UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

ALETHEIA

AgriTech

Versione 1.0.0

Data di rilascio:

03/06/2025

Sistemi Cooperativi A.A. 2024-2025

[Informatica e Tecnologie per la Produzione del Software]

Realizzato da

Calabrese Lorenzo 779727 l.calabrese28@studenti.uniba.it

Conforti Francesco 776628 f.conforti9@studenti.uniba.it

De Biase Giuseppe Pio 774957 g.debiase5@studenti.uniba.it

Zaharia Alexandru Gheorghe 774747 a.zaharia@studenti.uniba.it

Dipartimento di Informatica - Università degli Studi di Bari
Via Orabona, 4 - 70125 - Bari
Tel: +39.080.5443270 • +39.080.5442300 | Fax: +39.080.5442536

Laboratorio Aletheia aletheia.aziendale@gmail.com

Indice

INTRODUZIONE

AGRITECH

1. Descrizione informale del contesto e degli obiettivi

1.1 Contesto di business

Il progetto ha l'obiettivo di sviluppare un digital-twin per la protezione delle vigne da eventi atmosferici come pioggia e grandine. Attraverso l'uso di strutture smart, il sistema è in grado di monitorare in tempo reale le condizioni meteo e attivare in maniera autonoma un meccanismo di protezione tramite l'uso di tende .Queste, non solo difendono le colture dalle intemperie, ma sono anche progettate per raccogliere l'acqua piovana , convogliandola in apposite cisterne. Il sistema permette di monitorare e gestire il quantitativo d'acqua immagazzinato, ottimizzando l'uso delle risorse e migliorando la gestione idrica.

1.1.1 Problemi Attuali

Nonostante i progressi nelle tecnologie, sono stati riscontrati i seguenti problemi:

1. **Monitoraggio meteo**

   L'assenza di un monitoraggio meteo, in tempo reale e preciso, espone i vigneti a gravi rischi. Le previsioni meteo spesso non sono abbastanza accurate o tempestive, portando a situazione impreviste come piogge intense o grandinate che possono distruggere interi raccolti. L'incapacità di monitorare e anticipare questi eventi aumenta il rischio di ingenti perdite, compromettendo la qualità e la quantità del raccolto annuale.

2. **Protezione inadeguate dalle Intemperie**

   Le pratiche tradizionali di protezione dai fenomeni atmosferici come la grandine o le piogge forti sono spesso insufficienti. Le reti antigrandine e coperture temporanee, pur essendo utili, non offrono sempre una protezione adeguata in caso di eventi estremi o prolungati, lasciando le colture vulnerabili.

3. **Elevato Costo della Protezione Manuale**

   I metodi tradizionali di protezione come l'installazione manuale di reti antigrandine o l'uso di strutture temporanee sono costosi e richiedono molta manutenzione. Questa comporta non solo un dispendio di risorse finanziarie, ma anche un forte impegno in termine di tempo e lavoro da parte degli agricoltori.

1.1.2 Obiettivi

1. **Automazione della Protezione dalle Intemperie**

   Creare un sistema di protezione automatica tramite tende intelligenti che si attivano automaticamente in base alle condizioni meteo rilevate, riducendo la necessità di intervento umano e proteggendo efficientemente le colture da fenomeni atmosferici come la grandine e le forti piogge. Le tende devono adattarsi alle diverse intensità degli eventi atmosferici

2. **Raccolta e Gestione Efficiente dell'Acqua Piovana**

   Integrare una funzionalità che consenta la raccolta dell'acqua piovana, convogliando in cisterne per l'accumulo e l'uso futuro. L'obiettivo è ottimizzare l'uso delle risorse idriche, riducendo il consumo di acqua potabile per l'irrigazione e migliorare la sostenibilità delle coltivazioni

3. **Prevenzione e Riduzione dei Danni ai Raccolti**

   Minimizzare i danni ai raccolti causati da eventi atmosferici estremi, garantendo una protezione adeguata che riduca al minimo le perdite di prodotto, migliorando quindi la qualità e la quantità del raccolto annuale

ARCHITETTURA COMPLESSIVA

AGRITECH

2. ARCHITETTURA COMPLESSIVA

2.1 Architettura generale

Il sistema si articola in **sei componenti fondamentali**, che cooperano per garantire un flusso di dati efficiente e sicuro:

1. **Sensoristica e Hardware**
   - Composta da dispositivi basati su **ESP32** e altri sensori.
   - La comunicazione con i dispositivi avviene tramite **porta seriale (COM)** utilizzando **l'IDE di Arduino**.
   - I dati vengono rilevati e preprocessati localmente.
2. **Arduino IDE**
   - Strumento per la programmazione e l'interfaccia diretta con la scheda hardware.
   - Invia i dati al backend tramite **chiamate HTTP**.
3. **Backend e API**
   - Il backend è sviluppato in **Python con FastAPI** e viene eseguito in un ambiente **containerizzato (Docker)**.
   - Riceve i dati in formato JSON, li elabora e li memorizza nel **database MongoDB**.
   - Il backend converte i dati tra i formati **BSON** e **JSON** per l'integrazione ottimale con MongoDB.
4. **Database (Mongo DB)**
   - Sistema di archiviazione dati NoSQL, responsabile della memorizzazione dei dati rilevati e delle elaborazioni successive.
   - Integrato nel container per una gestione coerente e portabile.
5. **Sicurezza e Comunicazione (Cloudflare Tunnel)**
   - La comunicazione tra il server Linux e il backend utilizza **Cloudflare Tunnel**.
   - Il tunnel garantisce un canale sicuro tramite **QUIC** e **HTTP/2**, riducendo la latenza e migliorando la sicurezza.
6. **Integrazione con FlexSim**
   - Il backend, una volta elaborati i dati, comunica via **HTTP** con **FlexSim**, un ambiente di simulazione 3D.
   - Questo consente di valutare scenari e simulazioni in tempo reale in base ai dati rilevati e processati.

Diagramma architetturale del sistema

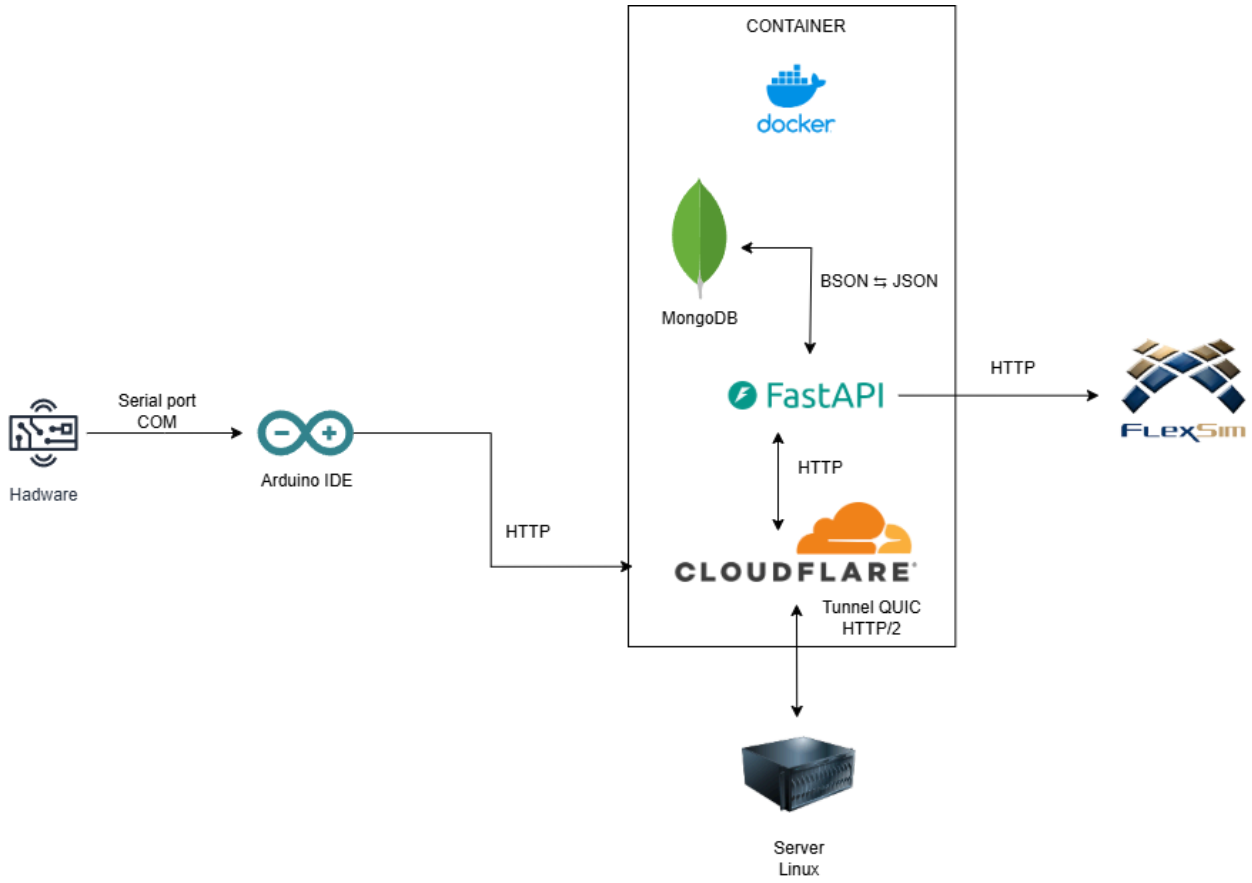


Figura 1. Diagramma architetturale del sistema

Diagramma logico del sistema

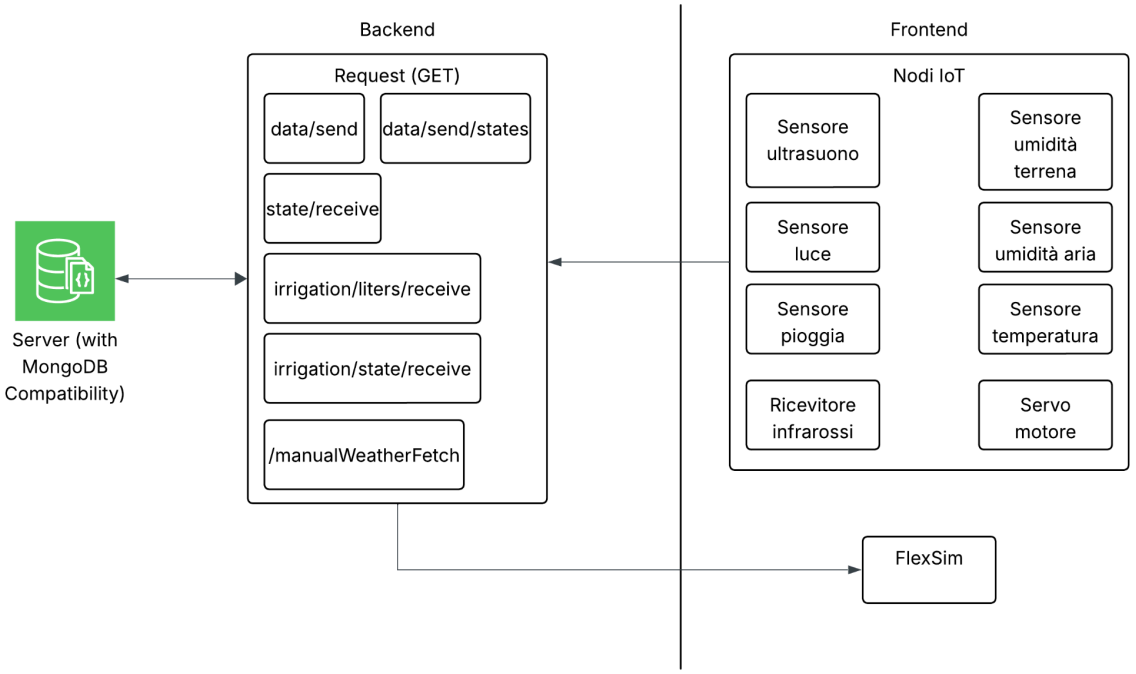


Figura 2 Diagramma logico del sistema.

2.2 Componenti principali

Componente	Ruolo principale
FlexSim	Esegue la logica di simulazione, riceve input dagli eventi reali e cambia lo stato degli elementi della simulazione.
Nodo IoT	Raccolgono dati dai sensori e li inviano ai microservizi tramite ESP 32 con chiamate API.
Microservizi	Espongono FastAPI per l'interscambio di messaggi, gestiscono i dati e la logistica di integrazione
Database NON Relazionale	Archivia i dati di evento

2.3 Tecnologie

Nome	Funzione	Utilizzo nel progetto
FlexSim	Software per la simulazione di aspetti della vita reale	Simulerà: • l'apertura e chiusura della tenda • Il riempimento della cisterna di acqua • l'irrigazione del campo
FastAPI	Libreria di Python per la gestione e creazione di chiamate API	FastAPI è stato utilizzato per la creazione degli endpoint necessari per il funzionamento del sistema.
SketchUP	Applicativo per la modellazione 3D	Ci permette di creare il modello 3D del vigneto, della tenda, della cisterna e dell'impianto idraulico
ESP32	Microcontrollore	ESP32 è stato utilizzato per la lettura dei sensori e per l'invio dei dati al Database
Docker	Software utilizzato per la containerizzazione dell'applicazione	Contenitore della sezione Backend e Database del progetto
MongoDB	Database NoSQL	Conterrà i dati ricevuti dai sensori
CloudFlare	Piattaforma che funge da CDN (Content Delivery Network)	CloudFlare è stato utilizzato per creare un tunnel virtuale che permette di accedere al backend del progetto attraverso un DNS personalizzato in maniera pubblica.

2.4 Flusso dei dati e interazioni

2.4.1 Rilevamento sensore pioggia -> FlexSim

- Il nodo IoT effettua una chiamata HTTP POST ogni 2.5 secondi, all endpoint /data/send del microservizi FastAPI;
- Quando il sensore rileva un cambiamento di stato, modifica il valore che il nodo IoT sta inviando;
- Il Backend valida il payload e lo salva nel DB ("Sensoristica") e nella collezione ("Record");
- Una serie di funzioni dedicate analizza i dati e valuta il tipo di dato da inviare a FlexSim;
- Il FlexSim chiama l'API di riferimento, ogni 2.5 secondi, a /state/receive per aggiornare lo stato della simulazione e far aprire o chiudere la tenda.

2.4.2 Rilevamento sensore ultrasuoni -> FlexSim (Raccolta acqua)

- Il nodo IoT effettua una chiamata HTTP POST ogni 2.5 secondi, all endpoint /data/send del microservizi FastAP;
- Quando il sensore rileva un cambiamento di stato, modifica il valore che il nodo IoT sta inviando;
- Il Backend valida il payload e lo salva nel DB ("Sensoristica") e nella collezione ("Record");
- Una serie di funzioni dedicate analizza i dati e calcola il litraggio all'interno della cisterna per inviare il valore corrispettivo a FlexSim;
- Il FlexSim chiama l'API di riferimento /irrigation/litres/receive per aggiornare lo stato della simulazione;

2.4.3 Rilevamento sensore temperatura -> FlexSim (Irrigazione)

- Il nodo IoT effettua una chiamata HTTP POST ogni 2.5 secondi, all endpoint /data/send del microservizi FastAP;
- Quando il sensore rileva un cambiamento di stato, modifica il valore che il nodo IoT sta inviando;
- Il Backend valida il payload e lo salva nel DB ("Sensoristica") e nella collezione ("Record");
- Una serie di funzioni dedicate analizza i dati e valuta, in base ad una soglia minima, se aprire i tendoni ed irrigare il terreno per l'alta temperatura;
- Il FlexSim chiama l'API di riferimento /irrigation/state/receive per aggiornare lo stato della simulazione per irrigare;

2.4.4 Innesco manuale apertura tenda -> FlexSim

- Quando premi il pulsante 2 del telecomando a infrarossi, il ricevitore capta il segnale;
- Una volta intercettato il segnale, il nodo IoT accende il led rosso sulla breadboard;
- Successivamente invia all endpoint /data/send/states un JSON, contenente lo stato della struttura (Aperto/Chiusa);
- Il Backend valida il payload e lo salva nel DB ("Sensoristica") e nella collezione ("States");
- Una logica dedicata verifica l'esistenza di cambiamenti di stato manuali e darà priorità a tale valore;
- Il FlexSim chiama l'API di riferimento /state/receive per aggiornare lo stato della simulazione;

2.4.5 Innesco manuale irrigazione -> FlexSim

- Quando premi il pulsante 1 del telecomando a infrarossi, il ricevitore capta il segnale;
- Una volta intercettato il segnale, il nodo IoT accende il led blu sulla breadboard;
- Successivamente invia all endpoint /data/send/states un JSON, contenente lo stato della struttura (Aperto/Chiusa);
- Il Backend valida il payload e lo salva nel DB ("Sensoristica") e nella collezione ("States");
- Una logica dedicata verifica l'esistenza di cambiamenti di stato manuali e darà priorità a tale valore;
- Il FlexSim chiama l'API di riferimento /irrigation/state/receive per aggiornare lo stato della simulazione per irrigare;

2.5 Elenco attività

Tipologia	Nome	Funzione
Endpoint	/data/send	Invio dei dati ricevuti dai sensori al Backend
Endpoint	/data/send/states	Invio dello stato modificato manualmente al Backend
Endpoint	/state/receive	Invio del cambio di stato della tenda
Endpoint	/irrigation/litres/receive	Invio del cambio di stato della cisterna in base al litraggio calcolato
Endpoint	/irrigation/state/receive	Invio cambio di stato dell'irrigazione
Funzione	get_litres()	Lettura del volume di liquido esistente nella cisterna.
Funzione	get_status()	Lettura dello stato di apertura della struttura.
Funzione	value_insert()	Inserimento dei valori inviati da Arduino nel Database
Funzione	get_irrigation_state()	Lettura dello stato di funzione dell'irrigazione.

PROGETTAZIONE E IMPLEMENTAZIONE DEI MODELLI DI SIMULAZIONE

AGRITECH

3. Progettazione e implementazione dei modelli di simulazione

3.1 Layout di simulazione

3.1.1 Descrizione dell'ambiente

Il modello simulato in Flexsim riproduce un sistema agricolo intelligente per la protezione e la gestione idrica delle vigne, progettato per reagire automaticamente a condizioni atmosferiche avverse e per ottimizzare la raccolta e l'uso dell'acqua piovana.

Il layout rappresenta una superficie **20 m x 10 m** suddivisa in **4 filari** paralleli **lunghi 14,80 m** ciascuno, **alto 1,50 m**, separati da corridoi di servizio di 2 m, ciascuno dotato di:

- Struttura portante per tende intelligenti motorizzate ad un'**altezza di 2 m**;
- Sensori meteorologici;
- Sistema idrico integrato nei pali, collegando le tende alla rete idrica sotterranea;
- Cisterna, posizionata a 2,5 m sotterranea, per la raccolta dell'acqua piovana e fonti esterne.
- Tubi per l'irrigazione posizionati ad un'altezza di 0,70 m dal suolo

Le tende sono progettate per **chiudersi automaticamente** in caso di pioggia o grandine, convogliando l'acqua raccolta verso una **cisterna sotterranea** attraverso un sistema di **tubazioni integrate nei pali di supporto**.

Lo stesso principio viene utilizzato anche per **l'irrigazione automatica dei filari**, che può essere attivata sia in modo autonomo dal sistema, sia manualmente dall'agricoltore.

Inoltre, le tende possono essere **aperte o chiuse manualmente** anche tramite un **telecomando in dotazione**, fornito insieme alla struttura.

Figura 2. Tende aperte

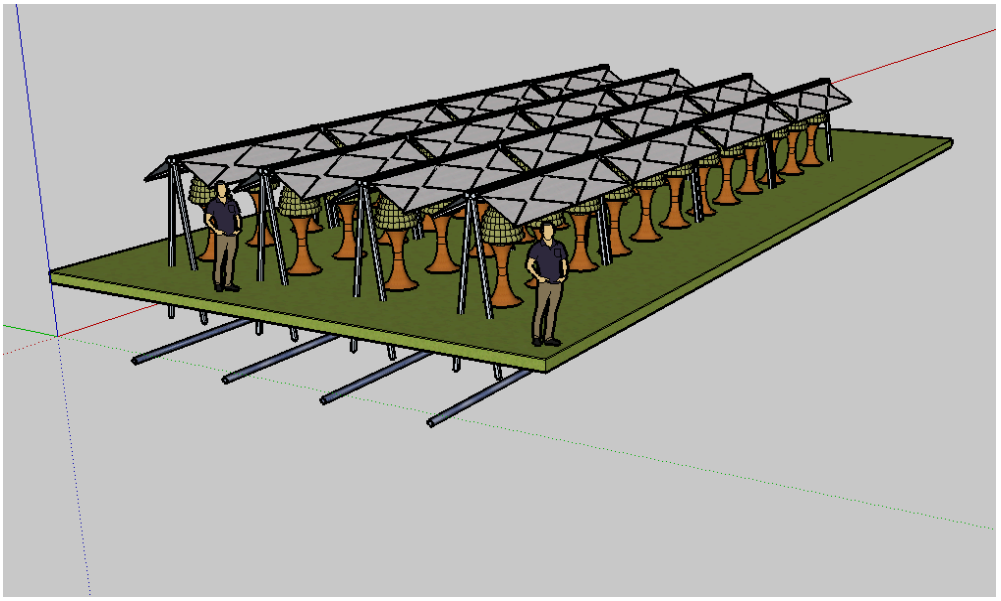
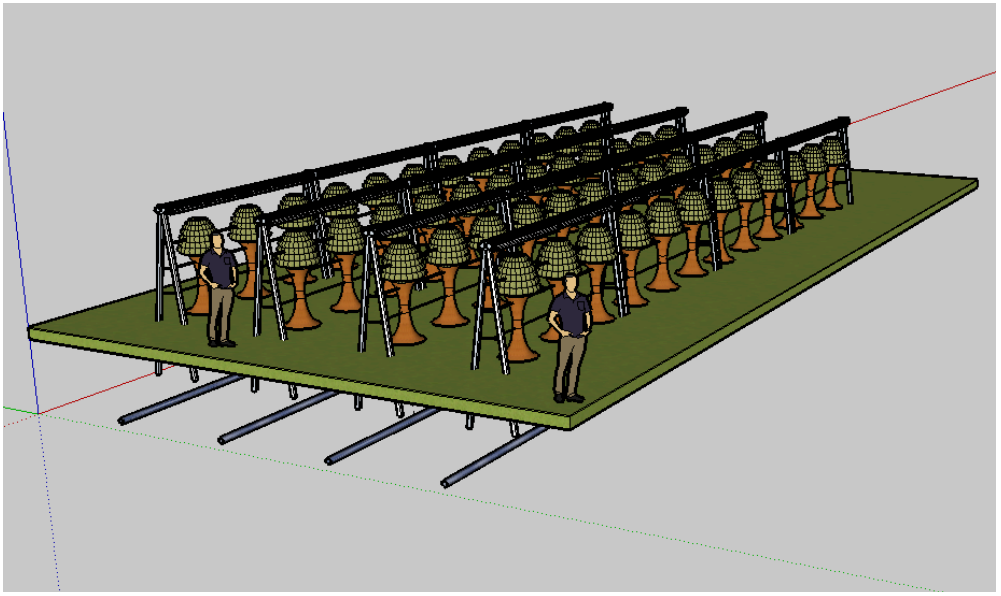


Figura 3. Tende chiuse



3.2 Elementi principali del layout

3.2.1 Variante apertura tende

Elemento	Tipo FlexSim	Descrizione
Filare di viti	**Shape 3D custom**	Rappresentato come serie di blocchi “Shape”. Ogni blocco è distanziato a 1,2 m l’uno dall’altro.
Tenda protettiva	**Fixed resource**	Modello “fixes resource” ha il compito di proteggere le viti; Si apre/chiude in base ad un Send Message.
Source	**Source**	Scriptato come blocco “Source": riceve comandi via Send Message per l’avvio o lo stop della generazione di fluidi per la simulazione dell’acqua piovana.
Cisterna	**Sink**	Elemento "Sink" che simula la raccolta dell’acqua piovana convogliata dalle tende; Nella simulazione ha il compito di eliminare l’elemento creato dal source.
Tubazione raccolta acqua	**Fixed resources**	Simulano il convogliamento dell’acqua dalla tenda alla cisterna.

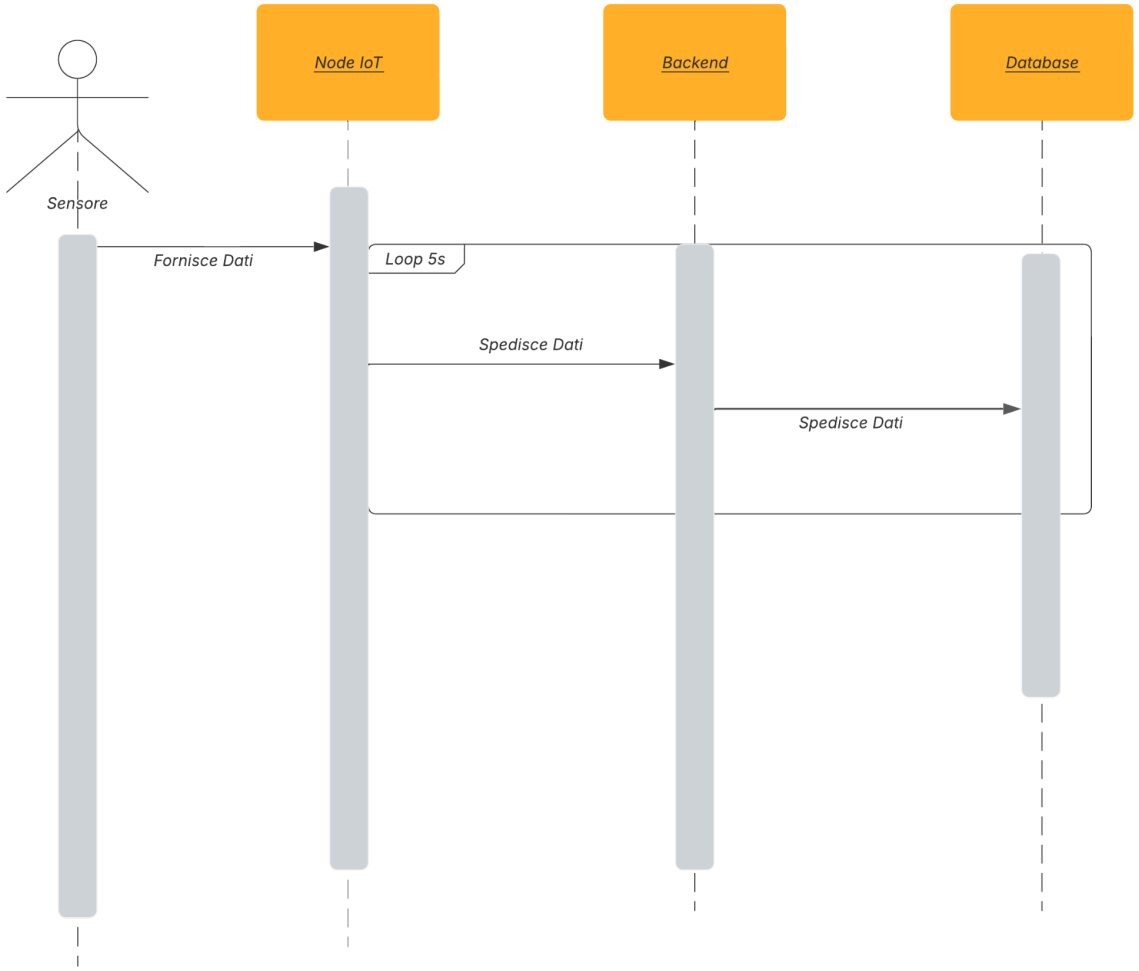
3.2.2 Variante irrigazione e raccolta d'acqua

Elemento	Tipo FlexSim	Descrizione
Filare di viti	**Shape 3D custom**	Rappresentato come serie di blocchi "Shape". Ogni blocco è distanziato a 1,2 m l'uno dall'altro.
Tenda protettiva	**Shape 3D custom**	Rappresenta le tende protettive.
Cisterna	**Source**	Rappresentata come un Source, la cisterna utilizza tre shape frame (Base Frame, Arancione e Verde). Ha il compito di immagazzinare l'acqua piovana, comunicare la quantità raccolta e avviare l'irrigazione.
Sistema di irrigazione	**Fixed resources**	Simulano l'irrigazione dell'irrigazione dalla cisterna al campo.
Sink nascosto	**Sink**	Il sink nascosto simula il punto di fine del flusso dell'acqua irrigata

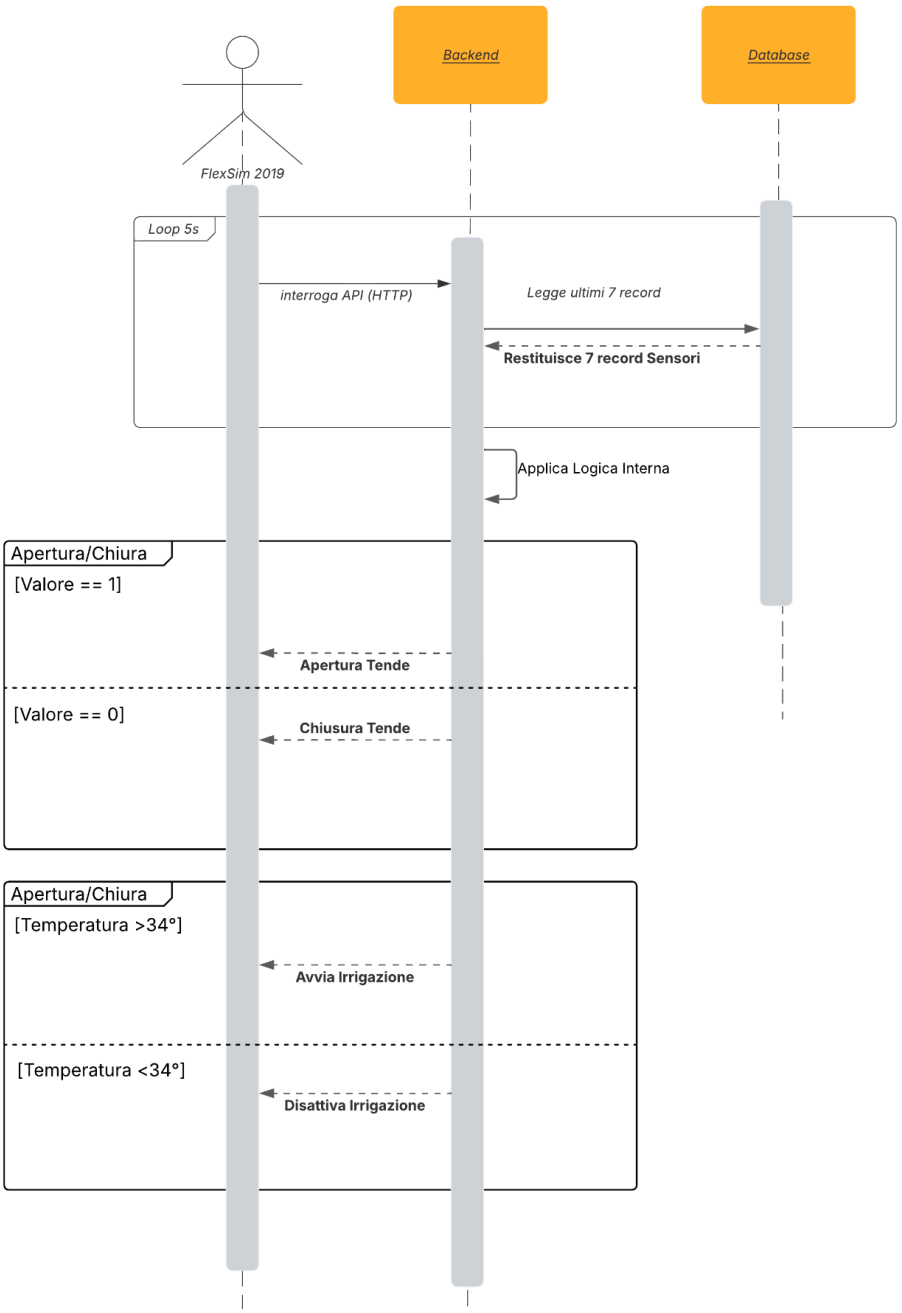
3.3 Regole di comportamento

3.3.1 Trigger ed eventi

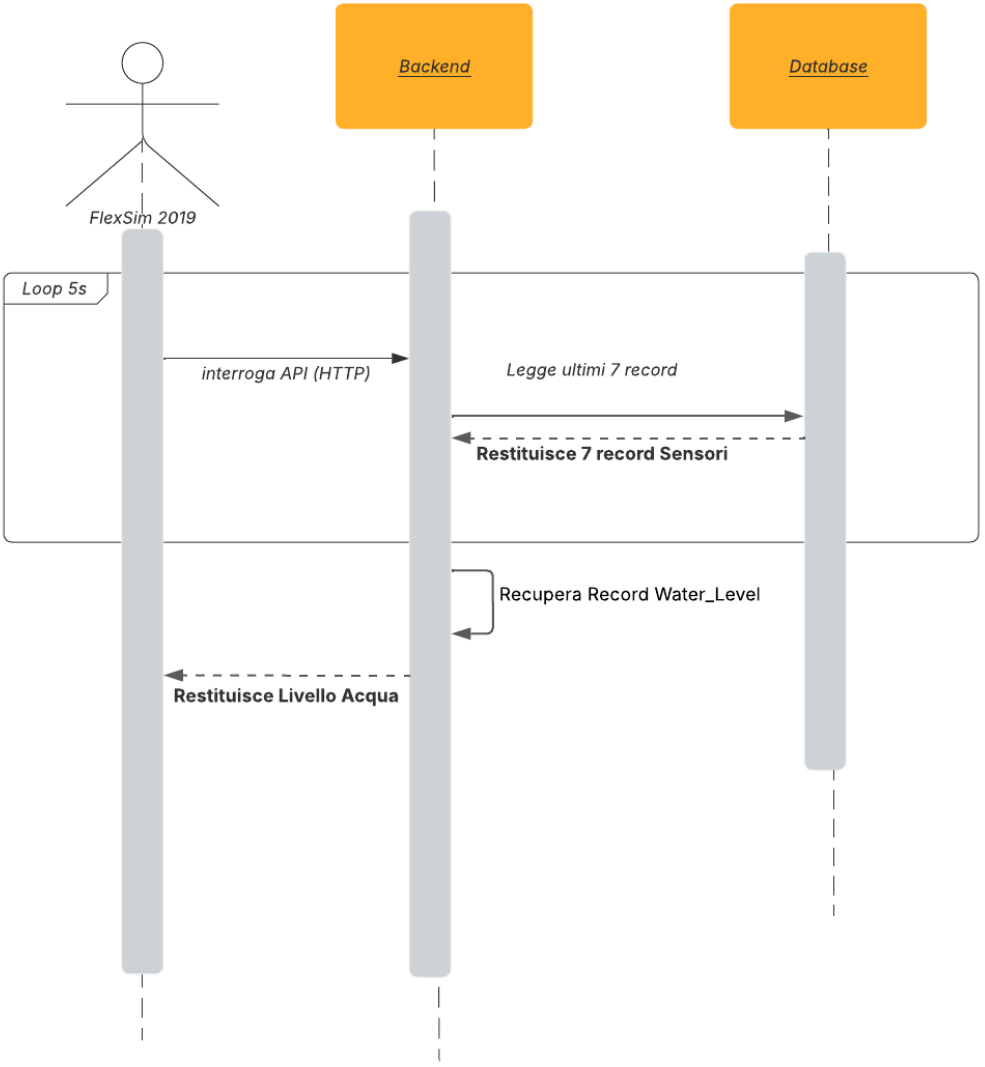
Comunicazione tra sensore e backend



FlexSim: Automazione tende e irrigazione con dati sensore



FlexSim: Recupera dati livello acqua



3.3.2 Logiche Temporali

Esse sono implementate tramite script FlexScript posizionati nei punti chiave del processo. Questi script gestiscono le interazioni dinamiche tra gli oggetti e coordinano le chiamate a servizi esterni per aggiornare in tempo reale gli stati e le visualizzazioni

3.3.2.1 Apertura/Chiusura tende e raccolta acqua

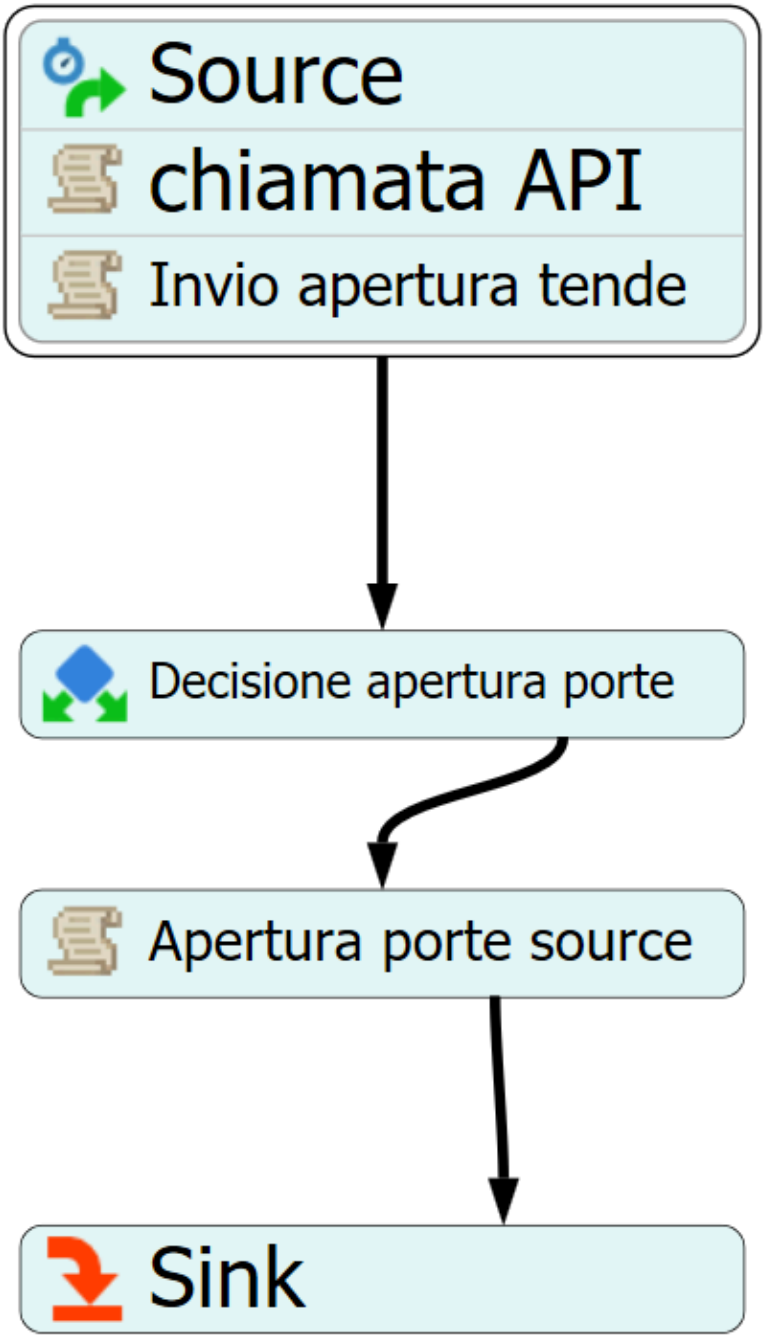
Durante questo processo, uno script personalizzato effettua una chiamata HTTP (API) verso il backend esterno. La logica prende in considerazione molteplici situazioni, tra cui:

a. Apertura manuale: In base alle scelte dell'utilizzatore, si possono aprire le tende premendo il tasto 2 del telecomando preso in dotazione..

b. Previsione meteo: La logica prende in considerazione la previsione meteo dell'ora successiva alla lettura attuale dei sensori. Se la previsione meteo indica che nell'ora successiva all'ora attuale c'è probabilità di temporale, grandine o acquazzone, vengono considerate le letture dei sensori (in particolare temperatura, umidità e luminosità) per calcolare un punteggio medio normalizzato. Questo punteggio medio normalizzato è ottenuto dalla media aritmetica dei tre indici (frequenza di pioggia, copertura nuvolosa e umidità), normalizzati su un intervallo tra 0 e 1. Se il valore medio supera una soglia predefinita del 53% (indicante la probabilità di eventi atmosferici intensi), la logica classifica la situazione come potenzialmente rischiosa e viene aperta la struttura di protezione.

In base alla risposta ricevuta, lo script FlexScript aggiorna la forma (shape) delle tende per riflettere visivamente questa condizione. Nello stesso momento, un source invia flowItem (acqua) verso una cisterna (sink) tramite dei tubi (convogliatori).

- Process Flow:



- Codice:

chiamata API - Custom Code

```
/**Custom Code*/
Object current = ownerobject(c);
Object item = param(1);
int rownumber = param(2);
Token token = param(3);
{ // ************* PickOption Start ************* //
/***Richiesta http*/
// Definizione dei parametri per la richiesta HTTP

Variant row = 1;
Variant col = 1;
string verb = "GET";
string server = "api.agritech.aletheialab.it";
string object = "/state/receive";
string data = "";
int silent = 0;
treenode result = node("/Tools/result",model());
applicationcommand("sendhttprequest", verb, server, object, data, silent, result);


    Variant risultato = gets(result);
    print("Valore:",risultato);
    token.stato = stringtonum(risultato);
    // Inserisci il valore nella cella

}
```

Invio apertura tende - Custom Code

```
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);
{ //************* PickOption Start *************\\
/***popup:SendMessage*/
/**Send Message*/
int NoDelay = -1;
print("Messaggio:", token.stato);
double delaytime = /** \nDelay Time: *//***tag:delaytime*//**/0/**list:NoDelay~0~10~current
treenode toobject = /** \nTo: *//***tag:to*//**/Model.find("Shape62")/**/;
treenode fromobject = /** \nFrom: *//***tag:from*//**/current/**/;
double param1 = /** \nParam1: *//***tag:par1*//**/(token.stato)/**/;
double param2 = /** \nParam2: *//***tag:par2*//**/0/**/;
double param3 = /** \nParam3: *//***tag:par3*//**/0/**/;
/**\n\nDelay Time:\nNoDelay: message sent immediately within trigger context\n0: delayed me
if (/** \nCondition: *//***tag:condition*//**/true/**/) {
    if (delaytime == NoDelay)
        sendmessage(toobject,fromobject,param1,param2,param3);
    else senddelayedmessage(toobject, max(0,delaytime), fromobject,param1,param2,param3);
}

} //******* PickOption End *******\\

```

Decisione apertura porte - Decision Point*

```
/**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);


// setto la global table
Table myTable = Table("StatoRaccoltaAcqua");

int row = 1;
int col = 1;

if (token.stato == 1) {
    // Aggiorna la cella con il valore ricevuto da backend
    myTable[row][col]=token.stato;

} else {
     myTable[row][col]=token.stato;

}

```

Apertura porte source - Custom Code

```
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);
{ //************** PickOption Start *************\\
/***popup:SendMessage*/
/**Send Message*/
int NoDelay = -1;
Table myTable = Table("StatoRaccoltaAcqua");
double delaytime = /** \nDelay Time: *//***tag:delaytime*//**/0/**list:NoDelay~0~10~currer
treenode toobject = /** \nTo: *//***tag:to*//**/Model.find("Source1")/**/;
treenode fromobject = /** \nFrom: *//***tag:from*//**/current/**/;
double param1 = /** \nParam1: *//***tag:par1*//**/myTable[1][1]/**/;
double param2 = /** \nParam2: *//***tag:par2*//**/0/**/;
double param3 = /** \nParam3: *//***tag:par3*//**/0/**/;
/**\n\nDelay Time:\nNoDelay: message sent immediately within trigger context\n0: delayed m
if (/** \nCondition: *//***tag:condition*//**/true/**/) {
    if (delaytime == NoDelay)
        sendmessage(toobject,fromobject,param1,param2,param3);
    else senddelayedmessage(toobject, max(0,delaytime), fromobject,param1,param2,param3);
}

} //******* PickOption End *******\\
```

3.3.2.2 Livello dell'acqua e irrigazione

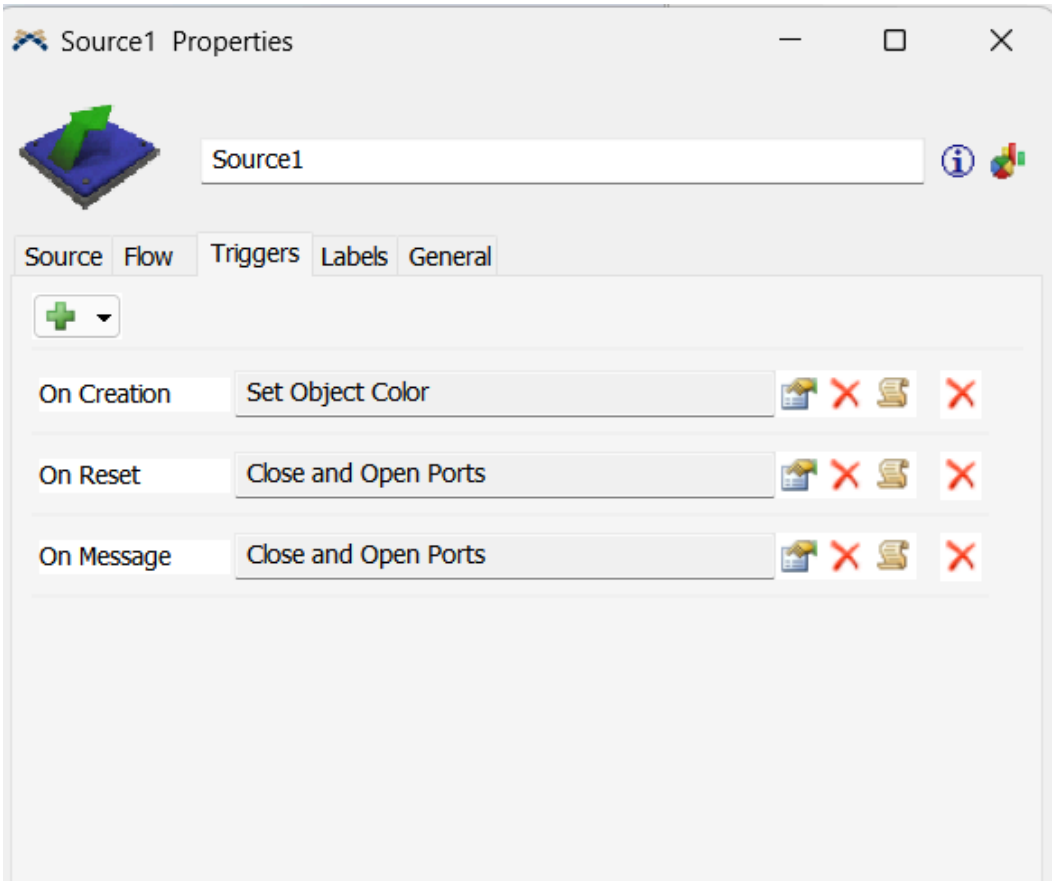
Durante questo processo, tramite un'API, il modello controlla la quantità di litri d'acqua presenti nella cisterna. Il backend restituisce il livello dell'acqua attuale, e in base a questo valore lo script FlexScript aggiorna la forma (shape) della cisterna con i seguenti colori:

- Verde: livello tra 9.000 e 15.000 litri
- Arancione: livello tra 5.000 e 9.000 litri
- Rosso: livello tra 0 e 5.000 litri

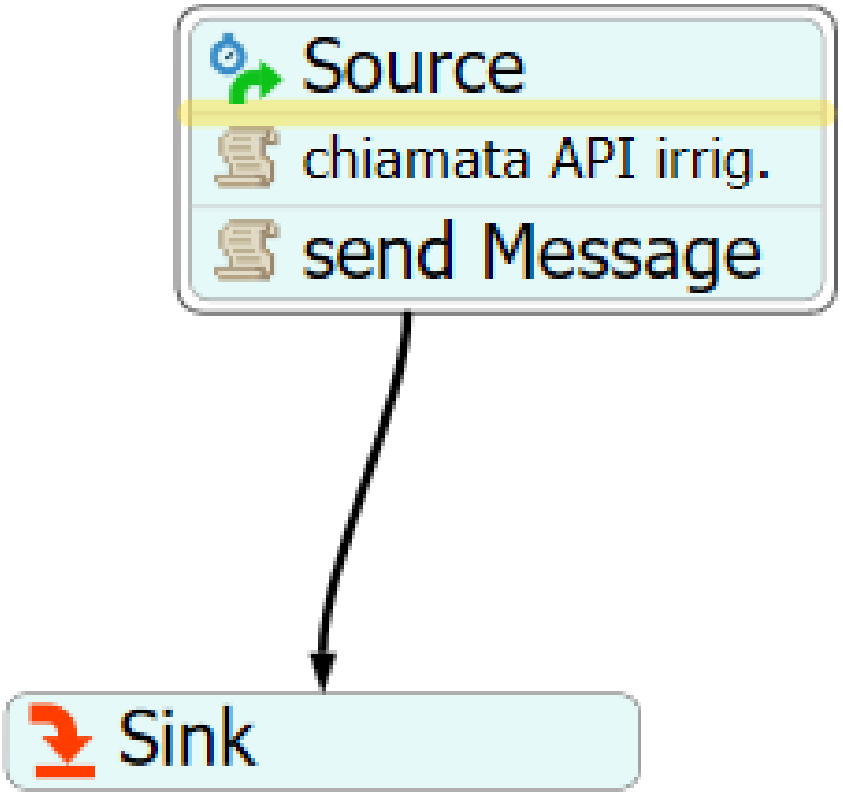
In questo modo, lo stato della cisterna è sempre visibile in tempo reale.

Nel modello le cisterne fungono da source che, al superamento di una temperatura di 34°C o l'avvio manuale, invia dei flowItem verso una coda (queue). Questa coda ha la funzione di smistare in maniera randomica l'acqua verso i convogliatori (tubi) per simulare l'irrigazione.

- Triggers



- Process Flow:



Codice:

chiamata API - Custom Code

```
/**Custom Code*/
Object current = ownerobject(c);
Object item = param(1);
int rownumber = param(2);
Token token = param(3);


{
Variant row = 1;
Variant col = 1;
string verb = "GET";
string server = "api.agritech.aletheialab.it";
string object = "/irrigation/state/receive";
string data = "";
int silent = 0;
treenode result = node("/Tools/result",model());
applicationcommand("sendhttprequest", verb, server, object, data, silent, result);


    Variant risultato = gets(result);


    token.irrigazione = stringtonum(risultato);



string server2 = "api.agritech.aletheialab.it";
string object2 = "/irrigation/litres/receive";
string data2 = "";
int silent2 = 0;
treenode result2 = node("/Tools/result",model());
applicationcommand("sendhttprequest", verb, server2, object2, data2, silent2, result2);


    Variant risultato2 = gets(result);
    print("Acqua presente:",risultato2);


    token.cisterna = stringtonum(risultato2);
}
```

send Message - Custom Code

```
1 Object current = param(1);
2 treenode activity = param(2);
3 Token token = param(3);
4 treenode processFlow = ownerobject(activity);
5 { //************** PickOption Start *************\\
6 /***popup:SendMessage*/
7 /**Send Message*/
8 int NoDelay = -1;
9 double delaytime = /** \nDelay Time: *//***tag:delaytime*//**/0/**list:NoDelay~0~10~curre
10 treenode toobject = /** \nTo: *//***tag:to*//**/model().find("Source1")/**/;
11 treenode toobject2 = /** \nTo: *//***tag:to*//**/model().find("Source27")/**/;
12 treenode fromobject = /** \nFrom: *//***tag:from*//**/current/**/;
13 double param1 = /** \nParam1: *//***tag:par1*//**/token.irrigazione/**/;
14 double param2 = /** \nParam2: *//***tag:par2*//**/token.cisterna/**/;
15 double param3 = /** \nParam3: *//***tag:par3*//**/0/**/;
16 /**\n\nDelay Time:\nNoDelay: message sent immediately within trigger context\n0: delayed
17 if (/** \nCondition: *//***tag:condition*//**/true/**/) {
18     if (delaytime == NoDelay){
19         sendmessage(toobject,fromobject,param1,param2,param3);
20         sendmessage(toobject2,fromobject,param1,param2,param3);
21         }
22     else{
23     senddelayedmessage(toobject, max(0,delaytime), fromobject,param1,param2,param3);
24     senddelayedmessage(toobject2, max(0,delaytime), fromobject,param1,param2,param3);
25 }
26 }
27
28 } //******* PickOption End *******\\
29
```

```
Object current = ownerobject(c);
Object fromObject = param(1);
Variant msgparam1 = param(2);
Variant msgparam2 = param(3);
Variant msgparam3 = param(4);

{
    Object involved = /** \nObject: */ /***tag:object*//**/current/**/;
    if (msgparam1 == 1) {
        openoutput(involved);
    } else {
        closeoutput(involved);
    }

    Object object = current;
    string framename = "";

    double litri = msgparam2.as(double);

    // Selezione frame in base al valore
    if (litri >= 0 && litri < 5000) {
        framename = "Base Frame";
    } else if (litri >= 5000 && litri < 9000) {
        framename = "Arancione";
    } else if (litri >= 9000 && litri < 15000) {
        framename = "Verde";
    } else {
        print("Fuori Range");
    }

    // Cambio frame effettivo solo se valido
    if (framename != "") {
        if (framename == "Base Frame" || framename == "0") {
            setframe(object, 0);
        } else {
            setframe(object, framename);
        }
    }
} // ******* PickOption End ******* //
```

COMUNICAZIONE TRA SIMULAZIONE E IL MONDO REALE

AGRITECH

4. Comunicazione tra simulazione e il mondo reale

La comunicazione tra la simulazione in Flexsim e il mondo reale è realizzata tramite un'architettura a microservizi, che consente di integrare in modo efficiente i dati raccolti dai sensori reali, comandare attuatori (tende) e monitorare lo stato del sistema in tempo reale.

4.1 Tecnologie

FastAPI

Il canale principale di comunicazione tra la simulazione e i servizi esterni è basato su **FastAPI,** progettate secondo il modello client-server.
Flexsim funge da **client** che effettua richieste HTTP verso i microservizi:

- **/state/receive:** Endpoint necessario per ricevere lo stato della struttura. (Aperta, Chiusa)
- **/irrigation/litres/receive:** Endpoint necessario per ricevere la quantità di liquido esistente nella cisterna. (restituisce l'ultimo record esistente dei litri)
- **/irrigation/state/receive:** Endpoint necessario per ricevere lo stato dell'irrigazione. (Aperta, Chiusa)
- **/data/send:** Endpoint necessario per l'inserimento dei dati letti
- **/data/send/states**: Endpoint necessario per l'inserimento degli stati aggiornati manualmente della struttura e dell'irrigazione

Arduino Language (per i dispositivi IoT)

I nodi reali (ESP32) utilizzano Arduino language per la comunicazione asincrona:

- Pubblicano dati dai sensori meteo.
- Ricevono comandi dai microservizi e li trasmettono agli attuatori.

4.2 Descrizione dati

4.2.1 Dati in ingresso alla simulazione (FlexSim)

Destinazione	Metodo	Descrizione
Micro-Servizio tenda	get_status()	Apertura e chiusura delle tende
Micro-Servizio cisterna	get_litres()	Volume d'acqua presente
Micro-Servizio irrigazione	get_irrigation_state()	Esegue l'irrigazione della piantagione

4.2.2 Dati in uscita alla simulazione (nella realtà)

Fonte	Tipo	Descrizione
Sensore pioggia	Int	Il sensore digitale rileva la presenza di pioggia secondo questa logica: - Restituisce il valore 0 in caso di assenza di pioggia. - Restituisce il valore 1 in caso di presenza di pioggia.
Sensore luminosità	Float	Il sensore analogico misura l'intensità della luce ambientale ha un range da 0 (buio) fino a 4048 (luce intensa)
Sensore temperatura	Float	Il sensore digitale DH11 rileva due valori: - Temperatura da 0° a 50° - Umidità da 20% a 90% RH
Sensore umidità terrena	Float	Il sensore analogico misura l'umidità del terreno con un range da 0 (terreno molto secco) fino a 4095 (terreno completamente bagnato).

Sensore ad ultrasuoni	Float	Il sensore a ultrasuoni misura la distanza dall'ostacolo più vicino con un range che va da 2 cm fino a 400 cm (distanza massima). Il valore viene convertite successivamente il volume di acqua esistente nella cisterna.
Sensore IR	Hexadecimal	Il ricevitore ad infrarossi rileva segnali a infrarossi emessi dal telecomando con un range tipico di 38 kHz.

4.2.3 Dati in ingresso dalla simulazione (nella realtà)

Destinazione	Metodo	Descrizione
Servomotore		
Led	checkIR()	Il ricevitore ad infrarossi capta i segnali del telecomando ed in base al pulsante premuto (1 per l'irrigazione, 2 per l'apertura della struttura) verranno accesi gli appositi led. (blu per l'irrigazione, rosso per la struttura)

DATABASE

AGRITECH

5. Database Non Relazionale Gestione degli eventi

5.1 Motivo della scelta

Nel contesto di questo progetto IoT, che prevede la raccolta di dati da sei sensori con una frequenza di aggiornamento di 2.5 secondi, è stata scelta un'architettura basata su un database **NoSQL** per le seguenti ragioni tecniche:

1. **Elevata frequenza di scrittura**

   I 5 sensori generano un flusso costante di dati (circa 172800 record al giorno). I database NoSQL, in particolare quelli orientati ai documenti o alle colonne, sono ottimizzati per gestire elevati volumi di scrittura in tempo reale, senza il sovraccarico dovuto alla gestione di transazioni complesse e schemi rigidi tipici dei database SQL.

2. **Scalabilità orizzontale**

   Le soluzioni NoSQL sono progettate per scalare orizzontalmente, consentendo di distribuire il carico su più nodi in modo semplice ed efficace. Questo è fondamentale in un ambiente IoT, dove la quantità di dati tende a crescere nel tempo con l'aggiunta di nuovi dispositivi.

3. **Flessibilità dello schema**

   I database NoSQL supportano schemi dinamici, permettendo di gestire con facilità dati eterogenei provenienti dai sensori. Questo è utile sia per gestire formati diversi di payload, sia per adattarsi a future modifiche del formato dei dati senza necessità di migrazioni complesse.

4. **Ottimizzazione per serie temporali**

   Molti database NoSQL sono ottimizzati per la gestione di **dati temporali**, rendendoli ideali per applicazioni IoT dove ogni dato è associato a un timestamp. Offrono query ottimizzate per aggregazioni temporali, downsampling e retention policy.

   Conclusione:

   Per un sistema IoT con flusso dati ad alta frequenza, requisiti di scalabilità, e necessità di gestione efficiente di dati temporali, l'adozione di un database NoSQL rappresenta una scelta più adatta rispetto ai tradizionali RDBMS, garantendo prestazioni superiori, flessibilità e facilità di evoluzione del sistema.

5.2 Database & Collections

5.2.1 Schema Collezione Record

La collezione **Record** è stata creata con un validatore che segue i seguenti criteri:

1. Schema **SensorValueIn** che accetta i seguenti valori:
   a. *sensor_id*: *str*
   b. *value*: *int*, *float*, *bool*
2. Schema **ReadingsIn** che accetta i seguenti valori:
   a. *temperature*: *SensorValueIn*
   b. *humidity*: *SensorValueIn*
   c. *soil_humidity*: *SensorValueIn*
   d. *rain*: *SensorValueIn*
   e. *light*: *SensorValueIn*
   f. *water_level*: *SensorValueIn*
3. Schema **RecordDataIn** che accetta il seguente valore:
   a. *readings*: *ReadingsIn*

Il validatore mostrato in precedenza richiede che l'invio dei dati effettuato tramite una richiesta **HTTP** di tipo **POST** al endpoint `/data/send` segua necessariamente la struttura definita in precedenza, rispettando le chiavi (segnate in *giallo*) ed il tipo (Composto, segnato in *azzurro*, Primitivo, segnato in *grigio.*). Questo porterà all'invio di un oggetto **JSON** con la seguente struttura:

```
{
  "readings": {
    "temperature": { "sensor_id": "ID", "value":  Value},
    "humidity": { "sensor_id": "ID", "value": Value},
    "soil_humidity": { "sensor_id": "ID", "value": Value},
    "rain": { "sensor_id": "ID", "value": Value},
    "light": { "sensor_id": "ID", "value": Value},
    "water_level": { "sensor_id": "ID", "value": Value }
  }
}
```

5.2.2 Schema Collezione States

La collezione **States** accetta oggetti di tipo **JSON** validati tramite Pydantic seguendo i seguenti criteri:

1. Schema **StatesIn** che accetta i seguenti valori:
    a. *irrigation_state*: *int*, *float*, *bool*
    b. *strucrute_state*: *int*. *float*, *bool*

Il validatore mostrato in precedenza richiede anch'esso che l'invio dei dati effettuato tramite una richiesta **HTTP** di tipo **POST** all'endpoint `/data/send/states` segua necessariamente la struttura definita in precedenza, rispettando le chiavi (segnate in *giallo*) ed il tipo (Primitivo, segnato in *grigio*). Questo porterà all'invio di un oggetto **JSON** con la seguente struttura:

```
{
  "irrigation_state": Value,
  "structure_state": Value
}
```

GESTIONE DEGLI EVENTI DAL CAMPO

AGRITECH

6. Eventi del campo

6.1 Descrizione sensori

1. **DHT11 - Sensore di Temperatura e Umidità**

   Misura: temperatura (0–50 °C, ±2 °C) e umidità (20–90% RH, ±5% RH).
   Alimentazione: 3,3.
   Uscita digitale su un singolo pin dati.
   Utile per monitorare le condizioni ambientali di una serra o orto.

2. **MH-RD Rain Sensor**

   Funzione: rileva la pioggia tramite una superficie a elettrodi.
   Uscite:

   - Digitale (uscita con soglia regolabile).
   - Analogica (valore proporzionale alla bagnatura).

   Alimentazione: 5V.
   Ideale per attivare sistemi di irrigazione automatica o per rilevare condizioni di pioggia.

3. **HC-SR04 - Sensore Ultrasonico di Distanza**

   Gamma: 2 cm – 4 m, con una risoluzione di circa 3 mm.
   Alimentazione: 5V.
   Pin:

   - TRIG (attiva la misura).
   - ECHO (durata del segnale di ritorno, proporzionale alla distanza).

   Utilizzato per misurare livello dell'acqua, altezza piante o ostacoli.

4. **Capacitive Soil Moisture Sensor V2.0**

   Misura l'umidità del suolo in modo capacitivo .
   Alimentazione: 3.3V.
   Uscite:

   - Analogica (valore proporzionale all'umidità).
   - Digitale (uscita con soglia regolabile tramite trimmer).

   Ideale per monitorare il fabbisogno idrico delle piante.

5. **Fotoresistenza (LDR)**

   È un resistore che cambia la sua resistenza in base alla quantità di luce:
   Alimentazione: 3.3V
   Uscite:
   Analogica (tensione proporzionale alla luce tramite partitore resistivo).
   Ideale per rilevare la variazione di luce in serre, orti o progetti di automazione domestica.

6.2 Descrizione nodi IoT

1. **ESP32 - Microcontrollore**

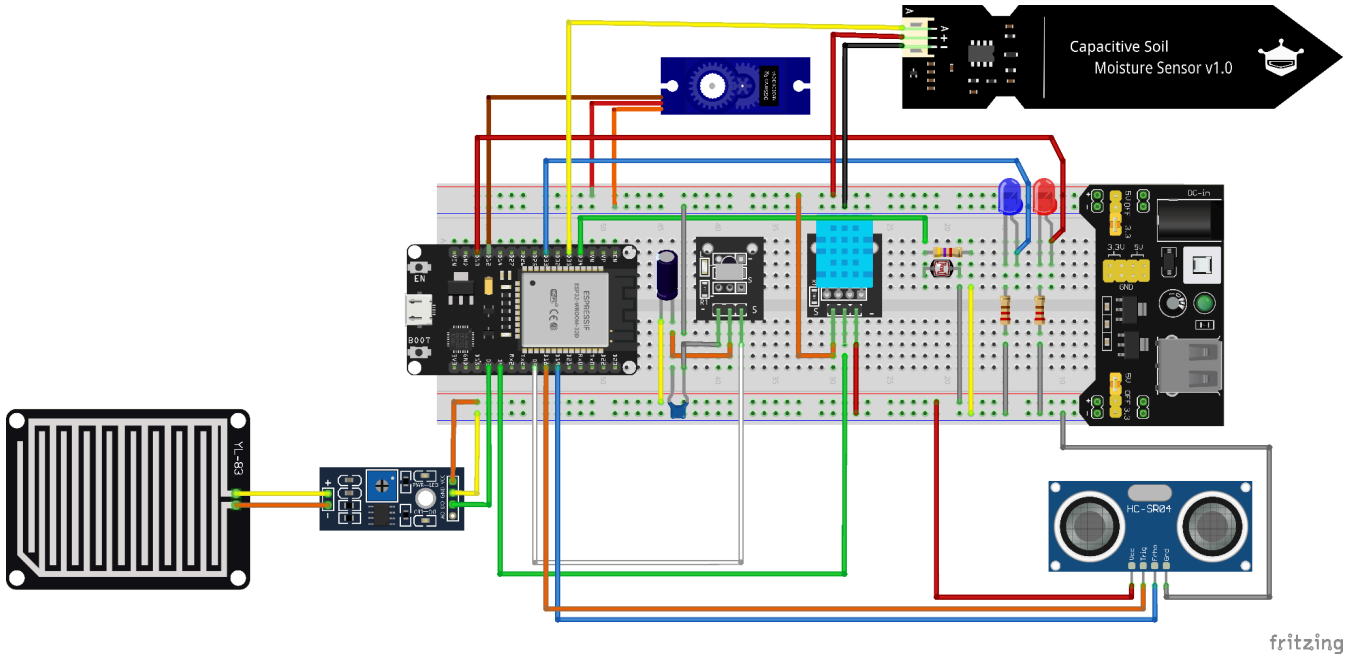
   Tensione logica: 3,3V.
   Alimentazione: 3,3–5V tramite regolatore interno.
   Caratteristiche: WiFi, Bluetooth, ingressi analogici (ADC), PWM, I2C, SPI, UART.
   Ideale per progetti di monitoraggio ambientale, automazione e IoT.

6.3 Accessori nodi IoT

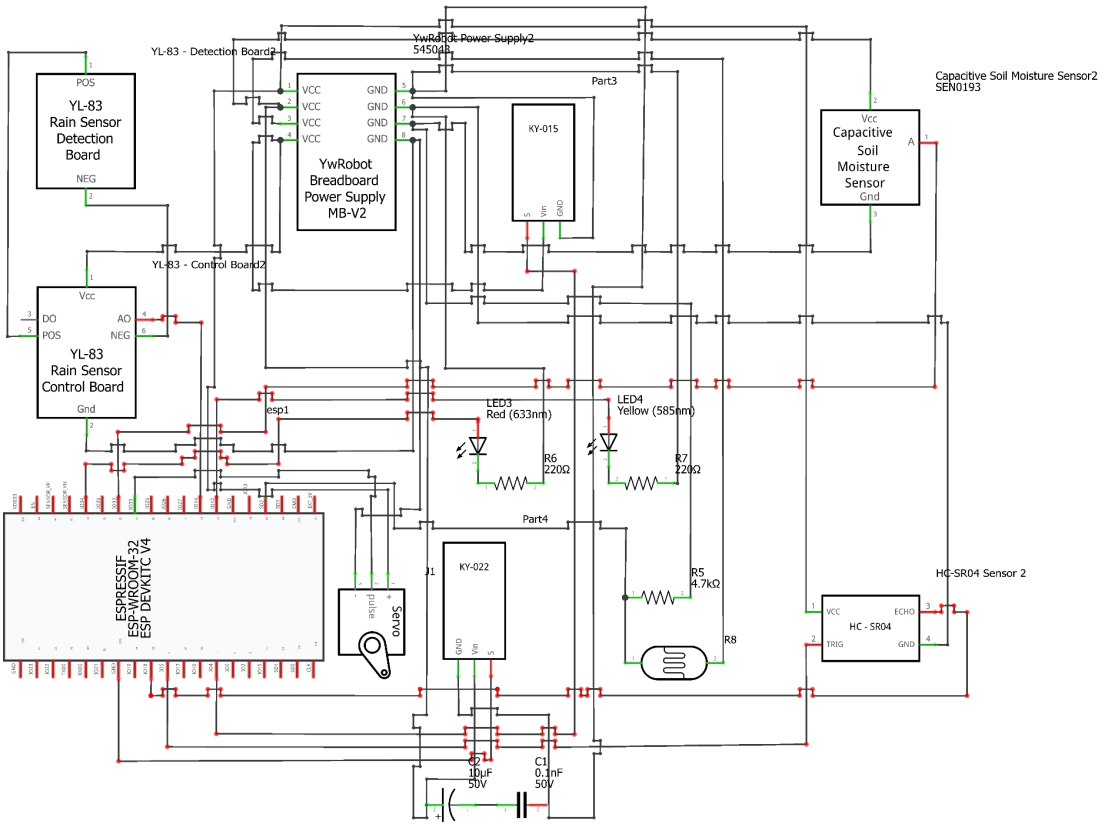
1. **Elegoo Power MB V2**

   Tensioni di uscita: 3,3V o 5V, selezionabili.
   Alimentazione in ingresso: 6,5–12VDC (tramite connettore barrel jack o pin header).
   Consente di alimentare facilmente moduli e sensori su breadboard.

6.4 Schema di Montaggio (Breadboard View)



6.5 Schema Elettrico (Schematic Diagram)



TEST E VALIDAZIONE

AGRITECH

7. Test e Validazione

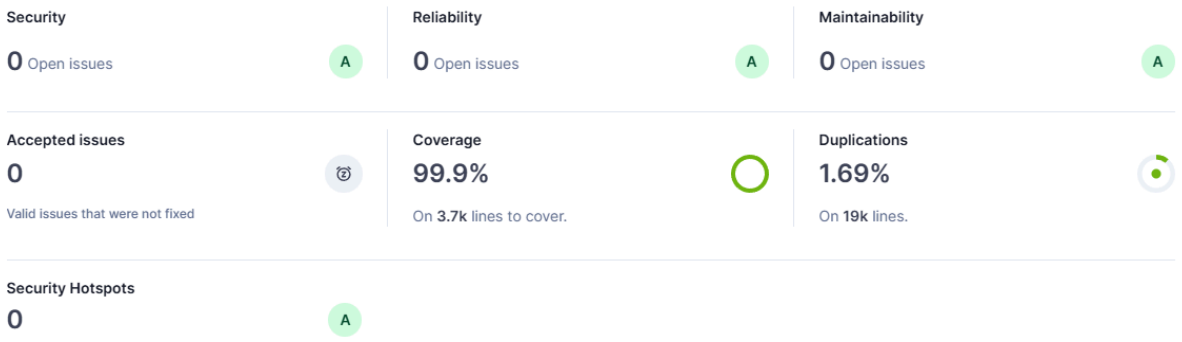
7.1 Descrizione delle attività di test

Per il backend sviluppato sono stati effettuati diversi tipi di test utilizzando strumenti specifici:

- **Sonarqube**: utilizzato per l'analisi statica del codice, al fine di individuare vulnerabilità, code smells e problemi di qualità del software.
- **Postman**: impiegato per testare le API, verificandone la correttezza delle risposte, le prestazioni e la conformità agli standard richiesti.

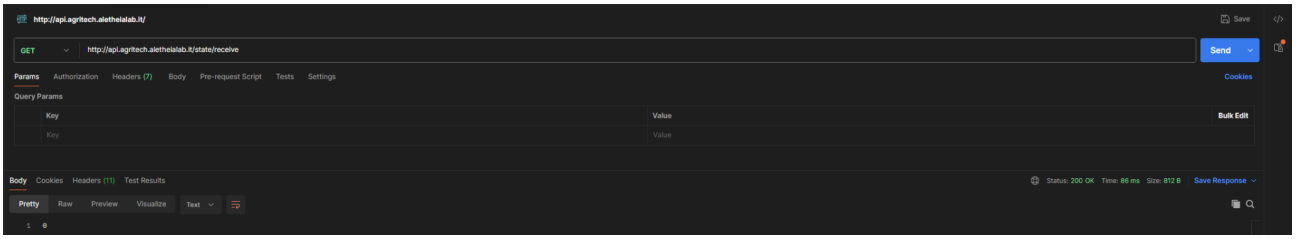
7.1.1 Sonarqube

Il test statico del codice backend, effettuato con lo strumento Sonarqube, ha prodotto un risultato eccellente, con una valutazione complessiva pari ad **A**. Questo punteggio riflette un'alta qualità del codice, una buona aderenza agli standard di sviluppo e una gestione efficace delle best practice. Di seguito il report del testing statico:



7.1.2 Postman

I test delle API del backend sono stati effettuati utilizzando Postman per verificare la correttezza delle risposte, l'aderenza agli standard RESTful e la robustezza generale. I risultati confermano che le API sono ben progettate, performanti e pronte per l'uso in produzione. Di seguito è riportato un esempio di API testata che restituisce un valore di **200 OK**. Tutte le API analizzate hanno riportato esiti positivi.



8. Manuale di istruzione

8.1 Avvio del sistema

1. **Accendi il sistema**
   a. Usa un **cavo di alimentazione da 9V o 12V**, visualizza *Figura 1*.
   b. Una volta collegato, il sistema si accende in automatico.
   c. Nella *Figura 2a/2b* è illustrato il punto esatto in cui inserire il cavo di alimentazione nella centralina.

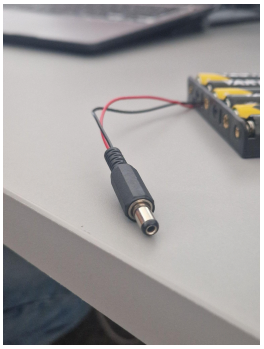


Figura 1

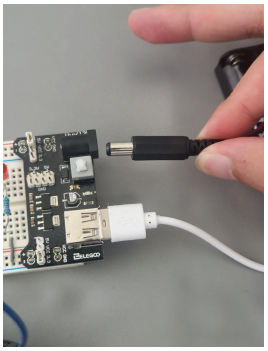


Figura 2a

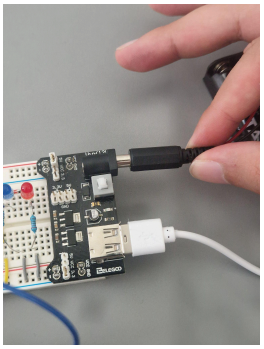


Figura 2b

2. **Avvia il programma**
   a. Il sistema è già configurato e si avvia autonomamente.
   b. Una volta acceso, controlla da solo pioggia e grandine.
3. **Controlla cosa succede**
   a. In caso di pioggia, le tende si aprono automaticamente per proteggere le vigne.
   b. Al termine della pioggia, le tende si chiudono da sole.
   c. L'acqua raccolta scorre attraverso tubazioni interne, raggiungendo una cisterna sotterranea.
   d. Se la temperatura supera i **34 °C**, il sistema attiva l'irrigazione automatica, utilizzando l'acqua raccolta.
4. **Avvio della tenda attraverso telecomando**

Pulsante Power:Invia dati presi dai sensori al database
1. Avvio dell'acqua
2. Apertura tenda
3. Attivazione Sensori

8.2 Cosa puoi fare

Cosa vuoi fare	Azione da compiere / Dove guardare
Chiudere/aprire una tenda a mano	Premi il pulsante sulla centralina o usa l'app.
Innaffiare manualmente	Premere il tasto “Irriga” (se presente sul pannello).

8.3 Cosa fa il sistema da solo?

Evento	Azione Automatica
Rileva Pioggia	Le tende si chiudono
Rileva grandine	Le tende si chiudono
Finisce la pioggia	Le tende si aprono dopo qualche minuto
Il terreno è secco	Parte l’irrigazione

8.4 Se qualcosa non funziona?

Problema	Soluzione
Le tende non si muovono	Controlla se è acceso il sistema
L’acqua non viene raccolta	Verifica che le tende siano chiuse bene
L’irrigazione non parte	Controlla che ci sia acqua nella cisterna

8.5 Quando spegnerlo?

Puoi lasciare il sistema acceso tutto il giorno.
Spegnilo solo se:

- “Devi fare manutenzione”
- “Finisce la stagione”

Per spegnerlo:

1. Spegni i dispositivi dal pulsante o togli la corrente.
2. Il sistema salverà i dati raccolti in automatico.

8.6 Assistenza

Se hai dubbi o problemi, puoi contattarci:

- Email: aletheia@aziendale.com
- Responsabile: Team Aletheia
- Telefono: +39 080 544 2335