilità di eseguire un "hot-reload" grazie al quale le modifiche, in fase di sviluppo, vengono immediatamente mostrate senza il bisogno di riavviare l'applicazione.
- **Foundation library:** scritta in Dart, fornisce classi e funzioni di base utilizzate per costruire applicazioni attraverso Flutter.
- **Widgets:** l'interfaccia utente in Flutter si compone sostanzialmente di widgets ovvero descrizioni immutabili dell'interfaccia utente come grafici, testo, forme ed animazioni. Per la creazione di un elemento grafico più complesso è possibile combinare molti widgets più semplici di diverso tipo.

2.6 Librerie di Arduino

Al fine di rendere il codice più leggibile e semplice sono state utilizzate alcune librerie, ufficiali e non, attraverso le quali si sono implementate le principali funzionalità del device. Andremo di seguito ad approfondire la loro struttura e le funzionalità esposte.

2.6.1 MKRGSM library

Questa libreria può essere facilmente installata sull'IDE di Arduino tramite il gestore delle librerie ed utilizzata all'interno del nostro codice inserendo la seguente riga: `#include <MKRGSM.h>`.
Ci permette quindi, utilizzando l'Arduino MKR GSM 1400, di eseguire la maggior parte delle operazioni possibili per mezzo di un telefono GSM. È infatti possibile effettuare e ricevere chiamate vocali, inviare e ricevere SMS e connettersi a Internet tramite una rete GPRS. Il modulo di bordo funziona di default utilizzando la tecnologia 3G, nel caso in cui quest'ultima non fosse disponibile si ripiega sulla connettività 2G.

Tipicamente ogni singolo comando è parte di una più ampia serie di istruzioni necessarie ad eseguire una delle tante operazioni possibili, inoltre la libreria è basata sulla GSM library dello shield GSM di Arduino con l'aggiunta di nuove funzionalità.
Poiché la libreria abilita più tipi di funzionalità, esistono diverse classi:

- **GSM**: si occupa dei comandi da fornire al modem radio, gestisce tutti gli aspetti di connettività del modulo e registra il device sulla rete GSM. Questo elemento è essenziale in qualsiasi progetto ove sia necessario l'utilizzo delle tecnologie: GSM, GPRS o entrambe. Le funzionalità disponibili vengono esposte dai seguenti metodi:
    - `begin()`: collega il device alla rete GSM identificata dalla carta SIM inserita all'interno dell'Arduino; si tratta della prima operazione necessaria all'utilizzo di tutte le altre funzionalità;
    - `shutdown()`: disconnette dalla rete GSM il dispositivo;
    - `getTime()`: restituisce il tempo in secondi trascorsi dal 1° gennaio 1970, questa funzionalità è utile per dare una scansione temporale alle diverse misurazioni effettuate.
- **GSM_SMS**: permette di inviare e ricevere messaggi di testo; nel nostro caso particolare viene utilizzata per allertare l'utente di valori anomali. L'invio dei messaggi è possibile per mezzo dei seguenti metodi:

- `beginSMS(number)`: identifica il numero di telefono, che va passato al metodo come argomento, al quale si vuole inviare l'SMS;
- `print(message)`: scrive un array di caratteri come messaggio SMS; anche qui il testo viene passato come argomento;
- `endSMS()`: comunica al modem che l'SMS è completo e successivamente lo invia.

- **GPRS**: permette al device di avere una connessione ad internet e quindi di comunicare con il cloud per mezzo del seguente metodo:
  - `attachGPRS(APN, user, password)`: si connette all'APN (Access Point Name) specificato per iniziare la comunicazione GPRS. È utile sapere che ogni provider di telefonia mobile ha uno specifico APN che funge da ponte tra la rete cellulare ed Internet. Talvolta sono presenti un nome utente e una password associati al punto di accesso.

- **GSMClient**: questa classe mette a disposizione le funzionalità utili ad un client, in modo analogo alle librerie Ethernet e Wi-Fi. Vediamo i principali metodi:
  - `connect(ip, port)`: avvia una connessione con uno specifico indirizzo IP su di una determinata porta, il valore di ritorno, un bool, indica la riuscita o il fallimento dell'operazione;
  - `connected()`: permette di sapere se il client è connesso all'indirizzo di nostro interesse o meno con la restituzione di un bool. Un client può risultare connesso anche se la connessione è stata chiusa poiché sono presenti ancora dati non letti;
  - `print()`: permette di scrivere dati al server in questione;
  - `read()`: legge il byte successivo ricevuto dal server a cui si è collegati;
  - `avaiable()`: ci restituisce il numero di bytes pronti per la lettura ricevuti dal server al quale siamo connessi.

Le classi e i metodi sopra riportati costituiscono la sola parte della libreria utilizzata per la realizzazione del sistema in oggetto.

2.6.2 SD library

La libreria SD consente di leggere e scrivere su schede SD, microSD nel nostro caso, e supporta i file system FAT16 e FAT32 su schede SD e SDHC standard. Per essere utilizzata è necessario inserire le seguenti righe di codice: `#include <SPI.h>, #include <SD.h>`. I nomi dei file passati alle funzioni della libreria SD possono includere percorsi separati da backslash, come ad esempio "directory/nomefile.txt". A partire dalla versione 1.0 la libreria supporta l'apertura di più file.

La comunicazione tra il microcontrollore e la scheda SD avviene per mezzo di SPI: un protocollo di trasmissione dati sincrono che usa come canale di comunicazione una seriale. Inoltre, è necessario utilizzare un altro pin per selezionare la scheda SD.
Vediamo le due classi che compongono la libreria:

- **SD**: fornisce funzioni per accedere alla scheda di memoria e manipolare i file e le directory contenuti al suo interno per mezzo dei seguenti metodi:
  - `begin()`: inizializza la libreria e la scheda SD restituendo un bool che ci indica il successo o meno dell'operazione;
  - `exist()`: verifica se un file o una directory esistono sulla scheda SD, ritorna vero in caso affermativo e falso in caso negativo;
  - `open(filepath, mode)`: apre un file presente sulla memoria il cui nome e percorso sono chiariti dal parametro "filepath". È possibile anche scegliere tra due modalità di apertura del file: FILE_READ che apre il file per la lettura e FILE_WRITE che è dedicato alla scrittura del file; nel caso in cui si apra il file per la scrittura e questo non sia presente, viene creato;
  - `remove()`: rimuove un file dalla scheda SD.

- **File**: questa classe ci permette di andare a leggere e scrivere su di un particolare file contenuto nella scheda SD e mette a disposizione un serie di metodi attraverso i quali è possibile accedere alle suddette funzionalità. Vediamo quelli utilizzati per questo progetto:

- `avaiable()`: controlla se ci sono byte disponibili per la lettura dal file ritornandone il numero;
- `close()`: chiude il file e si assicura che ogni dato scritto sia fisicamente salvato sulla scheda SD;
- `print()` e `println()`: entrambe scrivono i dati sul file che deve essere aperto in modalità scrittura, l'unica differenza è che println(), a differenza di print(), inserisce il carattere dopo essere andata a capo.
- `read()`: permette di leggere un file restituendo un byte o un character alla volta, se non c'è niente da leggere restituisce -1.

2.6.3 Arduino JSON library

La libreria ci permette di manipolare oggetti JSON in modo molto semplice ed efficiente utilizzando il microcontrollore Arduino MKR GSM 1400. La particolare operazione di interesse in questo progetto è la "deserializzazione" di un documento JSON, procedura che consente di estrarre le informazioni di nostro interesse da un file JSON. Questa libreria entra in gioco nella fase in cui il device deve ottenere dal cloud i parametri di funzionamento e quindi ricavare quest'ultimi dalla risposta fornitagli dal cloud.
Per il processo di "deserializzazione" vengono utilizzati i seguenti elementi:

- Un oggetto StaticJsonDocument utile a salvare il nostro documento JSON in memoria; utilizzando questa soluzione è necessario conoscere la dimensione del JSON. L'oggetto può essere istanziato nel seguente modo: `StaticJsonDocument<256> doc`.
- La funzione `deserializeJson()` che esegue effettivamente la "deserializzazione" del documento. Necessita di due argomenti: il documento JSON da elaborare e l'oggetto StaticJsonDocument dove salvare ciò che si è "deserializzato".

Una volta che l'oggetto contiene l'informazione di nostro interesse è possibile estrapolare i vari tipi di dati e assegnarli ad un variabile. Eccone qui un esempio:

```
postingInterval = doc["postingInterval"].as<unsigned int>();
measuringInterval = doc["measuringInterval"].as<unsigned int>();
updatingScriptInterval = doc["updatingScriptInterval"].as<unsigned int>();
```

Alla sinistra dell'uguale ritroviamo le variabili alle quali vogliamo assegnare i valori contenuti nel documento JSON, mentre `doc` rappresenta il nostro oggetto `StaticJsonDocument`.

3. Sperimentazione e risultati

3.1 Acquisizione e invio dei dati

Il sistema dedicato all'acquisizione e all'invio dei dati ad Atmosphere si compone di un microcontrollore, l'Arduino MKR GSM 1400, al quale sono collegati un sensore ad ultrasuoni, l'HC-SR04, che si occupa di misurare la distanza tra il sensore e il livello d'acqua all'interno della vasca ed un lettore di schede micro SD, l'MKR SD proto shield, utile a salvare i dati raccolti nel tempo che dovranno essere mandati al cloud.

La connessione tra microcontrollore e sensore ultrasonico è realizzata utilizzando, lato Arduino, due pin I/O digitali che sono collegati rispettivamente ai pin trig ed echo sul sensore ad ultrasuoni, il quale necessita inoltre di un'alimentazione fornitagli sfruttando i pin di uscita VCC e GND di Arduino, i quali si riferiscono rispettivamente a +3.3 V e a massa. La Figura 9 rappresenta una schematizzazione di tale collegamento.

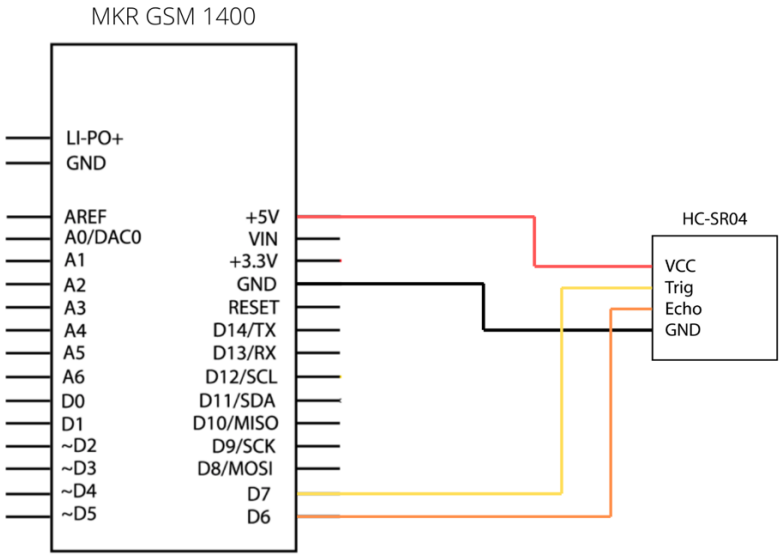

Figura 9: Scema di collegamento tra MKR GSM 1400 e HC-SR04

Per quanto riguarda lo shield MKR SD il collegamento è reso semplice dal fatto che sia la scheda principale, sia l'espansione contenente la microSD appartengono alla famiglia MKR. In Figura 10 è possibile vedere come le due schede possano essere collegate in modo semplice e preciso.

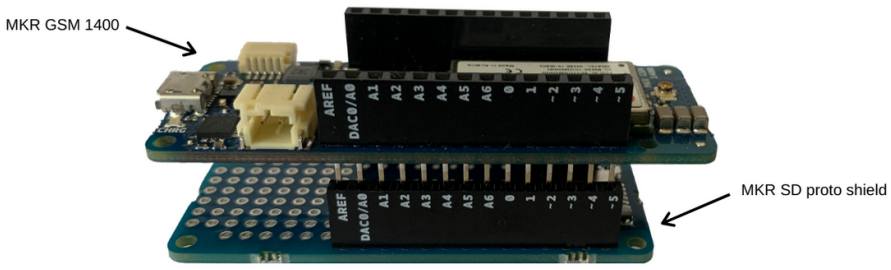

Figura 10: Collegamento tra MKR GSM 1400 e MKR SD shield

Questo sistema è stato installato all'interno di una vasca che costituisce la riserva d'acqua di un acquedotto che provvede alla fornitura di acqua potabile a buona parte della popolazione di Montoggio(GE), piccolo paese dell'entroterra ligure.
Per il funzionamento sul campo, vista la possibilità di collegare l'apparato alla rete elettrica, l'Arduino viene alimentato attraverso i pin VIN e GND con un alimentatore capace di erogare una tensione costante di 5V ed una corrente massima di 2000 mA, quest'ultima è sufficiente a garantire il corretto funzionamento di tutti e tre i dispositivi.
Infine si è provveduto ad inserire l'apparecchiatura all'interno di uno case stagno con il fine di proteggere l'elettronica da polvere ed umidità, come mostrato dalla Figura 11. Il case è caratterizzato da una

certificazione IP56 che garantisce protezione da polvere e acqua tale da non interferire con il funzionamento del dispositivo.

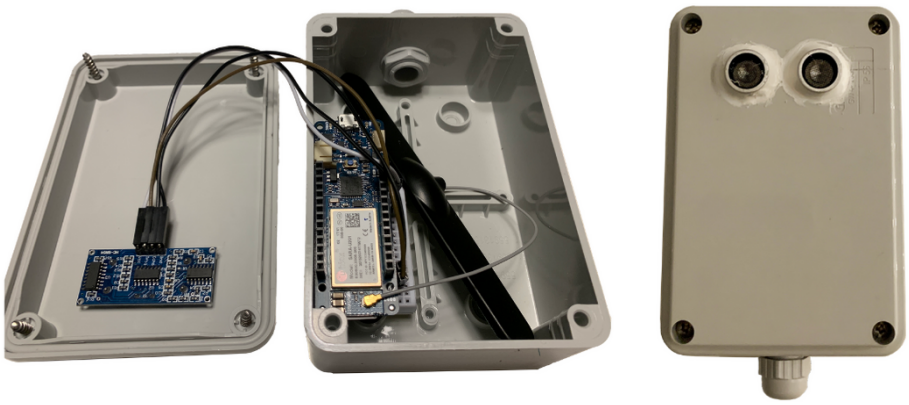

Figura 11: Prototipo utilizzato sul campo.

Il microcontrollore esegue le istruzioni fornitegli per mezzo di uno sketch che viene caricato su di esso utilizzando l'IDE di Arduino. Esistono due fasi principali di funzionamento: quella di setup e quella di loop.

La fase di setup viene eseguita solamente all'avvio del dispositivo o in caso di reset, all'interno di essa vengono svolte le seguenti attività:

- Viene verificato che la scheda microSD sia inserita e funzionate, in caso affermativo si procede all'eliminazione di eventuali dati che, se esistono, possono creare problemi durante il funzionamento.
- Si provvede ad aprire una connessione alla rete internet per mezzo della libreria MKRGSM.
- Infine, una volta connessi alla rete, si procede con lo scaricamento, da Atmosphere, dei parametri utili al funzionamento del device e si provvede alla sincronizzazione del tempo della scheda utilizzano la funzione `getTime()` della classe GSM.

La fase di loop consiste in una serie di istruzioni, ripetute durante l'intero periodo di funzionamento del microcontrollore, attraverso le quali si implementano le vere e proprie funzionalità di cui si è trattato in precedenza, quali: rilevamento del livello di riempimento, salvataggio dei dati in locale, invio di quest'ultimi al cloud e aggiornamento dei parametri di lavoro.
Al fine di temporizzare le diverse attività, viene calcolato il tempo trascorso dall'ultima volta in cui si è eseguita l'attività stessa andando poi a valutare se questo tempo è maggiore o minore di un valore prestabilito; nel primo caso l'operazione viene ripetuta, nel secondo non ne viene eseguita alcuna.
La Figura 12 mostra in maniera semplificata il processo svolto nella fase di loop, mediante un diagramma di flusso.

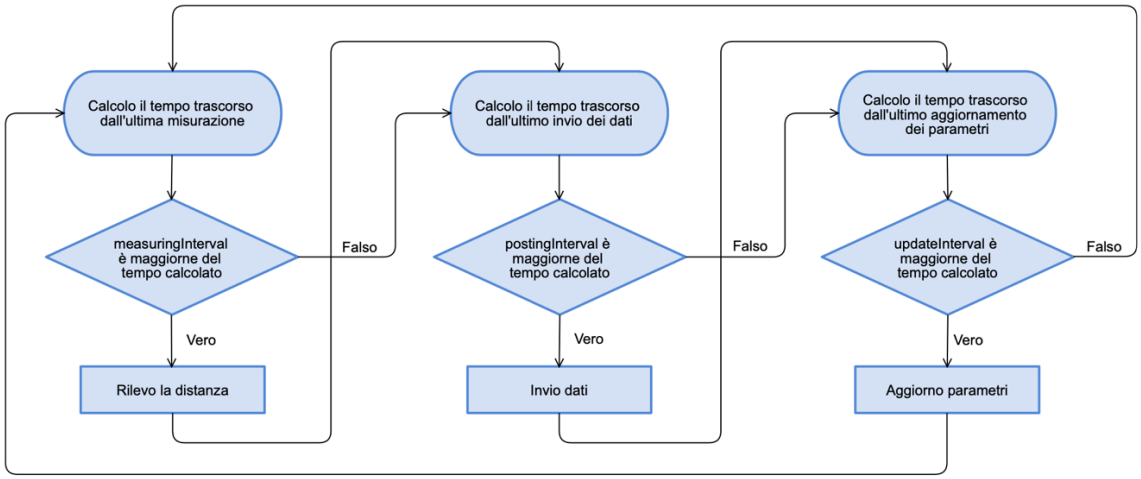

Figura 12: Diagramma di flusso del loop principale

Riporto qui sotto una parte del codice, presente nel loop, relativo al rilevamento della distanza tra sensore e livello dell'acqua:

```
if ((millis() - lastMeasureTime) > measuringInterval)
{
  lastMeasureTime = millis();
  int distance = measureDistance();
  addMeasureToBuffer(distance);
  if (levelAllarm.check(distance))
  {
    sendSms(levelAllarm.getAllarm(), false);
  }
}
```

Per controllare se è trascorso l'intervallo di tempo `measuringInterval` si utilizzano la funzione `millis()` che restituisce, in millisecondi, il tempo passato da quando Arduino ha iniziato ad eseguire lo sketch e `lastMeasureTime`. Ogni qual volta il tempo trascorso (`millis() - lastMeasureTime`) è maggiore di `measuringInterval` si procede con una nuova misurazione. Per portare a termine l'operazione sono necessarie diverse funzioni:

- `measureDistance()`: restituisce la distanza tra dispositivo e livello dell'acqua utilizzando il sensore ad ultrasuoni. Si è trattata questa funzione nel capitolo 2.3;
- `addMeasureToBuffer()`: salva il valore della distanza sulla scheda SD nel formato richiesto da Atmpsphere, il JSON. Nel caso in cui vengano salvate più misurazioni viene creato un array di oggetti JSON.

Utilizzando `levelAllarm.chek(distance)` viene controllato che il valore di distanza rilevato non sia superiore a due valori soglia. In caso affermativo si procede con l'invio di un SMS all'utente, funzionalità implementata dalla funzione `sendSms(levelAllarm.getAllarm(), false)` dove `levelAllarm.getAllarm()` restituisce il testo del messaggio che verrà inviato.
Vediamo adesso come viene temporizzato l'invio dei dati ad atmosphere:

```
if ((millis() - lastConnectionTime) > postingInterval)
{
  lastConnectionTime = millis();
  sendData();
}
```

Analogamente al caso precedente viene controllato se è trascorso un tempo maggiore a `postingInterval` confrontandolo con la quantità `millis() - lastConnectionTime()`. Nei casi in cui si debba procedere all'invio dei dati viene chiamata la funzione `sendData()` che andrà ad effettuare tutte le operazioni necessarie, in particolare:

- Si leggono i dati salvati in precedenza dalla scheda microSD e si preparano per l'invio, creando un documento JSON.
- Vengono mandati i dati ad Atmosphere per mezzo di una richiesta POST HTTP avente come body il JSON creato in precedenza, si è parlato di tale procedura nel capitolo 2.1. La richiesta è portata a termine da un'ulteriore funzione appositamente creata: `httpRequest(url, method, body, authorization)` come possiamo notare la funzione prende in input diversi parametri:
  - `url`: rappresenta l'indirizzo al quale le API possono essere chiamate;
  - `method`: indica il tipo di richiesta http (GET, POST, PUT);
  - `body`: parametro che contiene il corpo della richiesta, il documento JSON sopra citato in questo caso;
  - `authorization`: un bool che indica se una data richiesta necessita del token di autorizzazione o meno.

Infine la fase di loop deve regolamentare anche l'aggiornamento dei parametri di funzionamento; viene riportata qui sotto la relativa parte di codice:

```
if ((millis() - lastParametersUpdate) > updateParametersInterval)
{
  lastParametersUpdate = millis();
  updateParameters();
}
```

Trascorso un tempo maggiore o uguale a updateParametersInterval viene chiamata la funzione updateParameters() che andrà ad eseguire diverse operazioni:

- Esegue una richiesta GET HTTP al seguente URL: *http://test.atmosphere.tools/v1/scripts/water-level-script* la cui risposta contiene un documento JSON con i suddetti parametri.
- Per mezzo della libreria ArduinoJsnon, descritta nel capitolo 2.6.3, l'oggetto JSON viene "deserializzato" estrapolando le informazioni utili al funzionamento.
- Infine si assegnano i valori sopra ottenuti alle variabili utilizzate all'interno del codice. In particolare, vengono aggiornati i seguenti valori:
  - intervallo di rilevazione del livello e controllo di eventuali valori sotto soglia;
  - intervallo di invio delle rilevazioni effettuate al cloud;
  - intervallo di aggiornamento dei parametri stessi;
  - primo livello soglia di allarme;
  - secondo livello soglia di allarme;
  - distanza massima misurabile (vasca vuota);
  - distanza minima misurabile (vasca piena).

Questa funzionalità è utile nel momento in cui l'utente, o lo sviluppatore, per svariati motivi, vogliano andare a modificare il comportamento del prototipo senza bisogno di recarsi sul campo e caricare un nuovo sketch sulla scheda smontando il device e collegandolo al PC.

La scheda SD, oltre che costituire un buffer dove vengono inserite le misurazioni effettuate in attesa di essere inviate, permette di tenere traccia di eventuali errori o malfunzionamenti del device annotandoli su un file di log riportando data e ora nelle quali si sono verificati ed una loro breve descrizione.

Tale processo è svolto chiamando la funzione `logError(logText)` nel momento in cui, all'interno del codice, si riscontra una problematica. Tale funzione prende in input la descrizione del problema riscontrato andandola a scrivere all'interno di un file chiamato SystemLog.txt insieme a data e ora.

3.2 Mobile App

Al fine di monitorare e consultare i dati acquisiti, si è provveduto alla creazione di un'applicazione multipiattaforma per smartphone chiamata iTank, installabile su entrambi i sistemi operativi mobili attualmente più utilizzati (iOS e Android). Il framework per mezzo del quale è stato possibile sviluppare l'app è Flutter, trattato nel capitolo 2.5.

La realizzazione di tale applicazione si è resa necessaria per diversi motivi:

- in prima battuta per rendere completo lo sviluppo di tale sistema nel suo complesso, come descritto nel capitolo 1;
- secondariamente, trattandosi di un progetto che ha ritrovato un impiego pratico in un contesto reale dove l'utente che utilizza il sistema non ha conoscenze tecniche relative all'ambito di questa tesi, nasce l'esigenza di fornirgli uno strumento attraverso il quale possa consultare i dati raccolti in modo semplice ed in mobilità;
- infine, permette anche allo sviluppatore di consultare lo stato del prototipo ed i relativi dati in fase di sperimentazione avendo inoltre la possibilità di modificare alcuni parametri, i principali, senza doversi recare dove si trova fisicamente il dispositivo.

L'app è costituita da tre schermate: loginPage, homePage e settingsPage che hanno le funzioni di effettuare il login per mezzo di un nome utente ed una password, mostrare all'utente i dati raccolti dal campo e permettere la modifica dei parametri di funzionamento del sistema.

Partiamo analizzando layout e funzionamento della loginPage.

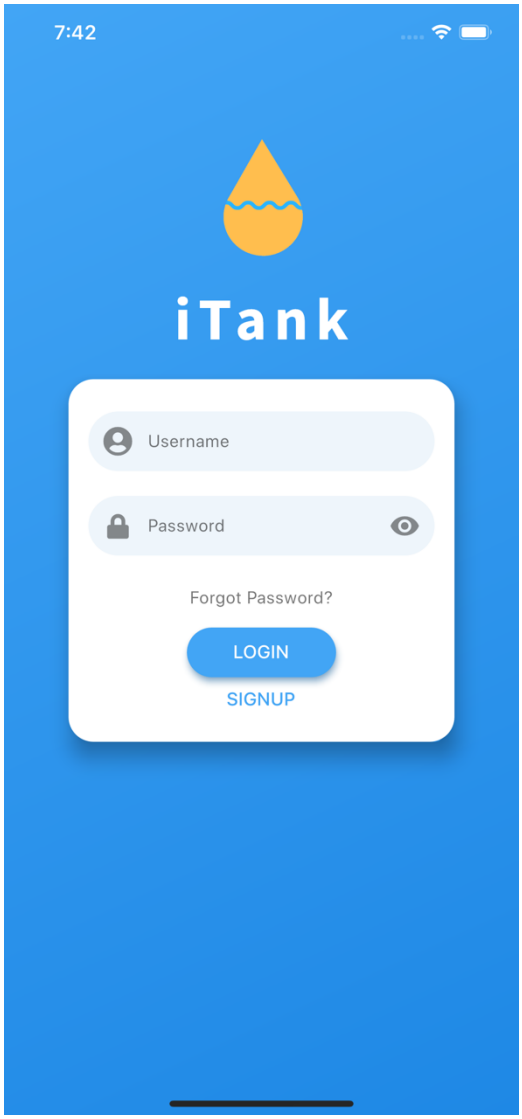

Figura 13: Schermata di login (loginPage)

La Figura 13 mostra la schermata di login che si compone dei seguenti elementi:

- logo e nome dell'applicazione nella parte superiore;
- due TextField all'interno delle quali vanno inseriti nome utente e password;
- un bottone che, se premuto, dopo aver inserito le corrette credenziali, rimanda alla schermata principale.

La funzionalità di accesso, presente in questa pagina, viene implementata per mezzo di una richiesta POST HTTP al seguente URL:

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

avente un body contente username e password inserite dall'utente, come illustrato nel capitolo 2.1. Sulla base della risposta HTTP ottenuta dalle API si valuta la validità o meno delle credenziali inserite, se lo status code della risposta è 200 significa che il login è andato a buon fine e che i dati di accesso forniti sono corretti, altrimenti si notifica all'utente che le credenziali sono errate. Nel caso in cui l'operazione sopra descritta vada a buon fine viene aperta la pagina principale e si procede al salvataggio delle credenziali sul particolare dispositivo utilizzato, in modo che, una volta inseriti i dati di accesso corretti, non vengano più richiesti all'avvio dell'applicazione.

Nel nostro particolare caso vengono utilizzati gli stessi nome utente e password indipendentemente da che la richiesta sia effettuata dal device o dall'applicazione, questo per motivi di semplicità in fase di sviluppo e poiché la sicurezza del sistema non è l'aspetto principale di questo progetto.

Passiamo ora ad analizzare la schermata principale della quale è raffigurata sotto l'interfaccia utente.

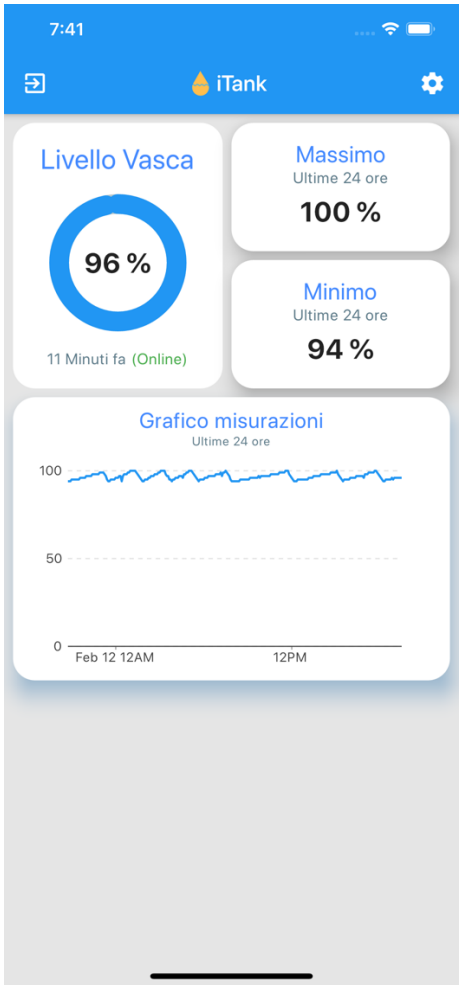

Figura 14: Schermata principale (homePage)

La Figura 14 rappresenta la schermata principale che si compone dei seguenti elementi:

- Una AppBar, situata nella parte alta dello schermo, contenente due pulsanti che ci consentono di effettuare il logout, ovvero tornare alla pagina di login, ed accedere alla pagina delle impostazioni.
- Individuato dalla scritta "livello vasca" un riquadro mostra il livello di riempimento della cisterna per mezzo di un indicatore circolare ed un valore percentuale. Riporta inoltre il tempo trascorso dall'ultima misurazione ricevuta da Atmosphere, in minuti, indicandoci anche lo stato di funzionamento del prototipo (online o offline).
- Individuati dalla dicitura "massimo" e "minimo" questi due riquadri mostrano rispettivamente il massimo e minimo livello raggiunto nelle ultime 24 ore.
- Troviamo infine un riquadro al cui interno è presente un grafico che ci permette di visualizzare l'andamento tenuto dal livello di riempimento nelle ultime 24 ore.

I dati visualizzati nella pagina vengono scaricati dal cloud mediante una richiesta GET HTTP, dato che siamo interessati alle sole misurazioni relative alla vasca (tank) e raccolte nelle ultime 24 ore è necessario applicare alcuni filtri in modo da ottenere solo i dati di nostro interesse, l'URL sarà quindi il seguente:

> *http://test.atmosphere.tools/v1/measurements?filter={"thing":"tank","startDate":{"$gt":"timestamp"}}&limit=300&page=1"*

dove il valore *timestamp* sarà sostituito con data e ora riferite a 24 ore prima dal momento in cui viene effettuata la richiesta.
Le API ci risponderanno dunque con un documento JSON contenete tutte le misurazioni delle ultime 24 ore, tali dati vengono convertiti dapprima in un oggetto JSON e successivamente viene creata una lista estraendo solamente le due informazioni a noi utili, data e valore delle misurazioni. Si cercano infatti, in tale lista, il valore massimo e minimo di riempimento e l'ultima misurazione effettuata dalla cui data si calcola il tempo trascorso dall'ultima rilevazione del livello. L'intero contenuto della lista è invece inserito nel grafico, avente sull'asse delle ascisse il tempo e sulle ordinate il valore percentuale del riempimento, che può assumere valori da 0 a 100.

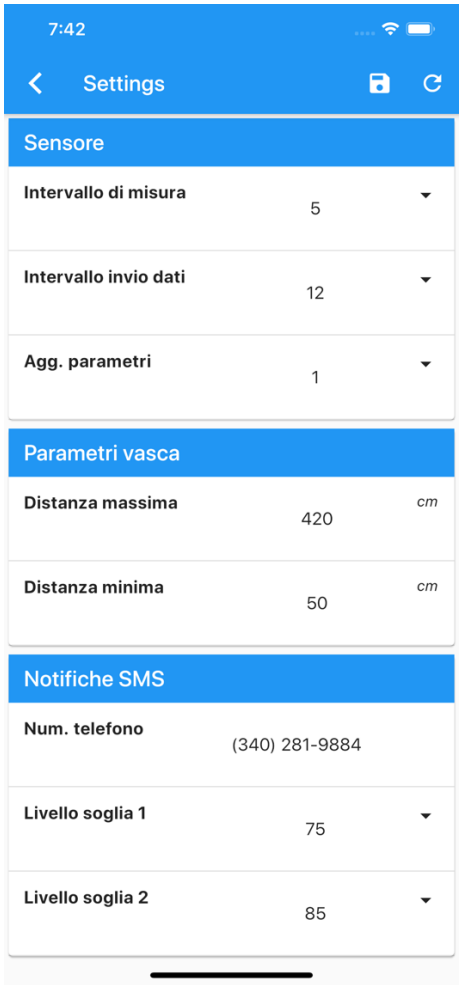

Figura 15: Schermata settaggio parametri (settingsPage)

Parliamo infine della schermata dedicata al settaggio dei parametri di funzionamento, mostrata dalla Figura 15. Quest'ultima si compone dei seguenti elementi:

- AppBar, situata nella parte alta dello schermo, contenente tre pulsanti: il primo ci consente di ritornare alla pagina principale, il secondo salva i parametri scelti nei vari campi sottostanti e l'ultimo serve semplicemente a ricaricare i parametri salvati su Atmosphere.
- Sezione dedicata al device dove possono essere impostati tre diversi valori:
    - intervallo di misura: indica il tempo che deve trascorrere tra una rilevazione del livello e l'altra da parte del dispositivo nella vasca, può essere settato tra 1 e 10 minuti;
    - intervallo invio dati: tempo che trascorre tra un invio delle misurazioni e l'altro, possono essere scelti valori tra 10 e 60 minuti;
    - aggiornamento parametri: intervallo temporale presente tra un aggiornamento dei parametri di funzionamento e l'altro, la scelta va da 1 ora a 24.
- Sezione dedicata alle caratteristiche della vasca dove si possono stabilire due grandezze:

    - distanza massima misurabile, misurata a recipiente vuoto;
    - distanza minima misurabile, rilevata a recipiente pieno.

- Sezione dedicata agli allert SMS dove possono essere scelti tre parametri:
    - Numero di telefono: recapito telefonico del gestore dell'infrastruttura monitorata che deve ricevere le eventuali comunicazioni via messaggio;
    - Livello soglia 1 e 2: indicano due valori percentuali di riempimento sotto i quali viene inviato l'SMS di avviso all'utente, settabili tra il 10% e il 90%.

Per il funzionamento della suddetta pagina vengono inizialmente scaricati dal cloud i parametri di funzionamento per mezzo di una GET HTTP e inseriti nei relativi campi; l'utente può allora modificarli a suo piacimento e decidere di salvarli mediante l'apposito pulsante. Il processo di salvataggio si compone di due fasi: nella prima, premuto il bottone "salva", viene generato un documento JSON con tutti i parametri appena scelti dall'utente verificando che siano valori conson; in seconda battuta tali dati vengono mandati ad Atmosphere effettuando una PUT HTTP all'indirizzo:

http://test.atmosphere.tools/v1/scripts/water-level-script

all'interno del body è contenuto il JSON appena creato, eccone un esempio:


```
{
   "code":"{\"postingInterval\":\"300\",\"measuringInterval\":\"120\",\"updat
   ingScriptInterval\":\"8000\",\"minHeight\":\"30\",\"maxHeight\":\"300\",\"
   phoneNumber\":\"3474205757\",\"allertLevel1\":\"80\",\"allertLevel2\":\"60
   \"}"
}
```

4. Contributo personale e considerazioni conclusive

Obbiettivo di questa tesi era realizzare un sistema di monitoraggio per la vasca di accumulo di un acquedotto, sito in Montoggio (GE), in modo da fornire al gestore dell'infrastruttura uno strumento utile ad evitare continui sopralluoghi di persona e la possibilità di ricevere notifiche in caso di eventuali anomalie.

Le soluzioni impiegate si sono rivelate conformi a quanto prefissato ed è stato possibile implementare le suddette funzionalità grazie all'utilizzo dell'Arduino MKR GSM 1400 che, per mezzo della connessione cellulare, ha reso semplice l'accesso ad internet e l'invio di SMS nonostante la struttura, oggetto del monitoraggio, si trovasse in una zona boschiva non coperta da connessioni via cavo.

Il sensore ad ultrasuoni HC-SR04 ha soddisfatto appieno quelle che erano le necessità, si tratta infatti di un dispositivo economico che permette di misurare distanze con buona precisione.

Infine Flutter ha permesso lo sviluppo di un'applicazione per smartphone in modo abbastanza semplice e veloce, quest'ultima rappresenta un elemento fondamentale per rendere possibile l'utilizzo del sistema all'utente privo di conoscenze tecniche in materia.

Il processo di sviluppo ha visto una prima fase più sperimentale, dove si è provveduto alla stesura del codice per mezzo di due editor: l'IDE di Arduino, utilizzato per comporre lo sketch da caricare sulla scheda e Visual Studio Code, impiegato per la scrittura dell'applicazione. Si è utilizzato inoltre Postman, un software realizzato per formulare richieste HTTP di ogni tipo, molto utile per testare e comprendere il funzionamento di Atmosphere.
La seconda fase ha riguardato invece la messa in opera del sistema e sono state svolte le seguenti attività:

- assemblaggio di un case IP56 per la protezione dell'Arduino e del sensore ultrasonico posizionati all'interno della vasca;
- creazione di un supporto metallico su misura per il fissaggio del prototipo;
- installazione in loco e collegamento del dispositivo alla rete elettrica preesistente;
- aggiornamenti dello sketch di Arduino sul campo dovuti a problemi iniziali di funzionamento;
- installazione dell'applicazione smartphone sul device del responsabile dell'acquedotto e relativa illustrazione del funzionamento;

A fronte di quasi due mesi di funzionamento il sistema ha soddisfatto pienamente le mie aspettative ed ha reso molto entusiasta la persona che lo utilizza, al punto che ha deciso di installarlo in futuro per monitorare altre vasche di cui è responsabile.

Sarà possibile in futuro apportare ulteriori migliorie al sistema, come ad esempio l'alimentazione del prototipo per mezzo di una batteria ed un pannello solare in modo da poterlo installare anche in zone ancor più remote ove non sia possibile sfruttare la rete elettrica.

5. Riferimenti bibliografici

[1] Arduino MKR GSM 1400 Datasheet

[2] HC-SR04 Datasheet

[3] Leonard Richardson, *RESTful Web APIs*, O'Reilly

[3] Dominique Guinard, Building the Web of Things, Manning

[3] Language Reference Arduino, https://www.arduino.cc/reference/en/

[4] Flutter Documentation, https://flutter.dev/docs

[5] Atmosphere Cloud API documentation, https://github.com/Atmosphere-IoT-Framework/api