



Università del Piemonte Orientale

Dipartimento di Scienze e Innovazione Tecnologica

Corso di Studi in Informatica

Relazione per la prova finale

**Implementazione di interfacce utente
vocali, touch e di configurazione per
MagicMirror**

Tutore interno:
Dott. Marco Guazzone

Candidato:
Riccardo Berto

Anno Accademico 2016/17

Indice

Abstract	3
Introduzione	4
1 Scopi e Problemi Affrontati	6
1.1 Applicazioni del MagicMirror	6
1.2 Interfaccia di controllo del MagicMirror	6
2 Tecnologie Implicate	9
2.1 Prefazione	9
2.2 Hardware	9
2.2.1 RasperryPi	9
2.2.2 Periferiche	10
2.3 Software	10
2.3.1 Raspbian	10
2.3.2 Electron	11
2.3.3 OpenCV	11
2.3.4 Google Speech API	12
2.3.5 Node.JS	12
2.3.6 Express	13
2.3.7 Mustache	13
2.3.8 MySQL	13
2.3.9 JavaScript	14
2.3.10 Python	14
2.3.11 GitLab	15
3 Struttura del MagicMirror	16
3.1 Il MM dall'alto livello	17
3.2 Avvio ed Esecuzione	18
3.2.1 Il file di Configurazione	19
3.3 Implementazione di un'applicazione	20

3.4	Messaggistica del MM	21
4	Applicazioni per il MM	22
4.1	Modulo per comandi vocali	22
4.1.1	Comunicazione con l'API di Google	23
4.1.2	Difficoltà incontrate	25
4.2	Modulo con Touch Board	25
5	Interfaccia web di configurazione	28
5.1	Struttura dell'interfaccia Web di configurazione	28
5.2	Individuazione e visualizzazione dei moduli nell'interfaccia . . .	29
	Conclusione	32
	Bibliografia	33

Abstract

La tesi affronta alcune problematiche presenti al giorno d'oggi in molti progetti informatici, dovute al continuo sviluppo di metodologie di interfacciamento sempre più dirette e complesse. Il software nel quale sono state affrontate queste problematiche è il MagicMirror, un progetto di domotica.

In particolare il problema è stato implementare metodi alternativi di interfacciamento, rispetto a mouse e tastiera, e metodi per configurare il software da remoto da parte di utenti non esperti.

Per affrontare il primo problema sono state implementate due interfacce utente: una vocale, per permettere l'impartizione di comandi tramite specifiche frasi, e una touch, per eseguire specifiche operazioni tramite movimenti delle dita su un'area di un dispositivo. Per il secondo problema è stata sviluppata un'interfaccia web di configurazione remota che permettesse di modificare facilmente le impostazioni del software principale.

Per le interfacce utente di controllo sono stati utilizzati servizi esterni all'ambiente in cui è stato sviluppato il progetto. Per l'interfaccia web di configurazione, invece, sono state usate tecnologie per la maggior parte già implementate nel software principale.

Introduzione

Nello stage che ho svolto presso Lab121 ho esteso il [19]MagicMirror (abbreviato in *MM*), un software di domotica sviluppato da Michael Teeuw, che gira sul calcolatore portatile RaspberryPi[3], sul quale ho svolto due principali attività:

- Lo sviluppo di applicazioni (dette anche *Moduli*) mirate al controllo del MagicMirror tramite l'impartizione di comandi vocali o input trasmessi dal movimento delle dita su una periferica touchpad.

La scelta dello sviluppo di queste applicazioni è stata presa con l'obiettivo di dotare il MagicMirror di un'interfaccia uomo-macchina avanzata. Infatti, negli ultimi decenni, con lo sviluppo di nuove tecnologie, sono stati sviluppati ed evoluti metodi non tradizionali per il controllo di dispositivi.

Per esempio si possono osservare la maggior parte degli Smartphone moderni che permettono lo sblocco dello schermo con il riconoscimento del volto oppure l'interazione con esso tramite sintesi vocale, invece dell'utilizzo di tastiere analogiche o digitali. Inoltre il progresso tecnologico di calcolatori e periferiche sempre più piccoli ed economici, ha permesso la nascita di una moltitudine di progetti casalinghi in ambito interazione uomo-macchina, contribuendo alla crescita dei software di questo tipo, oggi reperibili in quantità anche open source.

- Lo sviluppo di un'interfaccia grafica web, la quale, tramite un pannello di amministrazione, permette di modificare il file di configurazione del MM e delle applicazioni che si appoggiano su di esso, evitando così di dover accedere a file potenzialmente critici ed eliminarne parti essenziali ad opera di utenti non esperti. Infatti, oltre alle tecnologie, un altro aspetto che si è evoluto negli ultimi anni nel campo delle Applicazioni Web, è l'implementazione di interfacce grafiche di amministrazione ad

alto livello sempre più intuitive e facili da comprendere. Capita spesso che, in assenza di queste interfacce più evolute, un utente inesperto sia costretto ad utilizzare interfacce a basso livello (come strumenti di linea di comando o file di configurazione), con il rischio di causare potenziali danni.

Nei prossimi capitoli:

- Nel capitolo 1 verranno trattati scopi e problemi affrontati nello stage
- Nel capitolo 2 verranno elencate e spiegate le tecnologie che sono state utilizzate durante lo stage
- Nel capitolo 3 verrà spiegata la struttura del MagicMirror
- Nel capitolo 4 verranno spiegati i moduli creati per il MagicMirror
- Nel capitolo 5 verrà esposto il funzionamento dell'interfaccia web di amministrazione

Capitolo 1

Scopi e Problemi Affrontati

In questo capitolo verranno esposti gli scopi e i problemi incontrati durante l'attività di stage divisi in due argomenti. In particolare si tratta lo sviluppo delle applicazioni per il MM (sezione 1.1) e lo sviluppo dell'interfaccia web di amministrazione (sezione 1.2).

1.1 Applicazioni del MagicMirror

Il MagicMirror è un software che espone API, le quali permettono l'integrazione di applicazioni sviluppate da terzi e forniscono funzionalità per la comunicazione, gestione ed organizzazione tra le applicazioni. Lo scopo nello sviluppo delle applicazioni, durante lo stage, è stato di creare programmi che permettessero l'interazione tra il software principale e l'essere umano. Nello specifico si è voluto implementare la possibilità, da parte di un utente, di controllare le applicazioni attive sul MagicMirror tramite l'impartizione di comandi vocali o movimenti delle dita su una scheda Touch Board. In questa parte si è dovuto affrontare il problema di far comunicare i diversi dispositivi hardware con il calcolatore principale, e di programmare le applicazioni con linguaggi diversi, a seconda della compatibilità di un linguaggio e delle librerie a disposizione di una determinata periferica.

1.2 Interfaccia di controllo del MagicMirror

Il MM possiede all'interno delle sue cartelle un file di configurazione, il quale permette di configurare sia MM che le applicazioni caricate su quest'ultimo.

Prima dello stage per modificare la configurazione, bisognava accedere direttamente al file di configurazione appena presentato e modificarlo. Questo approccio, oltre ad essere scomodo, può anche causare errori, dal momento che è necessaria una sintassi corretta all'interno del file.

Preso atto di questo, nella seconda parte dello stage si è svolta la progettazione e lo sviluppo di un'interfaccia Web di configurazione remota il cui scopo è di permettere ad un utente autenticato di gestire le configurazioni del MM, e dei moduli implementati su di esso, da una pagina web anziché accedendo alla macchina fisica su cui gira l'applicazione. L'interfaccia permette anche di disattivare (o attivare) le applicazioni presenti nel MM, la cui procedura, prima, richiedeva di aggiungere manualmente un pezzo di codice nella configurazione del software principale. Lo scambio dei messaggi tra il MM e la pagina web avviene in formato JSON, lo stesso formato usato per il file di configurazione.

Un altro problema riscontrato in questa parte è stata la difficoltà dell'utilizzo di Javascript perchè è un linguaggio orientato ad eventi, ovvero alcune funzioni vengono eseguite senza attendere il risultato della precedente. Il linguaggio, per poter sincronizzare le funzioni, mette a disposizione le callback: esse sono funzioni passate come parametro ad un'altra funzione, cosicché quest'ultima può svolgere un determinato compito (quello svolto nella callback) che non è noto al momento della scrittura del codice. Senza buone pratiche di programmazione si rischia di incontrare un fenomeno chiamato "CallbackHell", ovvero callback che chiamano a loro volta altre callback creando funzioni annidate più volte, come mostrato in figura 1.1.


```
CallEndpoint("api/getidbyusername/hotcakes", function(result) {  
    CallEndpoint("api/getfollowersbyid/" + result.userID, function(result) {  
        CallEndpoint("api/someothercall/" + result.followers, function(result) {  
            CallEndpoint("api/someothercall/" + result, function(result) {  
                CallEndpoint("api/someothercall/" + result, function(result) {  
  
                });  
            });  
        });  
    });  
});
```

Figura 1.1: Esempio di CallbackHell

Nella figura il codice esegue delle query attraverso delle API esposte da un servizio. La funzione *CallEndpoint()* chiama la funzione dell'API passandogli un percorso, il quale restituisce il risultato di una query. Per poter procedere a quella successiva è richiesto il risultato della precedente e per farlo è necessario l'utilizzo di un'altra callback. Questa dipendenza si ripresenta anche nelle query successive creando un annidamento sempre maggiore conosciuto come CallbackHell.

Capitolo 2

Tecnologie Implicate

In questo capitolo verranno discusse le tecnologie che sono state studiate ed utilizzate durante le attività di stage. In particolare il capitolo è diviso tra tecnologie hardware e tecnologie software.

2.1 Prefazione

Lo sviluppo del progetto di stage è stato svolto nell'ambiente Raspbian[1], una distribuzione Debian[2] che gira sul dispositivo RaspberryPi[3]. Inoltre sono state adottate diverse tecnologie nel campo del riconoscimento vocale e di immagini (OpenCV [4] e Google Speech API [5]), nello sviluppo di applicazioni web (Electron [6], Node.JS [14], Express [8], Mustache [10]) e della gestione dati (MySQL [9]). Inoltre sono stati adottati diversi linguaggi di programmazione (JavaScript [11], Python [12]). Infine, per la gestione distribuita di progetti software e di versionamento del codice, è stata usata la piattaforma GitLab [13].

2.2 Hardware

2.2.1 RaspberryPi

RaspberryPi è un calcolatore elettronico, montato su una singola scheda elettronica, caratterizzato dal basso costo, dal consumo energetico ridotto e, per le sue dimensioni ridotte, dalla facile portabilità. Rilasciato per la prima volta nel 2012 è diventato un prodotto utilizzato per una moltitudine di progetti sia aziendali che casalinghi. Il modello usato durante lo stage è RaspberryPi 3 model B e monta:

- CPU con architettura Advanced RISC Machine (ARM)

- 1 porta HDMI
- 1 porta LAN
- 1 uscita Aux
- 4 porte USB
- 40 pin General Purpose Input/Output(GPIO)
- 1 scheda di rete wireless
- Alimentazione microUSB 5V
- un bus camera serial interface(CSI), ovvero una porta per telecamera con Flexible flat cable(FFC)
- ingresso per microSD

Il sistema operativo per Raspberry deve essere installato su una microSD opportunamente formattata e configurata con un Master Boot Record (MBR).

2.2.2 Periferiche

Nella creazione delle applicazioni per il Magic Mirror sono state usate diverse periferiche, tra cui un microfono USB, per catturare la voce in input e un componente Skydriver Touch Board (94mm x 122mm) di Piromoni collegabile tramite i 40 pin GPIO del calcolatore principale, per catturare input fisici tramite il movimento delle dita sulla Touch Board.

2.3 Software

2.3.1 Raspbian

Raspbian è una distribuzione del sistema operativo Linux derivata da Debian, completamente libera, ottimizzata per Raspberry. Fu sviluppata da Mike Thompson e Peter Green come progetto non affiliato alla compagnia RaspberryPi Foundation, per tenere in considerazione la limitata capacità di calcolo dei processori ARM. La prima versione venne rilasciata nel 2012.

2.3.2 Electron

Electron è un framework open source rilasciato per la prima volta nel 2013, ma la prima versione stabile è uscita nel giugno 2017. È disponibile per i sistemi operativi Windows, MacOS e Linux ed è scritto in C++ e Javascript. Il framework permette la creazione di applicazioni multiplatforma utilizzando tecnologie già esistenti per lo sviluppo del lato client e del lato server (Javascript, Node.JS, V8 [15]). All'avvio di Electron viene inizializzata una pagina con Chromium [17](un web browser installato insieme all'applicazione) nel quale viene mostrata una pagina web, e un server in Node.JS. Un'applicazione sviluppata con Electron ha bisogno di 3 componenti principali:

- Il package.json, un file JSON, che deve contenere almeno il nome dell'applicazione, la versione dell'applicazione creata, la descrizione di quest'ultima e il nome del file principale dell'applicazione (necessaria per l'avvio), come mostrato nella seguente immagine:

```
1 {  
2   "name": "magiemirror",  
3   "version": "2.1.1",  
4   "description": "The open source smart platform",  
5   "main": "js/electron.js"  
6 }
```

Name è il campo che assegna il nome dell'applicazione, *version* è il campo che assegna la versione dell'applicazione, *description* è una stringa che descrive il programma, *main* è il campo che punta al file Javascript da eseguire all'avvio.

- Un file HTML che contiene il template della pagina generata dall'applicazione
- Un file JavaScript che contiene il codice di esecuzione dell'applicazione come ad esempio la creazione di una finestra o la visualizzazione di una pagina.

2.3.3 OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria software sviluppata intorno al 2000 utilizzata nell'ambito della visione artificiale per l'acquisizione e il riconoscimento di immagini da parte di una macchina per mezzo di input digitali, ottenuti tramite telecamera o fotocamera.

La libreria è disponibile per i linguaggi C++(linguaggio in cui è scritta), C,

Python e Java e per diversi sistemi operativi, compresi quelli specifici per i dispositivi mobili.

OpenCV prende in input un'immagine o uno stream (come un video o una serie di immagini) e, utilizzando specifici algoritmi di Machine Learning, è in grado di individuare e riconoscere oggetti specifici.

2.3.4 Google Speech API

Negli ultimi anni Google ha ampliato sempre di più il suo catalogo per quanto riguarda i servizi cloud e web API. Tra questi si può individuare anche Google Speech API, il quale è un servizio che, ricevendo in input un file o uno stream audio, ottenuto per mezzo di un'acquisizione da un dispositivo di audio input, traduce il parlato in testo scritto tramite algoritmi avanzati di riconoscimento della voce.

L'API supporta oltre 110 lingue e si possono usare su diverse piattaforme dato che le librerie sono disponibili nei linguaggi C#, GO, Java, Node.JS, PHP, Python e Ruby. Inoltre Google Speech API dispone di alcune varianti:

- Una con interfaccia REpresentational State Transfer(REST), che comunica per mezzo di URI
- Una con gRPC, un sistema di chiamata di procedura remota

2.3.5 Node.JS

Node.JS già distribuito con Electron, ed è stato usato per la creazione del server per la gestione dell'interfaccia web per il MM. Node.JS è una piattaforma open source che permette l'esecuzione del linguaggio Javascript, sfruttando il motore JavaScript V8 sviluppato da Google, anche per il lato server, ovvero la parte del sistema che è adibita alla manipolazione e all'elaborazione dei dati in modo completamente trasparente all'utente.

Node.JS esegue delle operazioni al verificarsi di uno specifico evento, che può essere un accesso ad una porta del server o la richiesta di una pagina.

Per gestire i pacchetti di questo framework viene utilizzato NPM[16], uno strumento che permette di scaricare ed installare librerie private o pubbliche salvate su un database.

2.3.6 Express

Express è un framework per Node.JS che permette di creare applicazioni Web in JavaScript, offrendo strumenti e pacchetti che implementano più facilmente tutte le funzionalità offerte da Node.JS. Il software viene usato per creare e gestire il back-end di un server, che è composto da 3 entità importanti:

- il Controller, che definisce le funzioni associate ad un determinato modello.
- il Routing, utilizzato per determinare come il server debba rispondere ad un determinato metodo di richiesta, ricevuta sottoforma di URI, inoltrando la stessa alla funzione del controller del rispettivo modello a cui fa riferimento.
- i Modelli, creati per ogni entità-oggetto che esiste all'interno del server, ad ognuno dei quali viene associato un controller. Inoltre, tramite i modelli si accede al database per estrarre i dati e spedirli al controller per la manipolazione, ricevendoli successivamente modificati e salvandoli, se necessario.

2.3.7 Mustache

Mustache è un sistema di template, ovvero un sistema che prendendo in input dati e web template genera automaticamente pagine web, permettendo così di riutilizzare elementi statici dei template. Dal momento che Mustache è disponibile in diversi linguaggi tra cui Javascript è stato usato per l'interfaccia web di configurazione del MagicMirror. Mustache è un sistema logic-less perchè è privo di istruzioni per il controllo del flusso (come ad esempio l'"if" e l'"else"), quindi tutti i controlli di questo tipo sono data-driven.

2.3.8 MySQL

MySQL è stato usato durante lo stage come database per l'archiviazione di account di utenti accreditati all'autenticazione per accedere all'interfaccia web di configurazione del MM.

MySQL è uno tra i più famosi database open source sviluppato da Oracle,

più precisamente è sistema per la gestione di basi di dati basato sul modello relazionale. MySQL è composto da una semplice riga di comando e un server web, ma sono implementati anche programmi per l'amministrazione del database come, ad esempio, il famoso phpMyAdmin. La prima versione fu rilasciata nel maggio del 1995 sviluppata da Oracle e di proprietà di MySQL AB, distribuito sia con licenza commerciale sia con licenza libera.

2.3.9 JavaScript

Javascript è un linguaggio di programmazione ad alto livello orientato agli oggetti e ad eventi, che è supportato dalla maggior parte dei browser per lo scripting delle pagine web, e che supporta la programmazione procedurale. Inizialmente usato per il lato client ha subito un'evoluzione che lo ha portato ad essere utilizzato per lo sviluppo di back-end e web app.

ECMAJavascript(ES) è lo standard di Javascript che negli ultimi anni ha sviluppato ed evoluto il linguaggio in diverse versioni. Uno degli aspetti su cui il processo di standardizzazione si focalizzato è la semplificazione della gestione degli eventi e delle funzioni asincrone (cioè di quelle che riconsegnano il controllo al chiamante prima della loro terminazione). Le pratiche utilizzate per sincronizzare queste funzioni in ECMAJavascript 5 (ES5), ovvero la versione usata durante lo stage, sono l'utilizzo delle callback.

In ECMAJavascript 6 (ES6) è stato introdotto il concetto di Promises, che ha sostituito le funzioni di callback. Una Promise rappresenta un proxy per un valore non necessariamente noto quando la Promise è stata creata. Questa consente di associare degli handlers con il successo o il fallimento di un'azione asincrona, restituendo il valore prodotto, nel primo caso, o la motivazione nel secondo.

Sono tutt'ora in via di sviluppo nuove versioni di Javascript.

2.3.10 Python

Python è un linguaggio di programmazione open source, ad alto livello, con semantica dinamica, orientato ad oggetti, usato per lo sviluppo di applicazioni e scripting.

Questo linguaggio è stato particolarmente utile per l'interfacciamento con la telecamera e la Touch Board, essendo a disposizione librerie apposite per gestirle.

Come molti altri linguaggi, supporta l'implementazione di pacchetti esterni che possono essere salvati in una repository pubblica o su piattaforme esterne. Nel primo caso Python utilizza un suo gestore di pacchetti: Pip Installs Packages o Pip Installs Python (pip, acronimo ricorsivo). Nel secondo caso i pacchetti possono essere inseriti senza dover utilizzare il gestore di pacchetti appena introdotto.

2.3.11 GitLab

GitLab è una piattaforma web che implementa le funzionalità offerte dal software Git e altri servizi tra i quali la possibilità di creare wiki e un servizio di issue tracking, utile per tenere traccia di eventuali richieste o problemi. La comodità di questa piattaforma sta nel poter creare dei Commit, ovvero ad ogni modifica del codice si può salvarne lo stato assegnandoli un'etichetta in locale, per poi spedirlo al repository in remoto del progetto. In ogni momento si può tornare ad una versione vecchia tramite gli strumenti di versionamento offerti dalla piattaforma web.

Capitolo 3

Struttura del MagicMirror

Il MagicMirror (detto anche *MM*) è un progetto ideato e sviluppato da Michael Teeuw, successivamente esteso nelle sue funzionalità da una moltitudine di utenti su GitHub. La prima versione è stata scritta completamente in Python, mentre nella versione successiva è stato utilizzato Electron, che ha comportato una variazione di linguaggio, a favore di Javascript. In questo modo è stato possibile implementare un'interfaccia esteticamente più gradevole e API più intuitive. L'idea dell'autore è nata rifacendosi allo specchio magico della rinomata fiaba scritta dai fratelli Grimm: Biancaneve e i Sette Nani.

Il software viene mostrato attraverso un comune monitor, trasmettendo immagini poste su uno sfondo completamente nero. Applicando sopra una semplice pellicola a specchio (la quale da un lato permette di specchiarsi e dall'altro di vedere attraverso) si crea un effetto particolare per cui una persona riesce a specchiarsi e allo stesso tempo riesce a vedere le scritte o le immagini trasmesse dal monitor, come mostrato in figura 3.1. In quest'ultima si può vedere il MM che, oltre alla sua funzione base di specchio, mostra alcune informazioni come l'ora, la temperatura, il meteo e un messaggio.

In questo capitolo verrà spiegata la struttura generale del MM (sezione 3.1), le classi principali che vengono caricate all'avvio (sezione 3.2), la struttura del file di configurazione (sezione 3.2.1), come viene implementata un'applicazione (sezione 3.3) e il funzionamento del sistema di messaggistica (sezione 3.4).



Figura 3.1: MagicMirror by Michael Teeuw

3.1 Il MM dall'alto livello

Il MM è una applicazione composta da diversi componenti software con cui è possibile comunicare attraverso le API messe a disposizione da MM. Si possono individuare alcuni elementi principali rappresentati in figura 3.2:

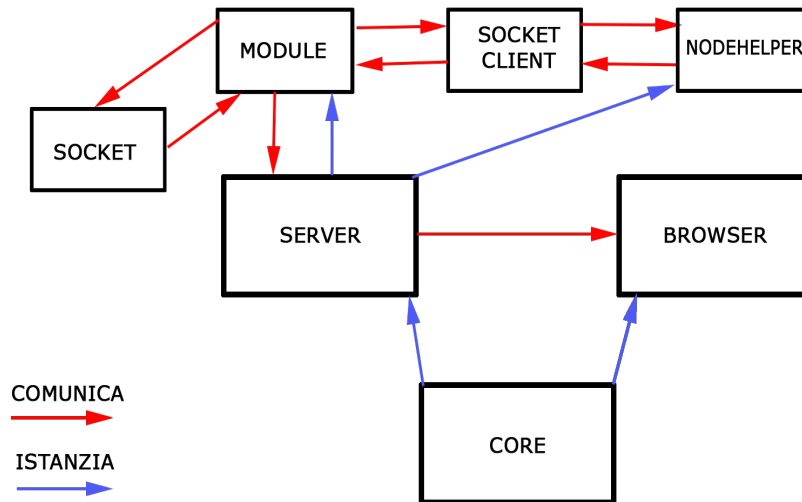


Figura 3.2: Struttura MagicMirror

- Il core, il cuore del MM dove gira il codice principale.
- Il server, che gira in locale, usato per restituire la pagina HTML, con i dati, al browser di Electron e comunicare con le API
- I Moduli, entità che espone le API per l'implementazione di applicazioni
- I NodeHelper, entità che espone le API per l'implementazione di funzioni di supporto al Modulo
- Il Socket, entità che fornisce un canale di comunicazione tra i Moduli
- Il SocketClient, entità che fornisce un canale di comunicazione tra un Modulo e il proprio NodeHelper.

3.2 Avvio ed Esecuzione

Il MM viene avviato tramite linea di comando, la quale esegue il codice di un file Javascript, il cui percorso è indicato nel documento *package.json* di

Electron, descritto nella sezione 2.3.2. Il file contiene il codice di Electron, che si occupa dell'avvio di MM, caricandone tutte le strutture, e della creazione di una nuova finestra browser, il cui compito è di mostrare l'output dei moduli. La pagina costruita e l'output dei moduli sono un insieme di Document Object Model (DOM), ovvero oggetti HTML che compongono una pagina web, i quali vengono mostrati solo dopo l'avvio di tutti i moduli attivi. Le strutture principali caricate dal MM sono le seguenti:

- gli end-point delle API per le applicazioni, che sono le interfacce usate per leggere e caricare le applicazioni inserite nel MM.
- gli end-point delle API per la gestione dei NodeHelper. Sono strutture opzionali usate per collegamenti esterni al MM (per esempio, con API di un servizio cloud). Ogni applicazione ha il proprio NodeHelper con cui può comunicare tramite messaggi in modo simile a come comunicano le applicazioni tra di loro.
- un "Socket" e un "Socket-Client", entità principali che definiscono le funzioni per lo scambio dei messaggi tra le applicazioni e i rispettivi NodeHelper.
- un Logger, implementato per tenere i log dell'applicazione e degli eventuali errori. Usato principalmente per il debugging.
- un file di configurazione, nel quale sono specificati il nome e le coordinate per la posizione dei vari moduli all'interno della pagina.

Inoltre viene inizializzato un server, il cui compito è quello di trasmettere la pagina HTML generata, con gli output delle varie applicazioni, al browser precedentemente avviato.

3.2.1 Il file di Configurazione

Come discusso nella sezione 1.2, il MM carica un file di configurazione, che è composto dai seguenti campi:

- la porta del server
- una whitelist, ovvero un IP oppure un range di IP che possono collegarsi allo specchio

- la lingua principale del sistema
- il formato dell'ora (12h o 24h)
- l'unità di misura (ad esempio pixel), usata per gestire la distanza degli oggetti HTML
- una lista di moduli (in formato JSON) da caricare durante l'avvio del MM con la relativa posizione nella pagina. Per ogni applicazione deve necessariamente comparire il nome e la posizione; opzionalmente si possono inserire un campo *header* e un campo *config* specifico per l'applicazione, come mostrato nell'esempio in figura 3.1.

```

1 {
2     "module": 'nome dell'applicazione',
3     "position": 'la posizione dell'applicazione
4                 all'interno dello specchio',
5     "header": 'una stringa che viene stampata
6                 sopra all'applicazione',
7     "config": { opzioni varie in formato JSON }
8 }
```

Listing 3.1: Campi di configurazione di un modulo nel MM

La lingua, il formato dell'ora e l'unità di misura usata sono parametri messi a disposizione dal sistema per la creazione di un'applicazione (ad esempio, il display di un orologio).

3.3 Implementazione di un'applicazione

La modifica del file di configurazione del MM appena descritta serve per "notificare" la presenza delle applicazioni a quest'ultimo, ma perchè possano funzionare è necessario che rispettino alcune specifiche regole. Per inserire il codice dell'applicazione all'interno dello specchio è necessario creare una cartella con un nome identificativo dell'applicazione nella directory *Modules*. Dentro la cartella appena creata devono essere inseriti:

- 1 file Javascript (JS), ovvero il documento principale con lo stesso nome della cartella appena creata. Contiene il codice dell'applicazione, il quale conterrà a sua volta il codice per la creazione dei DOM
- 1 file Cascading Style Sheets (CSS), per modificare l'estetica del DOM della relativa applicazione (opzionale)

- 1 file `node_helper.js`, che è il NodeHelper associato alla specifica applicazione (opzionale)
- Altri file necessari all'applicazione (immagini, JSON, etc)

3.4 Messaggistica del MM

Nella sezione 3.1 è stato già accennato che il MM implementa un meccanismo di messaggistica sfruttando un sistema di socket integrato, utile per l'organizzazione e la moderazione delle applicazioni tramite l'utilizzo di funzioni messe a disposizione dall'API. Sono presenti due classi socket:

- `Socket`, classe che fornisce le funzioni per ricevere e mandare messaggi tra i moduli del MM. Con questo socket la spedizione del messaggio è in broadcast usando il metodo `sendNotification(notification, payload)`. Il primo parametro è una stringa che identifica il messaggio, il secondo parametro è opzionale e può essere usato per inserire il corpo del messaggio. La ricezione del messaggio viene gestita con il metodo `notificationReceived(notification, payload, sender)`, i primi due parametri sono uguali a quelli della funzione per spedire, il terzo parametro contiene il nome del modulo che ha mandato il messaggio
- `SocketClient`, classe che fornisce le funzioni per ricevere e mandare messaggi tra il modulo e il suo NodeHelper. Per spedire i messaggi viene usata la funzione `sendSocketNotification(notification, payload)`, mentre la ricezione viene gestita con la funzione `socketNotificationReceived(notification, payload)`. I campi di queste due funzioni sono gli stessi descritti nel punto precedente.

Capitolo 4

Applicazioni per il MM

In questo capitolo vengono spiegati i requisiti dei moduli sviluppati durante lo stage e la loro implementazione. In particolare verranno esposte le dipendenze per il modulo dei comandi vocali (sezione 4.1.2), le metodologie per accedere ai servizi di Google (sezione 4.1.2), come avviene la comunicazione tra i servizi Google e il programma (sezione 4.1.1) e l'implementazione del modulo con Touch Board (sezione 4.2).

4.1 Modulo per comandi vocali

La prima applicazione implementata, è stata il controllo del MM tramite comandi vocali. Nello specifico l'applicazione deve, tramite delle opportune frasi, gestire le altre applicazioni presenti nel MM, sfruttando funzioni offerte dallo stesso.

Nella figura 4.1 è rappresentata la struttura e il funzionamento del modulo. Il microfono cattura l'audio, che viene elaborato, all'interno del NodeHelper, dal componente "Elaborazione Audio". Successivamente, l'audio così elaborato, viene passato all'API Google Speech, la quale lo inoltra al Servizio Google Speech e si mette in attesa di una risposta. Al ritorno di questa, viene passata al modulo che ha il compito di validare il comando e di inoltrarlo in broadcast a tutti i moduli, se corretto. La validazione del comando è un semplice controllo di stringa tra il comando ricevuto dal NodeHelper e una lista di stringhe in un array che corrispondono alla lista di comandi. Alcuni comandi di esempio sono: *mostra ora*, per mostrare l'ora, *che giorno è?*, per mostrare il calendario, oppure *mostra -nome modulo-*, per visualizzare uno specifico modulo.

Per poter utilizzare l'API è stato necessario scaricare un software per l'elaborazione dell'audio, e, tra le diverse possibilità, è stato scelto il software

Sound eXchange (*SoX*).

Inoltre è richiesta un'autenticazione per poter usufruire dei servizi Google.

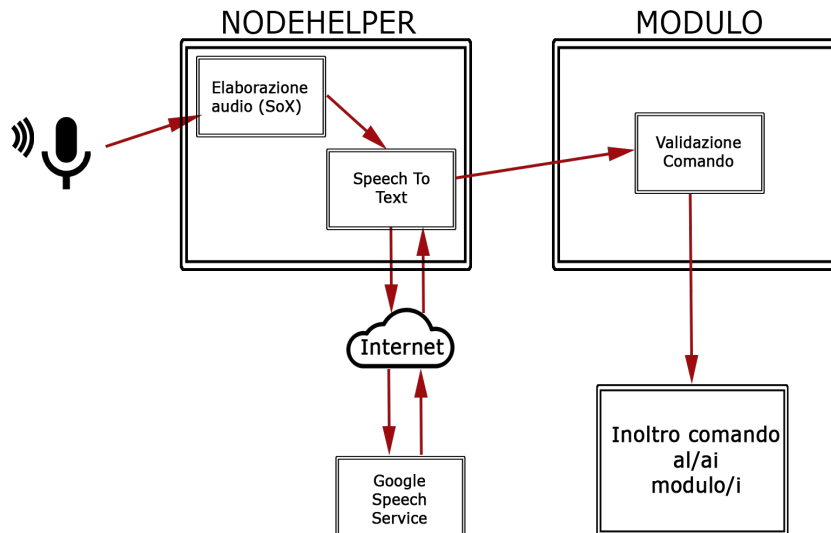


Figura 4.1: Struttura del modulo per comandi vocali

4.1.1 Comunicazione con l'API di Google

Il NodeHelper dell'applicazione si occupa di gestire lo streaming con l'API e di mandare i risultati (o gli errori) all'applicazione tramite i metodi di *SocketClient* messi a disposizione dal MM, esposte nella sezione 3.4. Il seguente codice viene usato per creare un canale streaming con l'API:

```
1      const recognizeStream = speech.streamingRecognize
      (request)
2      .on('error', sendSocketNotification("error"))
3      .on('data', (data) =>
4          if(Transcription: ${data.results[0].
              alternatives[0].transcript})
5              sendSocketNotification('limit_reached')
6          else
7              sendSocketNotification('response', data.
                  results[0])
```

Listing 4.1: Codice per l'inoltro dell'audio al Servizio Google

speech.streamingRecognize(request), richiama la funzione dell'API per aprire una connessione, dove *request* è il flusso audio. La funzione si mette successivamente in attesa di una risposta dall'API la quale può essere di due tipi:

- errore, nel caso ci sia stato un errore di connessione. In questo caso viene mandato al modulo un messaggio di errore
- dati di risposta, nel caso di risposta senza errori, ma che può essere divisa ulteriormente in altre due risposte. La prima sia ha nel caso in cui viene raggiunto il limite di parole tradotte (Google mette a disposizione un limite giornaliero per chi vuole usufruirne gratuitamente). In questo caso il modulo lo notificherebbe a video con un messaggio di errore. La seconda risposta contiene una stringa con la frase tradotta, che il modulo validerebbe come comando, e, in caso di risposta positiva, la inoltrerebbe ai moduli.

Per passare il flusso audio alla funzione appena descritta bisogna creare una *pipe*, ovvero un meccanismo di Comunicazione tra due processi tramite il quale l'output del primo processo è inoltrato come input al secondo processo. Nel seguente codice:

```
1      // Start recording and send the microphone input
      to the Speech API
2      record
3      .start({
4          sampleRateHertz: 1600,
5          threshold: 0,
6          verbose: false,
7          recordProgram: 'sox',
8          silence: '20.0'
9      })
10     .on('error', sendSocketNotification('error'))
11     .pipe(recognizeStream);
```

record è una funzione oggetto con ascolto di eventi che imposta tramite il metodo *.start* i settaggi dello streaming (per esempio la frequenza) e ne inizia la cattura. L'evento *.on('error')* serve per sollevare un'eccezione in caso di errore, che poi viene inoltrata al modulo. L'evento *.pipe(recognizeStream)* crea una pipe tra la funzione di registrazione e *recognizeStream* descritta nel codice precedente, passando il flusso audio direttamente alla funzione.

4.1.2 Difficoltà incontrate

Sound eXchange (SoX)

Come discusso nella sezione 4.1, per permettere all'audio di venire correttamente elaborato per lo streaming, è necessario utilizzare SoX. Affinchè il microfono si colleghi correttamente al programma è necessario impostare correttamente i valori delle variabili d'ambiente *AUDIODEV* e *AUDIODRIVER*. La prima variabile corrisponde al dispositivo audio al quale il programma deve fare riferimento, la seconda variabile al driver audio da utilizzare; di solito il predefinito è Advanced Linux Sound Architecture(ALSA).

Autenticazione Google API

Per poter usufruire delle API di Google è necessario fornire un'autenticazione a livello di sistema. Per poterlo fare bisogna ottenere delle credenziali di sicurezza per un account Google, attivabili tramite Google Cloud Platform Console. Le credenziali consistono in un username, l'email dell'account Google e una chiave di sicurezza unica, il tutto contenuto in un file JSON che può essere scaricato e salvato in locale. Per permettere al sistema di utilizzare l'API, occorre che il file JSON, con le credenziali salvate, sia raggiungibile all'interno del sistema e per rendere possibile ciò, bisogna creare una variabile d'ambiente con il nome *GOOGLE_APPLICATION_CREDENTIALS* nel quale viene salvato il percorso dove si trova il file.

4.2 Modulo con Touch Board

Nell'implementazione del modulo con la Touch Board è necessario aver installato nel sistema il linguaggio di programmazione Python, descritto nella sezione 2.3.10.

La Touch Board si presenta come una scheda con 40 porte I/O (le quali devono essere collegate alle GPIO della scheda RaspberryPi) e con un sensore elettrico di prossimità, come mostrato in figura 4.2, che permette di catturare i movimenti fino a 5 cm di distanza. Le librerie Python, in dotazione con la scheda, offrono funzioni per catturare i diversi input trasmessi, come ad esempio la direzione di spostamento del dito, oppure la cattura di un tocco sulla scheda.

La struttura del modulo, mostrata in figura 4.3, è composta da un'entità *Riconoscimento del comando* che, al ricevimento di un input sulla Touch Board, elabora e riconosce il segnale ricevuto (che può essere la direzione di spostamento del dito sulla scheda oppure un tocco) e lo inoltra al modulo.

Il comando viene validato dall'entità *validatore di comando* tramite semplice comparazione tra la stringa ricevuta e un array di stringhe, dove ogni elemento corrisponde ad un possibile comando che si può impartire. In caso di risposta positiva, il comando viene inoltrato in broadcast agli altri moduli, con i metodi discussi nella sezione 3.4.

Per poter eseguire un programma Python sul MM è necessario usare la libreria Javascript *Python-Shell*, la quale permette di avviare una shell di Python in background e avviare, di conseguenza, i programmi. La comunicazione tra programma Python e il NodeHelper avviene tramite messaggi in JSON. Quando la scheda riceve un input, il programma Python comunica il risultato al NodeHelper che lo inoltra al modulo.

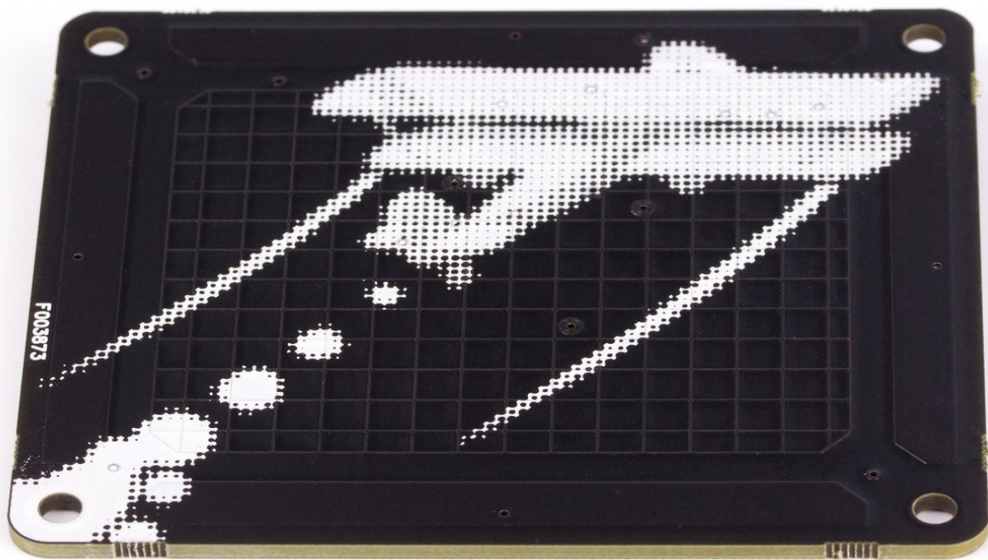


Figura 4.2: Touch Board Skydriver by Piromoni

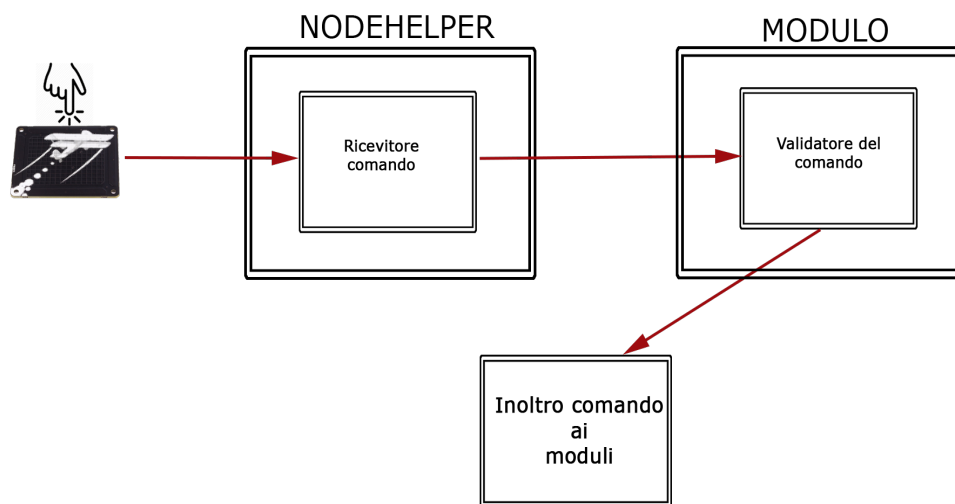


Figura 4.3: Struttura del modulo con la Touch Board

Capitolo 5

Interfaccia web di configurazione

In questo capitolo vengono esposti la progettazione e l'implementazione dell'interfaccia web di configurazione nel MM. Al fine di ospitare la pagina web che contiene il pannello di configurazione, è necessario creare una nuova classe server oltre a quella già esistente, citata nella sezione 3.1. Il server gira su una porta differente rispetto al primo server, la quale è indicata nel file di configurazione del MM, ed è accessibile da qualsiasi utente all'interno della sottorete. La struttura del server viene gestita con Express, spiegato nella sezione 2.3.6, che offre funzioni per gestire più facilmente le richieste delle pagine e le relative risposte. Inoltre, all'inizializzazione del server, viene passato come parametro il file di configurazione del MM.

Affinchè l'interfaccia funzioni completamente è stato necessario inserire una nuova dipendenza: per ogni modulo deve essere creato un file JSON con lo stesso nome, contenente i campi e le variabili della propria configurazione.

In particolare, nel capitolo, verranno esposte la struttura del funzionamento dell'interfaccia (sezione 5.1) e le metodologie per l'individuazione dei moduli presenti nel MM, con i relativi file JSON, al fine di renderli modificabili (sezione 5.2).

5.1 Struttura dell'interfaccia Web di configurazione

Dato che il server è implementato con Express, la struttura è caratterizzata dalle due entità (Controller e Modello), descritte nella sezione 2.3.6. Come

mostrato in figura 5.1, un utente (tramite il browser) fa richiesta di una pagina di un determinato modulo passando un Uniform Resource Identifier (*URI*), che viene ricevuto dal controller principale (passo 1) e inoltrato al modello (passo 2) a cui fa riferimento (in questo caso il modello è uno soltanto). Il modello ricerca in locale il file JSON del modulo richiesto (passo 3) e lo passa al controller (passo 4 e 5), il quale lo inoltra alla vista. Quest'ultima aggiorna il template della pagina inserendo i dati contenuti nel JSON che è stato passato, e la inoltra al browser che ne ha fatto richiesta. Non è stato necessario utilizzare il Routing dal momento che è stato necessario implementare un solo modello e, di conseguenza, un solo controller.

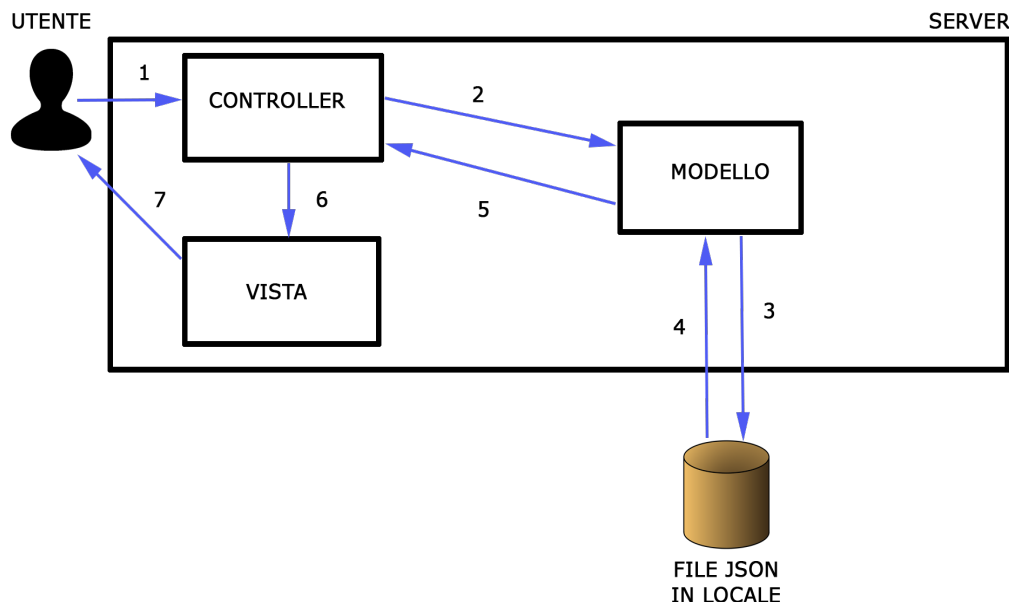


Figura 5.1: Struttura del server dell'interfaccia web

5.2 Individuazione e visualizzazione dei moduli nell'interfaccia

Come già discusso nella sezione 3.3, tutti i moduli sono contenuti nella cartella *Modules* del MM. Il server, tramite una funzione della libreria, legge i nomi di tutte le cartelle contenute all'interno di *Modules*, il cui percorso è

stato passato come parametro alla funzione. In seguito, salva tutti i nomi dei moduli in una lista, filtrando i file e le cartelle che non si riferiscono ad essi. La funzione per leggere le directory, appena citata, offre un metodo per leggere ed elencare le cartelle in modo sincrono, così da non rendere necessario l'utilizzo di callback.

All'interno di *Modules* è presente una cartella *default*, che contiene i moduli standard in dotazione con il MM. Per poter elencare anche questi ultimi è stato necessario utilizzare la funzione di libreria precedentemente usata anche su quest'ultima cartella e concatenare, successivamente, le due liste, per ottenere un'unica lista con tutti i moduli, necessaria per poter popolare il menù del pannello web di configurazione.

Tutte le pagine web che ritorna il server sono divise in 2 sezioni come illustrato nella figura 5.2:

- Il menù laterale, che contiene tutti i moduli messi nella lista (ad esempio *calendar* e *clock*). Ogni opzione del menù corrisponde ad un modulo, che inoltra la richiesta con il rispettivo URI
- Le varie impostazioni del modulo, visualizzate tramite *CodeMirror*[18], una componente Javascript che permette l'implementazione di un'area di testo per la modifica di codici all'interno di una pagina web.

Una pagina web consiste in un file con estensione *mustache* contenente codici HTML e variabili Mustache, motore di template esposto nella sezione 2.3.7. Il server, alla richiesta di una pagina di uno specifico modulo, ricerca il relativo file JSON all'interno della cartella del modulo stesso e lo passa come parametro sottoforma di oggetto JSON alla funzione *render*, necessaria per la creazione e visualizzazione della pagina. Insieme vengono passati i parametri *posizione* ed *header*, trattati nella sezione 3.2.1, estratti dal file di configurazione passato all'inizializzazione del server. Durante la creazione della pagina, i vari campi trasmessi, vengono inseriti dentro l'area di testo di *CodeMirror*, che permetterà di eseguirne le modifiche. Sotto all'area di testo è presente un bottone, che permette di attivare o disattivare il modulo nel MM; tuttavia affinché la modifica venga apportata è richiesto un riavvio manuale del software. Le modifiche effettuate vengono spedite al server che si occupa di salvarle nel file di configurazione principale del MM. Le metodologie e le pratiche per modificare e salvare le modifiche nel file di configurazione sono state sviluppate ed implementate da una mia collega.

Nel caso che durante una delle operazioni di fetch o di invio dei dati alla pagina si verifichi un errore, viene mostrata una pagina con un errore generico.

Invece, in caso di errata sintassi del file JSON modificato, viene ricaricata la pagina del modulo appena modificato mostrando una stringa di errore sopra all'area di editor di testo.

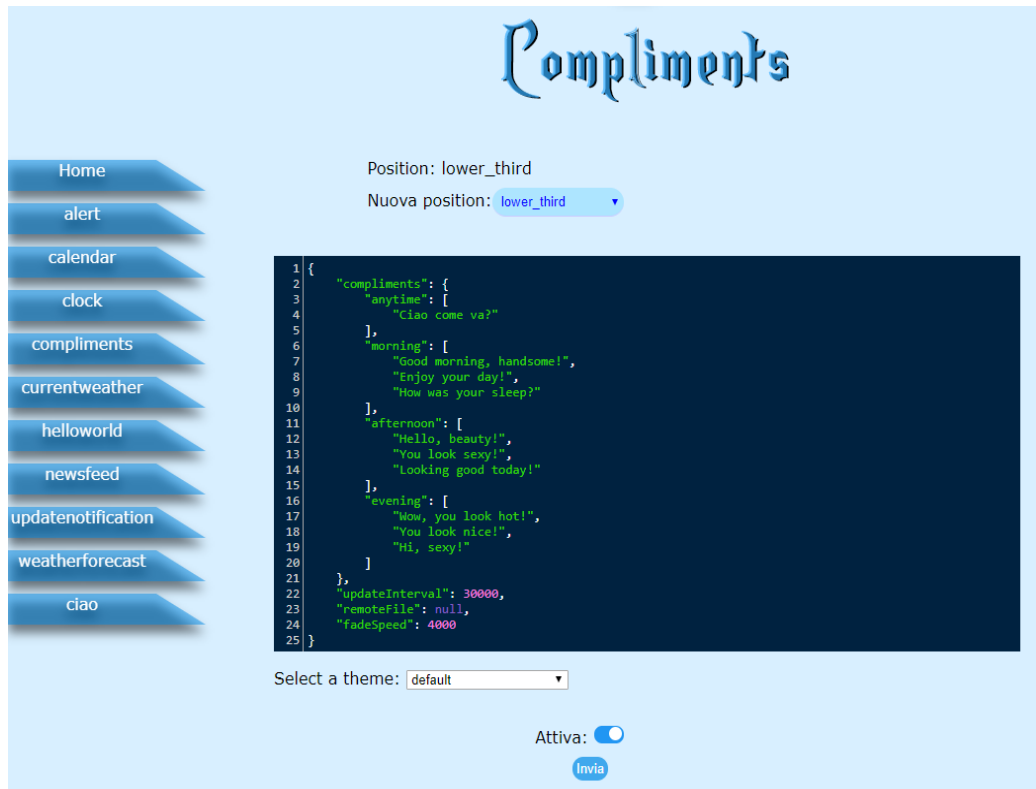


Figura 5.2: Interfaccia web di amministrazione

Conclusione

Per riassumere il lavoro svolto, si ricorda che durante la prima parte dello stage sono state sviluppate due interfacce utente per il MM:

- Una vocale, che permette l'impartizione di comandi vocali validi. In questa parte si è dovuto affrontare il problema di interfacciarsi ai servizi Google e di far comunicare correttamente il microfono con le API esposte, tramite il software SoX
- Una touch, che permette l'impartizione di comandi validi tramite il movimento delle dita su una scheda touch. Il problema riscontrato con questa interfaccia è stato implementare un canale di comunicazione tra il programma python che cattura i controlli sulla scheda e il NodeHelper.

Nella seconda parte è stata implementata un'interfaccia web di configurazione remota che permette di configurare più facilmente il MM e i suoi moduli. I problemi riscontrati in questa parte sono stati:

- La corretta individuazione, visualizzazione e modifica dei file JSON dei moduli, contenuti all'interno dei file locali del MM
- L'utilizzo di una buona pratica di programmazione in Javascript per evitare fenomeni che portassero ad una scrittura del codice molto confusionaria.

Alcune parti di questi progetti sono ancora arcaici o poco sviluppati: infatti un possibile sviluppo futuro delle interfacce utente è l'inserimento di un'unità logica che possa validare i comandi impartiti (sia vocali che touch), in modo più concreto rispetto ad un semplice controllo di stringa.

Invece, per l'interfaccia web di configurazione, si può implementare un sistema per il riavvio automatico del software, anziché farlo manualmente, insieme ad una crittografia dei messaggi scambiati, in modo da inserire un livello di sicurezza.

Bibliografia

- [1] Raspbian wikipedia, <https://it.wikipedia.org/wiki/Raspbian>
- [2] Debian wikidia, <https://it.wikipedia.org/wiki/Debian>
- [3] Raspberry official website, <https://www.raspberrypi.org/>
- [4] OpenCV official website, <http://opencv.org/>
- [5] Google Speech to Text API documentation,
<https://cloud.google.com/speech/>
- [6] Electron official website, <https://electron.atom.io/>
- [7] MVC Wikipedia, <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- [8] Express official website, <http://expressjs.com/it/>
- [9] MySQL official website, <https://www.mysql.com/it/>
- [10] Mustache Git Site, <https://mustache.github.io/>
- [11] JavaScript, <https://www.javascript.com/>
- [12] Python website, <https://www.python.it/>
- [13] GitLab website, <https://about.gitlab.com/>
- [14] Node.JS website, <https://nodejs.org/it/>
- [15] V8 wikipedia,
[https://it.wikipedia.org/wiki/V8_\(motore_JavaScript\)](https://it.wikipedia.org/wiki/V8_(motore_JavaScript))
- [16] npm docs,
<https://docs.npmjs.com/getting-started/what-is-npm>
- [17] Chromium wikipedia, <https://it.wikipedia.org/wiki/Chromium>

- [18] CodeMirror website, <https://codemirror.net/>
- [19] MagicMirror website, <https://magicmirror.builders/>