

Relazione assignment 3
Smart Garden
Corso di “Embedded systems and IoT”

Filippo Pilutti

23 giugno 2024

Indice

1	Introduzione	3
2	Garden SensorBoard	3
3	Garden Service	4
4	Garden Dashboard	5
5	Garden Controller	6
5.1	Task	6
5.2	Variabili globali	7
5.3	Macchine a stati finiti	7
6	Garden Mobile App	9
7	Sistema hardware	10

1 Introduzione

L'obiettivo di questo terzo assignment era quello di creare un prototipo di un sistema IoT per implementare un prototipo di un giardino smart, ovvero un sistema smart in grado di monitorare e controllare lo stato di un giardino.

Questo sistema é composto da 5 diversi sottosistemi:

- **Garden SensorBoard** - ESP32
- **Garden Service** - Backend PC
- **Garden Dashboard** - Frontend PC
- **Garden Controller** - Arduino
- **Garden Mobile App** - Android

Ognuna di queste componenti svolge una specifica funzione e deve comunicare con le altre per garantire il corretto funzionamento dell'intero sistema. In particolare, il Garden Service funziona come centro di controllo centrale, ricevendo i dati dei sensori dal Garden SensorBoard tramite HTTP e inviandoli, sempre tramite HTTP, al Garden Dashboard e alla Garden Mobile App. Inoltre, comunica con il Garden Controller su Arduino tramite la linea seriale e la Garden Mobile App comunica con il Garden Controller tramite un collegamento Bluetooth.

La parte hardware di questo sistema comprende due dispositivi differenti: una scheda ESP32 che raccoglie dati attraverso un fotoresistore e un sensore di temperatura analogico TMP36, e un microcontrollore Arduino UNO che controlla fisicamente il giardino tramite quattro LED verdi e un servomotore, collegandosi all'applicazione Android tramite un modulo Bluetooth HC-05.

La parte software prevede, oltre ai programmi C++ caricati sulle due schede, un servizio eseguito sulla porta locale 8080, una semplice interfaccia utente per visualizzare i dati e lo stato complessivo del sistema, e un'app Android, tutti sviluppati in linguaggio Java.

2 Garden SensorBoard

Questa componente raccoglie i dati fisici di luminosità e temperatura del giardino e li invia al Garden Service. Per fare ciò, utilizza una scheda ESP32 a cui sono collegati un fotoresistore e un sensore analogico di temperatura. Il sottosistema include anche un LED verde, che si spegne nel caso in cui venga rilevato uno stato di allarme. La scheda ESP32 offre funzionalità di

connettività wireless, permettendo di collegarsi a una rete Wi-Fi e inviare i dati raccolti sotto forma di oggetti JSON tramite HTTP. È importante sottolineare che il servizio con cui si comunica è attivo su localhost:8080 e viene reso accessibile dall'esterno utilizzando un servizio di tunneling come ngrok.

Il programma caricato sulla scheda è stato sviluppato in C++ con l'IDE Visual Studio Code utilizzando il framework Arduino per la gestione dei pin, seguendo lo schema *setup()* e *loop()*. Il programma prevede:

- connessione alla rete WiFi: per gestire la connessione alla rete viene utilizzata la libreria WiFi utilizzando SSID e password.
- gestione sensori: sono state create due classi apposite (TempSensor, PhotoResistor) per la lettura dei dati raccolti dai due sensori.
- invio dei dati: per l'invio dei dati viene utilizzata la libreria HTTP-Client per inviare una stringa in formato JSON attraverso un HTTP POST all'endpoint `/api/data`.
- rilevazione dello stato: il programma rileva lo stato attuale del sistema inviando una richiesta HTTP GET all'endpoint `/api/status`. Il parsing dell'oggetto JSON ricevuto viene effettuato utilizzando la libreria ArduinoJson.

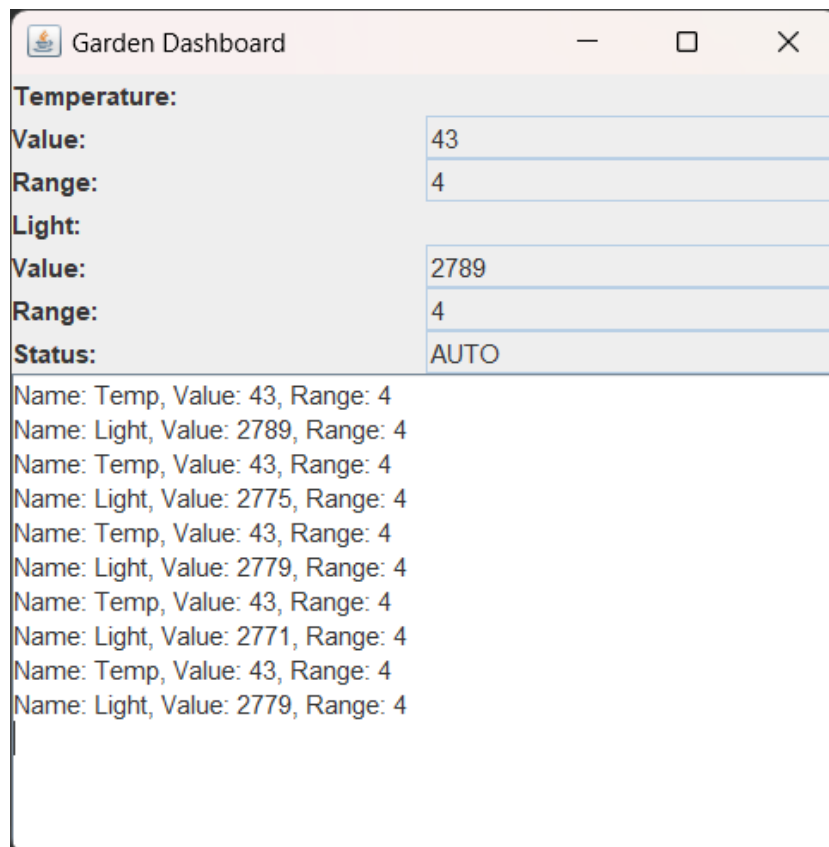
3 Garden Service

La componente Garden Service funge da centro di controllo dell'intero sistema, permettendo e gestendo la comunicazione tra tutti i vari sottosistemi. Per fare ciò utilizza un servizio che viene eseguito sulla porta locale 8080 e viene reso accessibile con un tunnelling tramite ngrok. Il servizio, chiamato DataService, è stato sviluppato in Java su IDE Eclipse utilizzando il toolkit Vert.x, che offre un ambiente di esecuzione asincrono per applicazioni reattive e per la gestione efficiente di richieste HTTP. Il servizio si occupa di:

- gestione comunicazione seriale: attraverso la classe SerialCommChannel e utilizzando la libreria jssc per la gestione delle porte seriali, rende possibile l'invio e la ricezione di messaggi con Arduino attraverso la sua linea seriale.
- monitoraggio dello stato del sistema: il servizio periodicamente legge lo stato del sistema attraverso la comunicazione seriale e lo rende visibile agli altri componenti.

- gestione richieste HTTP GET e POST: il servizio gestisce le richieste HTTP GET e POST sull'endpoint `/api/data` per la comunicazione dei dati raccolti dai sensori, mentre per l'endpoint `/api/status` gestisce solo le richieste HTTP GET. Ogni volta che il servizio riceve un HTTP POST su `/api/data` comunica il dato corrispondente ad Arduino attraverso il canale di comunicazione seriale.
- gestione dei dati: il servizio mantiene una lista di dieci oggetti JSON che rappresentano gli ultimi dieci dati ricevuti dal Garden SensorBoard e un oggetto JSON che rappresenta lo stato attuale del sistema. Gli oggetti JSON rappresentanti i dati hanno tre campi: `name`, `value` e `range`, mentre quelli rappresentanti lo stato ne hanno solo uno: `status`.

4 Garden Dashboard



Temperature:	
Value:	43
Range:	4
Light:	
Value:	2789
Range:	4
Status:	AUTO

```

Name: Temp, Value: 43, Range: 4
Name: Light, Value: 2789, Range: 4
Name: Temp, Value: 43, Range: 4
Name: Light, Value: 2775, Range: 4
Name: Temp, Value: 43, Range: 4
Name: Light, Value: 2779, Range: 4
Name: Temp, Value: 43, Range: 4
Name: Light, Value: 2771, Range: 4
Name: Temp, Value: 43, Range: 4
Name: Light, Value: 2779, Range: 4

```

Figura 1: Interfaccia utente Garden Dashboard.

Questa componente consiste in una semplice interfaccia utente sviluppata in Java Swing su IDE Eclipse per la visualizzazione dei dati raccolti dai sensori del Garden SensorBoard e dello stato attuale del sistema. Questo programma utilizza la classe `Timer` di java per aggiornare i dati visualizzati ogni 2 secondi attraverso i metodi `fetchData()` e `fetchStatus()`. Questi due metodi effettuano richieste HTTP GET rispettivamente sugli endpoint `/api/data` e `/api/status`, aggiornando l'interfaccia con i nuovi dati ottenuti. Le richieste HTTP sono gestite sempre utilizzando `Vert.x`.

5 Garden Controller

Il sottosistema Garden Controller prevede un sistema embedded basato su un microcontrollore Arduino UNO per la gestione fisica del giardino. Prevede la gestione di due impianti, uno per l'irrigazione basato su un servo motore e uno per l'illuminazione basato su quattro led verdi. Nella progettazione del sistema, è stata scelta un architettura basata su task, modellati attraverso macchine a stati finiti sincrone, rendendo possibile una scomposizione e modularizzazione dei suoi comportamenti.

Per la gestione degli attuatori sono state sviluppate delle relative classi: `Led`, `LedExt` e `ServoMotor`. La comunicazione con il Garden Service è gestita in una classe `MsgService` che permette di inviare e ricevere messaggi attraverso la linea seriale, mentre la classe `MsgServiceBT` sfrutta `SoftwareSerial` per inviare e ricevere messaggi attraverso la connessione bluetooth con il modulo HC-05.

5.1 Task

In fase di progettazione, ho scelto di scomporre il comportamento generale del sistema in tre task. I vari task sono stati sviluppati in delle corrispondenti classi, e la loro esecuzione è gestita da uno scheduler sviluppato per eseguirli a periodi di tempo predefiniti.

- **GardenStateTask:** rappresenta gli stati principali in cui il sistema può trovarsi durante la sua esecuzione, modifica le variabili globali per influenzare gli stati degli altri task e si occupa di comunicare con il Garden Service attraverso la linea seriale. Gli stati possibili sono: *auto*, *manual*, *alarm*. Il suo periodo è di 100 millisecondi.
- **LampsTask:** modella il comportamento dell'impianto di illuminazione, si occupa quindi di gestire l'accensione e lo spegnimento dei quattro led in modo automatico o manuale. I suoi stati possibili sono: *idle*,

auto-operating, *manual-operating*. Il suo periodo di esecuzione é di 200 millisecondi.

- **IrrigationTask**: modella il comportamento dell'impianto di irrigazione, gestisce quindi l'azionamento del servo motore in modo automatico o manuale. Gli stati possibili sono: *idle*, *auto-operating*, *manual-operating*, *pause*. Il suo periodo é di 100 millisecondi.

5.2 Variabili globali

Sono state utilizzate diverse variabili globali, salvate in un file `globals.h`, per consentire l'interazione tra i diversi task, in particolare variabili che indicano lo stato globale del sistema (*autoState*, *manualState*, *alarmState*), una variabile che segnala una particolare condizione del sistema (*irrigating*), variabili che tengono traccia di dati (*lightRange*, *tempRange*) e variabili che gestiscono l'interazione con gli attuatori (*manualLed1On*, *manualLed2On*, *manualLed3Intensity*, *manualLed4Intensity*, *manualServoOn*, *manualServoSpeed*).

5.3 Macchine a stati finiti

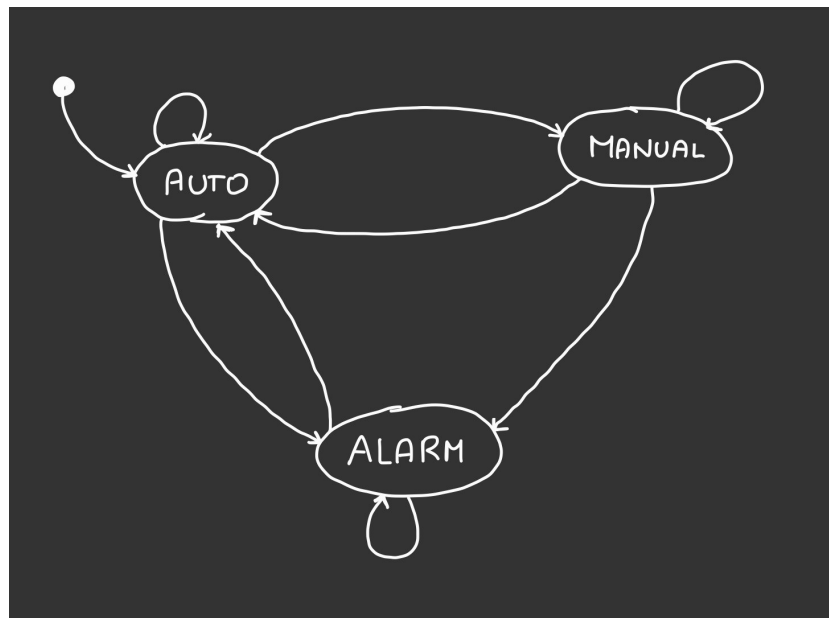


Figura 2: Macchina a stati finiti per GardenStateTask.

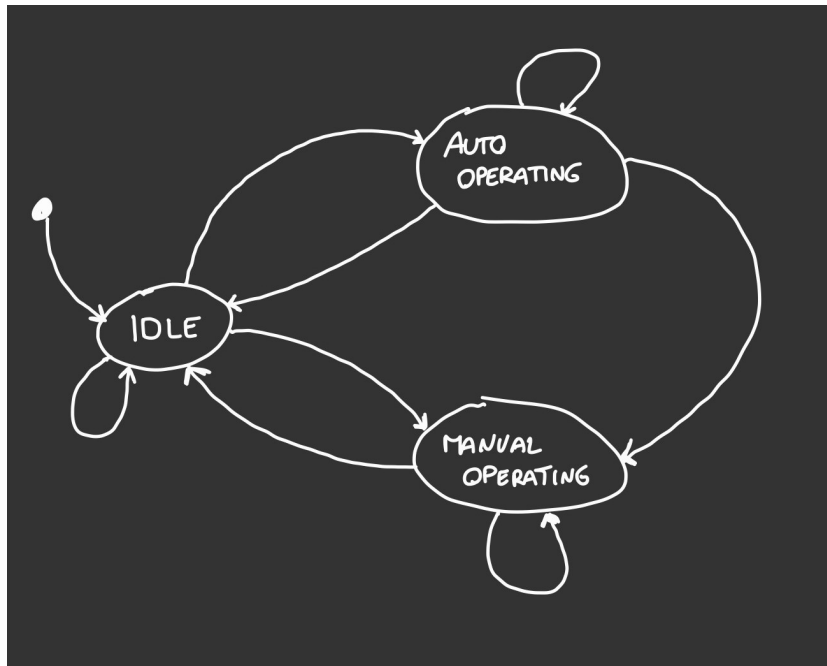


Figura 3: Macchina a stati finiti per LampsTask.

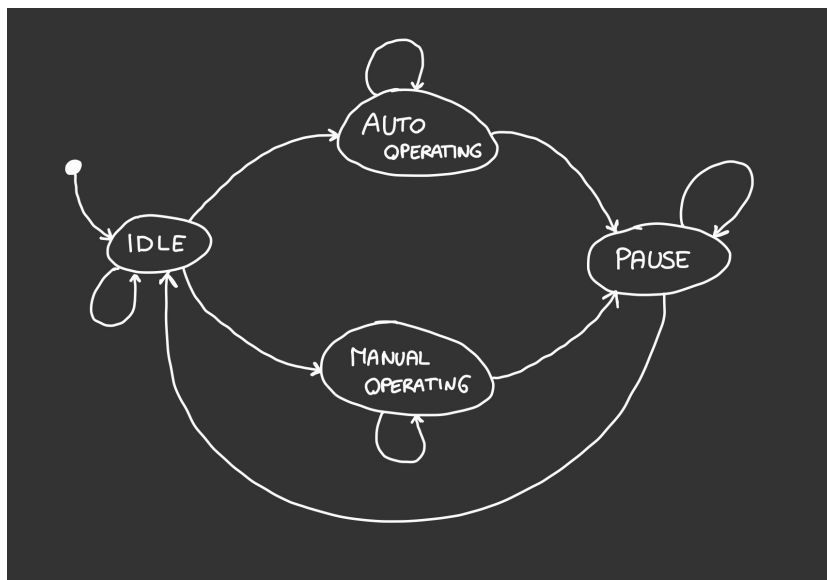


Figura 4: Macchina a stati finiti per IrrigationTask.

6 Garden Mobile App

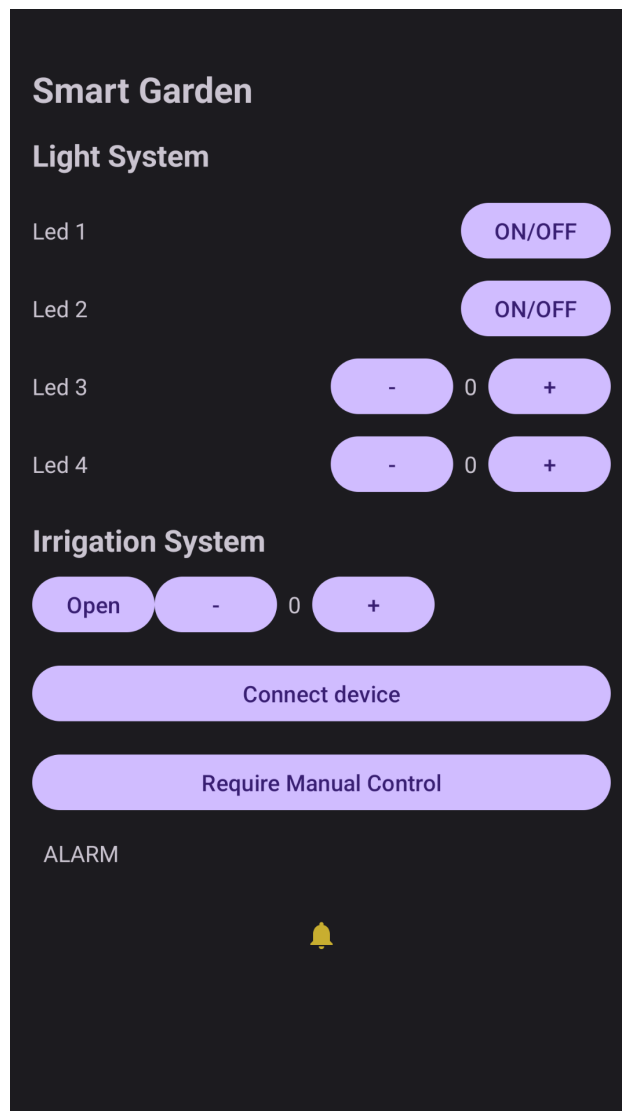


Figura 5: Screenshot dell'interfaccia utente della Garden Mobile App - stato di allarme

L'applicazione Garden Mobile App é stata sviluppata in linguaggio Java su IDE Android Studio. Si compone di una activity principale MainActivity, il cui layout é definito nel file activity_main.xml, che si occupa di gestire la connessione e la comunicazione bluetooth al modulo HC-05 del Garden Controller e le richieste HTTP GET al Garden Service per ottenere lo stato

attuale del sistema. Per la gestione delle richieste HTTP é stata utilizzata la libreria OkHTTP creando un metodo *getSystemStatus()* mandato in esecuzione periodicamente su un thread separato.

Per poter utilizzare le funzionalità del bluetooth e della connessione ad internet é stato necessario aggiungere i relativi permessi al file *AndroidManifest.xml*.

7 Sistema hardware

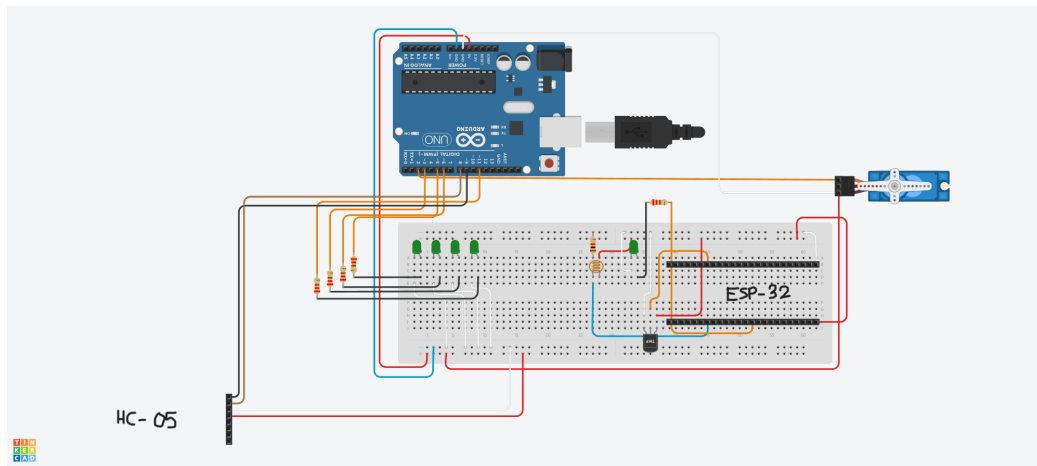


Figura 6: Schema collegamenti dei dispositivi

Link video

https://drive.google.com/file/d/1KU6f7kQjX_McoBHGKILK8pd4hr58AAP_/view?usp=drive_link