POLITECNICO DI TORINO

Report Progetto Finale Tecnologie per IOT

Gruppo: 14

Claudio Raccomandato 245488, Giovanni Gaddi 246574, Federico Giuranno 234437

Introduzione

Riassunto

Per il nostro progetto abbiamo realizzato un'applicazione Android che permette di controllare 5 diversi dispositivi IOT per smart home:

- Una sveglia
- Una serratura elettronica
- Un rilevatore per perdite di gas e acqua
- Una luce
- Un sistema di condizionamento

Ognuno di questi dispositivi è connesso ad internet e comunica tramite protocollo MQTT. Inoltre, possiedono tutti un ID unico e questo rende il sistema totalmente modulare e scalabile.
Sia i dispositivi che l'applicazione vengono registrati all'interno di un Catalog, il quale memorizza all'interno di un json anche tutti i dati significativi che vengono scambiati, in modo da poter essere utilizzato anche come forma di backup (accessibile con richieste GET dall'applicazione o publish MQTT dai dispositivi IOT)

JSON

Abbiamo scelto di costruirci un nostro modello di json per lo scambio di dati. L'abbiamo costruito in modo da contenere tutte le informazioni necessarie con il minore spazio possibile, infatti richiede solo 3 parametri:

- ID, formato dalla combinazione della tipologia del dispositivo (alarm, light…) più un numero identificativo nel caso ce ne fossero più di uno (es. alarm0, light1…)

- val, un array contenente i dati che si vogliono mandare, nel nostro caso non arrivano a più di 6
- t, il time-stamp

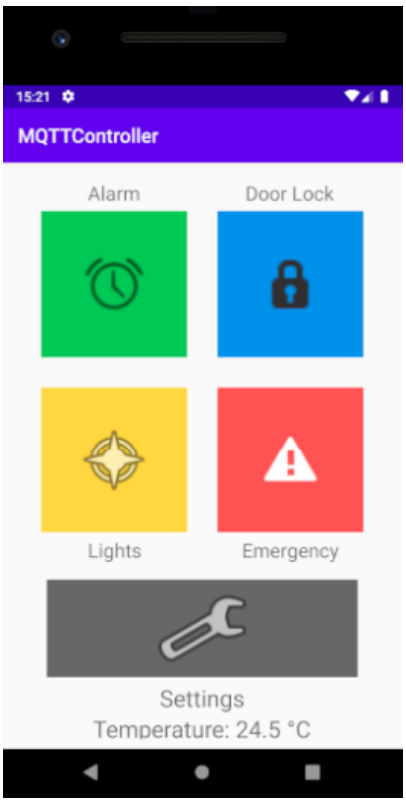
Ecco un esempio di json pubblicato al topic “tiot/14/light”

```
{
   "ID" : "light0",
   "val": [1],
   "t"  : 1234
}
```

Figura 1 - Esempio di struttura dei JSON

Applicazione

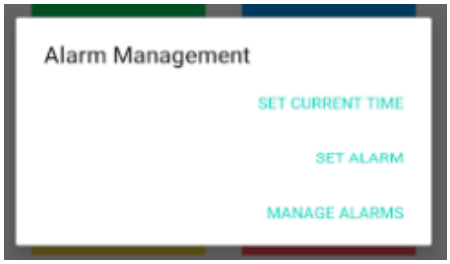
L’applicazione comunica con il Catalog tramite REST e con i dispositivi tramite il protocollo MQTT. Le richieste al Catalog vengono effettuate quando l’applicazione rileva una perdita dei dati salvati in locale, ad esempio non riconosce più lo stato delle luci.
La Home si presenta come una dashboard con 5 pannelli cliccabili: “Sveglia”, “Serratura”, “Luci”, “Emergenza” e “Impostazioni”, oltre a mostrare in basso la temperatura media della casa.



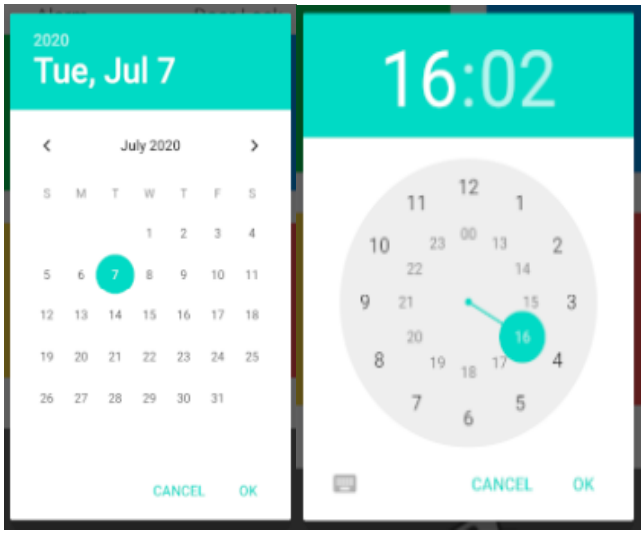
Screenshot 1 – homepage dell’applicazione

Sveglia

Permette di impostare l'ora attuale della sveglia (in caso di desincronizzazione), impostare nuove sveglie e gestire quelle già impostate.
Tutte e tre le opzioni comunicano col dispositivo inviando un JSON nel topic "tiot/14/alarm/[subtopic]".



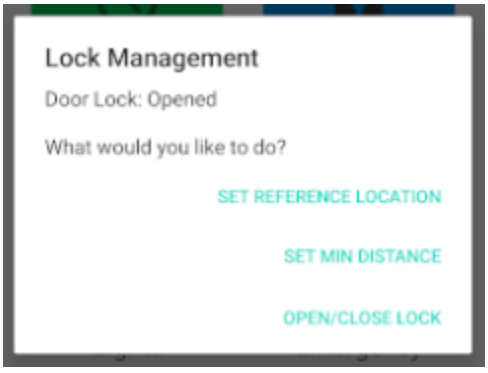
Screenshot 2 – menu pannello Sveglia



Screenshot 3 e 4 – impostazione di una nuova sveglia

Serratura

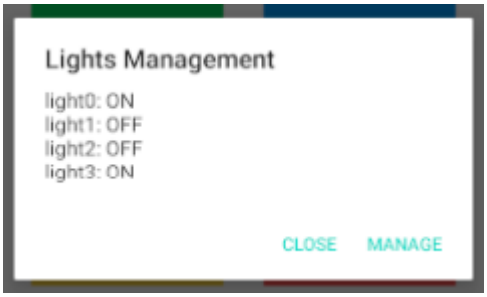
Il funzionamento della serratura è relativo alla posizione dell'utente, infatti quando l'applicazione rileva un allontanamento dallo Smart Lock, lo chiuderà automaticamente pubblicando un JSON nel topic "tiot/14/lock", che verrà ricevuto dal dispositivo. Allo stesso modo, all'avvicinamento verrà aperto automaticamente. E' possibile attivare o disattivare questa funzionalità nel pannello "Impostazioni".
Il menu permette di impostare l'attuale posizione come posizione di riferimento (ovvero dello Smart Lock), impostare la minima distanza di rilevamento per l'apertura/chiusura e aprire/chiudere manualmente la serratura.



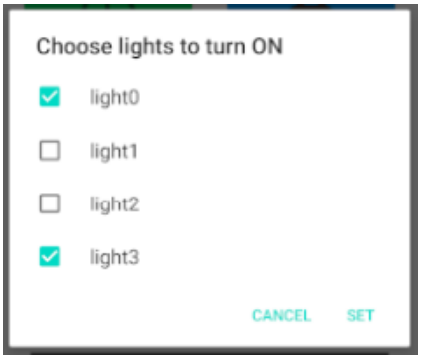
Screenshot 5 – menu pannello Serratura

Luci

Il pannello delle Luci permette di visualizzare lo stato corrente delle luci presenti nella casa e di impostare il loro stato inviando un JSON nel topic “tiot/14/light”.



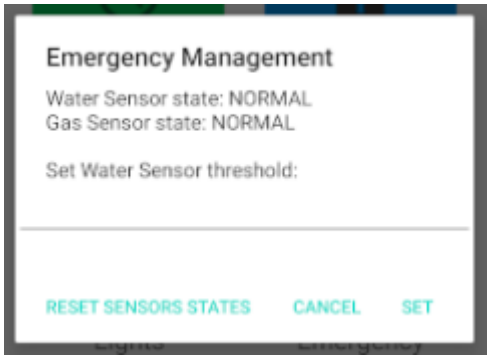
Screenshot 6 – menu pannello Luci



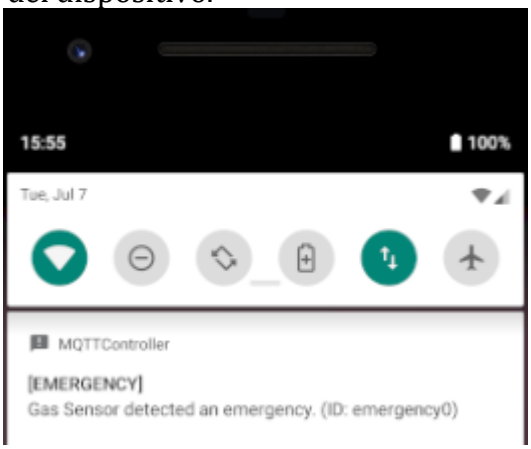
Screenshot 7 – gestione delle luci

Emergenza

Il pannello Emergenza mostra lo stato attuale dei sensori di emergenza (acqua e gas) e permette di impostare una soglia per il sensore dell’acqua o di resettare lo stato dei sensori.
Nel caso in cui uno dei due sensori rilevasse un’emergenza, invierebbe un JSON nel topic “tiot/14/datain/emergency”, al quale l’applicazione ha effettuato il subscribe. Alla ricezione di questo JSON, viene mandata immediatamente una notifica al dispositivo su cui è installata l’applicazione, specificando il tipo di emergenza e l’ID del dispositivo.



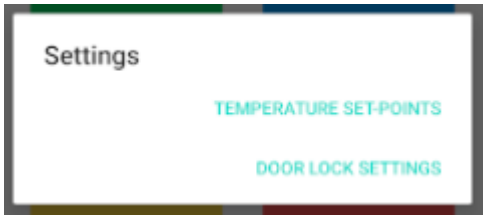
Screenshot 8 – menu pannello Emergenza



Screenshot 9 – notifica in caso di emergenza

Impostazioni

Nelle impostazioni è possibile impostare i Set-Points di temperatura per il sistema di riscaldamento/condizionamento e scegliere se usare o meno la funzionalità Smart Lock basata sulla posizione.



Screenshot 10 – menu pannello Impostazioni

Screenshot 11 – impostazione dei set-points (nell'immagine lo scenario Senza Presenza)

Dispositivi IOT

Generalmente ogni dispositivo ha 2 funzioni:

- RegisterDevice() che permette di iscriversi al Catalog tramite in json contenente l'ID del dispositivo e topic relativi alla tipologia di dispositivo. Viene inviato sul topic "tiot/14/deviceadd"
- DataUpdate() richiede al Catalog tramite Json sul topic "tiot/14/devicceUpdate" l'aggiornamento dei dati del dispositivo.

Sveglia

La sveglia è un dispositivo che sfrutta il clock interno dell'arduino per mantenere l'orario attuale, l'orario è aggiornato tramite json sul topic MQTT "tiot/14/alarm/clock", quando viene aggiornato l'orario tutte le sveglie antecedenti all'orario aggiornato vengono eliminate. Il dispositivo può immagazzinare 20 sveglie, che vengono comparate con l'orario attuale e poi attivate tramite suono di un buzzer, e disattivate tramite la pressione di un pulsante. La sveglia già attivata viene quindi cancellata inviando un json sul topic "tiot/14/alarm/delete". Per aggiungere sveglie il dispositivo riceve un json sul topic "tiot/14/alarm/clock", prima che sia effettivamente aggiunta alla lista viene controllata la validità, ovvero che non sia impostata prima dell'orario attuale. Analogamente è possibile eliminare le sveglie tramite un json sul topic "tiot/14/alarm/delete".
L'orario attuale viene quindi stampato per mezzo di un LCD.
Inoltre, è presente una fotoresistenza che viene utilizzata come sensore di luce tramite lettura di tensione analogica sul pin A0. Sono presenti due soglie per effettuare via software un processo di isteresi e al superamento di ciascuna delle due soglie viene inviato un json per comunicare l'aggiornamento del valore di luminosità.

La funzione DataUpdate viene chiamata all'avvio del dispositivo ed ogni 12 ore. risulta fondamentale in quanto permette il corretto funzionamento dell'orario ed un confronto tra le sveglie salvate nel dispositivo e nel catalog.

Serratura Elettronica

La serratura elettronica sfrutta un motore passo passo per chiudere ed aprire la serratura. In particolare, si è ipotizzato che per la chiusura sia sufficiente un giro completo, mentre per l'apertura si sfrutta un pulsante per segnalare il finecorsa. Il dispositivo inoltre segnala lo stato della serratura e della porta, nel secondo viene segnalato se la porta risulta chiusa o aperta tramite un semplice contatto. I comandi di apertura o chiusura riguardanti la serratura vengono ricevuti tramite json sul topic "tiot/14/lock".
Se la porta rimane aperta per più di 5 minuti o è arrivato il comando di chiusura della serratura mentre la porta risulta aperta il dispositivo lo segnala tramite json via MQTT.
Anche in questo dispositivo è presente la funzione DataUpdate che viene richiamata all'attivazione del dispositivo e ogni 12h aspettandosi come valore di ritorno lo stato della serratura.

Rilevatore perdite di gas e acqua

Abbiamo pensato di utilizzare due sensori per rilevare le due tipologie di perdite: un sensore di livello dell'acqua con uscita analogica ed un sensore per la quantità di gas con uscita digitale (1 non c'è gas, 0 c'è gas). L'uscita del rivelatore di gas è connessa in interrupt all'Arduino ed in caso di emergenza, tramite la funzione GasSensorDetector, imposta la variabile booleana gas = true. L'uscita del sensore di umidità viene invece letta in polling attraverso il pin analogico A0, quando il valore letto supera la soglia impostata dall'applicazione imposta la variabile booleana water = true e pubblica il json. Sempre in polling viene chiamata la funzione GasSensorManager, che nel caso in cui vi è la presenza di gas pubblica il json.
Infine, come per tutti i dispositivi, viene chiamata la funzione DataUpdate all'attivazione ed ogni 12 ore che permette di aggiornare il valore di soglia del sensore di livello dell'acqua.

Luce

La luce è un dispositivo particolarmente semplice, infatti l'unica sua funzionalità è quella di accendere o spegnere un led pilotato dall'applicazione. La funzione principale del codice è chiamata setLightValue che viene invocata quando viene pubblicato un json sul topic "tiot/14/light", contenente lo stato che deve avere la luce, rappresentata da un led nel prototipo.
Anche questo dispositivo possiede la funzione DataUpdate che all'attivazione ed ogni 12 ore permette di fare il refresh dello stato delle luci.

Sistema di condizionamento

Il sistema di condizionamento è composto da due parti: il condizionatore ed i sensori di temperatura e presenza.

Il condizionatore serve a controllare una centralina che gestisce la temperatura della casa, nel nostro prototipo formato da un led ed una ventola. I set point del dispositivo, in caso di presenza o meno, vengono pubblicati dall'applicazione su due topic, "tiot/14/temp/void" e "tiot/14/temp/presence". Oltre al dispositivo anche il Catalog memorizza nel file di storage i valori dei set point.

Il sensore di temperatura e presenza è un dispositivo che va posizionato nelle varie stanze e che pubblica ogni 3 secondi la temperatura ed in caso di variazione del valore precedente la presenza di persone nella stanza. Quando uno di questi valori viene pubblicato, il Catalog lo memorizza nel file di storage e, nel caso di valore di temperatura, calcola la media tra gli ultimi valori inviati da ciascun sensore, la memorizza e la pubblica in modo che il condizionatore regoli di conseguenza la temperatura della casa. Nel caso, invece, di valore di presenza, controlla se almeno un dispositivo ha rilevato la presenza di persone nella stanza e, nel caso il controllo risultasse positivo, associa al parametro presence_house il valore 1, dopodiché pubblica il valore di presenza di persone nella casa, in modo che venga regolata la temperatura dal condizionatore.
La funzione DataUpdate in questo caso è presente solo per il condizionatore e viene invocata all'attivazione ed ogni ora. Abbiamo infatti pensato che un refresh di controllo più frequente di set points, temperatura media e presenza fosse necessario al fine di evitare consumi eccessivi per il riscaldamento della casa.

Catalog

Abbiamo modificato il Catalog dell'esercizio 5 del laboratorio 2 della parte di software, in modo da permettergli di salvare come forma di backup i dati trasmessi dall'applicazione e dai dispositivi tramite MQTT.
Per prima cosa abbiamo registrato il Catalog ad alcuni topic utilizzati per trasmettere informazioni significative tra applicazione e dispositivi IOT, dopodichè abbiamo modificato la funzione on_message in modo da permettergli di leggere i json publicati e di salvare i dati che volevamo in un file json locale. La lettura del file, da parte dell'applicazione, avviene tramite richieste GET. Queste vengono gestite da una nuova classe chiamata DataStorage che va a leggere il file Storage.txt, scritto in precedenza, e restituisce un json contenente le informazioni lette nel solito formato ID-val-t. Per i dispositivi, invece, l'accesso al Catalog è totalmente diverso, questi devono infatti pubblicare un json contenente l'ID del dispositivo al topic "tiot/14/deviceUpdate". Una volta ricevuto il json il Catalog cerca nel file Storage.txt le informazioni relative al dispositivo che le chiede e pubblica un json nello stesso formato e allo stesso topic che ha utilizzato l'applicazione quando le ha scritte.
La struttura del file Storage.txt è basata su 5 tipologie di dispositivi:

- lights, per le luci
- alarms, per le sveglie
- locks, per le serrature
- emergencies, per i rilevatori di perdite di gas e acqua
- temp_set_points, per i set point dei sensori di temperatura

L'ultima key è temp_avg, ma non è un dizionario bensì contiene la media di tutti i sensori di temperatura della casa espressa in gradi Celsius. Questa rappresenta la temperatura mostrata dall'applicazione nella parte in basso della schermata principale.
Come per gli altri file di testo, Storage.txt viene creato automaticamente nel main quando non è presente:

```
if not path.exists("Storage.txt"):  # creo il file di backup
        with open("Storage.txt", "w") as file:
                file.write("{\"lights\": {}, \"alarms\": {}, \"locks\": {},
    \"emergencies\": {}, \"temp_set_points\": {},
    \"temp_avg\": null}")
                print(time.strftime("%H:%M:%S") + " - [Engine]INFO: New file
    Storage.txt created")
```

Figura 1 - creazione del file Storage.txt