Universitá del Piemonte Orientale

Dipartimento di Scienze e Innovazione Teconologica

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

Introduzione

Nello stage che ho svolto presso Lab121 ho esteso il 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 (detti 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'obbiettivo di dotare il MagicMirror di un'interfaccia uomo-macchina avanzata. Infatti, negli ultimi decenni, con lo sviluppo di nuove teconologie, 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 tecnolgico 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 teconologie, 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 teconologie 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 1.1 e lo sviluppo dell'interfaccia web di amministrazione 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, senza dover accedere alla macchina fisica su cui gira l'applicazione ed implementando una validazione per la sintassi del file di configurazione. 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 risulato della precedente. Il linguaggio, per poter sincronizzare le 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. Senza buone pratiche di programmazione si rischia di incontrarsi in un fenomeno chiamato "Callback Hell", ovvero Callback che chiamano a loro volta altre Callback creando funzioni annidate più volte, come mostrato in figura 1.1.

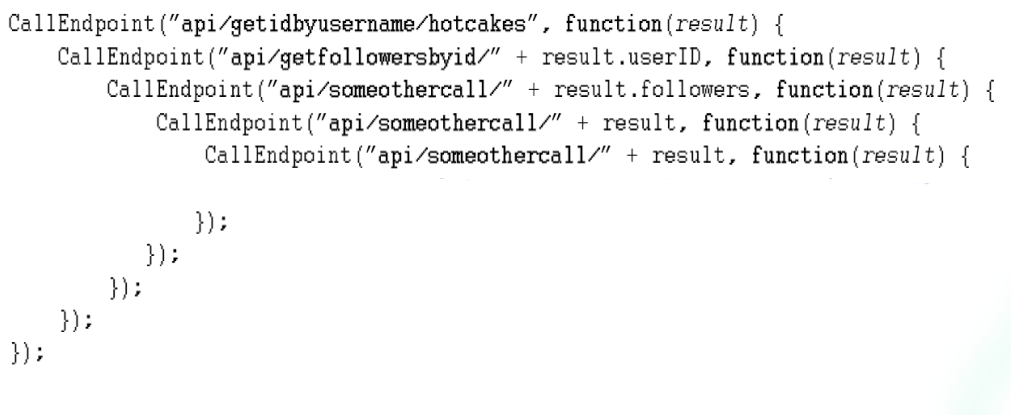

Figura 1.1: Esempio di Callback Hell

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

2.1 Prefazione

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 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]), 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]).

2.2 Hardware

2.2.1 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:

- 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 Fundation, 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 multipiattaforma 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 Electorn 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": "magicmirror",
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 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

2.3.5 Node.JS

Node.JS era già implementato 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 e API in JavaScript, offrendo strumenti e pacchetti che implementano più facilmente tutte le funzionalitè 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 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.

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 diverse lingue 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 controllli di questo tipo vengono fatti attraverso programmazione data-driven.
Il nome di questo sistema è dovuto all'utilizzo delle parentesi graffe, usate spesso all'intero della sua sintassi, che, se viste da una certa angolatura, assomigliano a un paio di baffi.

2.3.8 MySQL

MySQL è stato usato durante lo stage come database per l'archiviazione di account di utenti agibili all'autenticazione per accedere all'interfaccia web

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 da tutti i 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. In tutte il problema più trattato è il fatto che Javascript è orientato agli eventi, ed alcune funzioni necessitano di Callback, che possono degenerare se non vengono utilizzate buone pratiche di programmazione. Le callback sono necessarie perchè alcune chiamate di funzioni non vengono fatte direttamente, ma vengono fatte attraverso 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 asincrono perchè le funzioni e gli eventi vengono eseguiti in successione senza attendere il termine della precedente.

Le pratiche utilizzate per sincronizzare le funzioni in ECMAJavascript 5 (ES5), la versione usata durante lo stage, sono l'utilizzo delle Callback già discusse nella sezione 1.2.

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

2.3.10 Python

Questo linguaggio è stato particolarmente utile per l'interfacciamento con la telecamera e la Touch Board, essendo a disposizione librerie apposite per gestirle. Python è un linguaggio di programmazione open source, ad alto livello con semantica dinamica, orientato ad oggetti, usato per sviluppo di applicazioni e scripting.

Come molti altri linguaggi supporta 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).

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 assegnadoli un'etichetta in locale, per poi spedirlo al respository in remoto del progetto. In ogni momento si può tornare ad una versione vecchia tramite gli strumenti di reversering offerti dal servizio.

Capitolo 3

Struttura del MagicMirror

Il MagicMirror 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. L'idea dell'autore è nata rifacendosi allo specchio magico dell'omonima 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 questo capitolo verrà spiegata la struttura generale del MagicMirror 3.1, le classi principali che vengono caricate all'avvio 3.2, la struttura del file di configurazione 3.2.1, come viene implementata un'applicazione 3.3 e il funzionamento del sistema di messaggistica 3.4.

Figura 3.1: MagicMirror by Michael Teeuw

3.1 Il MM dall'alto livello

Il MM è una applicazione con diverse API e strutture che ne permettono il funzionamento. Si possono individuare alcuni elementi principali rappresentati in figura 3.2:

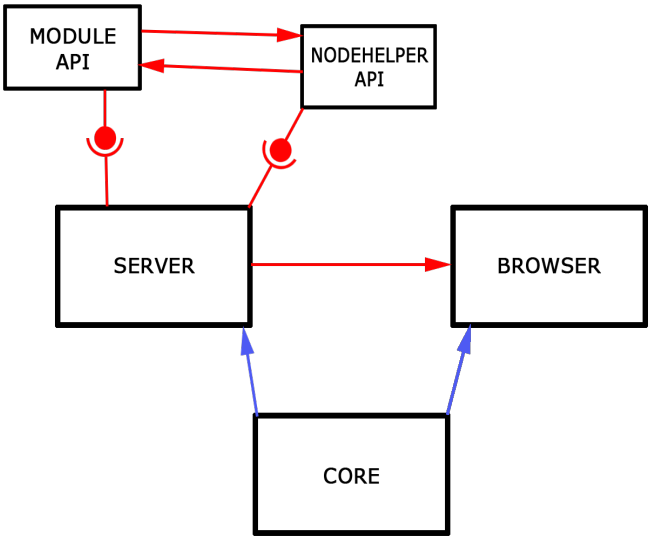

Figura 3.2: Struttura MagicMirror

- 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 gestisce le API
- Le API per i moduli, i metodi e funzionalità esposti dal MM per poter integrare i moduli creati
- Le API per i Node Helper, i metodi e funzionalità esposti dla MM per poter itegrare i NodeHelper dei relativi moduli
- Il Socket, entità che si trova nell'API dei moduli e fornisce un canale di comunicazione tra questi
- Il SocketClient, entità che si trova tra le API delle applicazioni e dei Node Helper, e fornisce il canale di comunicazione tra questi due.

3.2 Avvio ed Escuzione

Il MagicMirror viene avviato tramite linea di comando, la quale esegue il codice di un file Javascript, il cui percorso è indicato nel documento *package.json*, descritto nella sezione 2.3.2. Il file contiene il codice di Electron, che si occupa della creazione di una nuova finestra browser, il cui compito è di mostrare l'output dei moduli del MM, e l'avvio del codice che rappresenta il core, che carica tutte le strutture princiapli del MM. 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 MagicMirror.
- gli end-point delle API per i Node Helper, per la gestione di questi ultimi. Sono strutture opzionali usate per collegamenti esterni al MagicMirror (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" e un "Socket-Client", entità principali che definiscono le funzioni 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 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 resa, con gli output delle varie applicazioni, al browser precedentemente avviato.

3.2.1 Il file di Configurazione

Come discusso nel capitolo 1.2, il MagicMirror 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)
- unità di misura usata (ad esempio, metrica), usata per gestire la distanza dei DOM
- 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, come mostrato nell'esempio in figura 3.2.1.

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

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 MagicMirror 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)

3.4 Messaggistica del MM

Nella sezione 3.1 è stato già accennato che il MagicMirror 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 la funzione *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 la funzione *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
- Socket Client, 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 sevizi di Google (sezione 4.1.2), come avviene la comunicazione tra l'API 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, come è stato accennato nell'introduzione del capitolo 4, è stata il controllo del MagicMirror tramite comandi vocali. Nello specifico l'applicazione deve, tramite delle specifiche frasi, gestire le altre applicazioni presenti nel MagicMirror, 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 software Sound eXchange (*SoX*), un software per l'elaborazione e la manipolazione dell'audio. Successivamente, l'audio cos elaborato, viene passato all'API Speech To Text, 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 ai moduli, se corretto. Per poter utilizzare l'API è necessario installare il software SoX e fornire un'autenticazione per i servizi Google.

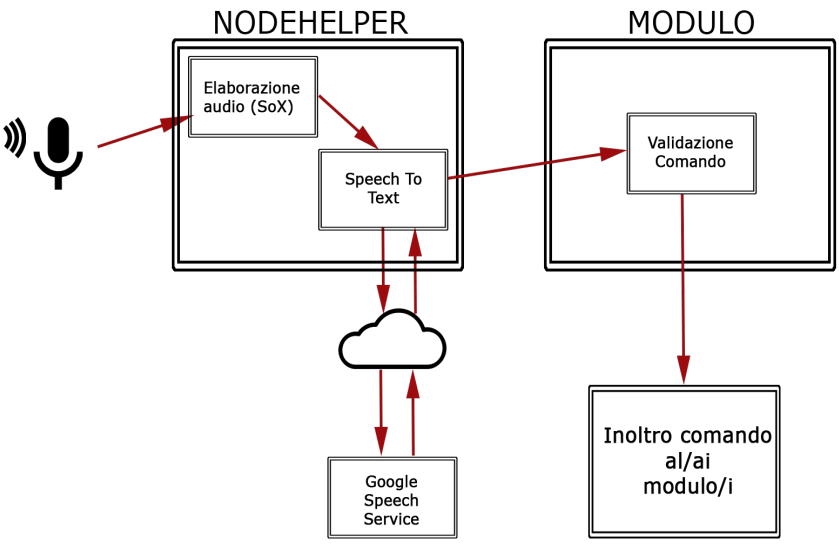

Figura 4.1: Struttura del modulo per comandi vocali

4.1.1 Comunicazione con l'API

Il NodeHelper 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 MagicMirror, 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 suc-

cessivamente 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 uteriormente 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 uno strumento per permettere a due processi di comunicare. 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 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)

Per permettere all'audio di venire correttamente elaborato per lo streaming, è necessario utilizzare Sound eXchange (SoX), citato nella sezione 4.1. Affinchè il microfono si colleghi correttamente al programma è necessario impostare correttamente i valori delle varibili d'ambiente *AUDIODEV* e *AUDIODRIVER*. La prima variabile corrisponde al dispositivo audio al quale il programma deve fare riferimento, la seconda varibile 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 poter permettere al sistema di utilizzare l'API occorre che il file JSON con le credenziali sia raggiungibile all'interno del sistema e per rendere possibile ci, bisogna creare una variabile d'ambiente con assegnato 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 4.3, 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 4.2, è composta da un'entità che, al ricevimento di un input sulla Touch Board, elabora e riconosce l'input ricevuto e lo inoltra al modulo. Il comando viene validato e, in caso di risposta positiva, viene inoltrato agli altri moduli. 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.

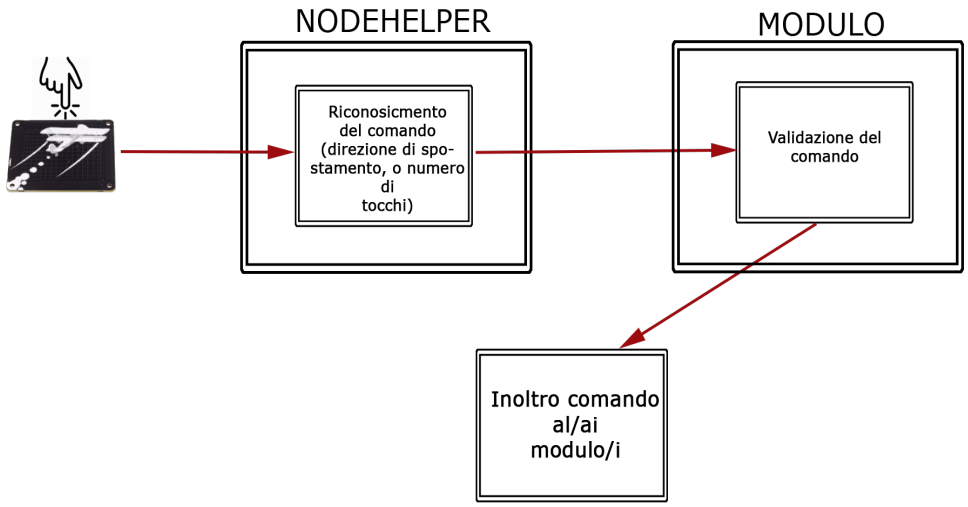

Figura 4.2: Struttura del modulo con la Touch Board

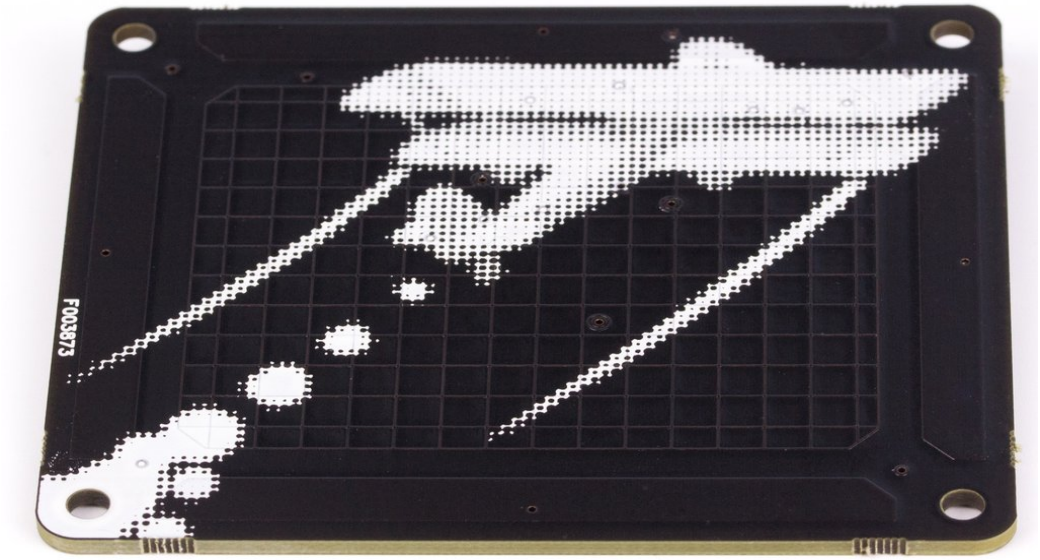

Figura 4.3: Touch Board Skydriver by Piromoni

Capitolo 5

Interfaccia web di amministrazione

In questo capitolo vengono esposti la progettazione e l'implementazione dell'interfaccia web di amministrazione nel MM. Al fine di ospitare la pagina web che contiene il pannello di amministrazione, è necessario creare una nuova classe server oltre a quella già esistente, citata nel sezione 3.1. Il server gira su una porta differente rispetto al primo server, la quale 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 amministrazione

Come già descritto nella sezione precedente il server è implementato con Express, quindi la struttura contiene le due entità (Controller e Modello),

descritte nella sezione 2.3.6, come mostrato in figura 5.1. Il browser fa richiesta di una pagina di un determinato modulo passando un Uniform Resource Identifier(*URI*), che viene ricevuto dal controller principale e inoltrato al modello a cui fa riferimento (in questo caso il modello è uno soltanto). Il modello ricerca in locale il file JSON del modulo richiesto e lo passa al controller, 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 esiste un solo modello e, di conseguenza, un solo controller.

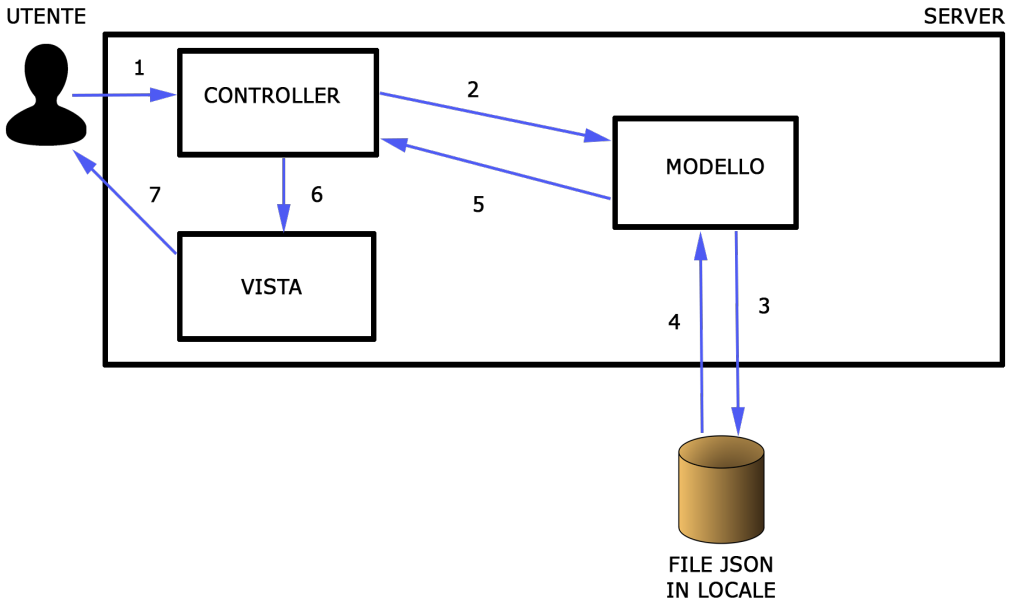

Figura 5.1: Struttura del server dell'interfaccia web

5.2 Individuazione e visualizzazione dei moduli nell'interfaccia

Come già trattato 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. 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 affinchè la modifica venga apportata è richiesto un riavvio del software. Le modifiche effettuate vengono spedite al server che si occupa di salvarle nel file di configurazione principale del MM.

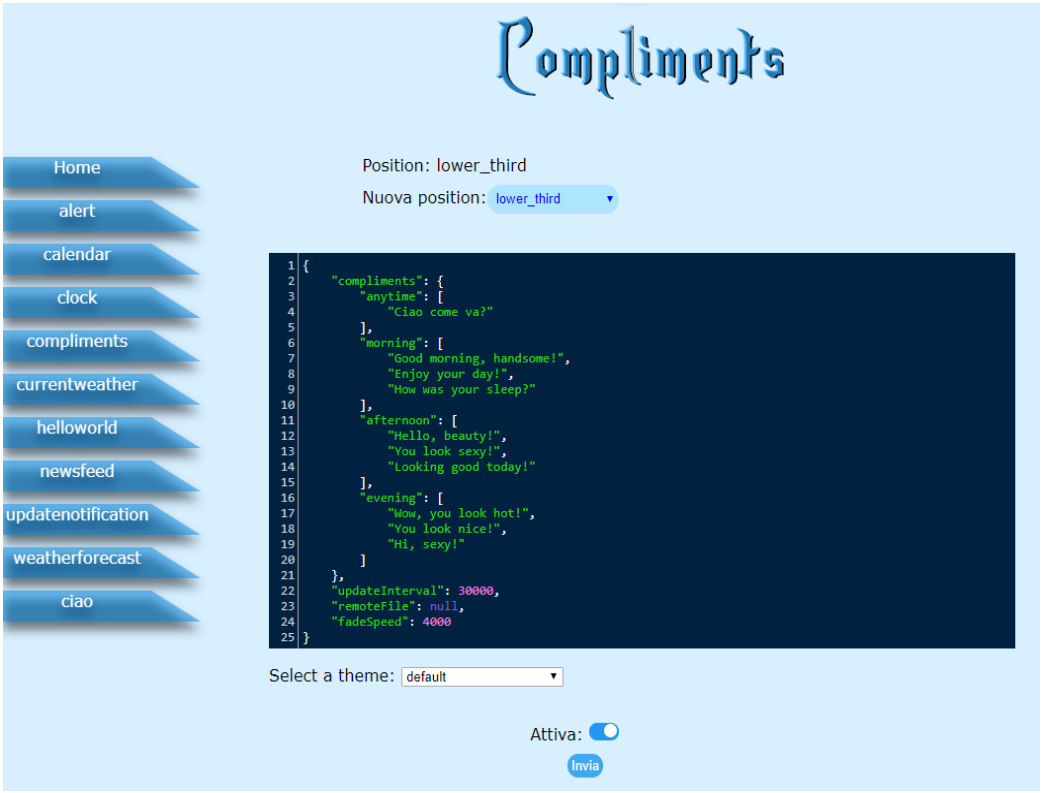

Figura 5.2: Interfaccia web di amminsitrazione

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)

[16] npm docs, https://docs.npmjs.com/getting-started/what-is-npm

[17] Chromium wikipedia, https://it.wikipedia.org/wiki/Chromium

[18] CodeMirror, https://codemirror.net/