Universitá del Piemonte Orientale

Dipartimento di Scienze e Innovazione Teconologica

Corso di Studi in Informatica

Relazione per la prova finale

MagicMirror(provvisorio)

Tutore interno:
Prof. Marco Guazzone

Candidato:
Riccardo Berto

Anno Accademico 2016/17

Indice

Introduzione

Nello stage che ho svolto presso Lab121 ho stato studiato il Magic Mirror, 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 (detti anche moduli) che si interfacciano con periferiche esterne, o con api (italic), e il software principale.
- Lo sviluppo di un'interfaccia web che permette di modificarne la configurazione senza dover accedere a file potenzialmente critici ed eliminarne parti essenziali ad opera di utenti non esperti.

Per la prima parte dello stage sono state create applicazioni mirate al controllo del Magic Mirror tramite l'impartizione di comandi vocali o input trasmessi dal movimento delle dita su una periferica touchpad.
Questa scelta è stata presa prendendo atto che nel corso degli ultimi decenni, con il perfezionamento delle teconologie e dei software, ha fatto nascere una tendenza a controllare macchine o dispositivi con metodi sempre meno "diretti". 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. Inoltre, lo sviluppo dei calcolatori e delle periferiche sempre più piccoli ed economici ha permesso la nascita di una moltitudine di progetti casalinghi con questo fine, contribuendo cosí alla crescita dei software in ambito relazione uomo-machcina, oggi reperibili in quantit anche open source.

Un altro aspetto che si è evoluto soprattutto negli ultimi anni nel campo delle applicazioni e delle Applicazioni Web (chiamate anche webb app) è l'implementazione di interfacce grafiche di gestione sempre più intuitive e facili da comprendere. Capita spesso che un'applicazione venga utilizzata da utenti

privi di conoscenze informatiche, i quali, se mancasse l'interfaccia di gestione, dovrebbero accedere direttamente ai file di configurazione per modificarli, e potrebbero causare potenziali danni.
A tal fine, nella seconda parte dello stage, è stata sviluppata un'interfaccia grafica web, la quale mostra un pannello di configurazione: il compito è di permettere la corretta modifica dei file di configurazione delle applicazioni e del Magic Mirror.

Nel capitolo 3 verrà spiegata la struttura del Magic Mirror.

Capitolo 1

Scopi e Problemi Affrontati

1.1 Applicazioni del MagicMirror

Il Magic Mirror è un software che implementa api, le quali permettono l'integrazione di applicazioni sviluppate da terzi e forniscono strumenti per la comunicazione, gestione ed organizzazione tra di esse. 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 Magic Mirror tramite l'impartizione di comandi vocali o movimenti delle dita su una scheda touchpad. 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

Nella seconda parte dello stage si è svolta la progettazione di un'interfaccia Web di configurazione remota il cui scopo è di permettere ad un utente autenticato di gestire le configurazioni delle applicazioni implementate nel software da una pagina web, senza dover accedere fisicamente alla macchina. Quest'ultima permette anche di disattivare (o attivare) le applicazioni presenti nel Magic Mirror, la cui procedura, prima, richiedeva di aggiungere manualmente un pezzo di codice nella configurazione del software principale.

Nello sviluppo dell'interfaccia si è affrontato il problema di dover modificare il documento di configurazione del software principale senza eliminarne parti essenziali o modificarlo in un formato sbagliato. Lo scambio dei messaggi tra il Magic Mirror e la pagina web avviene in formato JSON, lo stesso formato usato per il suo file di configurazione e delle sue applicazioni.
Un altro problema riscontrato in questa parte stata la difficolt dell'utilizzo di Javascript perch è un linguaggio asincrono, ovvero alcune funzioni venivano eseguite senza attendere il risulato della precedente. Il linguaggio, per poter sincronizzare la funzioni, mette a disposizione le callback: funzioni passate come parametro alla funzione principale che a loro volta avevano come paramentro il risultato, in modo da poter eseguire le operazioni solo dopo che la funzione chiamante avesse prodotto l'output. Il problema di questo approccio è stato il fenomeno "Callback Hell", ovvero Callback che chiamano a loro volta altre Callback creando funzioni annidate più volte, come mostrato in figura 1.1.

```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          })
        })
      })
    })
  })
});
```

Figura 1.1: Esempio di Callback Hell

Capitolo 2

Tecnologie Implicate

Lo sviluppo del progetto è stato svolto nell'ambiente Raspbian[1] una distribuzione Debian[2] che gira sul dispositivo RasperryPi[3]. Inoltre sono state adottate diverse tecnologie, recenti e non, nel campo della creazione di applicazioni(Electron [6], OpenCV [4], GoogleSpeechRecognition [5]), dei web server (NodeJS [14], Express [8], Mustache [10]), della gestione dati (MySQL [9]). Inoltre sono stati adottati diversi linguaggi di programmazione (JavaScript [11], Python [12]) e piattaforme per la gestione distribuita di progetti softwave (GitLab [13]).

2.1 Hardware

2.1.1 RasperryPi

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:

- 1 porta HDMI
- 1 porta LAN
- 1 uscita Aux
- 4 porte USB
- 40 pin General Purpose Input/Output(GPIO)

- 1 scheda di rete wirless
- Alimentazione microUSB 5V
- un bus camera serial interface(CSI), ovvero una porta per telecamere 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).

2.1.2 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 scheda.

2.2 Software

2.2.1 Raspbian

Raspbian è una distribuzione del sistema operativo Debian, completamente libero, ottimizzato per Raspberry. Fu sviluppato da Mike Thompson e Peter Green come progetto non affiliato alla compagnia RaspberryPi fundation, pensato apposta per la bassa potenza dei processori Advanced RISC Machine(ARM) montati sul dispostivo. La prima versione venne rilasciata nel 2012.

2.2.2 Electron

Electron è un Framework open source rilasciato per la prima volta nel 2013, ma la prima versione stabile è uscita solo di recente. È disponibile sui sistemi operativi Window, MacOS e Linux ed è scritto in C++ e Javascript. Il framework permette la creazione di interfacce grafiche (GUI) per applicazioni multi-piattaforma, utilizzando teconologie già esistenti per lo sviluppo del lato client e del lato server (Javascript, NodeJS, V8 [15]). All'avvio di Electron viene inizializza una pagina con Chromium [17](il quale viene installato con

l'applicazione) nel quale viene mostrata una pagina web, e un server in NodeJS che si occupa di trasmettere e interagire con il primo. Un'applicazione Electorn ha bisongo 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)

```
{
  "name": "magicmirror",
  "version": "2.1.1",
  "description": "The open source smart platform",
  "main": "js/electron.js"
}
```

- Un file HTML che contiene il template della pagina mostrata 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.

2.2.3 OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria software sviluppata intorno al 2000 utilizzata nell'ambito della visione in tempo reale 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 e dunque di cui ha l'interfaccia primaria), 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 algoritmi implementati al suo interno, riconosce oggetti o specifiche forme. Inoltre si può aumentarne la capacità applicando algoritmi di Machine Learning per individuare e riconoscere oggetti specifici.

2.2.4 Google Speech API

Negli ultimi anni Google ha ampliato sempre di più il suo catalogo per quanto rigarda 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 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 To Text 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

2.2.5 NodeJS

NodeJS è una piattaforma open source, utilizzata per progettare il backend di un server web, ovvero la parte del sistema che contiene applicazioni e programmi con cui l'utente non interagisce e che ne permette l'effettivo funzionamento e, nel caso, la manipolazione e l'elaborazione dei dati, sfruttando il motore JavaScript V8 sviluppato da Google.
NodeJS 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.

Express

Express è un framework per NodeJS che permette di creare applicazioni Web e API in JavaScript. Il software viene usato per creare e gestire il backend di un server, che è composto di 3 entità importanti:

- il Routing, utilizzato per determinare come il server debba rispondere ad un determinato metodo di richiesta, ricevuta sottoforma di URI, inoltrando la richiesta alla funzione del controller del modello a cui fa rifermento.
- 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 databse per estrarre i dati e spedirli al controller per la manipolazione, ricevendoli successivamente modificati e salvandoli, se necessario.

- il Controller, che definisce le funzioni associate ad un determinato modello.

2.2.6 JavaScript

Javascript è un linguaggio di programmazione di alto livello per oggetti ed eventi, che è supportato da tutti i browser per lo scripting delle pagine web, conosciuto per l'utilizzo in programmazione procedurale e per la programmazione orientata ad oggetti. Inizialmente usato per il lato client ha subito un'evoluzione che lo ha portato ad essere utilizzato per lo sviluppo di backend e web app. Gli oggetti in Javascript vengono creati a livello di codice collegando metodi e proprietà ad altri oggetti, anche vuoti, a tempo di esecuzione.

ECMAJavascript(ES) è lo standard di Javascript che negli ultimi anni ha sviluppato ed evoluto il linguaggio in diverse versioni. In tutte il problema più trattato è il fatto che Javascript è asincrono, ovvero non attende il completamento di alcune operazioni prima di eseguire le funzioni successive. Questo avviene perchè le chiamate di funzioni non vengono fatte direttamente, ma vengono fatte via messaggi, i quali vengono salvati in una coda di messaggi e vengono spediti sequenzialmente ad uno stack di chiamata dove viene salvata la corrispondente funzione per l'esecuzione. Questo metodo rende il linguaggio asincorno perchè le funzioni e gli eventi vengono eseguiti in successione senza attendere il termine della precedente.

Le soluzioni adottate in ECMA Javascrit 5 (ES5), la versione usata durante lo stage, sono l'utilizzo delle callback gi discusse nel cpaitolo precedente.

In ECMA Javascript 6 (ES6) sono state introdotte le Promises, ovvero al posto di far tornare una funzione Callback, ritorna una Promise(Promessa), la quale garantisce che una variabile/oggetto avrà un ritorno, mettendo cosi la funzione in attesa fino al ricevimento del valore o di un errore. Sono tutt'ora in via di sviluppo nuove versioni di Javascript.

2.2.7 Python

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

Come molti altri linguaggi supporta moduli e pacchetti anche sviluppati da terzi, salvati in una repository pubblica, attraverso un suo gestore di pacchetti Pip Installs Packages o Pip Installs Python (pip, acronimo ricorsivo).

Inoltre con Python è possibile creare ambienti isolati per l'utilizzo di pacchetti e moduli senza doverli installare all'interno del sistema. Per poterlo fare è disponibile virtualenv, uno strumento Python che permette, appunto, di creare ambienti virtuali (virtual enviroments). Questo linguaggio è stato particolarmente utile per l'interfacciamento con la telecamera e la Touch Board, essendo a disposizione librerie apposite per gestirle.

2.2.8 GitLab

GitLab è una piattaforma web 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 evenutali richieste o problemi. Il software Git è stato creato da da Linus Torvalds nel 2005 per la gestione e il versionamento del codice in Linux. Git è, in sostanza, un grafo diretto aciclico di oggetti, ognuno con un'identificativo univoco, ed è composto da 4 parti principali:

- i Blob, insieme di bytes che contengono i dati all'inteno dei file.
- i Trees, sono le cartelle che contengono, all'interno, un lista di file con alcuni bit i quali fanno riferimento al rispettivo blob o ad un'altro Tree.
- i Tag, sono etichette con dati aggiuntivi per identificare e gestire i nodi.
- i Commit, collegano più oggetti Tree ad uno storico nel quale sono salvati anche i nomi di tutti i commits parenti.

Capitolo 3

Struttura del Magic Mirror

Il Magic Mirror è un progetto ideato e sviluppato da Michael Teeuw, successivamente esteso nelle sue funzionalità da una moltitudine di utenti su GitHub. Una prima versione è stata scritta completamente in Python, mentre successivamente è stata creata una seconda versione nella quale si è preferito l'utilizzo di 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.

3.1 Perchè Magic Mirror?

L'idea dell'autore è nata rifacendosi allo specchio magico dell'omonima fiaba scritta dai fratelli Grimm, La Bella Addormentata.
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.

Figura 3.1: Magic Mirror by Michael Teeuw

3.2 Avvio ed Escuzione

Il Magic Mirror viene avviato tramite il comando *npm start*, il quale esegue il codice di un file Javascript, il cui percorso viene indicato nel documento *package.json*. Il primo file contiene il codice di Electron, che si occupa della creazione di una nuova finestra usata per rappresentare l'interfaccia mostrata dal browser. Quest'ultimo contiene a sua volta i Document Object Model (DOM) e il codice dell'applicazione, ovvero il core dello specchio. Quest'ultima carica tutte le strutture dello specchio, che sono le seguenti:

- gli end-point per le applicazioni, che sono le interfacce usate per leggere e caricare le applicazioni inserite nel Magic Mirror.

- gli end-point per i Node Helper, per la gestione dei questi ultimi. Sono strutture opzionali usate per collegamenti esterni al Magic Mirror (per esempio, con API di un servizio cloud). Ogni applicazione ha il proprio Node Helper con cui può comunicare tramite messaggi in modo simile a come comunicano le applicazioni tra di loro.
- un "Socket", entità principale che definisce le funzioni e le metodologie per lo scambio dei messaggi tra le applicazioni e i rispettivi Node Helper.
- un Logger, implementato per tenere i log dell'applicazione e degli evenutali errori. Usato pricipalmente per il debugging.
- un file *config.txt*, ovvero un file di configurazione dello specchio, nel quale sono segnate il nome e le coordinate per la posizione delle varie applicazioni all'interno della pagina

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

3.2.1 Il file di Configurazione

Come già menzionato, il Magic Mirror carica un file di configurazione, che 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 del timer (12h o 24h)
- unità di misura usata (ad esempio, metrica)
- una lista di applicazioni (in formato JSON) da caricare 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.

```
{
        "module": 'nome dell'applicazione',
        "position": 'la posizione dell'applicazione
            all'intenro dello specchio',
        "header": 'stampa sopra all'applicazione',
        "config": { opzioni varie in formato JSON }
}
```

La lingua, il formato del timer e l'unità di misura usata sono strumenti 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 Magic Mirror 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 Node Helper associato alla specifica applicazione (opzionale)
- Altri file necessari all'applicazione (immagini, JSON, etc)

Il file Javascript principale dell'applicazione viene caricato per primo dal Magic Mirror, e consiste in una chiamata di funzione con i seguenti due parametri:

```
{
        Module.register("Nome dell'applicazione", { /*
            lista JSON di oggetti contenenti funzioni o
            variabili */});
}
```

Il primo parametro è una stringa contenente il nome dell'applicazione: quest'ultimo deve essere necessariamente uguale al nome della cartella e al nome del file Javascript. Il secondo parametro, invece, è una lista JSON contenente funzioni o variabili (come in figura *da inserire*). Le funzioni offerte dalle API sono:

- defaults: , una lista di variabili che possono essere richiamate all'interno di una qualsiasi funzione tramite il comando *this.config.variabile*. I valori di queste possono essere sovrascritti modificando il campo *config* del relativo modulo nel file di configurazione del Magic Mirror.

- start: function(), funzione che viene eseguita quando tutte le applicazioni dello specchio sono state caricate (ovvero quando sono stati creati tutti i relativi DOM)

- getDom: function(), funzione che deve ritornare un DOM (un oggetto HTML contenente i dati da mostrare a schermo), creato tramite funzioni Javascript

- getStyles: function() return [], funzione che ritorna un array di file (in formato CSS) usati per l'estetica del DOM. Possono essere nella cartella dell'applicazione oppure ottenuti tramite link

- getTranslations: function() return en: "translations/en.json", de: "translations/de.json", funzione per tradurre l'applicazione in più lingue; se disponibile, viene caricata la traduzione in base alla lingua configurata nel software

- getHeader: function() return this.data.header;, funzione che stampa il campo *header* della configurazione, con la possibilità concaternarla ad una stringa o ad un parametro

- notificationReceived: function(notification, payload, sender) , funzione che serve per ricevere messaggi da altre applicazioni. Viene richiamata alla ricezione

- socketNotificationReceived: function(notification, payload), funzione che serve per ricevere messaggi dal Node Helper della relativa applicazione

Inoltre possono essere aggiunte delle funzioni necessarie all'applicazione implementandole con la stessa sintassi di quelle di default.

Il Node Helper, invece, viene caricato come libreria ed esportato insieme all'applicazione quando viene caricata dallo specchio. Per inizializzarlo sono necessari i comandi:

```
{
  var NodeHelper = require("node_helper");
  module.exports = NodeHelper.create({/* lista JSON di
     oggetti contenenti funzioni*/});
}
```

A differenza del file Javascript principale, l'unica funzione messa a disposizione dall' API è:

```
{
  start: function() {}
}
```

mentre è possibile implementare le proprie funzioni con la stessa sintassi.

3.4 Messaggistica del MM

In precedenza è stato già accennato che il Magic Mirror 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. Le funzioni usate per ricevere i messaggi, come già detto precedentemente, sono: *socketNotificationReceived(notification, payload)*, per la ricezione dei messagi da parte del Node Helper, e *notificationReceived(notification, payload, sender)*, per la ricezione di messaggi dalle altre applicazioni. Per spedire messaggi vengono chiamate le funzioni *sendSocketNotification(notification, payload)* e *sendNotificationReceived(notification, payload, sender)*, i cui campi indicano:

- notification, l'indentificatore della notifica, una sorta di header al quale si può assegnare un qualunque oggetto
- payload, il corpo del messaggio (opzionale)
- sender, usato solo nei messaggi tra i moduli, dove viene riportato l'identificativo dell'applicazione che lo ha mandato

sendNotificationReceived a differenza del suo corrispettivo per il Node Helper, manda il messaggio in broadcast a tutte le applicazioni attive sulla macchina, e deve essere filtrato dalle singole applicazioni in base al campo *notification*.

Capitolo 4

Applicazioni per il MM

4.1 Modulo per comandi vocali

La prima applicazione implementata, come stato accennato nell'introduzione, stata il controllo del Magic Mirror tramite comandi vocali. Nello specifico l'applicazione doveva, tramite delle specifiche frasi, gestire le altre applicazioni presenti nel Magic Mirror.
Il compito di trasformare da parlato a testo scritto e l'elaborazione delle frasi stato lasciato a carico di Google Speech API. Prima di spiegare il codice ci sono alcune dipendenze da spiegare.

4.1.1 Sound eXchange (SoX)

Per permettere al microfono di catturare ed elaborare correttamente l'audio nell'ambiente Raspbian necessario installare Sound eXchange (SoX), un software per la manipolazione dell'audio, utile per effettuare uno streaming dell'audio. Per installarlo correttamente necessario avere il Sistema Operativo aggiornato, usando il seguente comando da Shell:

```
$ sudo apt-get update
```

Il comando *sudo* un controllo speciale che permette di eseguire i comandi in modalit Super User, un utente speciale che ha i privilegi di amministrazione all'interno del sistema. *apt-get* uno strumento da riga di comando che permette di usare l'APT (Advanced Packaging Tool), un gestore di pacchetti dei sistemi Debian. Il parametro *get* indica l'operazione da eseguire con *apt*, in questo caso scaricare un pacchetto. *update* indica l'obbiettivo di cercare aggiornamenti del S.O. Una volta completato si pu installare il software sempre tramite comando da Shell:

```
$ sudo apt-get install sox
```

Il parametro *install* serve ad indicare di installare il pacchetot oltre a scaricarlo, mentre *sox* il nome del pacchetto che si vuole cercare. Al termine del processo necessario collegare il driver audio di Raspbian ALSA (Advanced Linux Sound Architecture) a SoX e indicare a quest'ultimo a quale periferica interfacciarsi, tramite i comandi:

```
$ export AUDIODEV=hw:1,0
$ export AUDIODRIVER=alsa
```

export un comando Linux usato per assegnare valori a Variabili d'Ambiente. *AUDIODEV* la variabile d'ambiente che identifica il dispositivo audio di Default, mentre *hw:1,0* indica l'assegnamento della periferica 0 presente sulla scheda 1. il comando *export AUDIODRIVER=alsa* opzionale, perch assegna alla variabile d'ambiente *AUDIODRIVER*, la varibiale che fa riferiemnto al driver audio del sistema, ALSA, il quale dovrebbe essere gi quello predefinito.

4.1.2 Autenticazione Google API

Per poter usufruire dell'API di Google necessario fornire un'autenticazione a livello di sistema. Per poterlo fare necessario ottenere delle credenziali di sicurezza per un account Google, attivabili tramite Google Cloud Platform Console. Le credenziali consistono in un username, la mail dell'account google, e una chiave di sicurezza unica, entrambi sono contenuti in un file JSON che pu essere scaricato e salvato in locale. Per poter permettere al sistema di utilizzare l'API bisogna fare in modo che le credenziali siano raggiungibili, inserendole in una Variabile d'Ambiente con il comando:

```
$ export GOOGLE_APPLICATION_CREDENTIALS='Percorso del
    JSON'
```

4.1.3 Node Heloer dell'applicazione

Il node Helper dell'applicazione si occupa di gestire lo streaming con L'API e di mandare i risultati (o gli errori) all'applicazione tramite le funzioni messe a disposizione dal Magic Mirror mostrate precedentemente.

```
var NodeHelper = require("node_helper");
const record = require('node-record-lpcm16');
const Speech = require('@google-cloud/speech');
const speech = Speech();
const request = {
  config: {
    encoding: encoding /*(Es. LINEAR16)*/,
```

```
      sampleRateHertz: sampleRateHertz /*(Es. 16000)*/,
      languageCode: languageCode /*(Es. it-IT)*/
    },
    interimResults: false /*usato se si vuole conoscere
        l'output prima dell'elaborazione*/
  };

module.exports = NodeHelper.create({
  start: function () {
        console.log("Starting Node Helper of module: "
           +this.name);
    },

    socketNotificationReceived: function(notification,
       payload) {
        var self = this;
        if (notification === "ready") {
            listen();
        }
    },

    listen: function(){
      // Create a recognize stream
      const recognizeStream = speech.streamingRecognize
         (request)
        .on('error', sendSocketNotification("error"))
        .on('data', (data) =>
          if(Transcription: ${data.results[0].
             alternatives[0].transcript}\n)
            sendSocketNotification('limit_reached')
          else
            sendSocketNotification('response', data.
               results[0])

      // Start recording and send the microphone input
         to the Speech API
      record
        .start({
          sampleRateHertz: 1600,
          threshold: 0,
          verbose: false,
```

```
            recordProgram: 'sox',
            silence: '20.0'
        })
        .on('error', sendSocketNotification('error'))
        .pipe(recognizeStream);
    }
})
```

4.1.4 Il Main

Il main dell'applicazione gestisce le risposte ricevute dal NodeHelper ed esegue i comandi impartiti da esse. Inoltre si occupa di mostrare a schermo quando il programma "in ascolto".

```
Module.register("voicecontrol",{
// Default module config.
defaults: {
  rosso: "modules/voicecontrol/icons/redpin.png",
  verde: "modules/voicecontrol/icons/greenpin.png",
  ready: false,
  modules: [],
  current: "",
}

start:function(){
  console.log("module"+ this.name+"started");
}

getDom: function(){
  var wrapper = document.createElement("div");
  var containerled = document.createElement("div");
  var containerlist=document.createElement("div");
  var list=document.createElement("ul");
  var led=document.createElement("span");
  led.className="ledcontainer";
  var imag = document.createElement("img");
  if(!this.config.ready){
    imag.src=this.config.rosso;
    led.innerHTML="Not Listening...";
  }
  else
```

```
    imag.src=this.config.verde;
    led.innerHTML="On Air";
  }
  imag.style.height = '30px';
  imag.style.width = '30px';
  for(var i=0; i<this.config.modules.length; i++){
    var option=document.createElement("li");
    if(modules[i]==current)
      option.style.color = 'red';
    option.innerHTML=modules[i];
    list.appendChild(option)
  }
  led.appendChild(imag);
  containerled.appendChild(led);
  containerlist.appendChild(list);
  wrapper.appendChild(containerled);
  wrapper.appendChild(containerlist);
  return wrapper;
}

notificationReceived: function(notification, payload,
   sender) {
                if (notification === '
                   ALL_MODULES_STARTED') {
                        MM.getModules().exceptModule(
                            this).enumerate(function(
                            module) {
                                this.config.modules.
                                   push(module.name);
        module.hide(1000, function() {
                                        //Module hidden
                                           .
                                    });
                                    });
                    });
            }
  if (notification === 'DOM_OBJECTS_CREATED') {
    this.sendSocketNotification('ready');
    this.config.ready=true;
    this.updateDom();
  }
```

```
},

socketNotificationReceived: function(notification,
    payload) {
    if (notification === 'limit_reached') {
      this.config.ready=false;
      this.updateDom();
    }
    else if(notification === 'error'){
      this.config.ready=false;
      this.updateDom();
    }
    else if(notification === 'response'){
      for(var j=0; j<this.config.modules.length; j++){
        if(payload===modules[j]){
          this.config.current=true;
        }
       MM.getModules().exceptModule(this).enumerate(
           function(module) {
                                    if(module.name==this.
                                       config.current)
           module.show(1000, function() {
                                          //Module show.
                                    });
                                    });
                    });
      this.updateDom();
    }
  }
}
```

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] Nodejs website, https://nodejs.org/it/

[15] V8 wikipedia, 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