SAPIENZA
UNIVERSITÀ DI ROMA

Università degli Studi di Roma "La Sapienza"

Progetto per il corso di Ingegneria del Software

Smart Greenhouse

Gruppo formato da:

Di Nella Daniel
Ciafardoni Luca

August 13, 2024

Contents

1 Introduzione al progetto

1.1 Descrizione Smart GreenHouse

Il nostro progetto si concentra sulla creazione di un ecosistema di automazione per una serra intelligente, offrendo agli utenti il completo controllo del loro ambiente di coltivazione. L'obiettivo è garantire un ambiente ottimale per la crescita delle piante, implementando un sistema di gestione intelligente che controlla la temperatura, l'umidità, l'illuminazione e l'irrigazione in modo preciso e efficiente. Inoltre, il sistema includerà funzionalità di monitoraggio e registrazione per tenere traccia delle condizioni ambientali e delle attività di coltivazione nel corso del tempo.

La versatilità della "SmartGreenHouse" si evidenzia attraverso la sua integrazione con una vasta gamma di sensori e dispositivi di automazione. Questi possono includere sensori di umidità del suolo, sensori di luce, sistemi di irrigazione automatica, e dispositivi per la regolazione della temperatura e dell'umidità. Il sistema offre anche la possibilità di connettersi con dispositivi smart esterni, consentendo agli utenti di massimizzare l'efficienza e il rendimento delle loro coltivazioni.

Questo programma genera in modo casuale una serie di azioni che simulano scenari realistici all'interno della serra, consentendo agli utenti di testare e ottimizzare le loro impostazioni di automazione in varie condizioni.

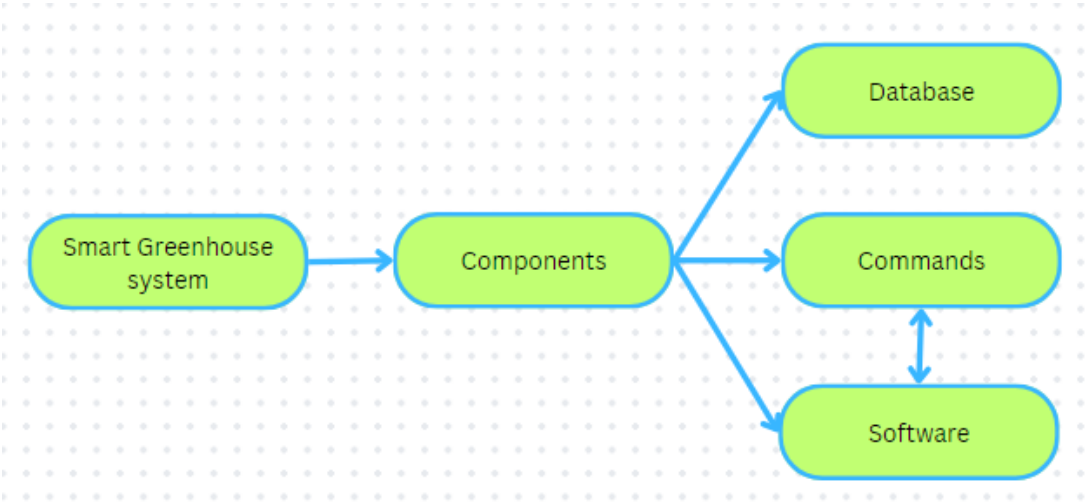

Figure 1: Panoramica su ambiente e sistema operativo

1.2 Descrizione interazione Components-Software-Commands

Per testare l'interazione dell'utente con la serra intelligente, utilizziamo tre componenti principali:

- **Main delle Componenti:** Eseguito dallo script "esegui_main.sh", questo componente gestisce le operazioni principali delle varie componenti della serra.
- **Software:** Questo componente attende i comandi che l'utente desidera eseguire e si occupa di trasmetterli alle componenti interessate.
- **Commands:** Responsabile dell'esecuzione dei comandi da eseguire.

Le componenti Software e Commands, oltre ad essere interconnesse tramite le comunicazioni Redis, sono entrambe strutturate con un sistema di "case". Ogni "case" comprende una componente che viene selezionata in modo casuale in Commands. Il Software riceve quindi la componente e il suo comando da Commands, la riconosce e pubblica il "risultato" grazie all'esecuzione del main della componente interessata.

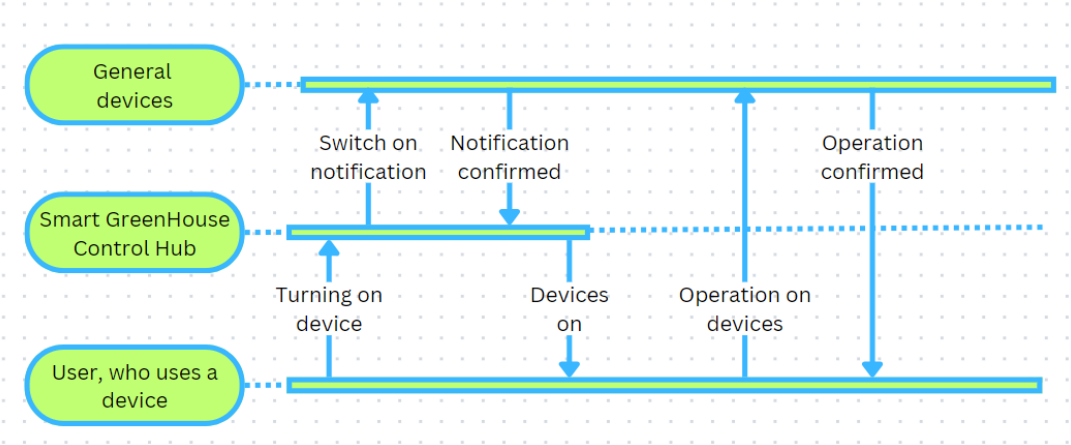

Figure 2: UML Message Sequence Chart

1.3 Descrizione connesioni Redis

Per facilitare la comunicazione tra il Software, Commands e i main delle varie componenti all'interno della serra intelligente, utilizziamo un sistema di più canali Redis.

- **Canale dell'Utente:** Questo canale permette all'utente di visualizzare tramite il Software se il comando inviato alla componente è stato eseguito con successo. È il principale punto di interazione tra l'utente e il sistema.
- **Canali per Ogni Componente:** Viene utilizzato un canale dedicato per ogni componente della serra. Questi canali permettono al Software di inviare i comandi ai main delle varie componenti tramite Commands, garantendo una comunicazione efficiente e specifica per ciascuna componente.
- **Canale per le Risposte:** Questo canale funge da monitor per verificare se le risposte attese dal Software impiegano troppo tempo. Viene utilizzato per testare la tempestività delle risposte e garantire che il sistema sia reattivo.

Tutte queste comunicazioni sono gestite attraverso l'uso di comandi Redis come "Subscribe Channel" e "Publish Channel", insieme all'utilizzo delle varie risposte ("reply"). Ad ogni interazione, viene attentamente verificato che la risposta venga eseguita con successo, garantendo l'affidabilità e la robustezza del sistema.

2 User Requirements

2.1 Gestione della coltivazione

La funzione "Gestione della coltivazione" offre il controllo automatizzato e manuale degli elementi della serra, incluso l'irrigazione e l'illuminazione interna.
Specifica:

- **Gestione dell'irrigazione:** Fornisce la capacità di programmare e controllare l'irrigazione delle piante in base alle loro esigenze specifiche di acqua.
- **Gestione dell'illuminazione interna:** Consente di controllare l'illuminazione dall'interno della serra per fornire la quantità e la qualità di luce ottimali per la crescita delle piante.
- **Monitoraggio del sensore ambientale:** Integrazione di sensori ambientali per monitorare parametri come temperatura e umidità all'interno della serra.

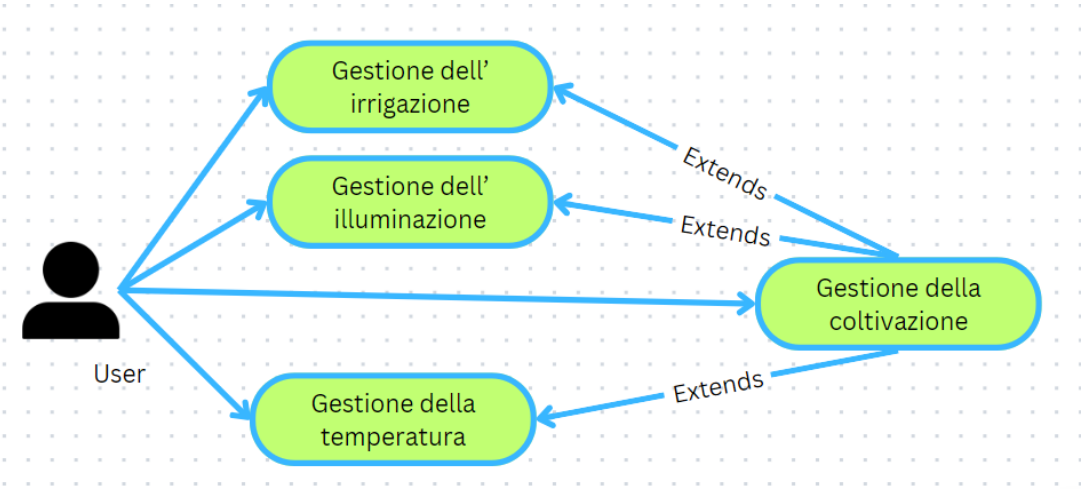

Figure 3: Use-Case UML di Gestione della coltivazione

2.2 Gestione dei dispositivi

Gestione personalizzabile e modificabile di dispositivi per la vita quotidiana nella Smart Greenhouse.
Specifica:

- **Accendi:** Attiva i dispositivi.
- **Spegni:** Disattiva i dispositivi quando non sono necessari o in caso di emergenza.
- **Programmazione oraria:** Possibilità di programmare l'accensione e lo spegnimento dei dispositivi in base a orari specifici.

2.3 Gestione dell'illuminazione

Gestione personalizzabile e modificabile dell'illuminazione all'interno della serra.
Specifica:

- **Accendi:** Attiva l'illuminazione interna della serra.
- **Spegni:** Disattiva l'illuminazione quando non è necessaria.
- **Cambia colore:** consente di regolare il colore delle luci LED per adattarle alle esigenze delle piante.
- **Cambia intensità:** Regola l'intensità luminosa dell'illuminazione per fornire la quantità di luce ottimale per le piante in crescita.

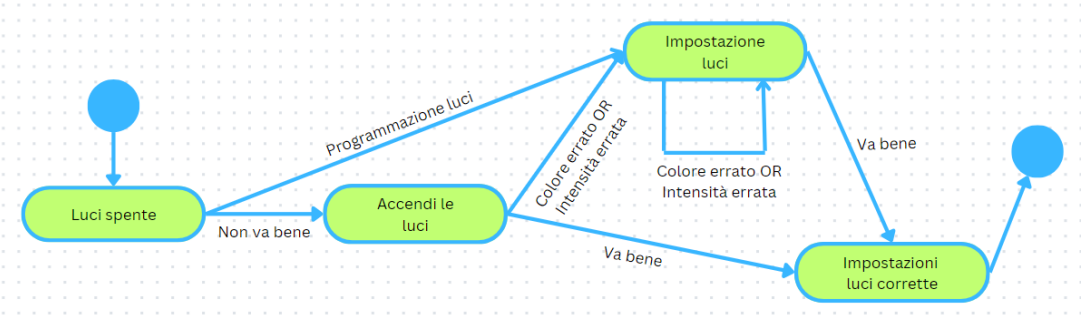

Figure 4: UML Activity Diagram delle luci

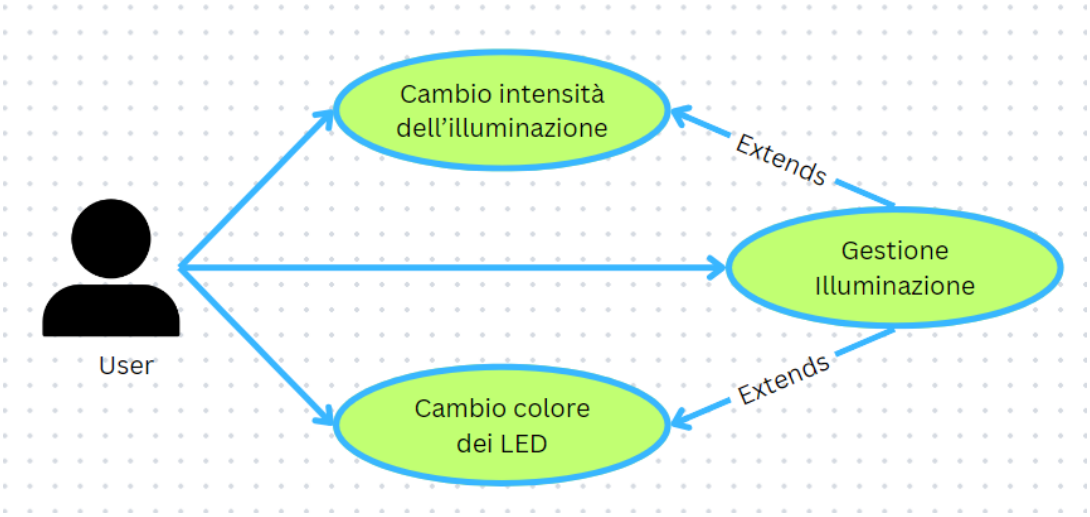

Figure 5: Use-Case UML di Gestione dell'illuminazione

2.4 Gestione della temperatura interna

La funzione "Gestione della temperatura" fornisce il controllo automatico e manuale della temperatura all'interno della serra.
Specifica:

- **Accendi:** Avvia il sistema di riscaldamento o di raffreddamento per mantenere la temperatura desiderata.
- **Spegni:** Arresta il sistema di riscaldamento o di raffreddamento quando la temperatura è ottimale per le piante.
- **Cambia temperatura:** Consente di regolare manualmente la temperatura desiderata.
- **Monitoraggio della temperatura:** Fornitura di monitoraggio in tempo reale della temperatura all'interno della serra.

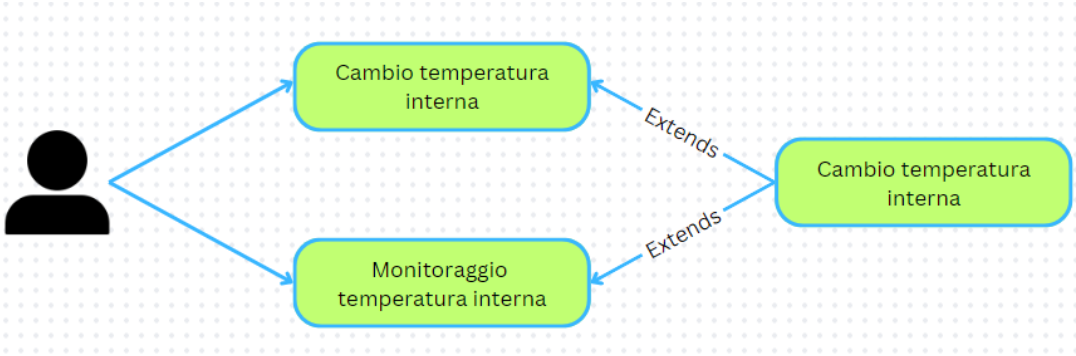

Figure 6: Use-Case UML di Gestione della temperatura

2.5 Gestione dell'irrigazione

La funzionalità "Gestione dell'irrigazione" fornisce il controllo automatico e manuale dell'irrigazione all'interno della serra
Specifica:

- **Accendi:** Avvia il sistema di irrigazione per bagnare le piante desidarate.
- **Spegni:** Arresta il sistema di irrigazione quanto le piante non necessitano ulteriore acqua.
- **Impostazione modalità d'irrigazione:** Consente di regolare manualmente il raggio di irrigazione decidendo tra 3 modalita (Low, Medium, High) da utilizzare.

2.6 Gestione della sicurezza e sensori

La funzionalità "Gestione della sicurezza" garantisce la protezione delle piante attraverso l'uso di sensori e allarmi intelligenti. Questo sistema include sensori per l'umidità del suolo, che assicurano che le piante ricevano la quantità d'acqua necessaria, e sensori per la temperatura, che monitorano e mantengono le condizioni climatiche ottimali nel giardino. Gli allarmi intelligenti avvisano prontamente in caso di condizioni fuori dai parametri stabiliti, permettendo interventi tempestivi per salvaguardare la salute delle piante.
Specifica:

- **Sensori di umidità del giardino:** Monitoraggio dei livelli di umidità del giardino.
- **Sensori di temperatura del giardino:** Monitoraggio della temperatura del giardino.
- **Videosorveglianza e sensori di movimento:** Integrazione di telecamere di sorveglianza e sensori di movimento per rilevare intrusioni o danni alla serra.

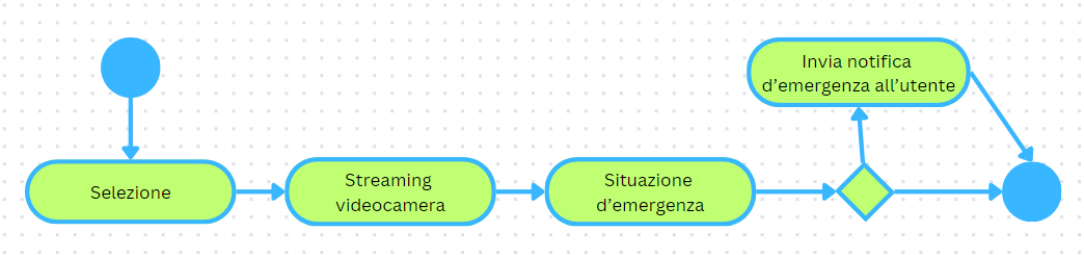

Figure 7: UML Activity Diagram delle videocamere

3 System Requirements

3.1 Prestazioni

Prestazioni minime richieste dall'utente per garantire un funzionamento efficiente della serra intelligente.
Specifica:

- **Tempo di risposta:** Il sistema deve garantire una risposta rapida, con un tempo massimo di 30 secondi per le richieste dell'utente, ad esempio per l'attivazione dell'irrigazione o il cambiamento delle impostazioni di illuminazione.
- **Affidabilità:** Il sistema deve essere operativo almeno il 99

3.2 Usabilità

Specifiche per l'interfaccia utente della serra intelligente, garantendo un'esperienza intuitiva per gli utenti.
Specifica:

- **Interfaccia utente:** L'interfaccia utente della serra intelligente dovrebbe essere intuitiva e facile da usare, anche per gli utenti senza particolari competenze tecniche, consentendo loro di controllare facilmente l'irrigazione, l'illuminazione e altre funzionalità della serra.

3.3 Gestione manutenzione

Requisiti relativi alla manutenzione periodica del software e dell'hardware della serra intelligente.
Specifica:

- **Aggiornamenti software:** Gli aggiornamenti software dovrebbero essere rilasciati periodicamente, con notifiche chiare agli utenti in caso di manutenzione, garantendo così la sicurezza e l'efficienza del sistema.
- **Manutenzione preventiva:** Il sistema dovrebbe essere sottoposto a manutenzione preventiva ogni mese per garantirne la stabilità e prevenire eventuali problemi o guasti.

3.4 Scalabilità

Miglioramento della scalabilità della serra intelligente per consentire l'aggiunta di nuovi dispositivi e funzionalità.
Specifica:

- **Capacità di espansione:** Il sistema deve consentire l'aggiunta di nuovi sensori, dispositivi di automazione e funzionalità senza causare problemi di prestazioni, garantendo così la flessibilità e l'adattabilità della serra intelligente alle esigenze dell'utente.

4 Implementazione

Il sistema di Smart GreenHouse è composto dai seguenti componenti:

- Luci
- Condizionatore
- Videocamere
- Irrigatore
- Sensore da giardino
- Sensori
- Dispositivi

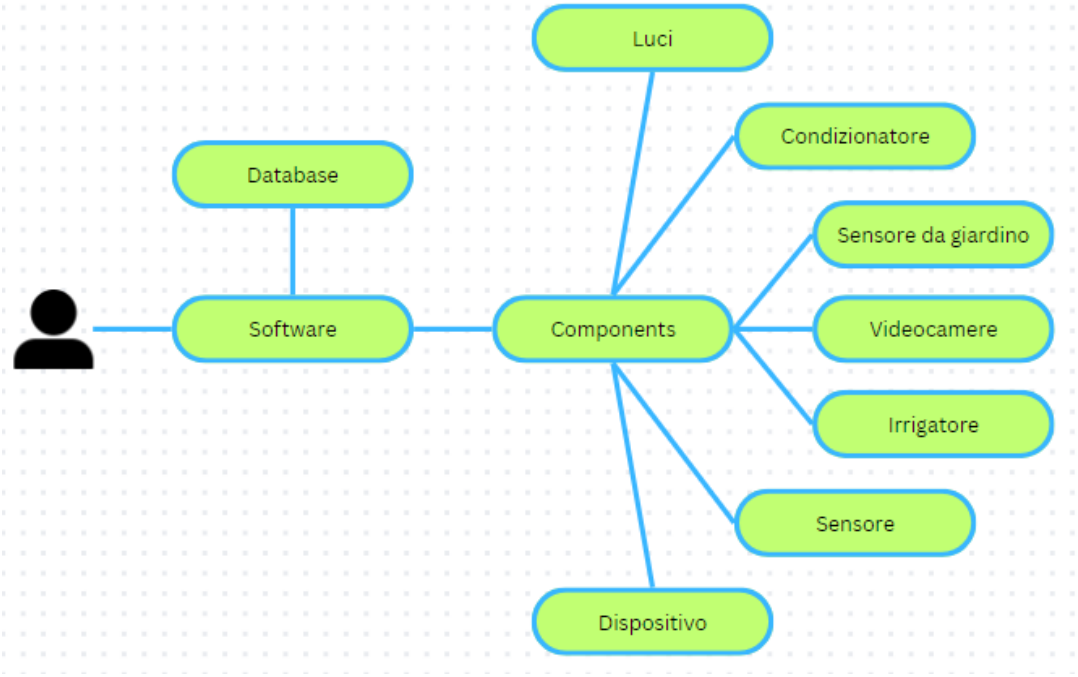

Figure 8: Diagramma architettura di sistema

4.1 Inizializzazione

- **Inizializzazione del Database e delle Variabili:** Il sistema avvia un database per archiviare i dati rilevanti e le variabili necessarie per il funzionamento ottimale delle componenti.
- **Impostazione della Comunicazione Redis:** Viene istituita una comunicazione tramite Redis, un sistema di memorizzazione chiave-valore in memoria ad alte prestazioni, che consente lo scambio efficiente di dati tra le diverse componenti del sistema.
- **Inizializzazione delle Componenti:** Ogni componente del sistema viene avviato con valori iniziali predefiniti per garantire un funzionamento uniforme e coerente.
- **Impostazione del Tempo di Avvio:** Il sistema imposta il tempo di avvio per il monitoraggio e il controllo delle componenti.

4.2 Test Periodici

Le componenti del sistema vengono testate periodicamente per verificare il loro stato attuale e rilevare eventuali anomalie. Questo processo avviene attraverso le connessioni Redis con il Software e Commands. Commands comunica con il Software, il quale elabora le informazioni e le trasmette alla componente interessata. I risultati dei test vengono accuratamente registrati nel database per analisi e monitoraggio successivi.

4.3 Monitoraggio e Log

- **Monitoraggio del Tempo di Risposta:** Durante i test, il sistema monitora il tempo di risposta delle componenti per garantire che le risposte avvengano entro limiti di tempo prestabiliti. Questo assicura un funzionamento efficiente e tempestivo del sistema.
- **Registrazione dei Dati Rilevanti:** I dati rilevanti, tra cui gli stati delle componenti e i tempi di risposta, vengono registrati accuratamente nel database. Questo permette un'analisi dettagliata delle prestazioni del sistema e fornisce informazioni utili per eventuali miglioramenti o regolazioni future.

4.4 Fine del programma

Dopo un numero predefinito di cicli di test, ad esempio dieci, i dati raccolti vengono registrati nel database e il programma termina il suo ciclo di funzionamento. Questo garantisce una gestione efficiente delle risorse e un controllo adeguato del sistema nel tempo.

4.5 Conclusione

In conclusione, l'implementazione del sistema di gestione intelligente per la serra prevede un ciclo completo di monitoraggio delle componenti, dalla loro inizializzazione fino alla registrazione dei dati e all'analisi delle prestazioni. Questo approccio consente agli utenti di rilevare lo stato delle componenti in tempo reale, monitorare i tempi di risposta e ottimizzare le prestazioni complessive del sistema per garantire una crescita ottimale delle piante all'interno della serra.

5 Pseudo-codice componenti

5.1 Pseudo-codice Sensore da giardino

Il sensore da giardino monitora l'umidità del suolo e la temperatura esterna, garantendo condizioni ottimali per la crescita delle piante. Fornisce dati in tempo reale per un'irrigazione e una gestione climatica precise ed efficienti.

- **Attributi**
  - **id: intero**, identifica in modo univoco un sensore del giardino.
  - **state: sensorGarden_type**, stato attuale del sensore da giardino.
  - **humidity: intero**, livello di umidità rilevato dal sensore da giardino.
  - **temperature: intero**, temperatura rilevata dal sensore da giardino.
- **Metodi**
  - **Costruttore(id: intero, state: sensorGarden_type, humidity: intero, temperature: intero): SensorGarden**, Inizializza un oggetto SensorGarden con l'id, l'umidità, la temperatura e lo stato specificati.
  - **getId(): intero**, Restituisce l'ID univoco del sensore da giardino.
  - **getState(): sensorGarden_type**, Restituisce lo stato attuale del sensore da giardino.
  - **setState(newState: sensorGarden_type): void**, Imposta lo stato del sensore da giardino al valore specificato.
  - **getTemperature(): intero**, Restituisce la temperatura attuale del sensore da giardino.
  - **setTemperature(newTemperature: intero): void**, Modifica la temperatura del sensore da giardino.
  - **getHumidity(): intero**, Restituisce l'umidità attuale del sensore da giardino.
  - **setHumidity(newHumidity: intero): void**, Modifica l'umidità del sensore da giardino.
- **Enumerazione sensorGarden_type**
  - **SensorGardenON:** sensore da giardino acceso
  - **SensorGardenOFF:** sensore da giardino spento

5.2 Pseudo-codice Luce

La classe Luci è responsabile della gestione delle luci all'interno della serra. Questa classe fornisce metodi per controllare lo stato delle luci e per modificare il loro colore e intensità.

- **Attributi**
  - **id: intero**, identifica in modo univoco una specifica luce all'interno del sistema.
  - **state: light_type**, lo stato corrente delle luci.
  - **color: light_color**, il colore attuale delle luci.
  - **intensity: intero**, l'intensità attuale delle luci.
- **Metodi**
  - **Costruttore(id: intero, state: light_type, color: light_color, intensity: intero): Light**, Inizializza un oggetto Luce con l'id, lo stato, il colore e l'intensità specificati.
  - **getId(): intero**, Restituisce l'id univoco della luce.
  - **getState(): light_type** , Restituisce lo stato attuale delle luci.
  - **setState(newState: light_type): void**, Imposta lo stato delle luci al valore specificato.
  - **getColor(): light_color,** Restituisce il colore attuale delle luci.
  - **setColor(newColor: light_color): void,** Imposta il colore delle luci al valore specificato.

  - **getIntensity(): intero,** Restituisce l'intensità attuale delle luci.
  - **setIntensity(newIntensity: intero): void,** Imposta l'intensità delle luci al valore specificato.

- **Enumerazione light_type**
  - **LightON:** luci accese
  - **LightOFF:** luci spente
  - **change_intensity:** cambio intensità
  - **change_color:** cambio colore

- **Enumerazione light_color**
  - **WHITE:** luce generica.
  - **RED:** utile per influenzare la germinazione dei semi in alcune specie.
  - **BLUE:** ben nota per sopprimere lo "stretching" dello stelo.
  - **GREEN:** efficace nell'accelerare la fioritura in un certo numero di specie.
  - **PINK:** combinazione di rosso e blu, spesso utilizzata per promuovere una crescita equilibrata e fioriture abbondanti.
  - **ORANGE:** utile per stimolare la produzione di frutta e fiori, favorendo la maturazione.
  - **PURPLE:** completa i colori vuoti e motiva la crescita vegetativa.

5.3 Pseudo-codice Condizionatore

La classe Condizionatore gestisce i condizionatori all'interno della serra intelligente. Questa classe fornisce metodi per controllare lo stato e la temperatura del condizionatore.

- **Attributi**
  - **id: intero,** identifica in modo univoco un condizionatore specifico all'interno del sistema.
  - **state: conditioner_type,** stato corrente del condizionatore.
  - **temperature: intero,** temperatura attuale del condizionatore.

- **Metodi**
  - **Costruttore(id: intero, temperature: intero, state: conditioner_type): Conditioner,** Inizializza un oggetto Condizionatore con l'id, la temperatura e lo stato specificati.
  - **getId(): intero,** Restituisce l'ID univoco del condizionatore.
  - **getState(): conditioner_type,** Restituisce lo stato attuale del condizionatore.
  - **setState(newState: conditioner_type): void,** Imposta lo stato del condizionatore al valore specificato.
  - **getTemperature(): intero,** Restituisce la temperatura attuale del condizionatore.
  - **setTemperature(newTemperature: intero): void,** Modifica la temperatura del condizionatore.

- **Enumerazione conditioner_type**
  - **ConditionerON:** condizionatore acceso.
  - **ConditionerOFF:** condizionatore spento.
  - **change_temperature:** cambio temperatura.

5.4 Pseudo-codice Videocamera

La classe Videocamera è responsabile della gestione delle videocamera di sorveglianza all'interno della serra. Questa classe fornisce metodi per controllare lo stato delle telecamere e per attivare la registrazione in caso di rilevamento di movimento da parte dei sensori.

- **Attributi**
  - **id: intero,** identifica in modo univoco una specifica telecamera all'interno del sistema.
  - **state: camera_type,** stato corrente della telecamera.
  - **recording: booleano,** flag booleano che indica lo stato di registrazione della telecamera.
- **Metodi**
  - **Costruttore(id: intero, state: camera_type): Camera,** Questo metodo inizializza un oggetto Camera con l'id specificato e lo stato iniziale fornito come argomento.
  - **getId(): intero,** restituisce l'id univoco della videocamer.
  - **getState(): camera_type,** restituisce lo stato attuale della videocamera.
  - **setState(newState: camera_type): void,** permette di impostare lo stato della videocamera al valore specificato.
  - **getRecording(): booleano,** restituisce lo stato corrente di registrazione della telecamera.
  - **setRecording(rec: booleano): void,** imposta lo stato di registrazione della videocamera in base al valore booleano fornito come argomento.
- **Enumerazione camera_type**
  - **CameraON:** La videocamera è accesa
  - **CameraOFF:** La videocamera è spenta

5.5 Pseudo-Codice Irrigatore

La classe Irrigatore gestisce gli irrigatori all'interno della serra. Questa classe fornisce metodi per controllare lo stato ed impostare i parametri di irrigazione

- **Attributi**
  - **id: intero,** identifica in modo univoco un irrigatore specifico all'interno del sistema.
  - **state: irrigator_type,** lo stato corrente dell'irrigatore, che può essere acceso o spento.
  - **p: pressure,** la pressione esercitata dall'irrigatore durante la fase di irrigazione.
- **Metodi**
  - **Costruttore(id: intero, state: irrigator_type, p: pressure): Irrigator,** Inizializza un oggetto Irrigatore con l'id, stato e pressione specificati.
  - **getId(): intero,** restituisce l'id univoco dell'irrigatore.
  - **getState(): irrigator_type,** restituisce lo stato attuale dell'irrigatore.
  - **setState(newState: irrigator_type): void,** imposta lo stato dell'irrigatore al valore specificato.
  - **getPressure(): pressure,** restituisce la pressione d'irrigazione attuale.
  - **setPressure(newP: pressure): void,** imposta la pressione d'irrigazione al parametro specificato.
- **Enumerazione irrigator_type**
  - **IrrigatorON:** irrigatori accesi.
  - **IrrigatorOFF:** irrigatori spenti.
  - **change_pressure:** cambio della pressione d'irrigazione.
- **Enumerazione pressure**
  - **LOW:** pressione d'irrigazione bassa.
  - **MEDIUM:** pressione d'irrigazione media.
  - **HIGH:** pressione d'iirigazione alta.

5.6 Pseudo-codice Sensore

La classe Sensore gestisce i sensori di sicurezza all'interno della serra. Questa classe fornisce metodi per controllare lo stato e rilevare movimenti.

- **Attributi**
  - **id: intero,** identifica in modo univoco un sensore specifico all'interno del sistema.
  - **state: sensor_type,** lo stato corrente del sensore.
  - **check: booleano,** indica la presenza o l'assenza di movimento rilevato dal sensore.
- **Metodi**
  - **Costruttore(id: intero, state: sensor_type): Sensor,** Inizializza un oggetto Sensore con l'id e lo stato specificati.
  - **getId(): intero,** restituisce l'ID univoco del sensore.
  - **getState(): sensor_type,** restituisce lo stato attuale del sensore.
  - **setState(newState: sensor_type): void,** imposta lo stato del sensore al valore specificato.
  - **getCheck(): booleano,** restituisce il valore corrente di movimento rilevato dal sensore.
  - **setCheck(value: booleano): void,** imposta il valore di movimento del sensore.
- **Enumerazione sensor_type**
  - **SensorON:** sensori accesi.
  - **SensorOFF:** sensori spenti.

5.7 Pseudo-codice Dispositivo

La classe Dispositivo gestisce i dispositivi all'interno della serra. Questa classe fornisce metodi per programmare l'accensione di un determinato dispositivo in un intervallo orario specifico e per ottenere informazioni sullo stato del dispositivo.

- **Attributi**
  - **id: intero,** identifica in modo univoco un dispositivo specifico all'interno del sistema.
  - **state: device_type,** lo stato corrente del dispositivo.
  - **nome: nome_type,** componente su cui adopera il dispositivo.
- **Metodi**
  - **Costruttore(id: intero, state: device**$_t ype, nome : nome\_type) : Device, Inizializza un oggetto Dispositivo$
  - **getProgrammed(): tuple(intero, intero),** Restituisce una tupla contenente l'inizio e la fine dell'intervallo programmato per il dispositivo.
  - **programmed_device(intervalloPrimo: intero, intervalloSecondo: intero,): void** programma l'attivazione del dispositivo in un intervallo di tempo specificato.
  - **getState(): device_type,** Restituisce lo stato attuale del dispositivo.
  - **setState(newState: device**$_t ype) : void, Imposta lo stato del dispositivo al valore specificato.$**getNome(): no**
  - **setNome(new_Name: nome_type,): void** Imposta il nuovo componente su cui il dispositivo adopera.
- **device_type:**
  - **DeviceON:** Dispositivo acceso.
  - **DeviceOFF:** Dispositivo spento.
  - **programmed:** Dispositivo programmato.
- **nome_type:**

- **TV**
- **DISHWASHER**
- **WASHING_MACHINE**
- **COFFEE_MAKER**
- **MICROWAVE**
- **HEATED_BLANKET**
- **SPEAKERS**
- **LAWN_MOWER**

6 Schemi Database

Per descrivere i vari componenti all'interno del database, oltre a specificare gli attributi indicati nei rispettivi pseudo-codici, vi sono gli attributi t, pid e insertion che rispettivamente indicano l'iterazione dei test, l'identificatore del processo e il tempo di inserimento dell'istanza all'interno del database

6.1 Schema Luce

```
CREATE TABLE IF NOT EXISTS Light (
    t INT NOT NULL,
    id INT NOT NULL,
    stato VARCHAR(20) NOT NULL,
    color VARCHAR(20) NOT NULL,
    intensity INT NOT NULL,
    pid INT NOT NULL,
    temp VARCHAR(25) NOT NULL,
    PRIMARY KEY (t, pid)
);
```

6.2 Schema Condizionatore

```
CREATE TABLE IF NOT EXISTS Conditioner (
    t INT NOT NULL,
    id INT NOT NULL,
    stato VARCHAR(20) NOT NULL,
    temperature INT NOT NULL,
    pid INT NOT NULL,
    temp VARCHAR(25) NOT NULL,
    PRIMARY KEY (t, pid)
);
```

6.3 Schema Videocamera

```
CREATE TABLE IF NOT EXISTS Camera (
    t INT NOT NULL,
    id INT NOT NULL,
    stato VARCHAR(20) NOT NULL,
    recording INT NOT NULL,
    pid INT NOT NULL,
    temp VARCHAR(25) NOT NULL,
    PRIMARY KEY (t, pid)
);
```

6.4 Schema Irrigatore

```
CREATE TABLE IF NOT EXISTS Irrigator (
    t INT NOT NULL,
    id INT NOT NULL,
    stato VARCHAR(20) NOT NULL,
    pressure VARCHAR(20) NOT NULL,
    pid INT NOT NULL,
    temp VARCHAR(25) NOT NULL,
    PRIMARY KEY (t, pid)
)
```

6.5 Schema Sensore da giardino

```
CREATE TABLE IF NOT EXISTS SensorGarden (
    t INT NOT NULL,
    id INT NOT NULL,
    stato VARCHAR(15) NOT NULL,
    temperature INT NOT NULL,
    humidity INT NOT NULL,
    pid INT NOT NULL,
    temp VARCHAR(25) NOT NULL,
    PRIMARY KEY (t, pid)
);
```

6.6 Schema Sensore

```
CREATE TABLE IF NOT EXISTS Sensor (
    t INT NOT NULL,
    id INT NOT NULL,
    stato VARCHAR(15) NOT NULL,
    movement INT NOT NULL,
    pid INT NOT NULL,
    temp VARCHAR(25) NOT NULL,
    PRIMARY KEY (t, pid)
);
```

6.7 Schema Dispositivo

```
CREATE TABLE IF NOT EXISTS Device (
    t INT NOT NULL,
    id INT NOT NULL,
    stato VARCHAR(20) NOT NULL,
    nome VARCHAR(20) NOT NULL,
    inizio INT NOT NULL,
    fine INT NOT NULL,
    pid INT NOT NULL,
    temp VARCHAR(25) NOT NULL,
    PRIMARY KEY (t, pid)
);
```