CONSEGNA 4 – “SMART DOOR”

LUCA RAGAZZI

SEIOT 17-18

E’ richiesto di creare un sistema embedded che funga da varco intelligente, controllando l’accesso ad un ambiente mediante una porta.

Il sottosistema DOOR è realizzato mediante Arduino e rappresenta la porta.

La piattaforma DOOR comprende inoltre un sensore di temperatura per rilevare la temperatura nell’ambiente. Inoltre sono presenti un sensore di presenza che mantiene la porta (e la sessione su mobile) aperte se rileva una presenza ed un led il cui valore può essere cambiato tramite mobile e visualizzato dal servizio web in funzione sul gateway.

Alla porta è possibile connettersi via bluetooth ed interagire con essa mediante applicazione mobile (sviluppata per Android).

La porta a sua volta accede ad un gateway, rappresentato dal Raspberry Pi che ospita un database ed autorizza l’accesso solo qualora le credenziali fornite dall’utente combacino con credenziali presenti nel db. Oltre a ciò il gateway ospita anche un servizio web (realizzato in Node.js) che fornisce una pagina in cui poter visualizzare la temperatura corrente e il valore attuale del led.

ARDUINO

Per il seguito definiamo:

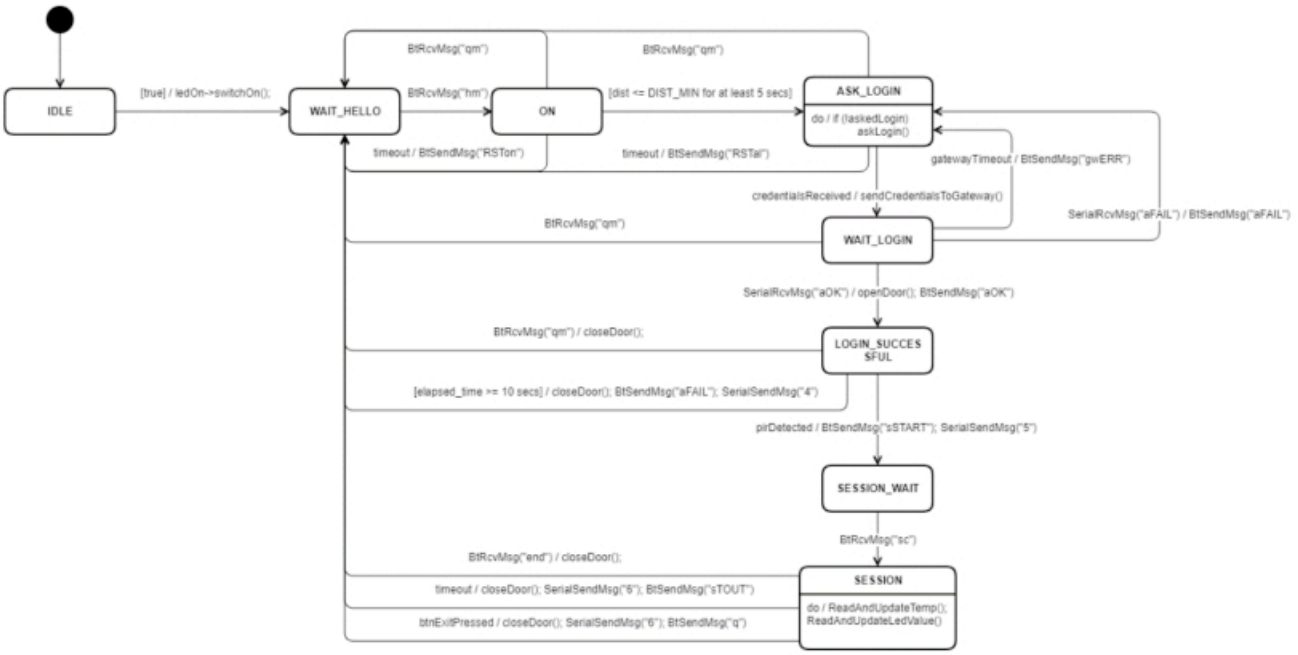
- RESET_TIMEOUT: 30 secondi
- GW_ERR_TIMEOUT: 10 secondi
- MAX_DELAY: 10 secondi

Il sottosistema DOOR (Arduino) è composto da 4 task (in esecuzione su scheduler cooperativo):

- Door: è il task principale. Rappresenta la porta.
- CommandReaderTask: Si occupa della ricezione dei messaggi provenienti da seriale hardware (collegata al gateway RasPi) e di quelli provenienti da bluetooth (software serial). Sulla base dei messaggi ricevuti setta opportunamente i valori di variabili presenti in uno SharedContext condiviso fra tutti i task e alla base delle interazioni fra essi.
- TempUpdateTask: Si occupa della lettura del sensore di temperatura (DHT11: sensore di temperatura digitale basato su protocollo one wire) ed invia i dati raccolti all’applicazione mobile e al gateway. Gli aggiornamenti vengono inviati ogni 2 secondi (per limite fisico del sensore).

· LedValueUpdateTask: Si occupa dell'invio del valore del led al mobile e al gateway ogniqualvolta l'intensità del led cambia.

Door Task



Appena il sistema parte si transita dallo stato IDLE allo stato ON e viene accesso il led ON che indica che il sistema è in funzione.

Dopodiché il sistema rimane in stato WAIT_HELLO finché qualcuno si avvicina alla porta con l'applicazione mobile aperta. Quest'ultima infatti invia un messaggio (*hm*) all'arduino per segnalargli che l'utente vuole accedere all'ambiente. Quando l'arduino rileva questo messaggio transita in stato ON in cui rileva la presenza di qualcuno ad una distanza inferiore a 50 cm per almeno 5 secondi. Se ciò non avviene entro RESET_TIMEOUT secondi l'arduino manda al mobile un messaggio di reset (*RSTon*) e torna nello stato WAIT_HELLO.

Se invece l'hello va a buon fine l'arduino transita in stato ASK_LOGIN in cui invia al mobile un messaggio di hello (*h*) e attende di ricevere da quest'ultimo le credenziali immesse dall'utente. Se entro RESET_TIMEOUT secondi non le riceve, invia al mobile un messaggio di reset (*RSTal*) e torna in WAIT_HELLO.

Se invece entro la scadenza del timeout l'arduino riceve le credenziali, le invia al gateway e transita nello stato WAIT_LOGIN. In questo stato attende una risposta dal gateway circa l'autorizzazione o meno ad accedere all'ambiente.

Se l'arduino non riceve una risposta entro GW_ERR_TIMEOUT secondi invia al mobile un messaggio (*gwERR*) e torna in stato ASK_LOGIN in cui attende che l'utente riprovi il login.

Se il gateway risponde con un messaggio di autenticazione avvenuta con successo (*aOK*) la porta si apre (servo ruota da 0 a 180°) e transita nello stato ACCESS_SUCCESSFUL.

Se invece il gateway nega l'accesso, l'arduino (*aFAIL*) invia al mobile un messaggio che segnala che l'accesso è fallito (MSG_ACC_FAIL) e torna in stato ASK_LOGIN.

Nello stato ACCESS_SUCCESSFUL l'arduino rimane in attesa che il PIR rilevi la presenza di qualcuno. Se entro MAX_DELAY secondi il PIR non ha rilevato alcuna presenza la porta viene chiusa e il sistema torna in stato WAIT_HELLO.

Se, al contrario, è stata rilevata la presenza di qualcuno entro il timeout, l'arduino manda al mobile un messaggio che indica che la sessione può iniziare (*MSG_SESS_START*) e transita in SESSION_WAIT.

Lo stato SESSION_WAIT rappresenta un punto di sincronizzazione con l'applicazione mobile. L'introduzione di questo stato è stata necessaria per evitare che gli handler per il settaggio del valore della temperatura e del led nell'app venissero chiamati prima che la creazione dell'activity fosse stata completata.

Quando l'arduino riceve il messaggio dal mobile che indica che l'activity della sessione è stata creata (*sc*) invia al gateway un messaggio che indica che qualcuno è dentro l'ambiente (MSG_INSIDE) e va nello stato SESSION.

Nella transizione allo stato SESSION viene anche settato il flag sessionStarted nello SharedContext che permette ai task TempUpdateTask e LedValueUpdateTask di inviare aggiornamenti sullo stato corrente della temperatura e del led anche al mobile.

Dopodiché l'arduino rimane in SESSION finché la sessione non termina.

La terminazione della sessione può avvenire per 3 motivi:

- L'utente decide di terminare la sessione dal mobile premendo il pulsante END CONNECTION sulla app (che provoca l'invio di un messaggio che segnala la fine della sessione (*end*) all'arduino).
- L'utente preme il pulsante fisico BTEXIT sulla porta.
- Il PIR non rileva alcuna presenza nell'ambiente per un tempo superiore a SESS_TIMEOUT secondi.

Al termine della sessione l'arduino torna nello stato WAIT_HELLO in attesa di una nuova richiesta di accesso.

Negli stati ON, ASK_LOGIN, WAIT_LOGIN, LOGIN_SUCCESSFUL se l'arduino riceve il messaggio "*qm*" dal mobile, ad indicare che l'app mobile è stata chiusa, ritorna in stato WAIT_HELLO. (In molti casi questo non avviene perché in caso di scadenza del timeout, l'arduino ritorna in stato WAIT_HELLO automaticamente).

LedValueUpdateTask

All'avvio del sistema questo task transita subito in stato ON. In questo stato se il valore del led è stato cambiato dall'applicazione mobile l'arduino invia un messaggio col valore del led

aggiornato al gateway. Inoltre il task invia il valore corrente del led al mobile se questo si è appena connesso.

TempUpdateTask

All'avvio del sistema il task transita subito nello stato ON. In questo stato ogni volta che il task viene eseguito (quindi ogni 2 sec. che è il periodo di scheduling del task), l'arduino legge il valore corrente della temperatura dal sensore, lo invia al gateway e anche al mobile se la sessione è iniziata.

GATEWAY (RASPBERRY PI)

Il ruolo del gateway è svolto da un raspberry pi. In esso è in esecuzione un processo node.js che da una parte funge da server fornendo il servizio internet da cui è possibile ispezionare il log degli accessi, il valore corrente del led e della temperatura, dall'altra si occupa dell'autenticazione degli utenti (richiestagli dall'arduino) interrogando un database MySQL presente al suo interno.

Essendo un processo node, l'applicazione è basata su un event loop. Quando riceve un messaggio dall'arduino (evento da seriale) ne fa il parsing ed esegue l'azione appropriata eseguendo uno switch sul codice del messaggio.

La comunicazione col frontend avviene utilizzando il protocollo WebSocket (per mezzo della libreria *socket.io*[1] per node) permettendo quindi di avere una comunicazione event-driven full-duplex real-time, in modo tale da aggiornare il frontend senza che esso debba costantemente richiedere aggiornamenti al server.

La gestione dei led LINSIDE e LFAILED_ACCESS avviene mediante la liberia onoff[2] per node.js che consente l'accesso ai GPIO.

Per far sì che l'applicazione node possa essere configurata in base alle proprie esigenze si è deciso di utilizzare la libreria *yargs*[3] che permette di creare un'applicazione interattiva effettuando il parsing degli argomenti da riga di comando (o eventualmente leggendo la configurazione da un file json.

Di seguito si espone la lista dei possibili argomenti (ottenibile anche eseguendo il comando *node server.js --help*):

ARGOMENTO	FORMA ABBREVIATA	TIPO	OBBLIGATORIO	DESCRIZIONE
--help		boolean	no	Visualizza l'help
--version		boolean	no	Mostra la versione dell'applicazione

1 https://socket.io
2 https://github.com/fivdi/onoff
3 http://yargs.js.org e sorgenti disponibili all'indirizzo: https://github.com/yargs/yargs

--config			no	Permette di definire un file json di configurazione (i.e. node server.js --config <path_to_your_config_file>). ATTENZIONE: il file passato al parametro –config DEVE ASSOLUTAMENTE avere estensione .json
--serverPort	--svp	Number (integer between 0 excluded and 65536 excluded)	sì	Porta su cui deve essere in ascolto il web server
--serialPort	--srp	string	sì	Porta seriale a cui è collegato l'arduino
--baudRate	--br	number (integer)	sì	Baud rate da utilizzare nella comunicazione seriale con l'arduino
--ledInsidePin	--lin	number (integer)	sì	Pin a cui è collegato il led LINSIDE
--ledAccessFailedPin	--lfail	number	sì	Pin a cui è collegato il led LACCESS_FAILED
--databaseHostName	--dbhost	string	sì	Hostname del server in cui si trova il database MySQL
--databaseUserName	--dbuser	string	sì	Username utilizzato per connettersi al database
--databasePassword	--dbpass	string	sì	Password utilizzata per connettersi al database
--databaseName	--dbname	string	sì	Nome del database a cui connettersi

L'applicazione è configurata per cercare un file con nome .smartdoorrc o .smartdoorrc.json a partire dalla directory in cui si trova lo script server.js che rappresenta l'applicazione node da eseguire.

Per gestire i led è stato ritenuto utile definire una classe Led che espone tre metodi:

- switchOn(): accende il led
- switchOff(): spegne il led
- blink(timeOn): esegue un lampeggio del led della durata specificata dal parametro timeOn

Della classe sono presenti 2 implementazioni: quella nella directory src/mock che al posto di eseguire le operazioni sui gpio stampa su console e quella nella directory src/real che è la reale implementazione che opera sui gpio. Questo è stato fatto per consentire di sviluppare l'applicazione anche senza dover montare fisicamente il circuito.

Inoltre è presente la classe AccessType (in src/AccessType.js) che rappresenta una enum contenente una descrizione del tipo di accesso utile per la registrazione del log su db e visualizzazione nel frontend.

Il modulo db.js (in src) fornisce una funzione connect che permette di creare la connessione mysql esplicitando hostname, username, password e databaseName.

Dato che il tipo enum non è presente di default in javascript, abbiamo utilizzato la libreria *enumify*[4] che è scritta in ES6. Purtroppo, node non supporta ancora ufficialmente tutte le caratteristiche di ES6 per cui si è reso necessario eseguire un transpiling via Babel dei sorgenti in ES5 che è pienamente supportato da node.

La cartella src contiene i sorgenti originali (pre-transpiling) mentre la cartella dist contiene il codice post-transpiling, perciò per poter eseguire correttamente l'applicazione occorre eseguire lo script *server.js* presente nella cartella *dist* non quello presente nella cartella *src*.

Il frontend è composto da una pagina html (contenuta nella cartella *public*) in cui sono presenti 3 elementi:

- Un termometro: realizzato utlizzando la libreria *canvas-gauges*[5]. All'aggiornamento del valore del termometro (*gauge.value*) in automatico il termometro si ri-renderizza.
- Una semplice section con uno span contenente l'intensità attuale del led.
- Una table contenente gli access logs della giornata realizzata utilizzando il plugin jQuery *DataTables*[6] che consente l'ordinamento in base a diverse colonne ed è dotata di una funzione di ricerca. Inoltre il plugin esegue la paginazione automatica dei risultati in base al numero di righe selezionato dall'utente.

Le colonne della tabella sono 3:

- Date: data e ora dell'accesso
- Username: username dell'utente che ha tentato l'accesso
- Description: Descrizione del tipo di accesso (Access succeeded, Access failed, Access failed. PIR hasn't detected anything within the timeout).

Inoltre è stato scelto di visualizzare una notifica toast (nella pagina, non push) a seguito di un tentativo di accesso per rendere subito evidente all'utente il tentativo di accesso riuscito o fallito senza che questi debba guardare la tabella dei log per accorgersene. Per fare ciò abbiamo utilizzato la libreria *toastr*[7].

---

4 https://github.com/rauschma/enumify

5 https://canvas-gauges.com o sorgenti disponibili qui: https://github.com/Mikhus/canvas-gauges

6 https://datatables.net

7 http://codeseven.github.io/toastr/ e sorgenti all'indirizzo: https://github.com/CodeSeven/toastr

Viene inoltre visualizzata una notifica toast se la socket lato client non riesce a connettersi a quella lato server (ad es. perché la socket lato server è down).

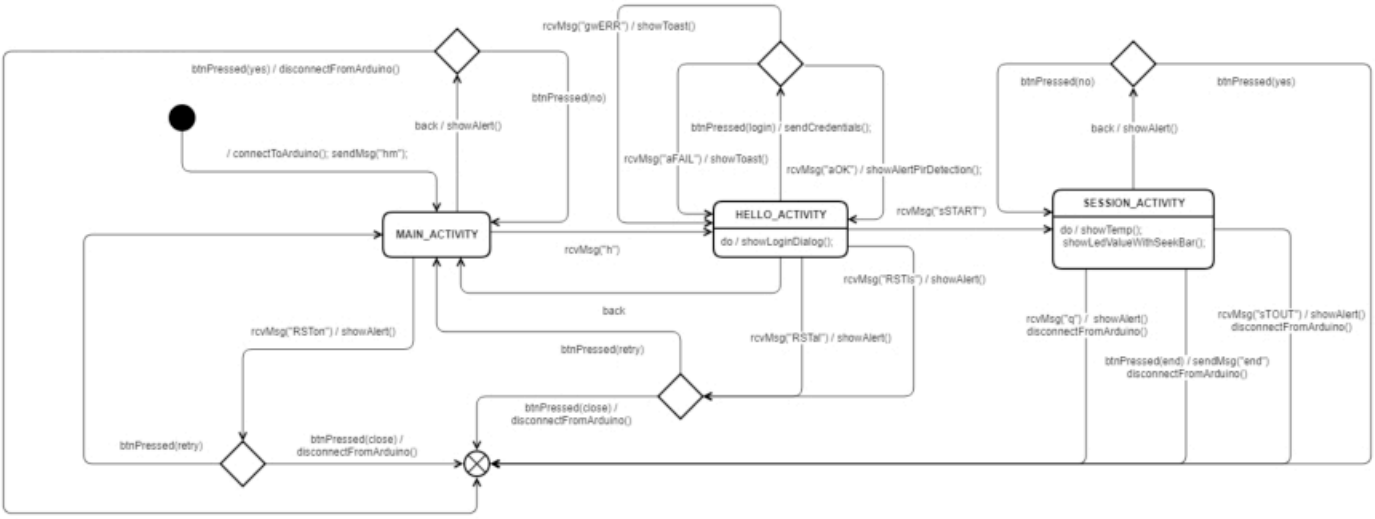
Il codice javascript necessario per il collegamento via WebSocket al backend, per la reazione agli eventi provenienti dalla socket e per l'istanziazione dei vari plugin è stato inserito direttamente nel tag script in fondo al file *index.html*.

PICCOLA GUIDA INSTALLAZIONE E RUNNING

Dopo aver scaricato lo zip o clonato la repo seguire questi passi:

1. Recarsi nella cartella contenente il file *package.json* dell'applicazione ed eseguire il comando *npm install* per installare i moduli node (non in tracking nella repo).
2. Creare il database utilizzando il file *smart_door_ddl.sql* presente nella cartella src subfolder della root (Consegna-04/src/smart_door_ddl.sql).
3. Editare il file *.smartdoorrc* presente nella cartella *src* impostando i parametri corretti per la propria installazione, in particolare prestare attenzione nel settare la giusta porta seriale.
4. Eseguire il comando *sudo npm start* per procedere al transpiling ed avviare l'applicazione
5. Da ora in poi se l'applicazione dovrà essere riavviata sarà possibile farlo invocando il comando *sudo node server.js* dalla cartella dist (o comunque dalla cartella dove sarà posizionato il codice transpilato).

ANDROID



Il sottosistema MOBILE è costituito da un dispositivo Android in grado di comunicare via Bluetooth con l'Arduino. Il suo scopo principale è quello di interagire con la DOOR, controllando la temperatura e impostando il livello di luminosità del LED collegato al microcontrollore.

L'aspetto cruciale del sottosistema è rappresentato dalla gestione della comunicazione, la quale deve essere mantenuta tramite l'utilizzo di un AsyncTask che avvia un Thread in background in grado di inviare e ricevere messaggi con il microcontrollore che sta dall'altro lato della connessione. In quanto il sensore Bluetooth dell'Arduino è una porta seriale, l'UUID da utilizzare è quello predefinito: 00001101-0000-1000-8000-00805F9B34FB.

Il problema principale è stato l'effettuare modifiche alla View dell'applicazione da Thread non principali (in questo caso il ConnectionManager): l'utilizzo di Handler è stato la soluzione a questa complicazione, permettendo così di svolgere qualsiasi azione necessaria a seconda dei diversi messaggi ricevuti dall'Arduino.