

RELAZIONE PROGETTO SISTEMI ORIENTATI AD INTERNET

ReactiveHomeAutomationSystem

Giulia Oddi & Luca Taverna - 25 Ottobre 2023

Introduzione

Descrizione Generale

Il progetto ha avuto come obiettivo lo sviluppo di un sistema che effettui la simulazione di una singola stanza automatizzata caratterizzata dalla presenza di alcuni sensori. Le varie componenti del sistema sono realizzate come microservizi e vengono eseguite in container Docker.

Le componenti sono:

- Web-app: fornisce una interfaccia utente per monitorare e modificare lo stato della stanza;
- Backend: permette il dialogo tra il frontend ed i vari sensori, propagando i comandi degli utenti all'attuatore e ricevendo gli stati aggiornati dai vari sensori, che verranno poi mostrati all'utente tramite web-app;
- Attuatore: riceve dal backend lo stato aggiornato del sistema ed i comandi inseriti dall'utente e ne controlla la validità;
- Weather-service: microservizio che periodicamente invia al backend una temperatura che simula quella esterna;
- Window-door: definisce il comportamento dei sensori porte e dei sensori finestre che possono essere aperte, chiuse o in stato di errore;
- Heatpump: definisce il comportamento dei sensori pompa di calore che può essere accesa, spenta o in stato di errore e la temperatura a cui deve operare;
- Thermometer: definisce il comportamento del sensore termometro, che simula il cambiamento di temperatura nella stanza in base agli stati degli altri sensori;

Esecuzione

L'esecuzione dell'applicazione avviene tramite l'utilizzo di Docker Compose ed in particolare l'applicazione viene avviata tramite il comando: `docker-compose up --build`.

Questo comando permette la creazione delle immagini e l'esecuzione dei singoli container definiti nel file `docker-compose.yml`.

Successivamente, utilizzando il browser **Firefox**, la web app è disponibile all'indirizzo: <http://oddi-taverna soi2223.unipr.it:8080/>. Per effettuare l'accesso all'applicazione web è necessario autenticarsi tramite un account Google.

Per far in modo che l'url indicato venga risolto nell'indirizzo della macchina locale, è necessario aver modificato, tramite permessi di root, il file `/etc/hosts` inserendo:

```
127.0.0.1 oddi-taverna soi2223.unipr.it www.oddi-taverna soi2223.unipr.it
```

Descrizione Architettuale

1. Containers Docker

Come detto in precedenza, l'esecuzione dei microservizi avviene tramite l'utilizzo di Docker Compose. In particolare, nel file `docker-compose.yml` viene definita una rete di tipo `bridge`, poiché i microservizi definiti all'interno dei containers sono standalone e necessitano di comunicare tra loro. La rete utilizza la subnet `10.88.0.0/16`, avente quindi una subnet mask di 16 bit. In alcuni servizi, per facilitare le connessioni websocket, è stato specificato tramite `depends_on` quali container devono già essere in esecuzione (`condition: service_started`) prima di potersi avviare.

Sono stati definiti i seguenti containers:

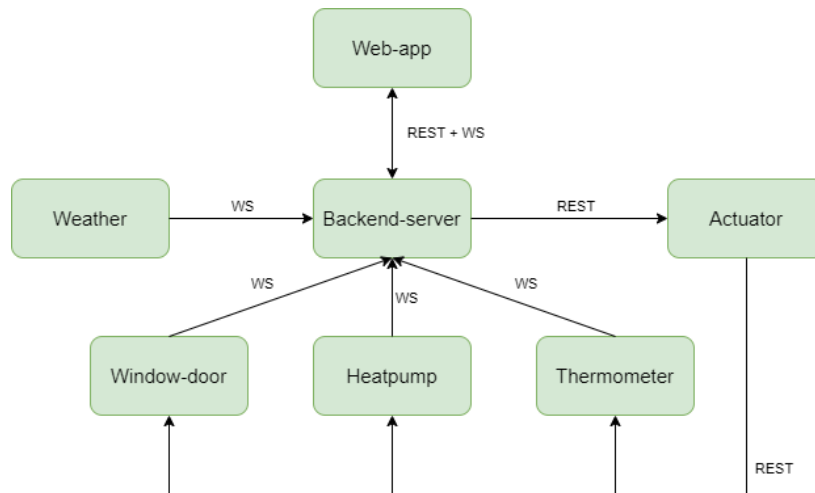
- **weather-service**: crea l'immagine a partire dalla directory `./weather-service`.
- **window-door**: crea l'immagine a partire dalla directory `./sensor-service/window-door`.
- **heatpump-service**: crea l'immagine a partire dalla directory `./sensor-service/heat-pump`.
- **thermometer-service**: crea l'immagine a partire dalla directory `./sensor-service/thermometer`.
- **actuator-service**: crea l'immagine a partire dalla directory `./actuator-service`.
- **backend-server**: crea l'immagine a partire dalla directory `./backend` e gli assegna l'indirizzo `10.88.0.11`. Dipende da: `weather-service`, `window-door`, `heatpump-service`, `thermometer-service`.
- **web-app**: crea l'immagine a partire dalla directory `./frontend`. Dipende da: `backend-server`.

Inoltre in ogni servizio è stato inserito `restart: unless-stopped`, per permettere ai container di riavviarsi in caso di spegnimento.

Ogni container viene costruito tramite il suo specifico Dockerfile, nel quale sono contenute le istruzioni necessarie alla sua esecuzione. Ad esempio, nel Dockerfile del backend viene specificato di utilizzare `node:12-alpine` come immagine base e vengono specificate le directory necessarie da copiare nel file system del container per poi eseguire il file `server.js` tramite `npm`.

2. Microservizi

Tutti i microservizi comunicano tra loro, tramite le interazioni riportate nel diagramma:



Frontend

Per permettere le richieste HTTP all'url <http://oddi-taverna soi2223.unipr.it:8080/> e la comunicazione tramite websocket con il backend, è stato modificato il file di configurazione di Apache (`my-httpd.conf`) inserendo:

```
<VirtualHost *:80>
    ServerName oddi-taverna.soi2223.unipr.it
    ServerAlias www.oddi-taverna.soi2223.unipr.it oddi-taverna.soi2223.unipr.it

    ProxyPass      /api/      http://10.88.0.11:8000/
    ProxyPass      /ws        ws://10.88.0.11:7000/
    ProxyPassReverse /ws        ws://10.88.0.11:7000/
</VirtualHost>
```

Accedendo all'indirizzo <http://oddi-taverna.soi2223.unipr.it:8080/>, all'utente viene chiesto di effettuare l'autenticazione, che avviene utilizzando **OpenID** fornito da Google. Per fare ciò, sono state aggiornate le credenziali di Google OAuth e inserite nel file `.env`:

```
OIDC_CLIENT_ID="851422523733-satu711g8moegjea5rt74n5rlpf59mbn.apps.googleusercontent.com"
OIDC_SECRET="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"
OIDC_REDIRECT="http://oddi-taverna.soi2223.unipr.it:8080"
```

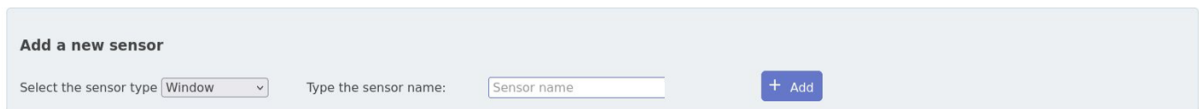
Per il login è quindi necessario un account Google. Inserendo email e password, il frontend comunica tramite REST con il backend per validarle e permettere, tramite l'uso di un token, l'accesso all'applicazione.

Dopo che l'utente è stato autenticato, la web app comunica con il backend tramite le websocket, per ottenere gli aggiornamenti sugli stati dei sensori della stanza ed inoltre per inoltrare i comandi specificati dall'utente tramite interfaccia grafica. In particolare, il frontend si sottoscrive al websocket server del backend reperibile all'indirizzo `10.88.0.11:7000` tramite il messaggio `{"type": "subscribe", "target": "room_properties"}`.

In questo modo tramite il paradigma **publish-subscribe** il frontend riceve costantemente aggiornamenti sullo stato della stanza. Il frontend utilizza poi la stessa websocket per inviare i comandi per richiedere un cambio di stato da parte dei sensori.

Lo stato della stanza viene mostrato all'utente tramite una Single Page Application caratterizzata da 4 differenti sezioni:

1. Add a new sensor:



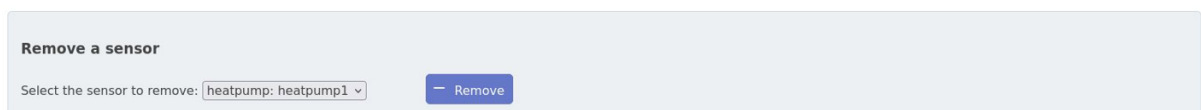
Sezione dedicata all'aggiunta di nuovi sensori. In questa sezione viene chiesto all'utente di scegliere, tramite un menu a tendina, il tipo del sensore da aggiungere tra door, window, thermometer oppure heatpump e il nome ad esso associato. Inoltre, se l'utente decide di aggiungere una pompa di calore, viene chiesto di specificare anche la temperatura desiderata, che deve essere compresa tra 15 e 35. Il sistema, oltre che a controllare la validità della temperatura inserita, controlla che non esista già un sensore dello stesso tipo con il nome specificato.

Se i dati inseriti sono validi, tramite il button “Add” vengono inviati i seguenti dati al backend:

```
addSensor = {  
  action: ADD,  
  sensor_type: type,  
  sensor_name: name,  
  state: OFF_CLOSE,  
  temperature: temperature};
```

Come si può notare, lo stato viene impostato di default a chiuso oppure spento, in base al tipo di sensore.

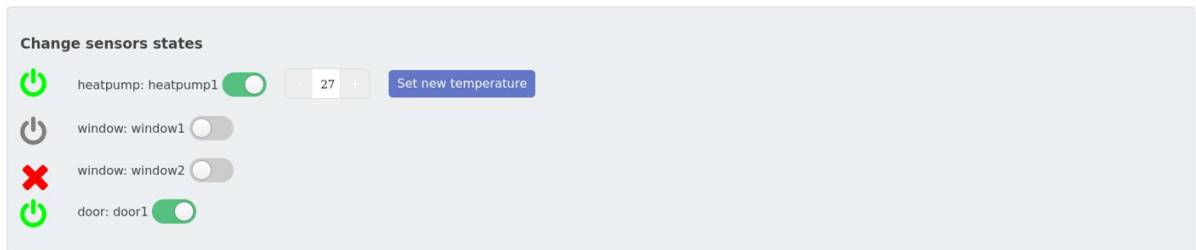
2. Remove a sensor:



Sezione dedicata alla rimozione di sensori già esistenti. L'utente può scegliere il sensore da rimuovere tramite un menu a tendina, che viene aggiornato dal frontend ogni volta che riceve la lista dei sensori aggiornata. Tramite il button “Remove”, quindi, viene inviato al backend il messaggio per la rimozione del sensore:

```
removeSensor = {  
  action: REMOVE,  
  sensor_type: type,  
  sensor_name: name };
```

3. Change sensors states:



Sezione dedicata ai cambiamenti di stato dei vari sensori. Di ogni sensore viene indicato, a sinistra del nome, lo stato corrente, tramite un'icona:

- verde: acceso o aperto;
- grigio: spento o chiuso;
- rosso: stato di errore.

Tramite un toggle switch, l'utente può inviare un comando di cambiamento di stato:

```
changeState = {  
    action: state,  
    sensor_type: type,  
    sensor_name: name };
```

Inoltre, se il sensore è in stato di errore, cliccando sul toggle switch è possibile riportare il sensore in uno stato operativo.

Il toggle switch non permette di cambiare nuovamente lo stato di un sensore fino a che l'ultimo cambiamento non ha avuto l'effetto desiderato. Nel caso in cui lo si ripremesse prima di ciò, il frontend invierà nuovamente al backend l'ultimo comando di cambio stato inviato. Per esempio, se è stato premuto il toggle switch di una finestra per aprirla, esso non potrà inviare il comando per chiuderla fino a che l'icona associata non sarà diventata verde (sensore acceso), se viene comunque premuto il frontend invierà nuovamente il comando di aprire quella finestra.

Per quanto riguarda le pompe di calore, in questa sezione è possibile anche modificare la temperatura operativa, tramite i pulsanti +/-, e inviarla tramite il button "*Set new temperature*". Questo avviene solo se la pompa di calore è accesa. In caso contrario, viene mostrato un messaggio di errore. Il messaggio inviato al backend è il seguente:

```
changeTemperature = {  
    action: ON_OPEN,  
    sensor_type: 'heatpump',  
    sensor_name: name,  
    state: state,  
    temperature: temperature };
```

La gestione dei cambiamenti degli stati avviene tramite un `BehaviorSubject (RxJS)`, che usa la lista dei sensori aggiornata come valore iniziale e che emette come eventi i cambiamenti di essa ricevuti dal backend. Questi cambiamenti vengono estratti tramite il metodo `.pipe()` di RxJS, in cui vengono cercati i sensori modificati (aggiunti oppure a cui sono state modificate le proprietà) ed i sensori rimossi. Successivamente, tramite il metodo `.subscribe()`, vengono gestiti e/o creati gli elementi di html che permettono di visualizzare e di modificare le proprietà dei sensori.

4. Charts:



Read only dashboard contenente i grafici relativi agli stati (in blu) ed alle temperature (in rosso) dei sensori della stanza e del weather-service.

Anche in questo caso, per la gestione degli eventi viene utilizzato il `BehaviorSubject` e quindi gli aggiornamenti della lista dei sensori sono gestiti tramite il metodo `.subscribe()`. In particolare, tramite la libreria **Chart.js**, ogni volta che viene aggiunto un sensore vengono creati i relativi grafici, tramite il metodo `.create_chart()`, e successivamente questi vengono aggiornati ogni volta che viene ricevuto un cambiamento dello stato e/o della temperatura tramite il metodo `.push()`, per inserire il nuovo valore arrivato, ed il metodo `.update()`, per mostrare il grafico aggiornato. Inoltre, se viene riscontrata la rimozione di un sensore, i grafici relativi ad esso vengono rimossi.

Backend

Il backend è un'entità che si occupa di inoltrare i messaggi e permettere a tutte le altre componenti di comunicare. In particolare, oltre che a comunicare con il frontend, riceve gli aggiornamenti dei cambi di stato dai vari sensori, sotto forma di lista, tramite l'implementazione del paradigma **publish-subscribe**.

Quindi, gestisce le comunicazioni:

- si sottoscrive tramite il messaggio `{"type": "subscribe", "target": "temperature"}` alla WS API esposta dal microservizio weather-service tramite `ws://weather-service:5000;`
- si sottoscrive tramite il messaggio `{"type": "subscribe", "target": "window-door", "list": null}` alla WS API esposta dal microservizio window-door tramite `ws://window-door:4000;`

- si sottoscrive tramite il messaggio `{"type": "subscribe", "target": "heatpump", "list": null}` alla WS API esposta dal microservizio heatpump tramite `ws://heatpump-service:4000;`
- si sottoscrive tramite il messaggio `{"type": "subscribe", "target": "thermometer_temperature", "list": null}` alla WS API esposta dal microservizio thermometer tramite `ws://thermometer-service:4000.`

Tramite il campo "list" nel messaggio di subscribe il backend può inviare ad un microservizio (window-door, heatpump, thermometer) appena riavviato la lista aggiornata dei sensori di cui si occupa, in questo modo il microservizio riesce a recuperare il suo stato prima di essere stato interrotto.

Ogni volta che il backend riceve un aggiornamento da queste API, ricerca all'interno della lista ricevuta i cambiamenti di stato e/o temperatura e gli eventuali sensori rimossi per aggiornare la lista `sensors_properties` in memoria. Dopodiché, la lista aggiornata viene inoltrata all'attuatore tramite REST all'indirizzo

`http://actuator-service:3000/sensor_properties.`

Inoltre, il backend mantiene anche una comunicazione attiva con il frontend. Ogni volta che la lista `sensors_properties` viene aggiornata, il backend la invia al frontend sottoscritto alla sua WS API.

Infine, il backend si occupa anche di propagare i comandi ricevuti dalla web app all'attuatore, tramite REST all'indirizzo `http://actuator-service:3000/command.`

Actuator

L'attuatore si mette in ascolto di messaggi ricevuti dal backend. In particolare, i messaggi vengono ricevuti su due endpoint differenti:

- `/sensor_properties`: tramite questo, l'attuatore riceve dal backend la lista dei sensori della stanza aggiornata. In questo modo l'attuatore viene sempre aggiornato dal backend sui sensori presenti nella stanza e sui loro stati attuali.
L'attuatore, salva gli aggiornamenti in memoria e successivamente inoltra la lista al microservizio thermometer, verso l'indirizzo `http://thermometer-service:3000/room_properties`, tramite REST.
- `/command`: tramite questo, l'attuatore riceve dal backend i comandi da inoltrare ai sensori inseriti dall'utente. In questo caso l'attuatore si occupa di validare la correttezza dei comandi ricevuti. Nel caso in cui il comando non sia considerato valido, ovvero il risultato del cambiamento di stato richiesto dall'utente è uguale allo stato corrente del sensore (ad esempio l'utente chiede di aprire una finestra già aperta) l'attuatore restituisce al backend una risposta di errore tramite il codice `304: Not Modified`. Se, invece, il comando è considerato valido, esso viene propagato al microservizio del sensore utilizzando il backchannel corretto. Questa comunicazione avviene tramite REST. In particolare, viene utilizzato l'endpoint `'/change-state'` se l'azione riguarda un cambiamento di stato (`ON_OPEN=1` o `OFF_CLOSE=0`), oppure l'endpoint `'/add-sensor'` se invece riguarda un'aggiunta (`ADD = 2`) o una rimozione di un sensore (`REMOVE = 3`).

Per simulare lo spegnimento del microservizio viene chiamato nel codice `process.exit()` dopo 3 azioni (aggiunte e/o rimozioni) di un sensore di tipo termometro. In questo modo grazie all'opzione `restart` in `docker-compose` il container viene riavviato senza causare l'interruzione dell'intero sistema.

Weather

Il weather service è un microservizio che simula il cambiamento della temperatura all'esterno della stanza. Dopo che il backend ha inviato un messaggio di sottoscrizione `{"type": "subscribe", "target": "temperature"}`, il weather-service invia periodicamente la nuova temperatura tramite websocket al backend. Il messaggio utilizzato per inviare gli aggiornamenti è il seguente:

```
msg = {type: 'temperature', dateTime: DateTime.now().toISO(), value};
```

Anche per il weather service è stato simulato lo spegnimento del container tramite l'utilizzo del metodo `_scheduleDeath()`, in cui viene generato casualmente il tempo per cui il container resterà attivo prima di chiamare `process.exit()`.

Sensori

Per la gestione dei sensori presenti nella stanza, sono stati realizzati 3 differenti container: `window-door`, `heatpump-service` e `thermometer-service`. In particolare, ogni container si occupa di tenere in memoria le proprietà di sensori dello stesso tipo, grazie all'utilizzo di una lista. Ad esempio la lista del microservizio `window-door` potrebbe essere la seguente:

```
sensors = [
  { type: 'window', name: 'window1', state: CLOSE},
  { type: 'window', name: 'window2', state: CLOSE},
  { type: 'door', name: 'door1', state: CLOSE},
];
```

Si è scelto di unire i sensori di tipo `door` e di tipo `window`, perché essi sono caratterizzati dagli stessi stati (`OPEN`, `CLOSE` e `ERROR`) e quindi anche dallo stesso comportamento.

Ogni sensore implementa il pattern `publish-subscribe`, tramite una WS API, per inviare aggiornamenti al backend. Periodicamente viene controllato se la lista delle proprietà dei sensori ha subito cambiamenti ed, in questo caso, viene inviata al backend tramite il messaggio:

```
msg = {type: 'sensors_list', dateTime: DateTime.now().toISO(), list: sensors};
```

Inoltre i sensori implementano una API REST per ricevere comandi dall'attuatore tramite `backchannel`. Tutti questi microservizi possono ricevere comandi relativi all'aggiunta o alla rimozione di sensori della tipologia di cui si occupano, tramite l'endpoint `/add-sensor`. Se l'azione riguarda l'aggiunta di un sensore, il microservizio in questione si occupa di aggiungere alla propria lista un sensore con le proprietà indicate nel messaggio. Se, invece, l'azione riguarda la rimozione di un sensore già esistente, il microservizio rimuove dalla sua lista il sensore in questione.

Per gestire queste operazioni, ed in generale tutte le modifiche alle liste dei microservizi contenenti le proprietà dei sensori della stanza, è stata utilizzata la programmazione funzionale. Ad esempio, se un microservizio riceve un comando inerente alla rimozione di un sensore, esso viene ricercato e rimosso dalla lista tramite una operazione di `filter`:

```
sensors = sensors.filter( item =>
item.type !== postData.sensor_type || item.name !== postData.sensor_name);
```


Per i sensori è stato simulato lo spegnimento dei container tramite l'utilizzo del metodo `_scheduleDeath()`, in cui viene generato casualmente il tempo per cui essi rimarranno attivi prima di chiamare `process.exit()`. Il container spento riparte grazie all'opzione `restart` in `docker-compose`, senza causare l'interruzione dell'intero sistema.

Window-Door

Come detto in precedenza, per i sensori di tipo window e di tipo door è stato realizzato un unico microservizio, dato che sono caratterizzati dagli stessi stati. Infatti, le porte e le finestre possono avere 3 differenti stati: OPEN, CLOSE oppure ERROR. Lo stato di errore è stato simulato tramite il metodo `_scheduleError()`, in cui, dopo un numero pseudocasuale di millisecondi, ad un sensore casuale appartenente alla lista viene modificato lo stato. Lo stato può poi essere ripristinato dall'utente tramite interfaccia grafica semplicemente cliccando sul toggle switch relativo al sensore in questione.

Inoltre, questo microservizio accetta tramite REST due diverse tipologie di messaggi:

- messaggi per il cambio di stato: questi messaggi vengono ricevuti sull'endpoint `/change-state`. In questo caso il servizio si occupa di analizzare il messaggio ricevuto ed aggiornare lo stato del sensore in questione.
- messaggi per la rimozione o l'aggiunta di un sensore: questi messaggi sono ricevuti sull'endpoint `/add-sensor`. Anche in questo caso la lista viene aggiornata.

Heatpump

Il microservizio `heatpump-service` è stato realizzato per gestire tutti i sensori che rappresentano pompe di calore. Questi sensori sono anch'essi caratterizzati da 3 diversi stati: ON, OFF e ERROR. Anche in questo caso è stato implementato il metodo `_scheduleError()`, avente lo stesso funzionamento descritto in `window-door`, per poter simulare lo stato di errore.

Le heatpump sono inoltre caratterizzate da una temperatura operativa, che può essere specificata dall'utente nel momento in cui decide di inserire una nuova pompa di calore o che può essere modificata solo nel caso in cui l'heatpump sia accesa.

Anche questo microservizio accetta tramite REST due diverse tipologie di messaggi:

- messaggi per il cambio di stato: questi messaggi vengono ricevuti sull'endpoint `/change-state`. In questo caso il servizio si occupa di analizzare il messaggio ricevuto ed aggiornare lo stato e/o la temperatura operativa del sensore in questione.
- messaggi per la rimozione o l'aggiunta di un sensore: questi messaggi sono ricevuti sull'endpoint `/add-sensor`. Anche in questo caso la lista viene aggiornata.

Thermometer

Questo microservizio viene utilizzato per tenere traccia di tutti i sensori di tipo termometro presenti nella stanza. Per semplicità, è stata inserita anche in questi sensori la proprietà `"state"` per avere una rappresentazione comune agli altri sensori e facilitare quindi la gestione delle liste. Lo stato dei termometri è però irrilevante. La proprietà fondamentale di questi sensori è la temperatura, che misura la temperatura interna alla stanza e che di default viene impostata a 20°C.

Questo microservizio accetta tramite REST due diverse tipologie di messaggi:

- messaggi per la rimozione o l'aggiunta di un sensore: come per i sensori descritti in precedenza, questi messaggi sono ricevuti sull'endpoint `/add-sensor` e in base al messaggio ricevuto la lista viene aggiornata.
- messaggi che riguardano la lista aggiornata dei sensori presenti nella stanza: questi messaggi vengono ricevuti sull'endpoint `/room_properties` e sono inviati dall'actuator. Ogni volta che il termometro riceve questi aggiornamenti, calcola due temperature importanti: la prima, che viene salvata in `temp_diff_weather`, indica la differenza tra la temperatura interna della stanza e quella all'esterno (relativa al `weather-service`); la seconda, salvata in `temp_diff_heatpump`, indica la differenza tra la temperatura interna della stanza e la temperatura operativa dell'heatpump. Nel caso in cui siano presenti più heatpump, viene considerata la temperatura operativa massima. Dopodiché, sulla base delle proprietà di tutti i sensori della stanza, viene aggiornata la temperatura rilevata dal termometro.

La simulazione del cambiamento di temperatura è stata implementata nel metodo `temperature_simulation()`. I cambiamenti di temperatura avvengono gradualmente, in base alla casistica con un diverso numero di iterazioni, ogni 2 secondi tramite un `.setTimeout()`. Al termine delle iterazioni, la temperatura del termometro si stabilizza se non ci sono modifiche allo stato della stanza. Nel caso in cui ci fosse un sensore in stato di errore, allora nel calcolo della temperatura viene considerato come se fosse spento (oppure chiuso).

La temperatura viene quindi calcolata e modificata in base alle caratteristiche attuali della stanza:

- se almeno una heatpump è accesa, con temperatura operativa maggiore di quella attuale della stanza:
 - se almeno una finestra è aperta:
 - se la temperatura all'esterno è maggiore di quella della stanza, allora la temperatura del termometro salirà in 3 iterazioni fino a raggiungere il massimo tra la temperatura esterna e quella della pompa di calore.
 - altrimenti significa che la temperatura all'esterno è minore di quella all'interno e quindi la temperatura del termometro aumenta più lentamente, con 8 iterazioni, fino a raggiungere la temperatura della pompa di calore.
 - altrimenti, se tutte le finestre sono chiuse, la temperatura salirà fino a raggiungere quella della pompa di calore in 5 iterazioni.
- se almeno una finestra è aperta e tutte le pompe di calore sono spente, oppure almeno una di esse è accesa ma con temperatura operativa inferiore di quella della stanza:
 - se nessuna porta è aperta, allora la temperatura del termometro aumenta con 3 iterazioni, fino a raggiungere la temperatura dell'esterno.
 - altrimenti, la temperatura del termometro aumenta più lentamente (5 iterazioni) dato il ricircolo d'aria con il resto della casa, fino a raggiungere la temperatura esterna.
- in tutti gli altri casi, ovvero se le finestre sono chiuse e le pompe di calore sono spente o con temperatura operativa inferiore rispetto a quella attuale della stanza, la temperatura del termometro non viene modificata.

Conclusioni e sviluppi futuri

In conclusione quindi, il sistema simula il cambiamento di temperatura di una stanza, in base al comportamento dei sensori di diversa tipologia al suo interno. L'utente ha la possibilità di interagire con il sistema e quindi apportare dei cambiamenti alle proprietà dei sensori grazie ad una interfaccia grafica.

In generale, il progetto è stato realizzato utilizzando i linguaggi HTML, CSS e JavaScript (NodeJS, RxJS e Chart.js) e Docker Compose per la gestione dei microservizi. Le comunicazioni tra i vari microservizi utilizzano due paradigmi differenti, REST e websocket.

Per quanto riguarda i possibili sviluppi futuri, un primo miglioramento da apportare è relativo alla sicurezza del sistema. Infatti, attualmente, è stata gestita l'autenticazione solamente nella comunicazione tra frontend e backend. Tutte le altre comunicazioni non sono protette, quindi, in un eventuale ambiente di produzione, sarebbe necessario implementare un meccanismo di autenticazione anche negli altri canali di comunicazione.

Inoltre, sarebbe possibile separare in due microservizi differenti la gestione delle porte e delle finestre, per poi quindi poterne differenziare il comportamento.

Per isolare maggiormente i singoli servizi, sarebbe ottimale realizzare un container differente per ogni sensore. In questo modo bisognerebbe gestire a runtime la creazione e lo spegnimento dei container sulla base dell'aggiunta o della rimozione dei sensori da parte dell'utente.

Il progetto è disponibile su GitHub all'indirizzo:

<https://github.com/giuliaOddi/ReactiveHomeAutomationSystem>.