Relazione di progetto di "Smart City e Tecnologie Mobili"

DodoHome

Numero del gruppo: 46

Componenti del gruppo: Edoardo Rosa

Indice

1 Introduzione

A partire dal 2014 con l'annuncio di *Amazon Alexa* come assistente personale intelligente e poi nel 2016 con l'annuncio di *Google Assistant* è esplosa l'euforia e la corsa verso lo sviluppo e l'utilizzo dei **home assistant** anche negli ambienti domestici con persone meno avvezze alla tecnologia.
All'inizio dell'era dell'**IoT**, infatti, le tecnologie e gli strumenti a disposizione non ancora del tutto maturi hanno portato alla creazione una nuova "classe" nel mondo informatico, quella dei **Makers**.

Secondo lo scrittore *Cory Doctorow*, autore del romanzo "Makers", sono "people who hack hardware, business-models, and living arrangements to discover ways of staying alive and happy even when the economy is falling down."[7]

Mentre Chris Anderson, imprenditore e direttore di Wired edizione USA dal 2001 al 2012 e autore di "Makers". The New Industrial Revolution, pensa che "the Maker Movement as the web generation meets the real world."[7]

Dale Dougherty, direttore della rivista "Make" e pilastro del movimento Maker, sintetizza in questo modo: "Makers want to hack this world the same way we used to hack computers."[7]

Grazie ai *Makers* lo sviluppo IoT ha aperto una nuova frontiera nel concetto di *computing*, *smart device* e *domotica*. Ciò che prima erano conoscenze esclusive di professionisti del settore, ora, grazie a molti movimenti **Open Source**, **Open Hardware** ed alla presenza di **community** molto ricche, tutti sono in grado di lavorare con queste tecnologie solo avendo alcune basi di programmazione ed elettronica.

L'IoT ha permesso quindi di creare od introdurre potenza computazionale ed intelligenza in ogni device.

Il device simbolo più noto e più utilizzato è il **Raspberry Pi** che a partire dal 2012 (anno di presentazione al pubblico) è stato il re indiscusso del mondo IoT e Makers.

Il *Raspberry* grazie alle sue dimensioni ridotte (da 64mmx54mm per la versione *A+* a 65mmx30mmx5mm per la versione *Zero*) e all'integrazione di Bluetooth e Wi-Fi, per le versioni più recenti, e la presenza di interfacce di I/O ha spostato l'interesse dei sistemi embedded dai microcontrollori alle single-board computer. In questo progetto verrà appunto utilizzato un *Raspberry Pi versione 2 modello B* come coordinatore dell'intero sistema.

Questo progetto vuole esplorare il mondo IoT e dimostrarne l'effetiva *utilità* ed *efficacia* nel mondo reale aiutando l'uomo durante ogni momento della giornata. In particolare il progetto andrà a gestire in maniera ottimali orari e spostamenti tra casa e lavoro; nel dettaglio si è preso in considerazione un lavoratore turnista a causa della frequente variazione di orari di inizio e fine lavoro (sia festivi che feriali).

2 Stato dell'arte

Il progetto si prefissa l'obiettivo di realizzare un servizio studiato per la gestione delle giornate del *turnista*, ovvero, chi lavora in orari e giorni variabili e non costante durante la settimana o il mese.
Questo servizio utilizzerà informazioni relative all'environment per schedulare al meglio i vari impegni, sveglie, spostamenti (in base al traffico o al meteo) sfruttando le API che alcuni servizi terzi mettono a disposizione per creare una aggregazione ad alto livello di questi dati.

Il progetto prenderà il nome di **DodoHome**.
Esempio di utilizzo basilare:

- utilizzo di Google Calendar come scheduler per i vari impegni: ora, luogo, durata;
- il servizio imposterà la sveglia diversi minuti prima dell'impegno per permettere di raggiungere l'appuntamento con il mezzo di trasporto preferito a seconda del traffico, coincidenze, meteo;
- se si preferisce andare a piedi ma è previsto brutto tempo il servizio cambierà mezzo di trasporto ed aggiornerà l'orario della sveglia.

Attualmente non esistono servizi identici a quello presentato con questo progetto ma sia Google[8] che alcuni privati[9] hanno creato soluzioni simili che cercano di trovare una soluzione allo stesso problema affrontato in questo progetto.
La principale differenza tra i servizi sopra citati e quello che verrà realizzato in questo progetto è la dinamicità e la reazione ad eventuali cambiamenti nei dati raccolti; in aggiunta il progetto utilizza altri dati per unificare le informazioni.

L'implementazione consisterà in una interfaccia di configurazione, diversi moduli per l'aggiornamento e il reperimento delle informazioni dai vari servizi terzi che agirà come backend per elaborare ed aggregare le informazioni per aggiornare i dati raccolti per l'utilizzo tramite l'applicazione Android. È prevista anche un'interazione tra utente e backend tramite un monitor di controllo e invio dei comandi tramite gesture per la visualizzazione degli eventi schedulati.

Il progetto terrà in considerazione gli aspetti legati alla *security* ed alla *privacy* dell'utente e garantire una soglia minima di tolerance in caso di disservizio garantendo nel miglior modo possibile il servizio all'utente.

3 Analisi dei requisiti

L'intento del progetto è creare un servizio che si interfacci con l'utente in maniera essenziale e che, tramite l'utilizzo di servizi trasparenti, scheduli gli orari lavorativi dell'utilizzatore.

3.1 Requisiti funzionali dal punto di vista dell'utilizzatore

- Configurazione del servizio una tantum: l'utende dovrà inserire una sola volta le sue preferenze
- Correttezza dei dati presentati.
- Facile e veloce interazione.

3.2 Requisiti funzionali del punto di vista del sistema

- Accesso a servizi di scheduling degli impegni utilizzati dall'utente
- Integrazione con i servizi normalmente utilizzati dagli utenti;
- Richiesta all'utente dei dati necessari per recuperare le informazioni necessarie
- Decisione basata su diversi parametri ritenuti generalmente accettati;
- Basso consumo di energia elettrica e di rete;
- Facile ed veloce interazione.

Il sistema DodoHome, una volta ottenuto l'accesso al calendario, andrà, tramite opportuni percorsi decisionali, ad aggiornare gli orari schedulati per permettere una ottimizzazione nei tempi per il lavoratore/utente in questione.

3.3 Requisiti non funzionali

La realizzazione di un sistema considerato *Smart* e *Assistant* deve prendere in considerazione tutte le problematiche che possono accadere in ambiente domestico o lato utente. Al contrario di progetti o servizi basate su architetture professionali, distribuite, fail safe ed non error prone come posso essere i servizi cloud, di hosting o serverless.

- Aperto: deve supportare la portabilità ed interoperabilità tra i vari utenti utilizzando servizi comuni.
- Modulare: ogni componente dovrebbe essere autonomo e con un certo grado di interoperabilità con il resto del sistema.
- QoS: fornire i servizi con vincoli di tempo, di disponiblità e di affidabilità anche in presenza di malfunzionamenti parziali che possono sempre verificarsi in un normale ambiente domestico. In quanto si tratta di un sistema centralizzato il rischio è molto alto.
- Trasparenza: devono essere mascherati i dettagli e le implementazioni dei servizi utilizzati permettendo una agevole progettazione e programmazione.
- Sicuro: evitare che cattive configurazioni della rete domestica aprano l'infrastruttura ad internet e proteggere il sistema stesso e gli utenti nella stessa rete domestica.

3.4 Casi d'uso

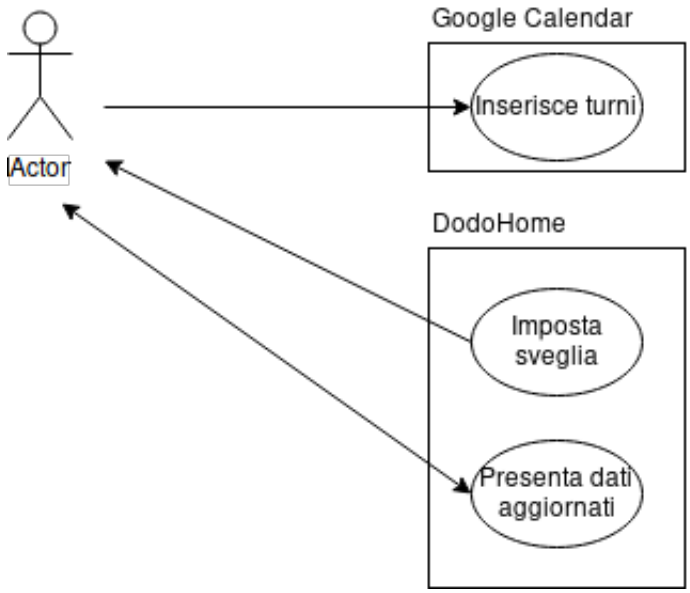


Figura 1: Caso d'uso 1

ID	UC1
ATTORI	Utente
PRECONDIZIONI	L'utente utilizza un servizio di calendario cloud per organizzare i propri impegni di lavoro.
SEQUENZA DEGLI EVENTI	1. Il caso d'uso inizia quando un utente decide di utilizzare DodoHome. 2. L'utente deve dare accesso a DodoHome ai propri calendari 3. DodoHome imposta le sveglie dell'utente.
POSTCONDIZONI	1. L'utente ha ottimizzato i propri tempi/spostamenti.

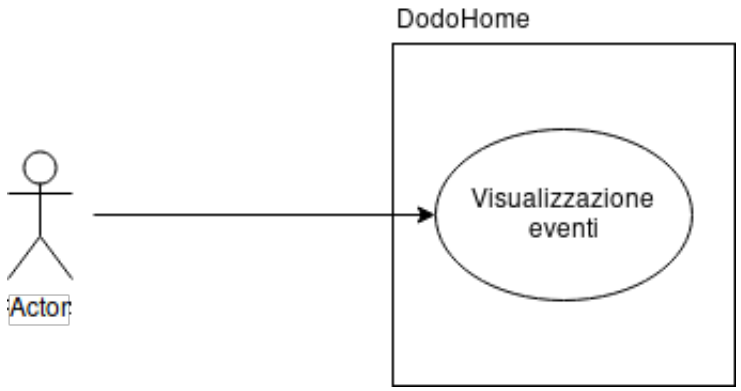


Figura 2: Caso d'uso 2

ID	UC2
ATTORI	Utente
PRECONDIZIONI	L'utente ha configurato DodoHome
SEQUENZA DEGLI EVENTI	1. Il caso d'uso inizia quando un utente decide di visualizzare i propri eventi. 2. L'utente si avvicina al dispositivo. 3. L'utente tramite gli appositi pulsanti sfoglia gli eventi.
POSTCONDIZONI	1. L'utente ha visualizzato i propri eventi schedulati.

4 Progettazione

In questa fase si andrà a definire quanto già enunciato in fase di analisi caratterizzando meglio la presenza e il legame tra le varie necessità di realizzazione evidenziate.

In questo progetto si prende come premessa che l'utente utilizzi **Google Calendar** come sistema di registrazione dei propri impegni lavorativi: un evento corrisponde ad un turno.
Si è scelto di utilizzare *Google Calendar* in quanto è il servizio di calendari maggiormente utilizzato ed alla portata del maggior numero di persone in quanto sia già preinstallato nella maggior parte dei sistemi Android, che rappresentano la quota di mercato maggiore nel mondo degli smartphone, e sia perchè permette una configurazione e facilità d'uso migliori rispetto ai concorrenti (*Outlook*, *ICalendar*, *DigiCal*, *Boomerang*, etc).

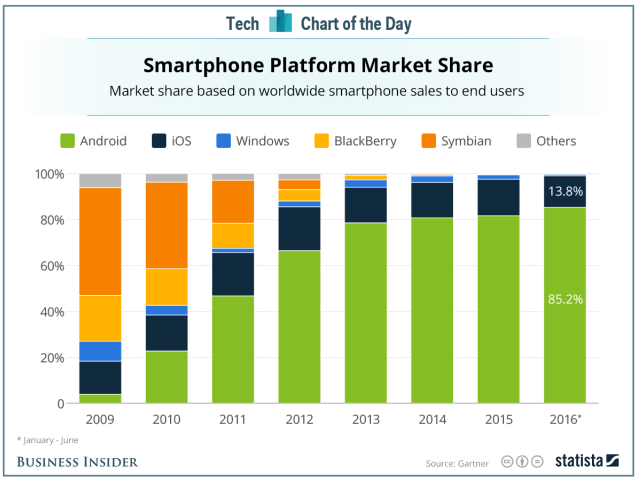


Figura 3: Quote di mercato OS per smartphone[15]

Al fine di ottenere le informazioni necessarie per il raggiungimento dell'obiettivo del progetto è necessario inizializzare il sistema con alcune informazioni di base che devono essere richieste all'utente.

Le informazioni necessarie sono:

- il **calendario** da cui importare gli eventi: è necessario che siano presenti solo gli eventi che si vogliono considerare al fine di evitare di ricevere notifiche per eventi non importanti;
- il **mezzo di trasporto** che si utilizza per andare al lavoro;
- il **mezzo di trasporto** alternativo che si utilizza per andare al lavoro se il primario non può essere disponibile;
- l'**indirizzo** del luogo di lavoro;
- l'**indirizzo** da cui si parte per andare al lavoro.

Le informazioni raccolte andranno poi ad essere integrate con quelle raccolte da altri servizi come **Google Maps** e **OpenWeatherMap** per poter essere elaborate per creare o aggiornare i relativi *reminder* degli eventi di Calendar.
Il lavoro di inserimento ed aggiornamento dovrà essere fatto costantemente durante la giornata così da avere dei dati sempre aggiornati e il più affidabili possibile. In quanto le API di OpenWeatherMap hanno una finestra temporale di previsioni meteo di *3 ore* si è deciso di schedulare la routine di aggiornamento dei reminder ogni 3 ore. In aggiunta alla schedulazione temporale si è deciso di prendere i considerazione i primi 5 eventi del calendario: in questo modo l'utente potrà pianificare ulteriori impegni in vista di quelli che saranno i suoi tempi di lavoro. I reminder verranno poi utilizzati dall'app Android per impostare correttamente le sveglie mattutine o come semplice notifica all'utente nel caso gli impegni non siano mattutini.
L'applicazione Android non è da considerarsi come il solo utilizzo possibile in quanto è possibile utilizzare o implementare altri applicativi con il solo requisito di avere accesso al calendario cloud dell'utente per poter leggere le informazioni inserite periodicamente.

In aggiunta ai reminder impostati su Calendar l'utente potrà avvalersi di un display per visualizzare in maniera sequenziale i propri eventi in arrivo.
Il servizio può essere diviso in più parti:

- parte fisica lato client: componentistica hardware

- parte fisica lato server: unità computazionale
- parte software lato client: gestione componentistica, utilizzo del servizio
- parte software lato server: tasks di elaborazione dei dati

4.1 Progettazione della parte fisica lato client

Dal punto di vista fisico si ha un **Raspberry PI[5] versione 2 modello B**:

- connesso ad una breadboard e a dei componenti hardware attraverso i pin *GPIO*.
- connesso all'alimentazione tramite apposito connettore standard micro-USB

Ogni componente della breadboard ha un proprio impiego e può essere distinto in *sensore* ed *attuatore* secondo la visione di sistema reattivo: è l'ambiente esterno che determina gli eventi che condizionano l'esecuzione del sistema.

L'interazione con l'utente è puramente informativa: una volta attivato il dispositivo tramite il sensore di presenza sarà possibile sfogliare i vari impegni schedulati tramite il calendario.
È quindi necessario utilizzare dei componenti di rilevamento della presenza, dei componenti per l'input e dei componenti per la visualizzazione dei dati.

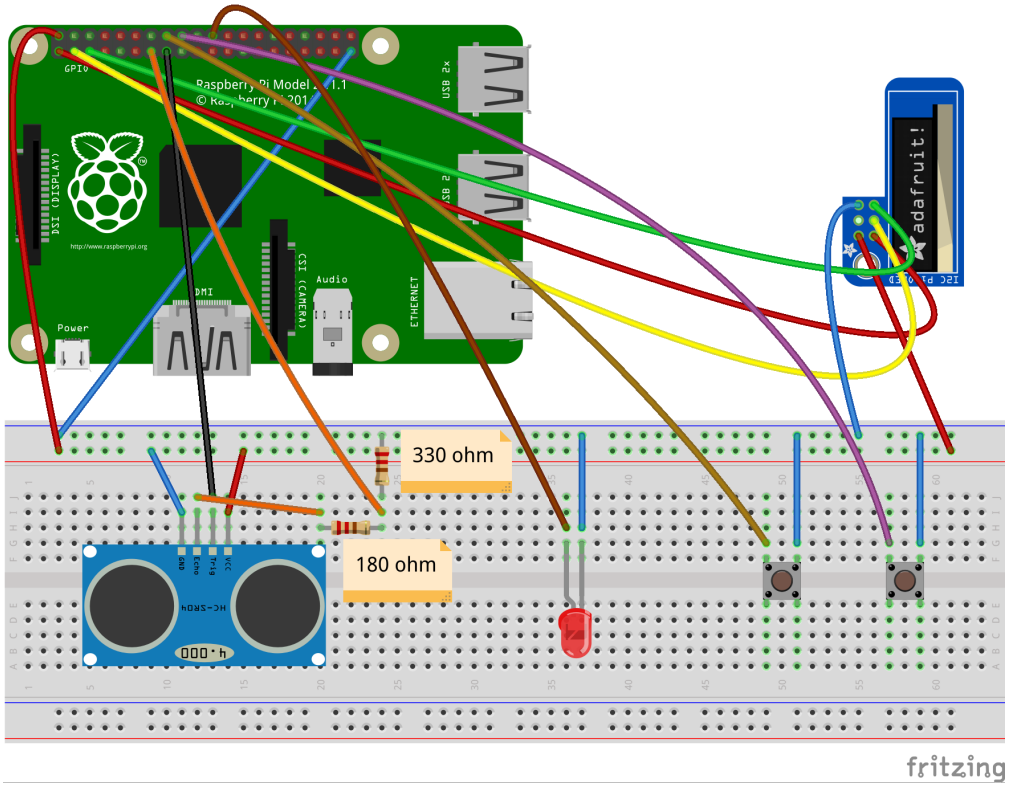


Figura 4: Sketch utilizzo breadboard e componenti - fritzing

Nel dettaglio sono stati utilizzati i seguenti componenti:

- Adafruit SSD1306 display
- HC-SR04
- Led
- Button
- Resistenze: 180 Ω e 330 Ω

4.2 Progettazione della parte logica lato client

Nella parte software dedicata all'interazione con l'utente tramite la componentistica hardware si è replicato il funzionamento di una *macchina a stati finiti* (FSM) che esegue dei task a seconda dell'input; mentre, per l'utilizzo finale del servizio,

si è deciso di implementare un'applicazione *Android* per l'impostazione dei promemoria.

Per la parte di utilizzo della componentistica è stata utilizzata una macchina a stati finiti (o automa a stati finiti) in quanto è il modello discreto più utilizzato e semplice per la progettazione di sistemi embedded.
Ogni FSM opera in una sequenza di passi (discreta) e la sua dinamica è caratterizzata da sequenze di eventi (discreti).

La decomposizione in task è un principio di progettazione molto importante che permette di rendere modulare il sistema:

- ogni modulo è rappresentato da un task (compito da eseguire);
- un task può essere decomposto in sotto-task in modo ricorsivo o un task complesso può essere definito come composizione di sotto-task più semplici.

I vantaggi ricavati sono:

- separtion of concerns;
- l'utilizzo di singoli moduli per una migliore comprensibilità del comportamento;
- debugging semplificato;
- supporto alla modificabilità, all'estensione ed al riuso del codice prodotto.

L'applicazione Android andrà a richiedere al servizio calendar gli eventi che schedulati dell'utente ed imposterà le opportune sveglie al fine di ottimizzare le tempistiche dell'utente.

4.3 Progettazione della parte fisica lato server

Dal punto di vista fisico si utilizza lo stesso dispositivo della parte client(**Raspberry Pi versione 2 modello B**) in quanto dal punto di vista progettuale risulta essere un device con bassi consumi ma prestazioni interessanti per una board di ridotte

dimensioni.

Il *Raspberry Pi* è un computer single-board sviluppato dalla **Raspberry Pi Foundation**. La board è *open hardware* (ad eccezione della *GPU*) ed è la piattaforma piu utilizzata in ambito makers e IoT e presenta le seguenti caratteristiche per la versione utilizzata in questo progetto:

- **CPU**: 900MHz quad-core ARM Cortex-A7
- **RAM**: 1GB RAM
- **GPU** :VideoCore IV 3D graphics core
- 4 porte USB
- 40 pin GPIO
- porta HDMI
- porta Ethernet (Mb/s)
- jack audo 3.5mm
- interfaccia per camera (CSI)
- interfaccia per display (DSI)
- slot Micro SD card

Il dispositivo è collegato in rete e funge da server per l'interfacciamento con l'utente per la configurazione iniziale e per lo scambio di informazioni con il servizio di calendario cloud.
Al fine di collegare in rete il server si è aggiunto un modulo WiFi (*TP-Link TLWN722N*) tramite una porta USB.

4.4 Progettazione della parte logica lato server

Per permettere all'utente la configurazione iniziale il server mette a disposizione una interfaccia web in cui tramite pochi e semplici passi viene effettuato il pairing tra il calendario cloud e il servizio offerto da DodoHome.
In aggiunta all'interazione con l'utente è necessario realizzare la logica core del servizio:

- Recupero dei dati dal calendario personale dell'utente
- Recupero dei dati riguardanti le preferenze dell'utente
- Recupero dei dati relativi a: location, orari, mezzi di trasporto, meteo, percorsi per gli eventi
- Elaborazione dei dati per creazione del prodotto finale
- Aggiornamento dei dati del calendario personale dell'utente con le informazioni raccolte

La routine è impostata per eseguire ciclicamente durante la giornata ogni 3 ore indipendentemente dall'interazione con l'utente ed in maniera transparente: l'utente si vedrà aggiornare i propri eventi con le nuove informazioni reperite dal sistema.
Per garantire massime performance e sicurezza si è deciso di utilizzare una distribuzione Linux come OS di base per la board, nello specifico ArchLinux nella versione per ARM, e le seguenti tecnologie opportunamente configurate:

- **NGINX**[10] come reverse proxy per il web server
- **uWSGI**[11] come web server
- **Redis**[13] come storage dei dati
- **ufw** come interfaccia di gestione per iptables

I parametri con cui sono stati scelti questi applicativi sono: consumo di risorse, efficacia e facilità di utilizzo.

5 Implementazione

In questa sezione si andranno a discutere le scelte e le soluzioni adottate durante l'implementazione del progetto. Inoltre, verranno presentate le librerie ed i framework utilizzati.
Importante è sottolineare che, per scelta etica e progettuale, si è preferito utilizzare librerie non solo *free* ma *open source* per poter andare ad agire puntualmente su eventuali problemi di compatibilità o bug osservati durante l'attività di implementazione; nonchè il contenimento dei costi di progetto.
La presenza di molte librerie open sviluppate sia da grosse aziende che da privati conferma la crescita e l'interesse verso il mondo dell'automazione a livello domestico o personale descritta nel capitolo 1.

Sono stati implementati i seguenti moduli e applicazioni:

- configurazione tramite interfaccia web
- gestione della componentistica ed interazione utente
- recupero ed aggregazione dei dati dalle diverse API per ottimizzazione tempistiche
- applicazione Android per impostazione delle sveglie

L'implementazione del software eseguito sul Raspberry è stato realizzato interamente in **Python** (tre dei moduli sopra elencati). I principali vantaggi del linguaggio sono: l'elevata *astrazione* che offre, la capacità di supportare diversi *paradigmi di programmazione* e alla varietà di *librerie* sviluppate per la gestione sia dei componenti hardware sia delle API utilizzate
.
Nel dettaglio sono state utilizzate le seguenti librerie e framework:

- **Flask**[12] come framework per sviluppare il backend dell'interfaccia web di configurazione;
- **jinja2**[24]: non una libreria o un framework ma un motore di template per applicazioni web;

- **materialize**[25]: framework per front-end;
- **google-auth**[16]: libreria per il meccanismo di autenticazione e accesso alle API Google;
- **googlemaps**[17]: libreria per l'utilizzo delle API di Google Maps;
- **requests-oauthlib**[18]: libreria basata su requests[19] per il supporto alle API con utilizzo OAuth;
- **redis**[20]: libreria per l'interfacciamento con Redis[13];
- **simplejson**[21]: libreria per codifica e decodifica di oggetti e stringhe JSON;
- **wiringpi**[22]: libreria python wrapper della libreria scritta in C per l'accesso ai PIN GPIO;
- **luma.oled**[23]: libreria per utilizzo del display OLED SSD1306;
- **pyowm**[26]: libreria per accesso ai dati messi a disposizione dalle API OpenWeatherMap.

Non sono state elencate le librerie richieste come dipendenze dalla librerie sopra elencate in quanto vengono utilizzate in maniera trasparente e non sono interessanti ai fini del documento stilato.

Le librerie core del progetto sono quelle per l'utilizzo delle API di Google e di OpenWeatherMap; per le API di Google sono state utilizzate le librerie ufficiali messe a disposizione da Google tramite GitHub mentre per *pyowm* si tratta di un wrapper scritto in Python per le API free di OpenWeatherMap.
Nonostante entrambe siano largarmente utilizzate e supportate hanno una scarsa documentazione ufficiale; in aggiunta le informazioni riportate dalla documentazione Google tramite il portale 'developers.google.com' si sono rilevate incomplete od errate.

5.1 Applicazione web

Durante il primo avvio all'utente sarà richiesto di accedere all'interfaccia web esposta da *DodoHome* per procedere alla configurazione iniziale.

Come già discusso il backend è stato realizzato tramite l'utilizzo del framework *Flask* sia per la gestione delle richieste che del contenuto da mostrare lato utente grazie al motore di template Jinja2. In aggiunta, al fine di rendere più fluida e piacevole la navigazione, sono state implementate alcune funzionalità tramite Javascript.

All'utente viene richiesto di inserire:

- il calendario da cui importare gli eventi: è necessario che siano presenti solo gli eventi che si vogliono considerare al fine di evitare di ricevere notifiche per eventi non importanti;
- il mezzo di trasporto che si utilizza per andare al lavoro;
- il mezzo di trasporto alternativo che si utilizza per andare al lavoro se il primario non può essere disponibile;
- l'indirizzo del luogo di lavoro trami apposita mappa e barra di ricerca
- l'indirizzo da cui si parte per andare al lavoro tramite apposita mappa e barra di ricerca.

Al fine di poter utilizzare le API messe a disposizione da Google e richiedere l'accesso al calendario dell'utente è stato necessario attivare gratuitamente le API per sviluppatori e richiedere le chiavi di utilizzo del sistema **OAuth** per applicazioni private.
OAuth è un protocollo open che viene utilizzato da Google per autenticazione e autorizzazione dei proprio servizi senza richiesta o storage delle credenziali degli utenti che si autenticano al servizio, in questo caso l'applicazione web o l'applicazione Android.
Una volta ottenuto il token *OAuth* è possibile accedere ai servizi Google come se fosse l'utente stesso ad accedere. A questo proposito infatti va dichiarato che gli unici privilegi che sono richiesti da DodoHome sono in lettura e scrittura esclusivamente alla piattaforma Calendar.

Il token *OAuth* ottenuto è salvato, assieme alle informazioni ottenute durante il processo di configurazione, per l'utilizzo da parte degli altri moduli, in *Redis* op-

portunamente configurato per effettuare persistenza dei dati.

5.2 Gestione componentistica

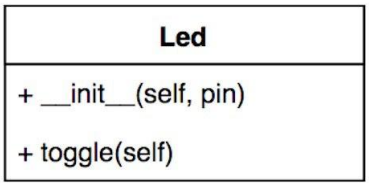
La parte di interazione fisica con il sistema, ovvero la visualizzazione degli eventi schedulati tramite il display OLED Adafruit, è stata realizzata in tasks implementando le classi: *Button*, *Led*, *Radar*, *OLed* e *Session*.

5.2.1 Button

La classe *Button* astrae l'interfacciamento con i pin di GPIO permettendo la creazione di un demone in ascolto di eventi così da permettere all'applicazione principale di agire alla pressione del tasto.
I PIN di utilizzo dei pulsanti sono stati configurati come *pull-up* in modo da sfruttare la resistenza interna dei GPIO ottenendo:

- dei campionamenti con un rumori ridotti;
- floating del button quasi annullato.

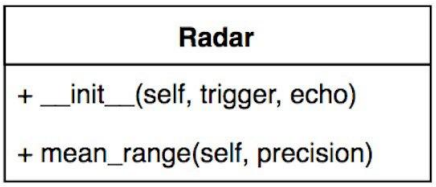
5.2.2 Led



La classe *Led* gestisce l'accensione e lo spegnimento del LED a 2 pin (controllo e massa) connesso al Raspberry come indicazione dell'attività del dispositivo.

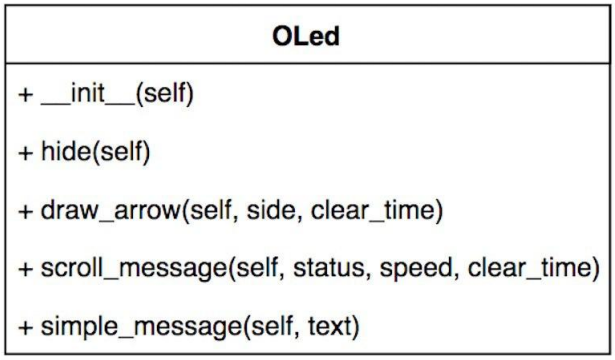
5.2.3 Radar

Il componente *HC-SR04* viene utilizzato come sensore di presenza: se l'utente è vicino al dispositivo allora è necessario attivare i pulsanti e il display. In questo



modo si ha anche un risparmio di energia attivando i componenti solo quando necessario. La classe *Radar* campiona in quasi real-time e se rileva, tramite threshold, la presenza di un ostacolo presente davanti al sensore attiva il resto del sistema.
Si è notato che l'utilizzo della libreria *wiringpi* rende più reattivo il campionamento ma non è stato possibile creare un sistema real-time che possa essere usato per l'implementazione di gesture tramite movimenti della mano davanti al sensore: per questo si è optato per un sistema più semplice utilizzando due pulsanti per la navigazione.

5.2.4 OLed

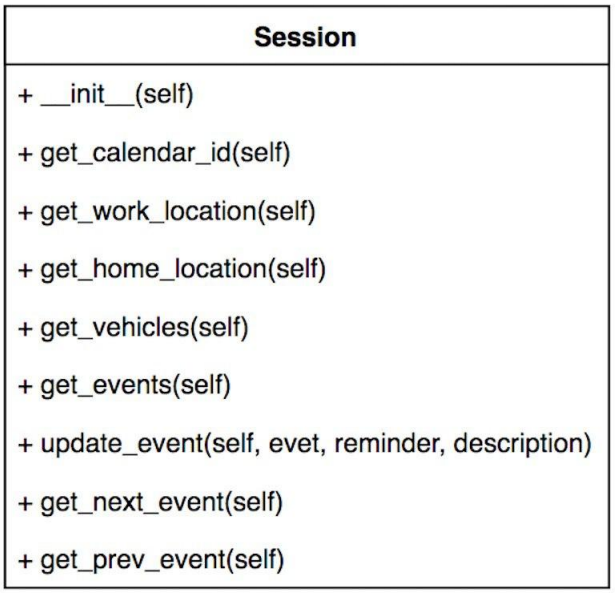


Il display *Adafruit SSD1306* utilizza i PIN I2C (Integrated Circuit) del Raspberry. La comunicazione seriale I2C risulta essere più semplice rispetto a quella SPI (Serial Peripheral Interface) ma al tempo stesso ha un throughput più basso e consumi più alti. Il problema dei consumi viene assorbito dalla tecnologia OLED in quanto i pixel sono accesi solo se sono utilizzati.

La classe implementata permette la visualizzazione del testo sullo schermo sia tramite scorrimento orizzontale sia come testo fisso. In aggiunta grazie alla libreria

messa a disposizione è possibile utilizzare diversi font e quindi poter anche mostrare simboli non ASCII.

5.2.5 Session



La classe *Session* viene utilizzata per reperire le informazioni inserite durante la fase di configurazione e quindi, tramite il token OAuth, leggere ed aggiornare i dati degli eventi presenti su Calendar.
Questa classe viene utilizzata sia per visualizzare gli eventi sul display OLED sia dallo script per l'ottimizzazione delle tempistiche orarie.

5.3 Recupero ed aggregazione dei dati

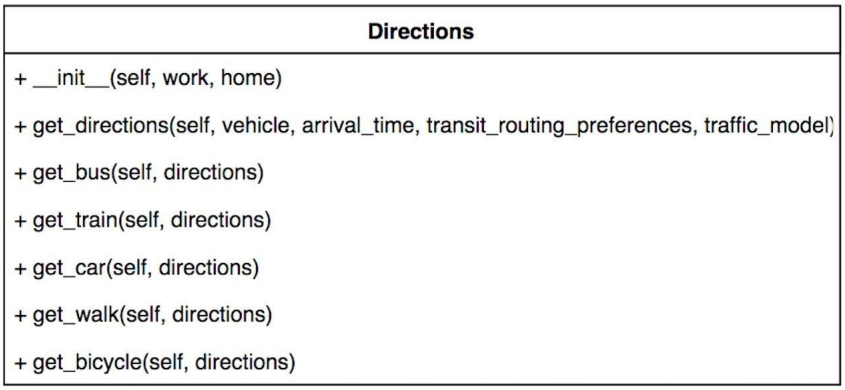
Il task per il recupero dei dati da Calendar, da Maps e da OpenWeatherMap è stato implementanto come una routine (*update_events.py*) che va ad aggiornare le informazioni dei 5 eventi futuri più recenti.

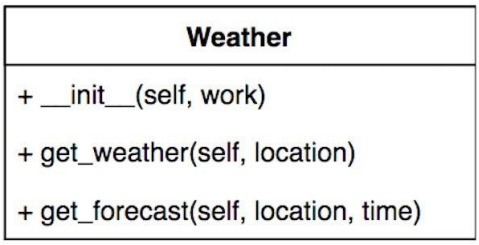
Il flow del task è riassumibile nei seguenti punti:

- recupero dei dati dei 5 eventi da considerare;

- inizializzazione della classe *Directions*;
- per ogni evento recuperare le informazioni riguardanti il meteo (classe *Weather*):
  - previsione meteo nella località associata al punto di partenza al momento della partenza: *Home*;
  - previsione meteo nella località associata al punto di arrivo al momento dell'inizio del turno di lavoro: *Work*;
  - previsione meteo nella località associata a *Work* al momento di fine turno;
  - previsione meteo nella località aossicata a *Home* al momento di fine turno;
- valutazione delle previsioni meteo in base alla tipologia di veicolo scelto;
- se la valutazione risulta negativa (cattivo tempo) allora si procede al ricalcolo dei tempi od alla sostituzione del mezzo di trasporto da primario a secondario e successivo ricalcolo delle tempistiche di traffico e tratta;
- se la valutazione risulta positiva (meteo favorevole) allora si procede al calcolo delle tempistiche di traffico e tratta per il veicolo scelto come principale;
- aggiornamento o inserimento del reminder per gli eventi elaborati;
- aggiornamento della descrizione degli eventi con le informazioni elaborate.

La possibilità di errori o la mancanza di connessione tra Raspberry e API potrebbe comportare notevoli disagi per l'utente; per questo è stato previsto che durante la fase di configurazione venga impostato un default reminder che permetta comunque all'utente di arrivare in tempo al lavoro.





La routine si avvale della funzione *find_optimal* per implementare il percorso decisionale sull'utilizzo dei mezzi di trasporto e che, tramite la classe *Directions*, recupera i dati da Google Maps per il cacolo dei tempi di percorrenza.

```
def find_optimal(event, primary, secondary, work, home, directions
                                    ):
  PADDING = timedelta(minutes=65)  # UTC-1 plus 5 minutes of bonus
  start_date = parse(event.get("start").get("dateTime"))
  if is_bad_weather(primary, work, home, event):
    # If bad weather: traffic model is pessimistic
    if primary == "driving":
      d, url = directions.get_directions(
        primary, start_date - PADDING, traffic_model="pessimistic"
                                    )
    # If bad weather: use less_walking parameter
    if primary == "bus" or primary == "train":
      d, url = directions.get_directions(
        primary,
        start_date - PADDING,
        transit_routing_preference="less_walking")
    # If bad weather: switch to the second vehicle choice
    if primary == "male" or primary == "bicycle":
      if is_bad_weather(secondary, work, home, event):
        # Get data for the second choice
        if secondary == "bus" or secondary == "train":
          d, url = directions.get_directions(
            secondary,
            start_date - PADDING,
            transit_routing_preference="less_walking")
        else:
          d, url = directions.get_directions(
            secondary,
            start_date - PADDING,
            traffic_model="pessimistic")
      else:
        d, url = directions.get_directions(secondary,
          start_date - PADDING)
  else:
    # Weather is good!
    d, url = directions.get_directions(primary, start_date -
                                    PADDING)
  return d, url
```

5.4 Sveglia (Applicazione Android)

L'applicazione sviluppata per piattaforma Android una volta richiesto l'accesso in lettura del calendario dell'utente va recuperare e a settare la sveglia per il successivo evento in programma.
Si è scelto di prendere in cosiderazione un singolo evento in quanto è possibile che nel corso di valutazioni successive, da parte del task di aggregazione, i riferimenti agli orari subiscano dei cambiamenti.

6 Privacy e sicurezza

Il trend imposto dal mercato sulla produzione di dispositivi per IoT sempre più performanti, economici e dalle dimensioni ridotte spesso porta a non valutare adeguatamente la progettazione di hardware e software che tengono conto della sicurezza di questi sistemi.
I primi attacchi mirati a queste tecnologie hanno avuto facilità di esecuzione ed una rapidissima diffusione [1].
In questo scenario, dalla parte dei produttori hardware si registra una *mancanza* o addirittura una *resistenza* per quanto riguarda la correzione di falle o il rilascio di aggiornamenti firmware. Questo potrebbe essere dovuto all'utilizzo di codice proprietario o la totale mancanza di supporto per il dispositivo che si utilizza. In alcuni casi si potrebbe parlare di **obsolescenza programmata**[2].
Anche da parte degli utilizzatori finali dei sistemi non si evince una particolare attenzione ai problemi relativi alla sicurezza; probabilmente a causa delle scarse conoscenze del sistema in uso (perchè complesso o non studiato approfonditamente) e delle tecnologie usate. È ormai tristemente noto che lo sviluppo di questi dispositivi perennemente connessi, non prenda quasi mai in considerazione le problematiche relative alla sicurezza derivata dalla connessione a sistemi più *fragili* o legate all'interazione con altri device (perdita di privacy, prestazioni o utilizzo improprio di risorse di rete).

[1] Breve lista di attacchi a sistemi IoT nel 2015 http://tinyurl.com/honlko2.
[2] Obsolescenza programmata - https://it.wikipedia.org/wiki/Obsolescenza_programmata.

DodoHome è stato sviluppato per garantire sicurezza sia all'utente che al sistema stesso.
In questo capitolo verranno presentate alcune delle soluzioni adottate e delle tecnologie utilizzate per l'implementazione dell'applicativo lato server e dei collegamenti di reti.

6.1 Sistema Operativo

Il sistema operativo scelto per il Raspberry PI è una versione *ARM a 32-bit* della distribuzione Linux **Arch Linux**.

6.1.1 Arch Linux ARM

- è *bleeding edge* per quanto riguarda l'upstream degli aggiornamenti (sempre aggiornata all'ultima versione dei software disponibile);
- generalmente considerata sicura;
- fornisce libertà maggiore per la configurazione del sistema;
- una forte e grande community.

6.1.2 Software orientato alla sicurezza

- **SSH** aggiornato all'ultima vesione con supporto ai cipher piu sicuri;
- firewall **UFW**, per rendere disponibili granularmente i servizi essenziali;
- **NGINX** configurazione per rispondere al meglio ad ogni volume o tipo di richieste.

Oltre all'installazione ed alla configurazione degli applicativi software sono stati configurati diversi parametri per l'hardening del kernel tramite sysctl ed un sistema di loggin per la registrazione degli accessi.

6.2 Applicazione web

L'applicazione web resa disponibile all'utente è il primo punto di ingresso che un potenziale attaccante potrebbe sfruttare per portare un attacco al sistema e per questo sono state seguite le indicazioni del *W3C* e del *OWASP* per una corretta e sicura implementazione dell'applicativo sia back-end che front-end.

In quanto il token di *OAuth* risulta essere presente in *Redis*, è necessario che solo il software di DodoHome ne abbia l'accesso; per questo, infatti, anche in caso di mal configurazione della rete domestica il firewall prevede alla protezione del sistema. L'applicativo web risulta disponibile solo in LAN mentre redis non è accessibile al di fuori del *localhost*.

7 Testing e performance

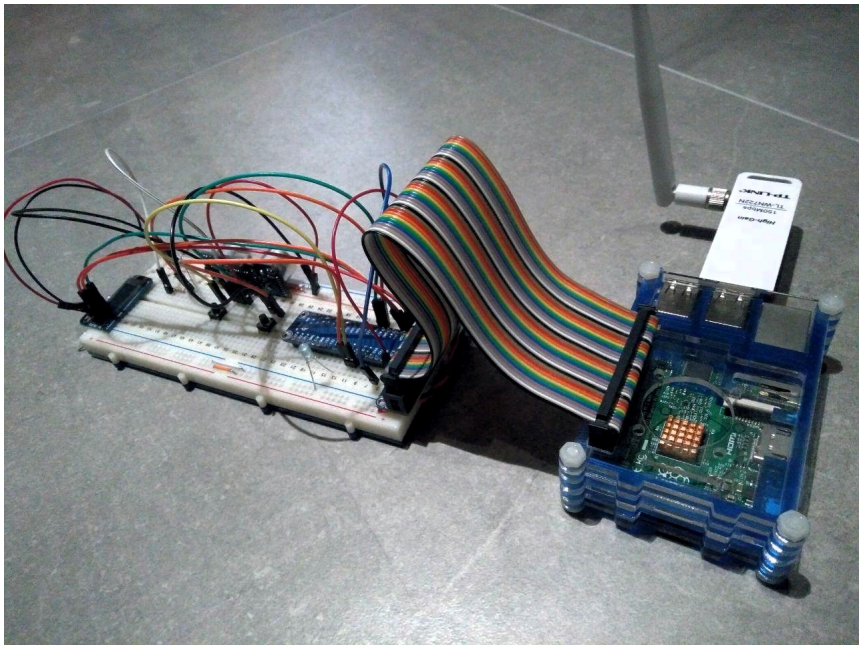


Figura 5: Reale implementazione

L'inserimento di un evento su calendar può essere fatto in qualsiasi momento in quanto la routine di valutazione ed aggiornamento esegue periodicamente ogni 3 ore.

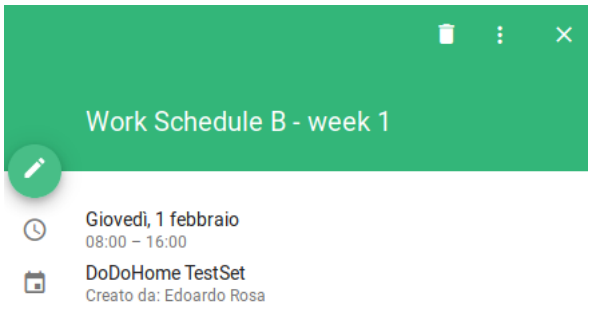


Figura 6: Esempio di evento base

Sono stati effettuati i test sulle performance della routine principale del progetto (update_events.py) e sui singoli sotto-task implementati: i test considerano i tempi di esecuzione, in secondi, di ogni funzionalità e sono stati eseguiti su 10 run della routine per l'aggiornamento di 5 eventi.
La routine in questione è stata eseguita dal Raspberry PI con le caratteristiche elencate al capitolo 4.3.

Funzione	**Secondi**
Get Events	0.68
Get Directions	0.67
Find optimal	1.18
Get Weather	0.12
Update Event	0.68
Global	12.34

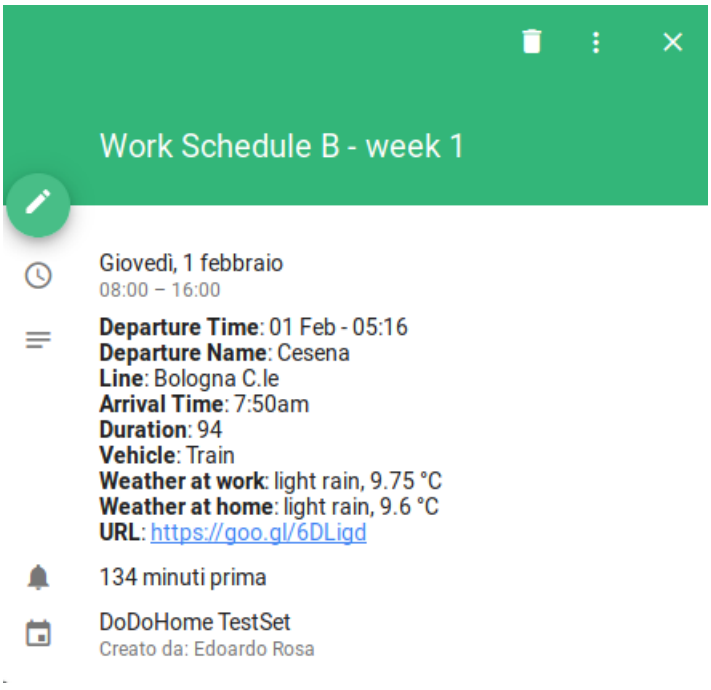


Figura 7: Esempio di evento aggiornato

Il web server (*NGINX + uWSGI*) è stato testato tramite il tool *wrk* con le seguenti modalità:

- 8 threads;
- keepalive per connessioni con status 200
- tempo di esecuzione di 60s

Al momento del test e subito dopo non sono stati rilevati cali visibili di prestazioni o indisponibilità dei servizi esposti.

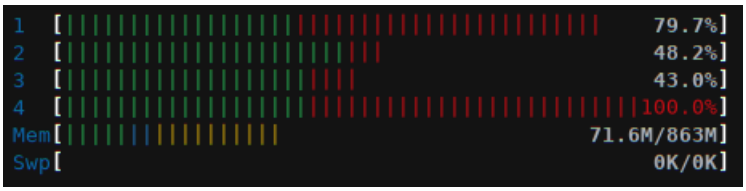


Figura 8: Carico di lavoro del Raspberry durante il test

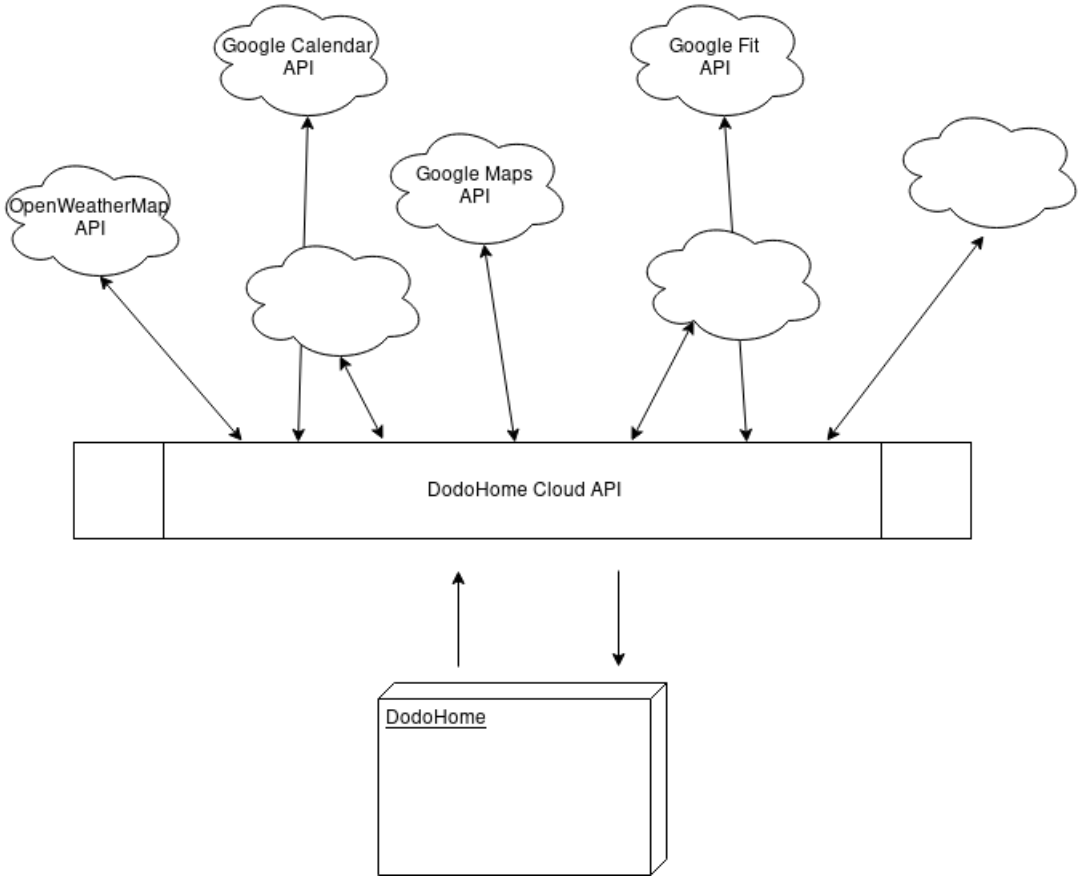
8 Analisi di deployment su larga scala

Prima di un deploy o produzione del progetto su larga scala è necessario procedere ad un tuning sia a livello hardware che software.

Per il parco software sarebbe opportuno effettuare un tuning dei parametri di valutazione e considerare l'introduzione di ulteriori possibilità di configurazione. La possibilità di introduzione di un motore per il profiling dell'utente potrebbe essere utile a migliorare il prodotto e creare in maniera trasparente un servizio più personalizzato ed efficace.
Il massiccio utilizzo di API e continuo accesso ad Internet rendono il progetto altamente vulnerabile in ambienti domestici in cui cadute di corrente o di connessione possono essere frequenti

L'hardware utilizzato in questo progetto si è rilevato leggermente sotto-performante in quanto non è stato possibile creare un sistema di gesture (molto accattivante per l'utente) e, in vista di ulteriori sviluppi ed aggiunta di funzionalità, potrebbe essere necessario l'utilizzo di una board più efficiente.

Una volta eseguiti questi opportuni potenziamenti è necessario introdurre ulteriori fonti da cui recuperare tutte le informazioni che sono ritenute necessarie al fine di migliorare notevolmente il comfort dell'utente: acquisto di biglietti per i mezzi pubblici o per parcheggi a pagamento, accesso ad API per il meteo più complete, utilizzo delle API Premium di Google, integrazione con sistemi di monitoraggio del sonno.
L'introduzione di tutte queste funzionalità deve prevedere un sistema di aggiornamento del software presente nei device commercializzati e una infrastruttura ad hoc cloud che venga utilizzata come middleware tra le API usate e DodoHome.



9 Conclusioni

Il progetto si mostra essere un piccolo esempio del concetto di Smart Home/Personal Assistant, in cui, in maniera non invasiva, si offre all'utente la possibilità di evitare di ripetere quotidianamente gli stessi task ed avere più tempo libero o più ore di sonno.
Benchè molto semplice come interazione e risultato, DodoHome è stato creato e studiato per problemi reali che affliggono o possono affliggere i lavoratori.

Il progetto potrebbe senz'altro essere portato avanti e migliorato (con maggior interesse iniziale l'applicazione Android per la Sveglia) e integrato con nuove feature derivate dalle esigenze nate durante il normale utilizzo quotidiano.

Riferimenti bibliografici

[1] https://it.wikipedia.org/wiki/Amazon_Alexa

[2] https://home-assistant.io

[3] http://internetofthingsagenda.techtarget.com

[4] https://wikipedia.org/wiki/Raspberry_Pi

[5] https://www.raspberrypi.org/

[6] http://ieeexplore.ieee.org/document/7389283/

[7] http://it.openmaker.eu/2017/09/05/movimento-maker/

[8] https://support.google.com/googlehome/answer/7029585?hl=en

[9] https://www.xda-developers.com/tasker-pro-calendar-based-alarm/

[10] https://nginx.org/

[11] https://uwsgi-docs.readthedocs.io/en/latest/

[12] http://flask.pocoo.org/

[13] https://redis.io/

[14] https://www.python.org/

[15] http://www.businessinsider.com

[16] https://github.com/GoogleCloudPlatform/google-auth-library-python

[17] https://github.com/googlemaps/google-maps-services-python

[18] https://github.com/requests/requests-oauthlib

[19] https://github.com/requests/requests

[20] https://github.com/andymccurdy/redis-py

[21] https://github.com/simplejson/simplejson

[22] https://github.com/WiringPi/WiringPi-Python

[23] https://github.com/rm-hull/luma.oled

[24] https://github.com/pallets/jinja

[25] http://materializecss.com/

[26] https://github.com/csparpa/pyowm

[27] https://github.com/wg/wrk