



**Università
di Genova**

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE
TELECOMUNICAZIONI

Tesi di Laurea Triennale

INGEGNERIA ELETTRONICA E TECNOLOGIE DELL'INFORMAZIONE

Febbraio 2026

**Sistema di controllo e monitoraggio di cisterne
d'acqua interconnesse**

Candidato: Zahi Abou Chakra

Relatore: Prof. Riccardo Berta

Correlatori: Dr. Matteo Fresta
Luca Lazzaroni

Indice

1. Introduzione	2
2. Dispositivi e Sistemi Utilizzati.....	3
2.1 Smart Thing System.....	3
2.2. Network and Cloud	5
2.3. User	5
3. Sistema di Controllo	6
4. Interfaccia API e Mobile App	10
4.2. Measurify e JSON.....	10
4.3. Applicazione Mobile.....	11
5. Sperimentazione e Funzionamento	15
6. Riferimenti.....	16

1. Introduzione

Il presente progetto consiste nella realizzazione di un sistema in grado di automatizzare il funzionamento di diverse cisterne d'acqua interconnesse, fornendo all'utente informazioni relative allo stato operativo e ai livelli d'acqua delle cisterne.

L'idea alla base del progetto nasce da una problematica riscontrata in ambito domestico, legata alla discontinuità e alla bassa pressione dell'erogazione idrica comunale, insufficienti a garantire il riempimento di un serbatoio situato al terzo piano dell'edificio. Per ovviare a tale criticità, è stato installato un contenitore di accumulo al piano terra, alimentato lentamente da una sorgente naturale. Una volta raggiunto il livello massimo, l'acqua viene trasferita al serbatoio superiore mediante una pompa elettrica.

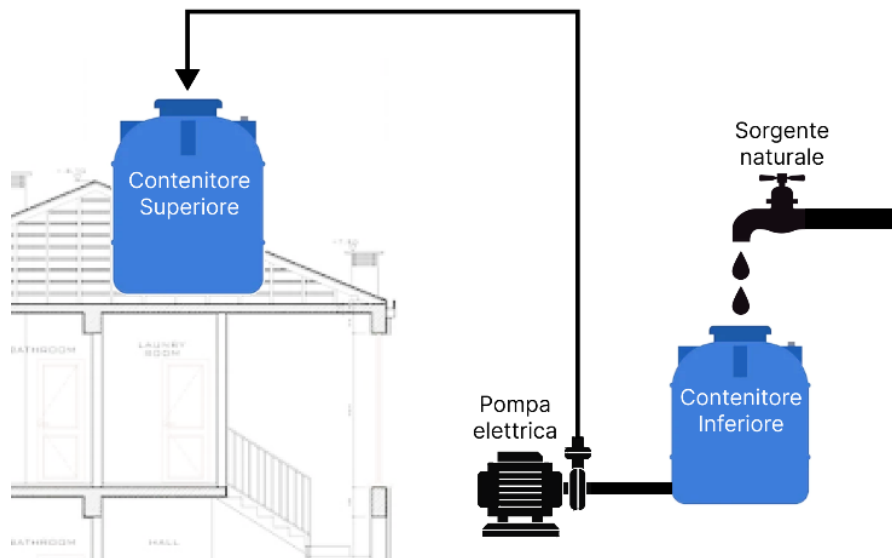


Figura 1: Configurazione delle cisterne.

Attualmente, l'intero processo richiede un intervento manuale sia per il monitoraggio dei livelli sia per l'attivazione della pompa.

Con la grande diffusione dei sistemi IoT (Internet of Things) e la loro evoluzione, che li ha resi sempre più accessibili, è stato creato un sistema basato su queste tecnologie al fine di automatizzare l'intero processo e fornire all'utente i dati sul proprio dispositivo mobile.

Il sistema è stato progettato per operare all'interno di una configurazione idrica costituita da una cisterna dedicata all'alimentazione idrica di una struttura edilizia come un edificio o un palazzo, e da una o più cisterne destinate all'accumulo dell'acqua.

2. Dispositivi e Sistemi Utilizzati

Il seguente schema presenta i componenti che formano il sistema complessivo e i collegamenti tra di loro:

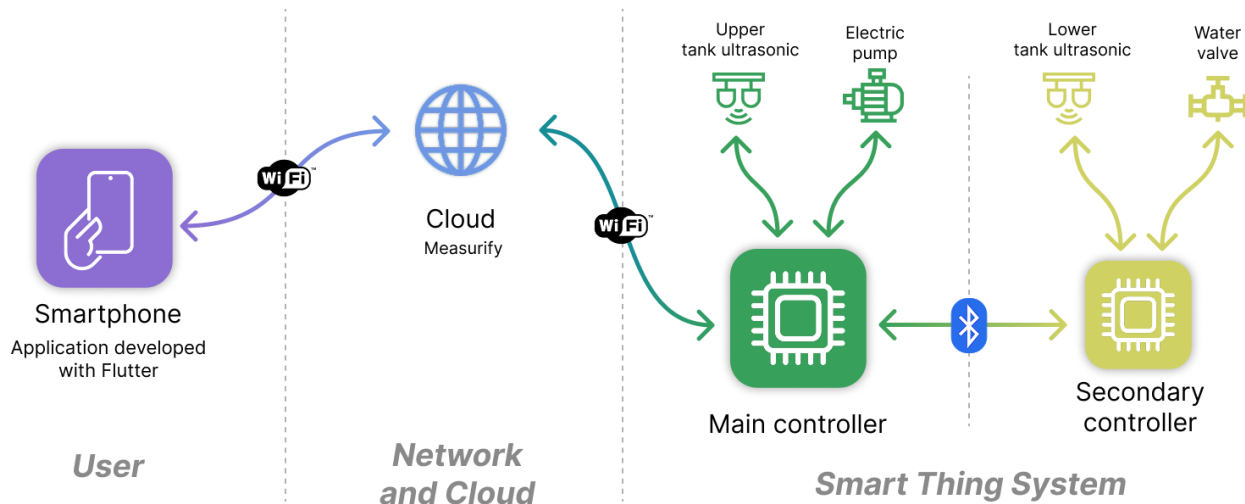


Figura 2: Schema del sistema.

È stato suddiviso in tre sotto-parti:

2.1 Smart Thing System

Lo Smart Thing System presenta la configurazione dei microcontrollori, sensori, pompa elettrica, elettrovalvola e le tecniche di comunicazione usate tra di loro.

Per suddividere questo sottosistema in piccoli sotto-problemi, è stato adottato il concetto leader-follower. Ogni cisterna è dotata di un microcontrollore in grado di misurare il livello d'acqua, mandare le misure al resto del sistema e controllare una pompa o una valvola. In questo caso un leader è una cisterna che è dotata di un microcontrollore principale (Main Controller) e i followers formano il resto delle cisterne con microcontrollori secondari (Secondary Controllers), che sono anche le cisterne accumulatori di acqua.

L'unica differenza tra un controllore principale e uno secondario (cioè un leader e un follower) è che quello principale, o il leader, non espone il suo livello alle altre cisterne e invece lo manda al cloud, che, in questo caso, è un database raggiungibile tramite Internet. Ora abbiamo che, tutte le cisterne secondarie (followers) mandano le misure alla cisterna principale, e quella principale condivide tutte le misure con il cloud.

Per semplicità, è stato studiato il caso in cui è presente una sola cisterna secondaria. Ovviamente il sistema può essere esteso al caso con più cisterne secondarie accumulatrici...

Per poter scegliere i microcontrollori adatti all'applicazione, sono stati studiati i propri ruoli. Il microcontrollore principale, per esempio, deve:

1. Controllare la pompa, cioè deve prendere decisioni cambiando lo stato di un pin (ON/OFF).
2. Calcolare il dato del sensore, che vuol dire manipolare un insieme di pin per fare le misure.
3. Ricevere i dati dalla cisterna secondaria connettendosi tramite qualche tecnica di telecomunicazione.
4. Mandare tutti i dati al cloud, quindi deve essere in grado di connettersi ad una rete WiFi.

Nella configurazione del problema iniziale domestico, l'area di copertura del WiFi include la cisterna principale, ma non copre la posizione della cisterna secondaria. Per quello è stato necessario trovare una tecnica di comunicazione alternativa. È stata scelta la tecnologia **BLE (Bluetooth Low Energy)** come mostrato in [figura 2](#), che, come il nome suggerisce, è di basso consumo di energia e offre un'area di copertura maggiore rispetto a quella del WiFi. Il BLE è capace di stabilire una connessione per una distanza di circa 100 metri in linea di vista, che si riduce a 10-30 metri in presenza di ostacoli. E nel caso si usassero trasmettitori ad alta potenza, l'area di copertura può arrivare oltre ad un kilometro.

I controllori secondarie devono avere le stesse specifiche tranne quella del WiFi. A questo punto si procede alla scelta del microcontrollore.

Esistono diversi microcontrollori che soddisfano le specifiche del progetto, Uno è stato l'**ESP32-WROOM-32** da Espressif Systems, riportato in figura a destra.



Figura 3:
Modulo ESP32-WROOM-32.

Questo modulo è stato scelto per la sua accessibilità e il suo basso costo. Inoltre, offre un MCU (Microcontroller Unit, l'ESP32) abilitato sia di WiFi sia di BLE, ed è capace di effettuare calcoli e prendere decisioni logiche semplici.

L'ESP32 è usato sia nella cisterna principale sia in quella secondaria, ed è stato programmato con l'Arduino IDE, spiegato nel [capitolo 3](#).

Per misurare il livello d'acqua dentro alla cisterna si usa un sensore ad ultrasuoni (figura 4) che viene operato tramite due pin: il trigger, che manda un'onda sonora ad alta frequenza, e l'echo che riceve la riflessione dell'onda sulla superficie dell'acqua. Il microcontrollore, a questo punto, misura il tempo impiegato dall'onda per viaggiare e rientrare nel sensore, calcolando la distanza fra l'acqua e il sensore. (moltiplicando il tempo impiegato per la velocità del suono e dividendo per due...)

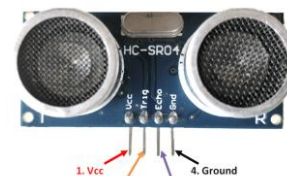


Figura 4:
Sensore ad ultrasuoni (HC-SR04)

La pompa elettrica invece, viene operata tramite un pin digitale, connesso direttamente ad un relè elettromeccanico, che consente di controllare una corrente ad alta potenza con un segnale a bassa tensione (accende o spegne la tensione di 220 volt con quella proveniente dal microcontrollore di 5 volt). Per poter effettuare i test in laboratorio, la pompa elettrica ad alta potenza è stata sostituita con una piccola di 5 volt.



Figura 5: Micro-pompa elettrica (3-5 volt)

L'elettrovalvola, quella che controlla il flusso d'acqua alle cisterne di accumulo, è controllata in modo simile ed è solo usata nelle cisterne secondarie.

2.2. Network and Cloud

Il cloud costituisce l'infrastruttura usata per poter immagazzinare e manipolare i dati ricevuti dal Smart Thing System sul cloud.

Come piattaforma del cloud, è stato usato **Measurify** basato su REST, che permette di salvare e manipolare dati provenienti da sensori, tramite chiamate API.

Con HTTP, il microcontrollore manda i propri dati e quelli ricevuti tramite BLE, alla piattaforma Measurify, dove vengono salvati e visualizzati in grafici e numeri.

Measurify definisce una struttura a 4 livelli:

- Thing: definisce una “cosa” che può essere un’oggetto o una persona su cui si fanno le misure. Nel nostro caso è la cisterna.
- Feature: La misurazione stessa, cioè il livello d’acqua.
- Device: il dispositivo che effettua le misure, cioè l’ESP32-WROOM-32
- Measurement: la misura effettuata dal dispositivo su una determinata Thing.

Per questo progetto, è stato fornito un “tenant” di Measurify, dove si possono memorizzare i dati, insieme a due Things: “CisternaA” e “CisternaB”, e una Feature per ciascuna chiamata “Livello_acqua”. Inoltre, è stato fornito un token per ogni cisterna, per riuscire a fare chiamate POST e GET con http.

A questo punto, il microcontrollore principale, connesso a WiFi locale, può fare chiamate HTTP (POST) per salvare le misure e gli stati di ogni cisterna su Measurify.

2.3. User

L'utente deve riuscire a visualizzare i livelli nelle cisterne e lo stato del sistema dal suo dispositivo mobile. Per questo, è stata sviluppata un'interfaccia utente (UI) con Flutter (nel linguaggio Dart) supportata dai sistemi operativi più diffusi (Android, IOS e Windows).

Per rendere la lettura dei dati facile all'utente, l'applicazione, chiamata Water Monitor, è stata studiata a livello di design di interfaccia usando Figma (strumento di prototipazione).

L'obiettivo è di informare l'utente del livello d'acqua e dello stato del sistema, quindi l'applicazione è costituita da tre pagine, discusse nel [capitolo 4](#).



Figura 6: Icona dell'app.

3. Sistema di Controllo

Dopo aver fatto le scelte del hardware, passiamo allo sviluppo software. Per ottenere i risultati desiderati dal progetto, la metodologia usata per lo sviluppo del programma dei microcontrollori è fondamentale.

Per il controllore principale, il codice sviluppato con Arduino IDE è sequenziale ed è diviso in un file principale chiamato `mainController.ino`, e altri header files (`.h`) per mantenere una struttura modulare:

1. `config.h`:

Questo header file, definisce tutti i parametri del sistema che vengono usati nel processo di controllo (`#define...`). Sono definiti:

- I numeri dei pin del microcontrollore usati per la pompa e il sensore (TRIG e ECHO)
- La misura del sensore quando la cisterna è piena (distanza tra il sensore e la superficie d'acqua).
- La misura del sensore quando la cisterna è vuota (distanza tra il sensore e il fondo della cisterna).
- Il numero di misure da fare (usato per ottenere una misura precisa, spiegato dopo...)
- Il livello minimo di acqua nella cisterna secondaria per accendere la pompa,
- Il livello minimo accettabile d'acqua nella cisterna principale.
- Il tempo fra i controlli del funzionamento corretto del sistema.
- Il tempo fra gli aggiornamenti del cloud.
- Sensor range: usato per dividere il livello massimo e per offrire flessibilità nel controllo.

Questi valori sono definiti per una specifica configurazione (taglia delle cisterne, tipo di microcontrollore, sensore...) e non vengono modificati nell'esecuzione del programma. Inoltre, offrono flessibilità nel caso viene cambiata la configurazione del sistema.

2. `globalVariables.h`:

Insieme a `globalVariables.cpp`, raccoglie tutte le variabili necessarie al corretto funzionamento del programma. Ad esempio, le variabili usate per il conteggio del tempo trascorso, per memorizzare un livello d'acqua precedente (usato per il confronto con quello attuale), per

memorizzare lo stato della cisterna, per memorizzare gli errori (error flags). Inoltre, sono definiti e memorizzate i vari identificativi del BLE e WiFi (UUIDs, SSIDs, Password...).

In questo file, sono anche descritti gli stati della cisterna: WAITING: attesa della cisterna secondaria prima di attivare la pompa. FILLING: pompa accesa. BLOCKING_ERROR: quando c'è un errore critico.

3. control.h:

Con il relativo control.cpp, definiscono le funzioni di basso livello:

- `get_level_sup()`: calcola il livello d'acqua nella propria cisterna. La superficie dell'acqua può presentare irregolarità che influenzano la precisione della misura. Quindi vengono effettuati diverse acquisizioni a intervalli di 10 millisecondi e successivamente viene calcolata la media tra di loro per ottenere una misura precisa. Infine, il valore viene memorizzato e usato da altre funzioni.
- `start_pump()` e `stop_pump()`: usate per attivare o disattivare la pompa, cambiando il valore del pin digitale relativo.
- `check_pumping()`: verifica il corretto funzionamento della pompa confrontando il livello dell'acqua precedente con quello corrente. Nel caso non si riscontri un incremento del livello, viene segnalata una condizione di errore alla funzione chiamante (return -1).

Le funzioni definite nel presente header file, operanti a questo livello di controllo, non prevedono la modifica degli error flags, che sono controllati a livelli superiori del programma.

4. logic.h

logic.cpp definisce le funzioni di livello più alto rispetto a quelle in control.h. Le funzioni definite sono quelle in grado di prendere decisioni logiche e cambiare lo stato del sistema:

- `handle_upper_tank()`: questa funzione è alla base dell'operazione della pompa. In questa fase, vengono verificati i livelli d'acqua sia nella cisterna secondaria sia in quella principale, sulla base dei quali viene determinata l'attivazione o la disattivazione della pompa.
Viene disattivata qualora il livello d'acqua nella cisterna secondaria non risulti sufficiente secondo i parametri definiti in config.h, per evitare il sovraccarico della pompa. Inoltre, viene disattivata nel caso la cisterna principale risulti completamente riempita.
- `handle_error()`: verifica il corretto funzionamento di tutti i componenti del sistema. In particolare, controlla la misura corrente del livello d'acqua e segnala un errore qualora la funzione di basso livello (`get_level_sup()`) restituisca -1. Successivamente, viene verificata l'eventuale presenza di errori di comunicazione BLE, e in tal caso, viene avviata la procedura di riconnessione.

- `log()`: fatta per gli sviluppatori per visualizzare i cambiamenti e gli errori del sistema nel Serial Monitor.

5. **connectivity.h**

Dichiara le funzioni responsabili della gestione della parte di comunicazione del sistema. Ad esempio, la comunicazione con la cisterna secondaria tramite BLE viene gestita in questa sezione, dove vengono definite le principali funzioni necessarie al suo corretto funzionamento (`notify`, `onConnect`, `onDisconnect`...). Inoltre, è definita la funzione `search_and_connect(...)` usata dallo stack del BLE nelle fasi di connessione iniziali per la ricerca e la connessione alla cisterna secondaria. Tale funzione viene anche chiamata in caso di errore BLE, per ristabilire la comunicazione.

In questo file, viene anche gestita la comunicazione col cloud tramite WiFi. La procedura iniziale della connessione ad un Access Point viene gestita una sola volta nella funzione `setup()` presente nel file `mainController.ino`. In caso di errore WiFi, l'ESP32 gestisce la riconnessione automaticamente. Le funzioni responsabili dell'aggiornamento dei dati su Measurify sono le seguenti:

- **`update_cloud()`**: Measurify è una piattaforma di tipo RESTful che usa JSON per rappresentare e scambiare i dati. In questa fase, viene creato un oggetto JSON che contiene i seguenti dati: data e ora della misura, percentuale dell'acqua nella cisterna e flag bits che contengono dati relativi allo stato del sistema. La struttura dell'oggetto JSON è descritta in dettaglio nel [capitolo 4](#).

A questo punto, viene preparato l'header del protocollo HTTP nel quale sono specificati il tipo di messaggio (in questo caso `application/json`) e le informazioni di autorizzazione, cioè il token.

Questa procedura viene ripetuta all'interno della stessa funzione anche per i dati relativi alla cisterna secondaria.

- **`wifi_error_handler(...)`**: questa funzione viene usata dallo stack WiFi per determinare il processo relativo agli eventi relativi alla connessione WiFi (connessione non riuscita, connessione interrotta...).

6. **mainController.ino:**

`mainController` è il file principale del programma che esegue le funzioni `setup()` e `loop()`.

In seguito, sono riportate le operazioni di ciascuna funzione, in ordine di esecuzione:

`setup()`:

- preparazione della comunicazione seriale.

- Indicazione del ruolo dei pin.
- Misurazione iniziale del livello dell'acqua, per assicurare la correttezza dei valori.
- Procedura di ricerca e connessione della cisterna secondaria tramite BLE.
- Preparazione della connessione WiFi, usando SSID e password specificati in `globalVariables`.

Questa funzione viene eseguita solo all'accensione del microcontrollore. Nel caso di malfunzionamento di qualche componente, viene segnalato l'errore negli `error flags`.

La funzione `loop()` viene eseguita ciclicamente dopo l'esecuzione di `setup()`:

`loop()`:

- Misurazione del livello dell'acqua, per garantire l'aggiornamento del valore ad ogni ciclo di esecuzione.
- Controllo degli errori tramite `handle_error()`, responsabile della gestione delle condizioni di errore e dell'aggiornamento dello stato del sistema.
- Controllo della pompa tramite `handle_upper_tank()`, responsabile della logica di controllo della cisterna principale.
- Aggiornamento dei dati su Measurify con `update_cloud()`.
- Per gli sviluppatori viene alla fine eseguita `log()`, che indica lo stato del sistema tramite la porta seriale.

4. Interfaccia API e Mobile App

4.2. Measurify e JSON

Come descritto nel capitolo 2, Measurify è la piattaforma cloud usata per la memorizzazione e trasmissione dei dati. È di tipo RESTful e usa un oggetto JSON per rappresentarli. Nel sistema di controllo, la funzione `update_cloud()` presente nel file `connectivity.cpp` è responsabile della preparazione dell'oggetto JSON che verrà spedito a Measurify tramite HTTP.

L'oggetto a il formato seguente:

```
1  {  
2    "timestamp": "1769295600004",  
3    "values": [84, 8]  
4  }
```

dove “timestamp” rappresenta la data e l’ora secondo l’epoch UNIX in millisecondi e “values” rappresenta due “Features”: il primo è `Livello_acqua` che è un percentuale del livello d’acqua, e il secondo rappresenta lo stato e gli errori del sistema (tipo di errore, attesa, riempimento...), codificato in formato binario come segue:

$0_{10} = 0000_2$: Nessun errore. Il sistema è in attesa o la cisterna è piena.

$1_{10} = 0001_2$: Errore del sensore (livello d’acqua fuori scala).

$2_{10} = 0010_2$: Errore della pompa (indicato da `check_pumping()`).

$4_{10} = 0100_2$: Errore del BLE (connessione alla cisterna secondaria fallita).

$8_{10} = 1000_2$: Pompa attiva.



Figura 7: Rappresentazione del secondo "Feature"

Inoltre, è stato aggiunto nel secondo byte meno significativo, il tempo rimasto per riempire la cisterna (se calcolabile) (nella figura 7 a sinistra), sempre in formato binario, che consente di memorizzare un tempo massimo di 255 minuti (8 bits).

Nelle fasi di sviluppo, è stata utilizzata la piattaforma Postman che permette allo sviluppatore di mandare chiamate HTTP al server. In altre parole, rende possibile modificare, cancellare e aggiungere dati al tenant di Measurify, accedendo col username e password, che è stato rilevante nella fase di test.

4.3. Applicazione Mobile

Per poter interpretare i dati memorizzati su Measurify, è stata sviluppata un'applicazione mobile con Flutter (Dart), supportata dai sistemi operativi più utilizzati come descritto nel [capitolo 2](#).

La progettazione dell'applicazione Water Monitor è stata effettuata mediante Figma, seguendo i principi di design adottati nelle applicazioni moderne, con l'obiettivo di garantire un utilizzo efficace.

L'applicazione presenta tre pagine accessibili tramite una barra di navigazione posizionata nella parte inferiore dell'interfaccia:

1. Home Screen

È la pagina iniziale che contiene una finestra (widget) per ciascuna cisterna. All'interno della finestra è disposto la percentuale d'acqua, lo stato della cisterna, eventuali commenti, e a destra, una grafica di una cisterna che mostra il livello di riempimento:

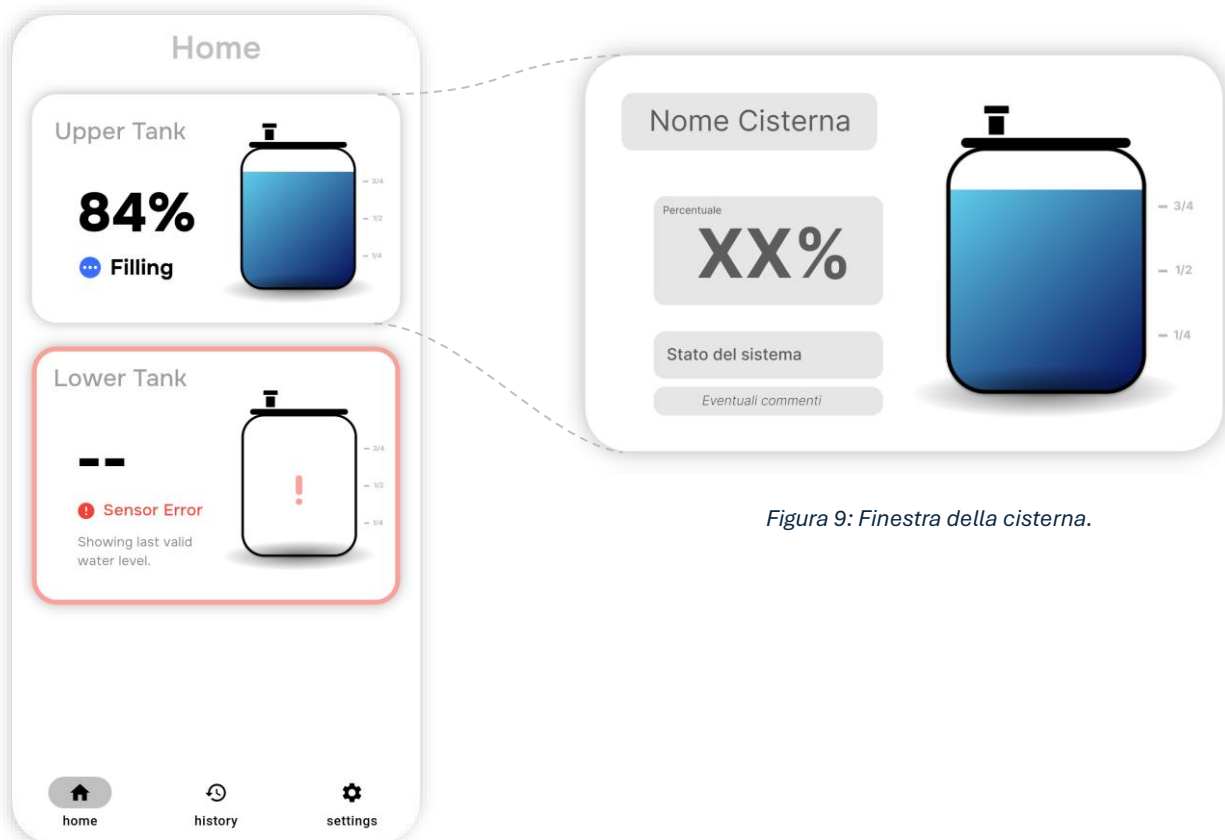


Figura 9: Finestra della cisterna.

Figura 8: Home Screen

La seguente figura mostra i possibili output della finestra:

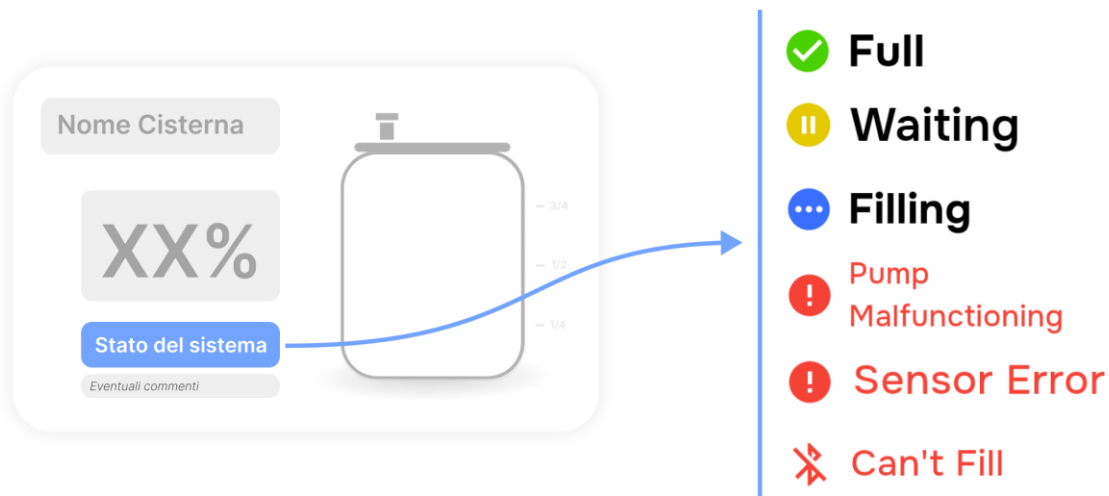


Figura 10: Possibili stati della cisterna.

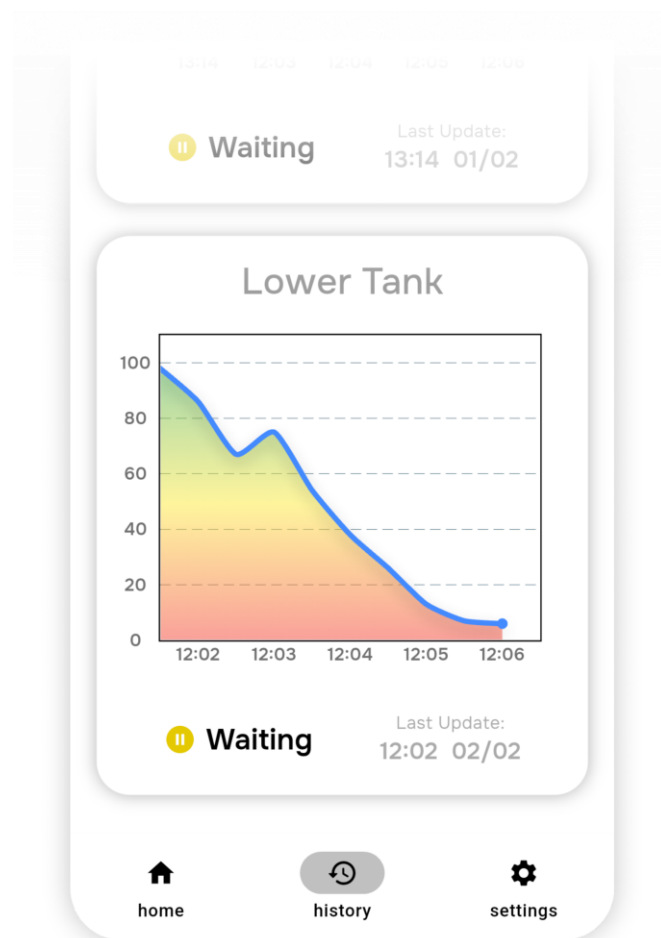
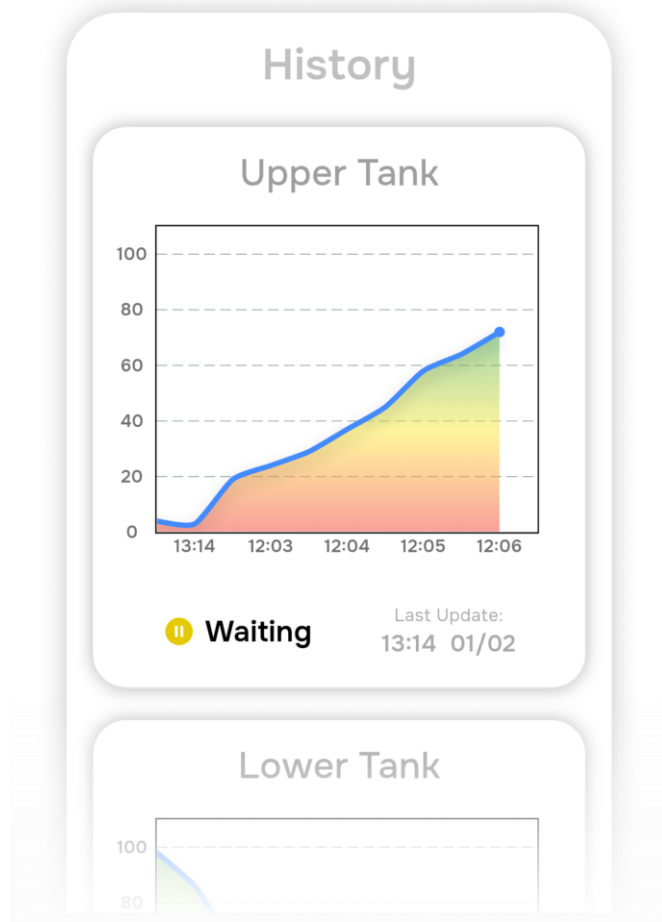
- **Full:** cisterna piena, pompa non attiva.
- **Waiting:** sistema in attesa. Il livello d'acqua nella cisterna secondaria non è sufficiente per attivare la pompa (attesa del riempimento di quella cisterna).
- **Filling:** pompa attiva / riempimento della cisterna.
- **Pump Malfunctioning:** errore di pompa. La pompa è stata attivata mentre il livello d'acqua è rimasto invariato.
- **Sensor Error:** i valori misurati dal sensore sono fuori scala.
- **Can't Fill:** la connessione alla cisterna secondaria, il livello d'acqua in quella cisterna non è disponibile, e la pompa non può essere attivata.

Per una cisterna secondaria, le possibili uscite sono Full, Sensor Error, Can't Accumulate e Filling, tra cui "Can't Accumulate" indica un problema con la sorgente naturale (sorgente naturale attiva mentre il livello d'acqua è invariato).

Nella sezione dei commenti, qualora lo stato sia "Sensor Error", viene visualizzato il commento "Showing last valid water level." per indicare che l'applicazione dispone l'ultima misura valida del livello d'acqua (mostrato in figura 8). Per lo stato "Filling", viene invece mostrato il tempo residuo per il riempimento della cisterna, qualora il dato sia disponibile.

2. History

Nella presente pagina sono visualizzati grafici che rappresentano la variazione nel tempo del livello d'acqua. È stata adottata la stessa struttura delle finestre (widgets) che contengono il grafico, lo stato della cisterna, la percentuale del livello d'acqua e la data dell'ultima acquisizione del dato:



Il grafico relativo alla cisterna riporta sull'asse delle ordinate la percentuale del livello d'acqua, e sull'asse delle ascisse è indicato l'orario dell'acquisizione. Quest'ultimo è determinato solo dal momento in cui è stata aggiornata l'informazione sul cloud, e non coincide necessariamente con l'ora di recupero del dato dall'applicazione. Questa scelta progettuale viene utile nel caso di errori (soprattutto errori WiFi) poiché consente di individuare l'ultima data in cui il sistema ha funzionato correttamente.

La parte inferiore della finestra dispone lo stato del sistema e la data e l'ora dell'ultimo aggiornamento da parte della cisterna.

3. Settings

Rappresenta la pagina delle impostazioni, che viene utile nel caso il progetto venga esteso.

Possibili impostazioni:

- Controllo del sistema dal dispositivo mobile (indicare orari di lavoro, fermare immediatamente il sistema, reset del sistema...)
- Modalità della visualizzazione dei dati (app theme, colori, sottofondo...)
- Impostazioni avanzate, che permettono allo sviluppatore di cambiare gli indirizzi http e token ed indicare nuove dimensioni delle cisterne, senza dover riprogettare l'intera applicazione.

5. Sperimentazione e Funzionamento

6. Riferimenti

- Espressif – ESP32 - Documentation:
[https://www.espressif.com/en/products/socs/esp32#:~:text=ESP32%20is%20highly%20Dintegrated%20with,Circuit%20Board%20\(PCB\)%20requirements.](https://www.espressif.com/en/products/socs/esp32#:~:text=ESP32%20is%20highly%20Dintegrated%20with,Circuit%20Board%20(PCB)%20requirements.)
- Libreria Bluetooth Low Energy: NimBLE-Arduino:
<https://docs.arduino.cc/libraries/nimble-arduino/>
- Libreria WiFi:
<https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFi.h>
- Corso di Flutter su Code Academy:
<https://www.codecademy.com/learn/intro-to-flutter>
- Video Introduzione a Flutter:
https://youtu.be/3kaGC_DrUnw?si=PShjmNGqaFkNzGHg
- Postman:
<https://www.postman.com/>