UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI **INGEGNERIA**

Corso di Laurea triennale **in ingegneria informatica e dell'automazione**

Sviluppo di un sistema di monitoraggio non invasivo delle abitudini di persone anziane

Development of a non-invasive monitoring system of habits of elderly people

Relatore:

Chiar.mo Prof. Adriano Mancini

Tesi di:

Mariusz Gabriel Wiazowski

A.A. **2018 / 2019**

Sviluppo di un sistema di monitoraggio non invasivo delle abitudini di persone anziane

Capitolo 1
INTRODUZIONE

Secondo il più recente rapporto annuale dell'Istituto Nazionale di Statistica (2019)[1], la vita media degli uomini è di 80,8 anni e quella delle donne di 85,2. Rispetto agli altri anni le aspettative di vita sono cresciute e risulta evidente che il paese sta invecchiando sempre di più, mentre le nascite continuano a calare. Tutto questo fa si che siamo di fronte ad un calo demografico, che succede quando il tasso di mortalità è superiore a quello di natalità. Quasi un terzo della popolazione italiana ha un'età superiore ai 60 anni[2]. Ed è dimostrato che a partire da questa età l'essere umano diventa più fragile e le condizioni di salute diventano via via più precarie. In aggiunta, noto il fatto che le donne vivono di più rispetto agli uomini, gli anziani spesso vivono da soli perché divorziati o vedovi.

D'altra parte, un altro dato statistico molto importante che riguarda sempre gli anziani è il numero degli infortuni domestici. Le persone più coinvolte da questo tipo di incidenti sono infatti le casalinghe e gli anziani, mentre il luogo dove questi incidenti si verificano più frequentemente è la cucina. Secondo l'ISTAT il numero degli incidenti avvenuti in casa è in crescita, a dimostrazione di questo fatto l'INAIL, l'Istituto nazionale Assicurazioni Infortuni sul Lavoro, da dicembre del 2019 propone ai propri iscritti una polizza assicurativa contro gli infortuni domestici[3].

L'Italia, secondo GSMA Intelligence[4], è il terzo paese al mondo per numero di smartphone, ben 83% degli italiani ne possiede uno. In una vita sempre più frenetica, essere sempre connessi con la propria famiglia è un'esigenza sentita. Quando sono chilometri a dividere membri della stessa famiglia, diventa difficile tenere sotto controllo la saluta dei componenti più anziani. Ovviamente, nei casi più gravi, queste persone richiedono assistenza specifica e continuativa, quindi un semplice monitoraggio non è sufficiente perché è richiesto l'intervento umano. Possono essere direttamente i figli a prendersi cura di loro oppure infermieri o persone qualificate per l'assistenza agli anziani. In alternativa esistono strutture specifiche come le case di riposo. Da un'analisi della Unione europea delle cooperative Uecoop[5] è emerso che il numero delle case di riposo in Italia è in forte crescita, in particolare nel settore privato ci è stato un aumento del 40% dei posti letto negli ultimi dieci anni, anche se la famiglia resta il principale ambito di vita degli anziani.

Esiste però una categoria di persone over 60 che non richiede assistenza continua, ma andrebbe monitorata costantemente per prevenire l'insorgere di patologie o complicazioni dovute all'avanzamento dell'età, ma anche per offrire una certa tranquillità ai propri cari sullo stato di salute della persona monitorata.

A titolo d'esempio, immaginiamo che tra i componenti di una famiglia vi sia una persona anziana che vive da sola. Tale persona recentemente si è rotta il femore ma il peggio è già passato ed è stata dimessa dall'ospedale per tornare a casa propria. In questo preciso esempio l'anziano non richiede assistenza particolare perché fondamentalmente sta bene, è autonomo e vuole essere

---

[1] https://www.istat.it/it/archivio/230897 - Rapporto annuale dell'ISTAT

[2] Informazioni prese dal sito https://www.tuttitalia.it/statistiche/popolazione-eta-sesso-stato-civile-2019/

[3] https://www.inail.it/cs/internet/attivita/assicurazione/assicurazione-infortuni-domestici.html - Polizza contro gli infortuni domestici di INAIL

[4] https://www.gsmaintelligence.com/markets/1687/dashboard/ - Dati di GSMA Intelligence sul mercato del mobile in Italia

[5] http://www.vita.it/it/article/2018/09/06/istat-uecoop-case-di-riposo-boom-dei-privati-40/148924/ - Articolo sull'indagine di Uecoop

indipendente. Però l'esperienza passata insegna che prevenire è meglio che curare, e a tal proposito la tecnologia può rappresentare una soluzione valida. Con il termine *Ambient Assisted Living*, abbreviato con AAL, si intende l'uso di nuove tecnologie nell'ambiente domestico per rispondere alle sfide di un società che invecchia. Tali soluzioni mirano a integrarsi il più possibile nell'ambiente in cui sono utilizzate, di modo che l'utente arrivi a sentirsi sicuro e a proprio agio nella vita di ogni giorno. Il sistema in oggetto della tesi nasce proprio con l'idea di offrire uno strumento di monitoraggio non invasivo delle abitudini delle persone anziane. Questo perché l'essere indipendenti e autonomi sono caratteristiche alla base della qualità della vita di un individuo e il sistema è stato sviluppato proprio per innalzare il livello di quest'ultima andando ad agire in diversi aspetti della loro vita, come comunicazione, comfort abitativo, ed aiuto nelle attività quotidiane. Il suo scopo è quello di rassicurare il paziente, di offrire consigli in merito allo stile di vita in casa e di notificare i componenti della famiglia, oppure gli operatori sanitari, in caso di situazioni anomale. Chiamato Nexty, dall'unione delle tre parole inglesi *next to you* (vicino a te), il kit comprende un gateway centrale e sei sensori ambientali capaci di rilavare parametri come la temperatura, la luminosità e l'umidità ma anche la permanenza nelle varie stanze della casa o l'assenza, la qualità e la quantità del sonno, l'apertura della porta d'ingresso e l'autenticazione degli ospiti. Nexty stesso è un "sensore di vitalità" sulla base del brevetto numero TO2011A001052 del 14 Novembre 2011 intitolato "Dispositivo di monitoraggio di un sistema di telemedicina e/o tele-assistenza e relativo uso e metodo".
Il target a cui si rivolge è rappresentato, quindi, dalle famiglie in cui ci sono componenti over 60 che vivono da soli, aziende ospedaliere o sanitarie e case di riposo che cosi facendo possono sopperire al bisogno di garantire servizi sanitari sempre più difficili da erogare, soprattutto in territori poco accessibili e ampliano la propria offerta di servizi includendo il monitoraggio dei propri pazienti in stato di convalescenza e riabilitazione.
L'obiettivo di questo elaborato è quello di presentare in dettaglio la struttura del software e dell'hardware che compongono il sistema Nexty. A tal proposito, nel capitolo 2 viene descritto il gateway di raccolta dati. Si tratta di un mini computer dotato di connettività Wifi e Z-Wave, che ha il compito di registrare ed elaborare i dati proveniente dai vari sensori ambientali della casa. Oltre a sensori fisici, come verrà spiegato più avanti nel capitolo, nel sistema sono stati inclusi dei moduli software che emulano il funzionamento di un sensore. Nel capitolo successivo si parla del server e dell'architettura dei servizi esposti tramite un API RESTful. In aggiunta viene descritta l'applicazione principale usata dall'utente per la consultazione delle informazioni raccolte dal kit. Strettamente collegato a questo argomento è poi il capitolo 4 che tratta la struttura del database scelto per l'archiviazione dei dati e offre una panoramica sulla tecnica usata per interfacciarsi con quest'ultimo. Infine nel capitolo 5 si parla del servizio di notifica di allarmi, caratteristica molto importante per questo tipo di sistema che sottolinea la natura proattiva del kit che svincola l'utente dal dovere consultare grafici e schemi in situazioni che non presentano anomalie.

Capitolo 2
GATEWAY DI RACCOLTA DATI

In questo secondo capitolo si analizza il funzionamento e la struttura del dispositivo elettronico che raccoglie dati dai sensori dell'abitazione e si connette con il server remoto. Nel paragrafo 2.1 vengono descritte le componenti hardware che compongono il gateway, mentre nel paragrafo 2.2 è presentata l'architettura del software e una descrizione dei moduli integrativi.

2.1 Hardware

Il gateway ha la forma di parallelepipedo ed è composto da materiale plastico di colore bianco. La struttura ha due fori su di un lato che servono per collegare l'alimentatore da 5V/2A e l'antenna GSM esterna. Al suo interno, le schede elettroniche sono disposte una sopra all'altra a formare una torre verticale. Alla base di questa torre si trova il mini computer Raspberry Pi, il modulo SIM800, più in alto c'è il controller per la rete Z-Wave ed infine il lettore di schede RFID. Nei sottoparagrafi 2.1.1 e 2.1.2 vengono descritti i componenti principali del gateway, ovvero il Raspberry Pi ed il controller Z-Wave. Per completare l'elenco delle componenti fisiche, nel sottoparagrafo 2.1.3 viene presentato il set di sensori wireless Z-Wave. L'hardware richiesto dal progetto per il gateway è elencato nella tabella 2.0.

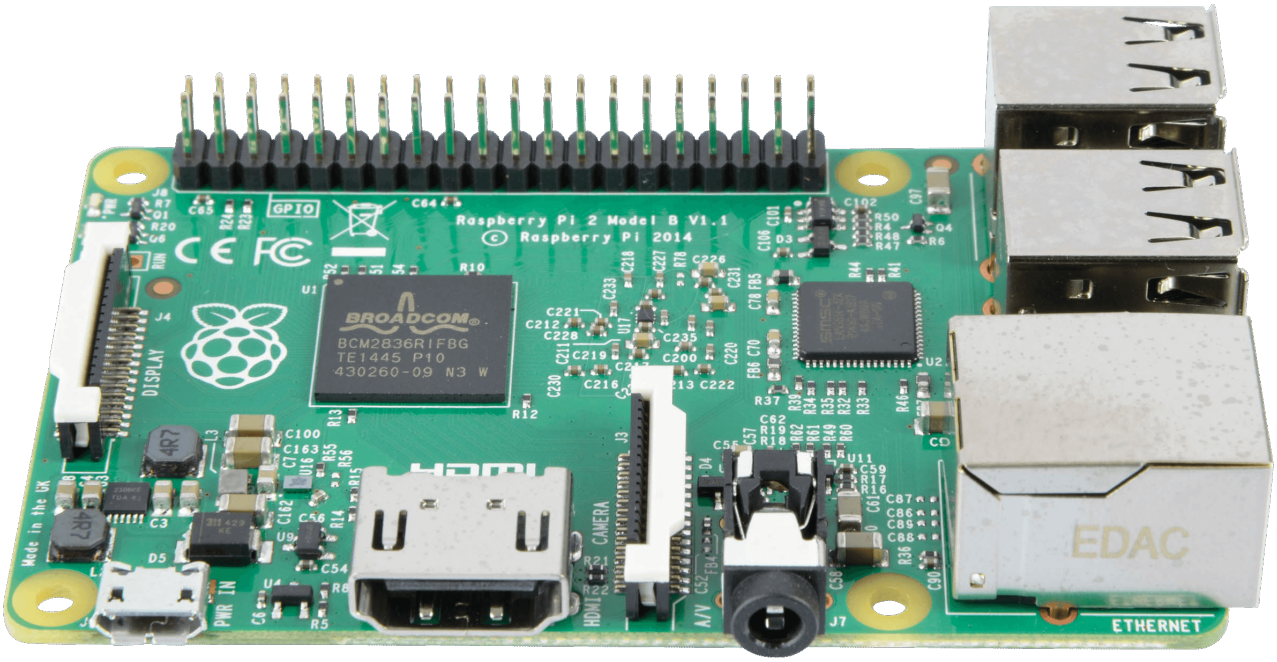
Tabella 2.0 - Elenco dei componenti per l'unità centrale di raccolta dati

Marca	Nome	Riferimenti
Raspberry	Pi 3 model B oppure Pi 2 model B	https://www.raspberrypi.org/products/
Z-Wave.Me	RaZberry Z-wave controller	http://razberry.z-wave.me/index.php?id=1
D-Link	DWA-121 Micro Adattatore USB Wireless N 150 (soltanto nel caso del Raspberry Pi 2)	http://www.dlink.com/it/it/products/dwa-121-wireless-n-150-pico-usb-adapter
Neuftech	Lettore Scheda ID Carta RFID USB	https://www.amazon.it/Neuftech-Lettore-Scheda-Prossimita-Portachiavi/dp/B018OYOR3E
Itead	Raspberry PI SIM800 GSM GPRS Add-On V2.0	https://www.itead.cc/raspberry-pi-sim800-gsm-gprs-add-on-v2-0.html
Qualsiasi	Scheda microSD da 16GB (anche da 8GB va bene)	https://www.amazon.it/s/ref=nb_sb_noss_1?__mk_it_IT=ÅMÅŽÕÑ&url=search-alias%3Daps&field-keywords=scheda+micro+sd+16+gb&rh=i%3Aaps%2Ck%3Ascheda+micro+sd+16+gb

2.1.1 Raspberry Pi

Il cuore pulsante del gateway è rappresentato da un computer single-board con architettura ARM, dalle dimensioni di una carta di credito e sviluppato dalla Raspberry Pi Foundation. Nella figura 2.1 è illustrato un dettaglio del modello utilizzato per la realizzazione del prototipo, ovvero il Pi 2 model B. L'architettura ARM (Advanced RISC Machine) indica una famiglia di microprocessori RISC principalmente impiegata in sistemi embedded e dispositivi mobili. Le motivazioni principali che hanno spinto a scegliere tale soluzione sono basate su fattori economici nonché sulla compatibilità con altre componenti del sistema e sulla effettiva comunità presente di sviluppatori e appassionati.
Il progetto di questo mini computer nasce nel lontano 2006 e nel 2012 viene rilasciato il primo modello. Da allora la comunità è cresciuta in modo significativo, oggi il numero degli iscritti sul forum officiale[6] è di 279 mila, mentre la più grande community italiana[7] ammonta a quasi 14 mila iscritti. La versatilità del Raspberry Pi è comprovata anche dal grande numero di progetti, più di 3300, che si possono consultare liberamente sul noto portale dei progetti fai da te Instructables[8].
La tecnologia di comunicazione usata dai sensori scelti per il kit è Z-Wave, quindi il Raspberry Pi deve disporre di un controller Z-Wave. Ciò può avvenire tramite un dongle USB oppure un modulo hardware interno. Per ridurre le dimensioni del case dell'unità si è scelto di adoperare il modulo Z-Wave.Me Razberry[9] da posizionare direttamente sopra alle porte dell'interfaccia GPIO (acronimo di General Purpose Input/Output) della scheda.

Figura 2.2 - Raspberry Pi 2 model B[10]



[6] https://www.raspberrypi.org/forums/ - Forum ufficiale di Raspberry Pi.

[7] https://forum.raspberryitaly.com - Community italiana Raspberry Pi.

[8] https://www.instructables.com/circuits/raspberry-pi/projects/ - Pagina web con i progetti che impiegano un Raspberry Pi.

[9] https://z-wave.me/products/razberry/ - Pagina web del modulo Z-Wave.Me Razberry.

[10] Fonte: https://www.reichelt.com/it/it/lampone-pi-2-b-v1-1-1-4x-900-mhz-1-gb-di-ram-raspberry-pi-2-b-p152728.html

Ad oggi, esistono in commercio ben undici modelli di Raspberry Pi con prezzi nell'intervallo 15-65€. Nel nostro caso, visto le ridotte potenze di calcolo richieste, la scelta è ricaduta sui due esemplari riepilogati nella tabella 2.1.

Tabella 2.1 - Due modelli di Raspberry a confronto

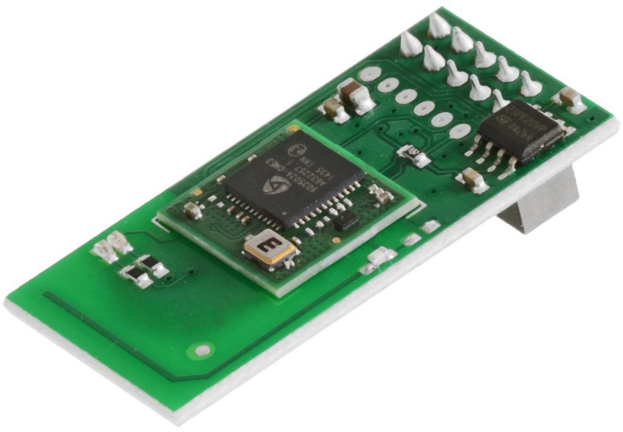
	CPU	**Memoria**	**Collegamenti di rete**
Raspberry Pi 2 model B	900 MHz 32-bit quad-core ARM Cortex-A7	1 GB (condivisa con la GPU) LPDDR2 (450 MHz)	Ethernet 10/100 Mbit/s (RJ-45)
Raspberry Pi 3 model B	1.2 GHz 64-bit quad-core ARM Cortex-A53	1 GB (condivisa con la GPU) LPDDR2 (900 MHz)	Ethernet 10/100 Mb/s, Wireless LAN 802.11n, Bluetooth 4.1

Entrambi sono dotati dello stesso numero di pin GPIO per ospitare il modulo Z-Wave e dispongono di 4 porte USB 2.0. Per maggiori dettagli si rimanda alla pagina del produttore (https://www.raspberrypi.org/products/).

2.1.2 Controller Z-Wave.Me Razberry

Z-Wave è un protocollo wireless che opera attorno ai 900 MHz progettato per la domotica e la telemedicina. Le reti Z-Wave sono di tipo mesh (a maglia) dove i nodi si possono dividere in due categorie: nodi controllori e nodi *slave*. I nodi controllori ospitano una tabella di indirizzamento dell'intera rete e possono calcolare i percorsi sulla base di essa. Tali nodi hanno la capacità di trasmettere i percorsi ai dispositivi slave in modo da abilitarli alla trasmissione. D'altra parte, i nodi slave non sono in grado di stabilire i percorsi e funzionano come unità di input e output nelle applicazioni Z-Wave. Esempi di questi nodi sono i dispositivi che rilevano la presenza, la temperatura e l'umidità, che controllano l'accensione e lo spegnimento di carichi, ed in generale dispositivi che contengono sensori a bordo.

Figura 2.2 - Dettaglio del modulo Razberry di Z-Wave.Me[11]



Per poter creare una rete Z-Wave almeno uno dei suoi nodi deve essere un controllore. Una singola rete Z-Wave può estendersi fino a 232 nodi. Il controller ha il compito di collegare il gateway ad un rete di tipo Z-Wave e di includere o escludere i sensori dalla rete stessa.

[11] Fonte: https://www.amazon.it/z-wave-me-RaZberry-peripheral-controllers-Z-Wave/dp/B00BL9QFH6/ref=sr_1_3?__mk_it_IT=ÅMÅŽÕÑ&keywords=raspberry+z-wave&qid=1581704905&sr=8-3

Per lo sviluppo del sistema in oggetto si è scelto il controllore Razberry, illustrato nella figura 2.2, della azienda russa Z-Wave.Me. La comunicazione tra il modulo e il Raspberry Pi è di tipo seriale. Viene sfruttata la porta SPI del Raspberry Pi, mentre il controller dispone di interfaccia UART/TTL. Le dimensioni del modulo mostrato nella figura 2.2 sono 44x42x14 mm e comprendono anche l'antenna PCBA, sono tali da permettere di alloggiare il controller direttamente sopra la sezione delle porte GPIO interessate. La scheda del controller ospita un modulo ricetrasmettitore Z-Wave Sigma Designs ZM5101 e una memoria flash SPI da 32 KB per i dati di rete. Il consumo energetico della scheda è in genere 18 mA a 3,3 V ma può raggiungere un picco fino a 40 mA quando il chip sta trasmettendo.

2.1.3 Sensori

Per il monitoraggio delle abitudini quotidiane dell'anziano sono stati scelti sensori ambientali che non devono essere indossati dalla persona, come ad esempio bracciali o orologi muniti di sensori, ma dispositivi di dimensioni ridotte da posizionare all'interno delle varie stanze della casa. In questo modo il kit risulta poco invasivo e l'anziano difficilmente si accorge di essere monitorato. Per mezzo di questi dispositivi, il sistema è capace di monitorare il sonno, l'attività motoria e gli spostamenti dell'anziano all'interno della propria abitazione. In aggiunta offre dati statistici sul comfort avvertito e sull'eventuale presenza di ospiti o intrusi. Nella configurazione standard il set è composto da sei dispositivi che sono elencati nella tabella 2.2. Tutti i sensori sono dotati di tecnologia Z-Wave e sono alimentati attraverso batterie, grazie a questo l'installazione è semplice ed immediata. In seguito si analizza le potenzialità di ciascun dispositivo.

Tabella 2.2 - Elenco dei sensori che compongono il kit

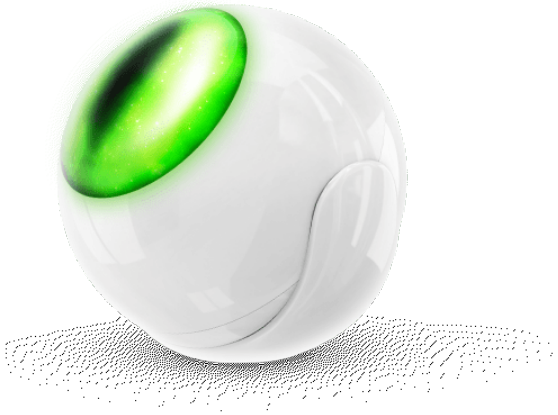
Quantità	Marca	Modello	Tipo batteria	Misure	Riferimenti
3	Fibaro	Motion Sensor	CR123A	Movimento, temperatura e luminosità	https://www.fibaro.com/en/products/motion-sensor/
2	Fibaro	Door/Window Sensor	ER14250 (1/2AA)	Apertura/chiusura e temperatura	https://www.fibaro.com/en/products/door-window-sensor/
1	Zipato	Mini RFID Keypad	2 x AA 1,5V	Identificativo del tag RFID	https://www.zipato.com/product/mini-keypad-rfid
1	Tunstall	Bed Occupancy Sensor	Nessuna	Presenza sul letto	https://www.tunstall.co.uk/our-products/product-catalogue/bed-chair-occupancy-sensor/

FIBARO MOTION SENSOR

Il *motion sensor* di Fibaro, illustrato nella figura 2.3, è un dispositivo Z-Wave dotato di tre sensori: un sensore ad infrarossi passivo, un sensore di temperatura e uno di luminosità. I suoi dati vengono inviati al gateway ogni qualvolta che c'è una variazione significativa dei parametri ambientali osservati. Il sistema elabora queste informazioni per fornire dati sulla posizione della

persona nell'arco della giornata, sulla quantità di movimento e sul comfort percepito nelle varie stanze.

Figura 2.3 - Sensore di movimento Fibaro motion sensor[12]



LETTORI RFID

All'interno del sistema sono stati previsti due lettori RFID (Radio-frequency identification), il primo è dotato di tecnologia Z-Wave, mentre il secondo è collegato al gateway tramite cavo USB. Il Mini RFID Keypad della Zipato (figura 2.4), oltre alla funzionalità di lettura, dispone di una tastiera numerica di quattro cifre e due pulsanti aggiuntivi, nasce come dispositivo per il controllo dell'attivazione/disattivazione di allarmi e può essere adoperato per l'autenticazione. Il lettore RFID USB della Neuftech è un dispositivo HID (Human Interface Device) appartenente alla categoria delle tastiere, nel sottoparagrafo 2.2.2 viene descritto il software che è stato sviluppato per interagire con esso. Il lettore RFID ha lo scopo di identificare le persone che entrano nella casa dell'anziano tramite lettura di schede elettroniche dotate di antenna e microchip. Il chip è dotato di una memoria non volatile e un codice univoco, il quale viene trasmesso tramite l'antenna all'apparato lettore al fine di fornire uno storico dei visitatori. Il sistema è in grado di segnalare eventuali presenze di sconosciuti grazie alle informazioni circa lo stato di apertura della porta fornite dal sensore dedicato.

Figura 2.4 - Lettore RFID Zipato Mini RFID Keypad[13]

[12] Fonte: https://www.fibaro.com/it/products/motion-sensor/

[13] Fonte: https://www.zipato.com/product/mini-keypad-rfid

FIBARO DOOR SENSOR E TUNSTALL BED SENSOR

Il sensore di apertura e chiusura della porta di Fibaro, illustrato in figura 2.5, ha un doppio uso all'interno del sistema, per questo motivo ne sono previsti due. Nella prima configurazione viene impiegato per rilevare le aperture e le chiusure della porta principale d'ingresso. Questo dato insieme alle informazioni sul movimento permette di determinare se l'anziano ha lasciato la casa oppure è rientrato. Inoltre, insieme al lettore RFID, funge da sistema di controllo di accessi nella casa. Nell'altra configurazione, il dispositivo viene usato insieme al tappetino Tunstall, presente in figura 2.6, per rilevare la presenza o l'assenza della persona sul letto. Infatti il *bed occupancy sensor* non dispone di interfaccia Z-Wave a bordo ma è collegato tramite filo al sensore della porta che fa da ponte radio.

Figura 2.5 - Sensore Fibaro door sensor[14]

Figura 2.6 - Tappetino Tunstall bed sensor[15]

Modulo SIM800

La scheda elettronica dotata di modulo GSM/GPRS SIM800, mostrata in figura 2.7, permette di estendere le potenzialità del kit e aumentare la robustezza del canale di comunicazione con il server di raccolta dati e il cliente finale. Infatti, lo scopo principale di questo modulo è di garantire una connessione al server anche in assenza o malfunzionamento della rete Wi-Fi e di inviare brevi messaggi di testo in caso si verificano condizioni anomale che sono trattate più in dettaglio nel capitolo 5. Le dimensioni della scheda sono 85mm X 56mm X 1.6mm e permettono di posizionare il modulo direttamente sopra il Raspberry Pi sfruttando le porte GPIO. Grazie all'interfaccia UART

[14] Fonte: https://www.fibaro.com/it/products/door-window-sensor/

[15] Fonte: https://www.tunstall.co.uk/our-products/product-catalogue/bed-chair-occupancy-sensor/

e al set di comandi AT è possibile inviare e ricevere messaggi SMS (Short Text Message), effettuare e rispondere a chiamate telefoniche e connettere il Raspberry Pi alla rete Internet. Il set di comandi Hayes (o comandi AT) è un linguaggio di comando specifico originariamente sviluppato da Dennis Hayes nel 1981. Il set di comandi è costituito da una serie di brevi stringhe di testo che possono essere combinate per produrre comandi per operazioni come la composizione, il riaggancio e la modifica dei parametri della connessione.

Figura 2.7 - Dettaglio del modulo SIM800[16]

2.2 Software

In questo paragrafo verrà tratta la componente software installata sul gateway, spesso chiamato semplicemente Raspberry Pi. Il sottoparagrafo 2.2.1 offre una panoramica sul sistema domotico Z-Way Home Automation sviluppato dalla stessa azienda del controller Z-Wave Razberry. Invece nel sottoparagrafo 2.2.2 vengono trattati i moduli software che fanno parte del kit e nell'ultimo sottoparagrafo si parla del server di configurazione e dell'interfaccia grafica utili all'utente nel momento della prima installazione del kit in casa della persona.

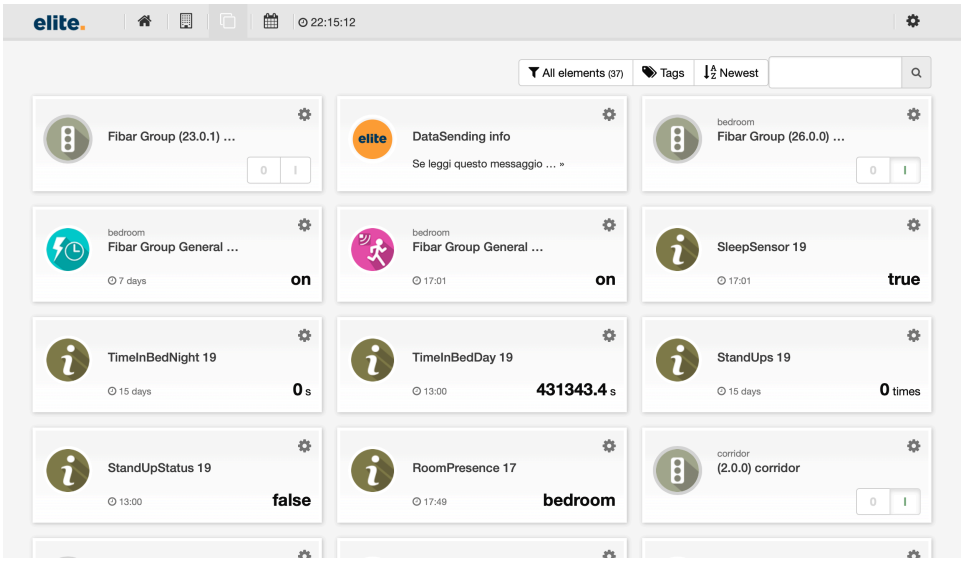
2.2.1 Z-Way Home Automation

Z-Way Home Automation è un sistema di automazione domotica basato sul linguaggio Javascript (usando il motore Javascript V8) e sviluppato per funzionare con i controller di Z-Wave.Me. Insieme al sistema di automazione vengono fornite due interfaccia utente entrambe create utilizzando il noto web framework AngularJS. La Expert UI è un'applicazione web dedicata al controllo avanzato della rete Z-Wave, tramite questa interfaccia grafica è possibile conoscere lo stato della rete Z-Wave, l'elenco dei suoi nodi, avviare la procedura di inclusione/esclusione ed impostare i parametri di ciascun sensore. Anche la SmartHome UI (figura 2.8) è un'applicazione web che serve principalmente per la consultazione dei dati dei sensori e per la gestione dei

[16] Fonte: https://www.itead.cc/raspberry-pi-sim800-gsm-gprs-add-on-v2-0.html

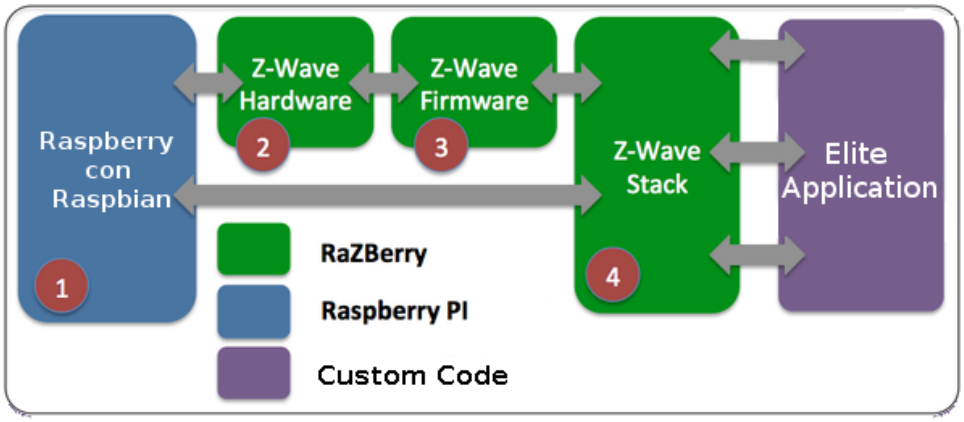
moduli scaricabili, per questi motivi è rivolta ad utenti meno esperti, a differenza della prima che è per utilizzatori avanzati.

Figura 2.8 - Schermata principale dell'applicazione SmartHome UI



La figura 2.9 mostra come è costituito il software all'interno del Raspberry (elemento 1 nella figura). Il controller RaZberry insieme al Z-Way Home Automation sono rappresentati dagli elementi 2, 3 e 4. Il blocco chiamato "Elite Application" internamente è composto da un set di moduli software e dal server di configurazione, entrambi scritti usando il linguaggio Javascript. La comunicazione tra i moduli ed il resto del sistema avviene tramite un meccanismo interno di tipo publish/subscribe, mentre il server di configurazione si deve avvalere dell'API RESTful esposta dal kit domotico. Tale differenza è dovuta la fatto che i moduli sono integrati nell'applicazione Z-Way Home Automation, invece il server di configurazione è un processo distinto. Il sistema domotico permette ai moduli di creare eventi, misure e di gestire notifiche provenienti dal bus di comunicazione per eseguire azioni personalizzate. Tra le più importanti funzioni che vengono gestite sicuramente c'è la comunicazione tra i sensori della rete Z-Way e il server remoto di raccolta dati.

Figura 2.9 - Architettura software del gateway di raccolta dati



L'intera applicazione Z-Way Home Automation si trova nella cartella chiamata z-way-server. Nella tabella 2.3 viene elencato il contenuto della directory principale. Per eseguire manutenzione e per installare componenti software aggiuntivi è di particolare interesse la sottocartella di automation chiamata user-modules, che non va confusa con la cartella modules. Infatti, quest'ultima è un

contenitore di moduli installati tramite l'interfaccia grafica, mentre la cartella user-modules contiene soltanto moduli software installati manualmente. Nel sottoparagrafo che segue (2.2.2) viene ulteriormente evidenziata questa distinzione.

Tabella 2.3 - Struttura della directory di Z-Way Home Automation

Nome	Descrizione
ZDDX	Cartella con i file descrittori dei device Z-Wave installati.
automation	Cartella contenente il sistema di automazione domotica e i moduli.
config	Cartella con i file di configurazione.
htdocs	Cartella che contiene tutti i file client-side serviti dal server, tra cui le due interfacce.
libs	Cartella con le librerie dinamiche (una sorta di DLL).
libzway	Cartella contenente file di intestazione di librerie per la comunicazione col controller Z-Wave.
modules	Cartelle di moduli software.
modules-includes	Cartella contenente file di intestazione per il motore JavaScript V8.
translations	Cartella con file XML per la traduzione.
z-get-tty-config	Cartella per il file di configurazione della porta seriale.

2.2.2 Moduli proprietari

Un punto di forza importante del sistema demotico Z-Way Home Automation è la modularità, infatti è possibile integrare moduli software di terze parti per estendere le funzionalità del sistema e per creare sensori virtuali. Un sensore virtuale non è un dispositivo fisico ma un specifico software che può generare misure ed eventi sfruttando le informazioni inviate da altri sensori. Il sistema domotico permette l'installazione di moduli da uno store gestito da Z-Wave.Me. Si tratta di un catalogo di moduli suddiviso per categorie che l'utente finale può sfogliare ed aggiungere al proprio kit. Ogni sviluppatore, previa registrazione, può caricare i moduli che ha sviluppato e renderli disponibili a tutti gli utenti. Facendo in questo modo si è creata una discreta comunità di sviluppatori che contribuisce estendendo le capacità del kit. Solo per citarne alcuni, sullo store si possono trovare moduli per integrare Amazon Alexa, Philips HUE e Apple HomeKit. Un altro modo per installare dei moduli aggiuntivi è quello manuale che non prevede la pubblicazione sullo store ma si deve operare direttamente sul file system del Raspberry Pi e copiare la cartella che rappresenta il modulo in una posizione specifica. Nel caso in esame, avendo sviluppato moduli dedicati a casi d'uso specifici e non volendo condividere il codice sorgente con terzi, si è scelto quest'ultimo approccio. Usando il metodo di installazione manuale è opportuno inserire i moduli in questa cartella */opt/z-way-server/automation/userModules*.

La struttura tipica di un modulo software è suddivisa nel seguente modo: il file denominato index.js contiene l'algoritmo che descrive le logiche del modulo, il file module.json è un descrittore del modulo stesso e permette, tra le svariate opzioni, di dichiarare il nome, la descrizione e la definizione dell'interfaccia grafica di configurazione. Infine, la cartella htdocs contiene l'icona del modulo, mentre nella cartella lang si trovano dei file JSON necessari alla localizzazione del software.

Nel file contente l'algoritmo, mostrato nell'esempio 2.0, si trova la definizione della classe principale che deve necessariamente discendere dalla classe AutomationModule. Durante la fase di inizializzazione è possibile creare un virtual device, che verrà trattato dal sistema come un dispositivo Z-wave fisico, nonché registrare il modulo a ricevere informazioni dal bus degli eventi oppure creare dei task basati sul tempo. Per completare la struttura del codice non devono mai mancare i metodi d'istanza per l'inizializzazione e lo stop del modulo. Durante lo stop è fondamentale rimuovere eventuali *listeners* e dispositivi virtuali registrati in fase di inizializzazione.

Esempio 2.0 - Struttura base in Javascript di un modulo Z-Way Home Automation

```
// ----------------------------------------------------------------------
// --- Class definition, inheritance and setup
// ----------------------------------------------------------------------

function MyModule(id, controller) {
    // Call superconstructor first (AutomationModule)
    MyModule.super_.call(this, id, controller);
}

inherits(MyModule, AutomationModule);

_module = MyModule;

// ----------------------------------------------------------------------
// --- Module instance initialized
// ----------------------------------------------------------------------

MyModule.prototype.init = function (config) {

    MyModule.super_.prototype.init.call(this, config);

    // Subscribe to events, setup timers, instantiate virtual devices,
register widgets and so on...

};

MyModule.prototype.stop = function () {

    // remove event listeners and bindings

    MyModule.super_.prototype.stop.call(this);
};

// ----------------------------------------------------------------------
// --- Module methods
// ----------------------------------------------------------------------

MyModule.prototype.myMethod = function () {
    // methods logic
};
```

Di seguito verrà descritto più in dettaglio ciascun modulo software che deve essere installato ed opportunamente configurato per soddisfare le specifiche del progetto.

DATA-SENDING
Il modulo data-sending riveste un ruolo fondamentale. Infatti, esso si occupa dell'invio dei dati al server di raccolta e della generazione di una parte degli eventi. Al momento della sua installazione viene caricato il file di configurazione del kit, in formato JSON e generato durante la procedura di prima installazione che viene ulteriormente trattata nel sottoparagrafo 2.2.3. Tale file contiene informazioni necessarie per eseguire una mappatura tra i sensori locali e le loro istanze create sul server di raccolta dati, cosi da associare alle misure e agli eventi un riferimento al sensore e al kit stesso. Bisogna precisare che localmente i sensori hanno identificativi diversi da quelli registrati sul server di raccolta e lo scopo del file è proprio quelli di fare una traduzione tra identificativi locali e quelli globali.
Come già accennato, il file di configurazione del modulo data-sending è in formato JSON, e la tabella 2.4 evidenzia gli elementi che lo compongono.

Tabella 2.4 - Elenco dei campi del file di configurazione del modulo data-sending

Voce	Descrizione
wifi	Contiene le informazioni per la connessione ad una rete WiFi, ovvero SSID e chiave.
user	Contiene il token per eseguire l'invio dei dati al server.
kit	Rappresenta il kit selezionato dall'utente.
devices	E' un elenco di dispositivi Z-Wave installati insieme al controller principale. Non è un array ma un dictionary per facilitare la ricerca dei devices in base al loro nodeId.
sensors	E' un elenco di sensori fisici e virtuali associati ai dispositivi installati. Non è un array ma un dictionary per facilitare la ricerca dei sensori in base al loro localId.
eventTypes	E' un elenco di eventi che un kit è in grado di generare. Non è un array ma un dictionary per facilitare la ricerca degli eventi in base al loro eventType.

Una volta che il data-sending è stato inizializzato, vengono registrati due task di cui uno ad intervalli regolari si occupa di valutare e generare eventi su base temporale, e l'altro invia le misure al server di raccolta dati non appena vengono notificate dal *event bus* interno al kit.

DAILY-ACTIVITIES
Il sensore virtuale chiamato daily-activities si occupa di aggregare i dati provenienti dai vari sensori di presenza è fornire un nuovo dato: la stanza in cui si trova la persona. Durante la sua configurazione, il modulo si registra ad ascoltare le notifiche provenienti da tutti i sensori di movimento della casa e dal sensore della porta principale di ingresso. E' importante notare che il modulo è capace di monitore soltanto una persona, infatti l'algoritmo si basa sostanzialmente sui dati di movimento e sulla collocazione dei sensori che hanno individuato un movimento. Quindi, se l'ultimo dato sul movimento è di un sensore in una stanza diversa, allora viene impostata la stanza associata al sensore come quella in cui si trova ora la persona. Il sensore di porta permette di determinare l'assenza dell'anziano nella casa. In particolare, la persona viene considerata assente se, in seguito ad una apertura della porta, non vi è alcun movimento nella casa nell'arco di un determinato intervallo temporale.

MOVEMENT-INDEX
Movement-index è il modulo incaricato di fornire informazioni sull'attività motoria dell'anziano. Alla base dell'algoritmo c'è l'osservazione dei dati dei sensori di movimento e del sensore virtuale daily-activites allo scopo di calcolare il tempo totale in cui ciascun sensore di movimento ha dato un valore positivo. Infatti, quando un sensore ha rilevato un movimento rimane in tale stato fintanto rileva movimento continuo e questo ci permette di dare una stima del tempo in cui l'anziano sicuramente non è fermo ma sta svolgendo attività motoria. Quando il sistema rileva una assenza dalla casa, l'algoritmo considera la persona in costante movimento.

SLEEP-SENSOR
Lo sleep-sensor è un modulo che monitora il sensore del letto per dare informazioni sulla qualità del sonno della persona. E' considerevole il fatto che il sensore è in grado di distinguere tra il riposo pomeridiano o diurno e il sonno vero è proprio, cioè quello notturno. L'attenzione maggiore è rivolta al sonno durante la notte, e al momento del risveglio vengono forniti dati sul numero delle alzate, la stanza visitata durante queste alzate e il tempo totale del sonno. Per quanto riguarda il riposo pomeridiano, il sistema fornisce soltanto la durata temporale. Tale distinzione è fatta su base oraria che può essere configurata secondo le abitudini della persona. Il modulo registra 5 sensori virtuali: quello principale indica se la persona sta attualmente dormendo oppure no, indifferentemente dalla tipologia del sonno, mentre il secondo sensore virtuale indica se la persona si è alzata durante la dormita notturna. I restanti sensori aggiornano i loro valori soltanto alla fine di una dormita, ovvero quando è possibile attribuire una tipologia a questa sulla base delle informazioni osservate, e offrono rispettivamente indicazioni sul numero delle alzate durante la dormita notturna e sulla durata del sono attribuito alla notte e al giorno.

RFID-USB-READER-SENSOR
Il RfidUSBReaderSensor è il modulo Z-Way Home Automation che permette l'interazione tra il sistema e il lettore RFID collegato tramite porta USB. L'architettura di questo modulo è più complessa perché il dispositivo non è un sensore Z-Wave, infatti l'algoritmo è stato scritto usando due linguaggi differenti: Python e Javascript. Dal momento che il lettore RFID viene riconosciuto a livello del sistema operativo come una tastiera, lo script Python si occupa di catturare il codice del tag letto e di salvarlo su un file temporaneo. In seguito, tramite una chiamata http al sistema demotico, viene scaturito un update del sensore virtuale, creato dal modulo, che legge dal file temporaneo il codice del tag e genera la misura.

ALTRI MODULI
Altri due moduli molto utili in fase di individuazione e correzione di errori e nel testing sono il debug-module e il debug-module-daily. Entrambi i moduli permettono di generare delle misure fittizie e di spedirle al server di raccolta dati usando il modulo data-sending oppure direttamente tramite chiamate http. A differenza dal debug-module, il secondo modulo permette di programmare la generazione e l'invio delle misure a diversi orari della giornata in modo da simulare le abitudini di una persona in modo automatico.

La tabella 2.5 riepiloga i vari moduli software sviluppati per il sistema Z-Way.

Tabella 2.5 - Repository dei moduli software

Nome	Link	Descrizione
daily-activities	https://bitbucket.org/elitefabriano/daily-activities	Modulo Z-Way Home Automation che monitora la posizione della persona nelle varie stanze della casa.

Nome	Link	Descrizione
data-sending	https://bitbucket.org/elitefabriano/data-sending	Modulo Z-Way Home Automation che si occupa dell'invio dei dati al server di raccolta e della generazione di alcuni eventi.
debug-module	https://bitbucket.org/elitefabriano/debug-module	Modulo Z-Way Home Automation utile per il debug, permette di generare misure fittizie.
debug-module-daily	https://bitbucket.org/elitefabriano/debug-module-daily	Modulo Z-Way Home Automation utile per il debug, permette di generare misure fittizie ad orari prestabiliti.
movement-index	https://bitbucket.org/elitefabriano/movement-index	Modulo Z-Way Home Automation che calcola l'indice di movimento della persona.
sleep-sensor	https://bitbucket.org/elitefabriano/sleep-sensor	Modulo Z-Way Home Automation che monitora il sonno giornaliero e notturno.
RfidUSBReaderSensor	https://bitbucket.org/elitefabriano/rfidusbreadersensor	Modulo Z-Way Home Automation che permette l'interazione con il lettore RFID.

2.2.3 Server di configurazione

L'applicazione el_care_raspberry (chiamata anche server di configurazione o wizard) è realizzata utilizzando il noto framework opensource NodeJS[17] versione 10.15.3. La parte del backend è stata implementata utilizzando il web framework SailsJS[18] versione 0.12.3, invece per la parte frontend si è utilizzato il framework AngularJS[19] versione 1.5.8 e AngularJS Material 1.1.5. Il codice sorgente del progetto si trova in una *code repository*, cioè uno spazio web protetto progettato appositamente per ospitare codice sotto controllo di versione GIT, raggiungibile all'indirizzo https://bitbucket.org/elitefabriano/el_care_raspberry.
SailsJS si definisce come un web backend framework basato sul pattern MVC (Model-View-Controller) sviluppato in cima all'ambiente NodeJS. E' rilasciato come software gratuito e open-source, ed è progettato per semplificare la creazione di applicazioni web e API RESTful integrate con diverse tipologie di database. D'altra parte, AngularJS è anch'esso un progetto open-source gestito da Google per lo sviluppo di moderne applicazioni web.
Il server di configurazione ha il compito di distribuire i file statici che compongono la parte frontend, e l'API esposta si limita alla creazione del file di configurazione per il modulo data-sending e all'esecuzione dei script bash che operano direttamente sull'interfaccia di rete del Raspberry Pi. Risulta evidente che il ruolo dell'applicazione SailsJS è soltanto di supporto al frontend, pertanto la descrizione dell'API e l'architettura non sono trattate per brevità.
Per comprendere meglio la struttura del software si può iniziare con la directory principale del progetto che si presenta suddivisa nelle cartelle elencate nella tabella 2.6.

[17] https://nodejs.org/it/about/ - Maggiori informazioni su NodeJS

[18] https://sailsjs.com/whats-that - Introduzione a SailsJS

[19] https://angularjs.org - AngularJS

Tabella 2.6 - Struttura della directory del server di configurazione

Nome	Descrizione
api	Cartella contenente modelli, controller e regole per la gestione del server.
assets	Cartella che contiene tutti i file client-side serviti dal server, cioè il frontend.
config	Cartella con vari file di configurazione, tra cui quelli inerenti la connessione con il database, i percorsi dell'API, la pianificazione di comandi periodici e lo script di avvio del applicazione NodeJS.
node_modules	Cartella delle dipendenze del progetto NodeJS.
tasks	Cartella dei task di Grunt che eseguono la compilazione, il linting e la minificazione del codice.
utils	Cartella contenente script bash per il passaggio da client mode ad AP mode e altri file di configurazione che vengono copiati in automatico dagli script.
views	Cartelle per i template, ovvero file con estensione EJS che vengono convertiti in HTML mentre il programma è in esecuzione.

Proseguendo l'analisi dei sorgenti del progetto, nella cartella chiamata utils si trovano vari script bash e file di configurazione dell'interfaccia di rete del sistema. Nella tabella 2.7 vengono descritti quelli più importanti.

Tabella 2.7 - File di scripting e di configurazione

Nome	Descrizione
connectToWifi.sh	Script bash per il passaggio in client mode.
createAP.sh	Script bash per il passaggio in AP mode.
el_care_raspberry.sh	Script bash per l'avvio automatico del server di configurazione.
testInternet.sh	Script bash per testare l'accesso ad internet.
update_raspberry_info.sh	Script bash per aggiornare le informazione HW del kit tramite una chiamata HTTP al server di raccolta dati.
config_file/wifi_config.txt	File usato in modalità client. Contiene la configurazione delle interfacce di rete.
config_file/ dhcpd_client_mode.conf	File usato in modalità client. Contiene la configurazione del client dhcp.
config_file/ wpa_supplicant.conf	File usato in modalità client. Contiene la configurazione per il wpa supplicant necessario per connettersi con una rete wi-fi.
config_file/ap_config.txt	File usato in modalità AP. Contiene la configurazione delle interfacce di rete.
config_file/ dhcpd_ap_mode.conf	File usato in modalità AP. Contiene la configurazione del client DHCP e indica di disabilitarlo se si è in AP.
config_file/dnsmasq.conf	File usato in modalità AP. Contiene la configurazione del server DHCP.
config_file/hostapd.conf	File usato in modalità AP. Contiene la configurazione per la creazione della rete wi-fi. Qui si può dettare il nome della rete e la password.

Nome	Descrizione
config_file/ hostapd_default.conf	File usato in modalità AP. Contiene la locazione della configurazione di default di hostapd.

Finiamo l'analisi delle componenti del progetto con la cartella che contiene l'applicazione AngularJS, ovvero l'interfaccia grafica per l'installazione del kit. Si deve precisare che in questa cartella non sono stati inclusi due template i quali rappresentano il punto di ingresso per l'applicazione. Questi due file si possono trovare in views/layout.ejs e views/index.ejs, rispettivamente nel primo viene definita una struttura base dell'interfaccia con l'inclusione di tutte le dipendenze di tipo CSS (Cascading Style Sheets) e Javascript, nel secondo si trova la definizione del corpo principale dell'app AngularJS. La tabella 2.8 mostra la struttura della directory della cartella assets.

Tabella 2.8 - Struttura della directory della applicazione AngularJS

Nome	Descrizione
images	Cartella contenente immagini.
js	Cartella che contiene tutti i file JavaScript del frontend. Qui possiamo trovare i controllers, i services, la dichiarazione dell'app AngularJS e la struttura degli stati dell'app.
libs	Cartella con i moduli per il frontend.
styles	Cartella per i file CSS.
templates	Cartella che contiene file HTML che rappresentano gli stati dell'applicazione.

Nella tabella 2.9 sono descritti tutti i moduli di terze parti utilizzati per il backend dell'applicazione.

Tabella 2.9 - Dipendenze software lato back-end del server di configurazione

Modulo	Link	Descrizione
ejs	https://github.com/tj/ejs	Libreria per generazione delle viste.
grunt	https://github.com/gruntjs/grunt	Tool di build per JavaScript.
grunt-contrib-clean	https://github.com/gruntjs/grunt-contrib-clean	Plugin di Grunt per eliminare cartelle o file.
grunt-contrib-coffee	https://github.com/gruntjs/grunt-contrib-coffee	Plugin di Grunt per compilare file CoffeeScript in JavaScript.
grunt-contrib-concat	https://github.com/gruntjs/grunt-contrib-concat	Plugin di Grunt per concatenare file.
grunt-contrib-copy	https://github.com/gruntjs/grunt-contrib-copy	Plugin di Grunt per copiare file e cartelle.
grunt-contrib-cssmin	https://github.com/gruntjs/grunt-contrib-cssmin	Plugin di Grunt per la compressione dei file CSS.

Modulo	Link	Descrizione
grunt-contrib-jst	https://github.com/gruntjs/grunt-contrib-jst	Plugin di Grunt per compilare file underscore in file JST.
grunt-contrib-less	https://github.com/gruntjs/grunt-contrib-less	Plugin di Grunt per compilare file LESS in CSS.
grunt-contrib-uglify	https://github.com/gruntjs/grunt-contrib-uglify	Plugin di Grunt per la compressione dei file JS.
grunt-contrib-watch	https://github.com/gruntjs/grunt-contrib-watch	Plugin di Grunt per osservare i cambiamenti di file.
grunt-sails-linker	https://github.com/sailsjs/grunt-sails-linker	Plugin di Grunt per l'autoinserimento dei tag "script" nei file HTML.
grunt-sync	https://github.com/tomusdrw/grunt-sync	Plugin di Grunt per copiare soltanto i file che hanno subito modifiche.
include-all	https://github.com/balderdashy/include-all	Modulo per NodeJS che permette di unire tutti i file di una cartella in un unico file.
rc	https://github.com/dominictarr/rc	Modulo per NodeJS per caricare i file di configurazione.
sails	https://github.com/balderdashy/sails	Framework MVC per NodeJS.
sails-disk	https://github.com/balderdashy/sails-disk	Modulo per Sails per il salvataggio dei dati in un file locale.
grunt-ng-annotate	https://github.com/mgol/grunt-ng-annotate	Plugin di Grunt per aggiungere le annotazioni ove non sono state aggiunte (nei sorgenti dell'app AngularJS).
request	https://github.com/request/request	Modulo per NodeJS per fare richieste HTTP.

Nella tabella 2.10 sono descritti i moduli di terze parti utilizzati per il frontend dell'applicazione.

Tabella 2.10 - Dipendenze software lato front-end del server di configurazione

Modulo	Link	Descrizione
angular-animate	https://github.com/angular/angular.js	Modulo per AngularJS per gestire le animazioni.
angular-aria	https://github.com/angular/angular.js	Modulo per AngularJS usato per migliorare l'accessibilità dell'interfaccia utente.
angular-material-datetimepicker	https://github.com/logbon72/angular-material-datetimepicker	Componente grafico per la selezione della data.
angular-material	https://github.com/angular/material	Libreria di componenti grafici per l'interfaccia dell'app.

Modulo	Link	Descrizione
angular-ui-router	https://ui-router.github.io	Modulo per AngularJS per gestire gli stati dell'app.
angular	https://github.com/angular/angular.js	Framework per web app.
font-awesome	https://github.com/FortAwesome/Font-Awesome	Set di icone.

La procedura iniziale ha lo scopo di collegare il kit domotico alla rete Internet tramite un modem router wireless ed a creare un file di configurazione usato dal modulo data-sending (descritto nel sottoparagrafo 2.2.2). Normalmente il gateway di raccolta dati è in modalità wireless access point, questo significa che permette all'utente di accedere in modalità wireless alla rete locale da cui si ha l'accesso alla pagina di configurazione che si trova all'indirizzo http://192.168.0.1. Il processo di configurazione può essere suddiviso in due fasi: la prima fase serve a collegare il kit ad Internet. Dopo che l'utente ha inserito il SSID[20] e la chiave della rete wireless desiderata, il kit tenterà di connettersi a tale rete e, quindi di verificare la connessione Internet. Se non riesce a connettersi al router oppure è impossibile verificare la connessione Internet, il gateway tornerà in modalità wireless access point. Invece, se la connessione col router è stata stabilita e Internet è accessibile, allora il kit provvederà a comunicare le informazioni che permettono di identificarlo, che sono indirizzo IP locale e numero seriale, al server di raccolta dati per poi terminare definitivamente la procedura di configurazione della connessione ad Internet. Durante questo processo di identificazione l'utente viene posto in uno stato di attesa che prevede una procedura di fallback. Se dopo un tempo prestabilito l'applicazione web non riesce ad ottenere le informazioni sul kit dal server di raccolta dati, allora l'utente verrà invitato a riprovare la procedura dall'inizio. Soltanto se il kit è raggiungibile, ovvero quando il processo di connessione ad Internet è stato completato, l'utente viene reindirizzato per continuare con la seconda fase della configurazione. La seconda fase della configurazione permette di registrarsi al servizio per consultare i dati messi a disposizione sul server remoto e di inserire le informazioni riguardo la persona monitorata. In figura 2.10 è mostrata una schermata d'esempio di questa fase, in particolare quella relativa al posizionamento dei sensori nell'abitazione. Oltre a questo vengono chiesti i contatti necessari all'invio di notifiche email ed SMS e la procedura termina con il salvataggio locale del file di configurazione in formato JSON e l'avvio del monitoraggio dell'anziano. Il diagramma di flusso della figura 2.11 mostra la procedura di configurazione iniziale del kit appena presentata.

[20] SSID è acronimo di service set identifier ed indica il nome di una rete WLAN.

Figura 2.10 - Interfaccia grafica di installazione del kit

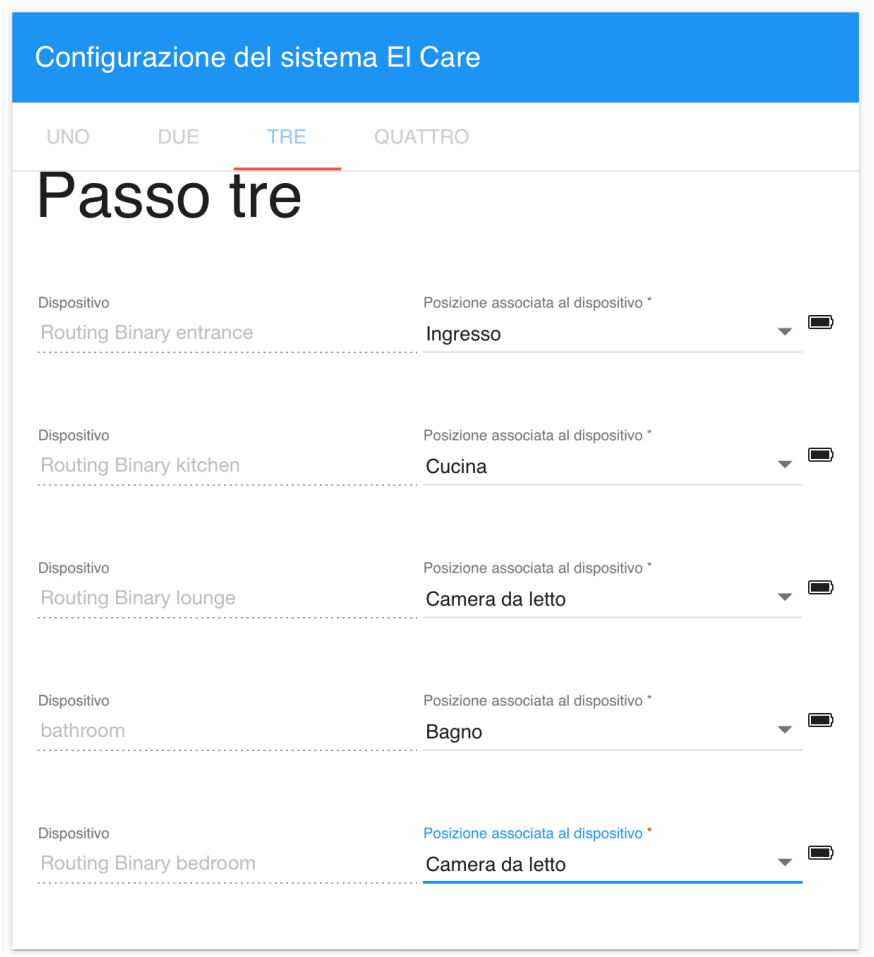
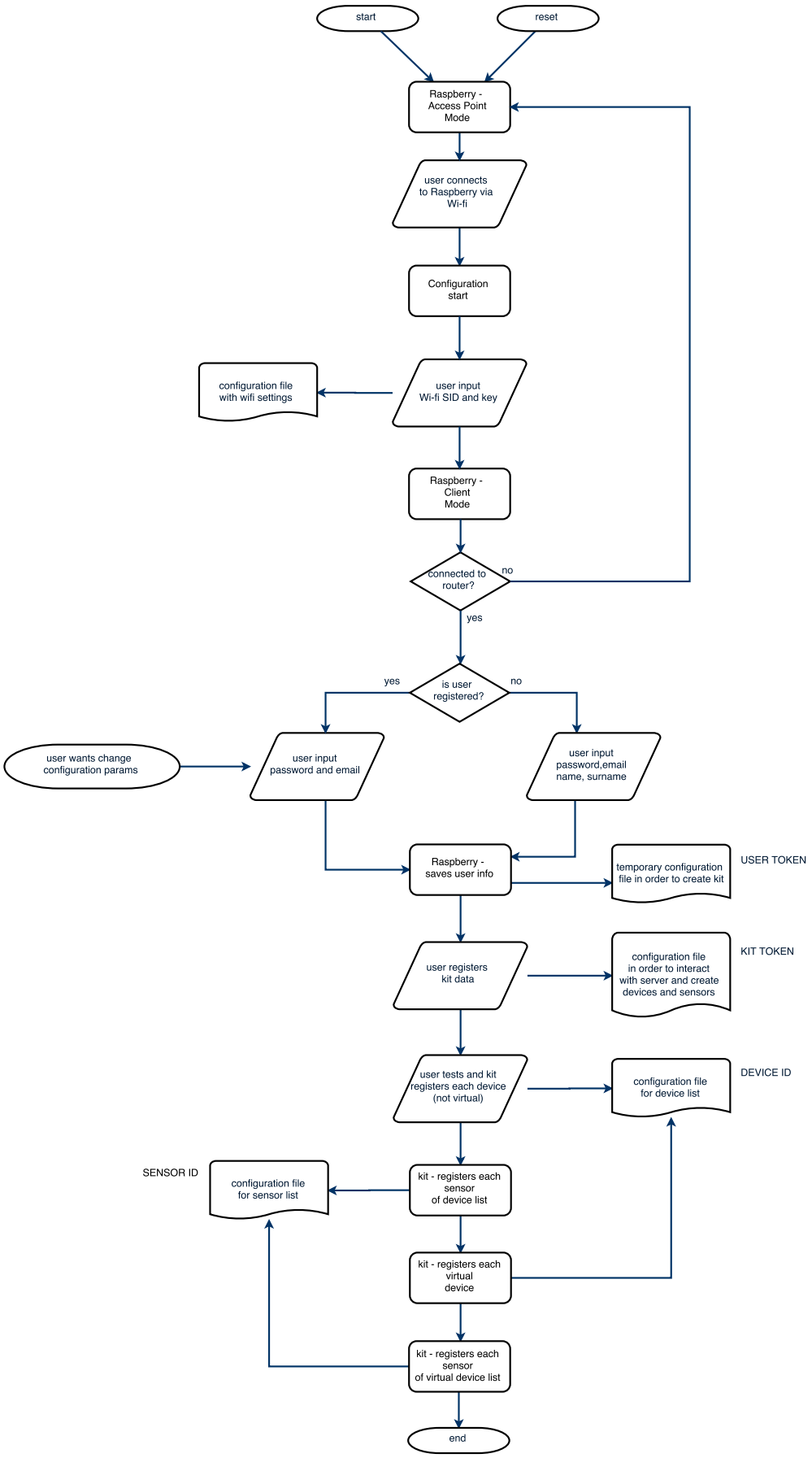


Figura 2.11 - Diagramma di flusso del processo di configurazione iniziale del kit



Capitolo 3
SERVER DI RACCOLTA DATI

Nel terzo capitolo viene descritto il progetto software del server di raccolta dati. Quest'ultimo, oltre ad esporre al pubblico un'interfaccia di tipo RESTful, distribuisce quelle che in gergo si chiamano risorse statiche, cioè file HTML, CSS e JS che insieme compongono l'interfaccia grafica di consultazione dei dati. Il codice sorgente del progetto si trova in una *code repository* sotto versionamento GIT. Nel paragrafo 3.1 viene presentata la struttura del backend, in particolare si prosegue con una panoramica dell'intera architettura del server (sottoparagrafo 3.1.1), per finire con la descrizione dell'API. Invece nel paragrafo 3.2 si parla del frontend, cioè l'interfaccia che permette all'utente finale di consultare le informazioni raccolte dal kit e modificare parametri legati all'account. Nel sottoparagrafo 3.2.1 viene trattato il concetto delle Single Page Application che è alla base delle web app moderne. Infine, nel sottoparagrafo 3.2.2 si descrive la struttura dei file sorgenti inerenti la parte frontend.

3.1 Backend

3.1.1 Architettura

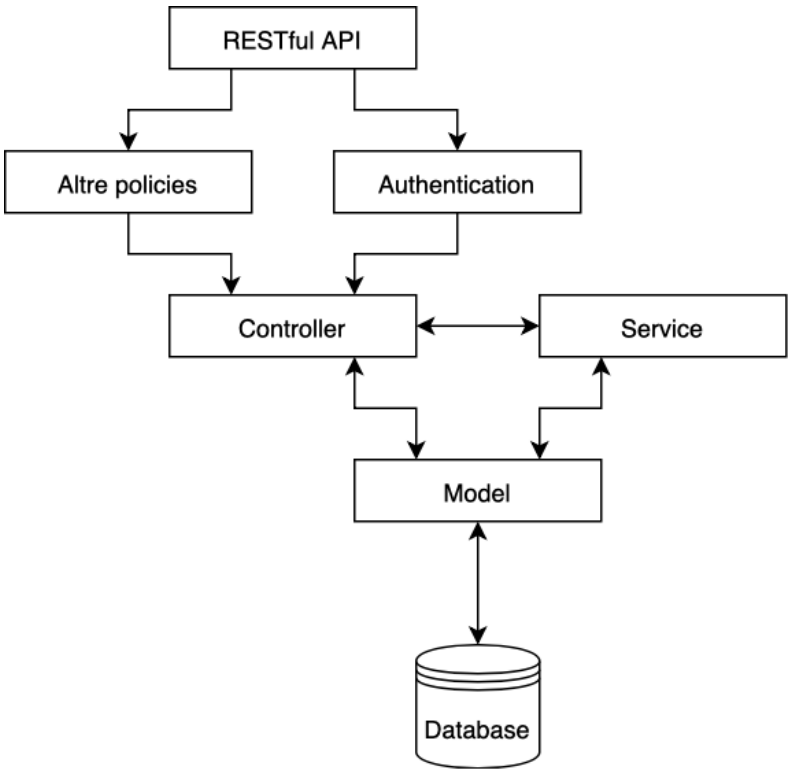
Il progetto contenente il codice sorgente per il server di raccolta dati è chiamato el_care. Esattamente come il server di configurazione (chiamato anche wizard), descritto nel sottoparagrafo 2.2.3, anche questo progetto è un'applicazione NodeJS versione 6.12.3 sviluppata usando il framework SailsJS versione 0.12.3. L'idea che sta alla base dell'architettura del server è il principio di separazione dei compiti (in inglese *separation of concerns*) e il pattern MVC (Model-View-Controller). La separazione dei compiti è un principio usato per suddividere un programma software in sezioni distinte in modo tale che ognuna si occupi di un compito diverso. Tale concetto è noto nella programmazione ad oggetti con il termine incapsulamento, grazie al quale è possibile limitare l'accesso diretto agli elementi dell'oggetto. Nel caso dello sviluppo del server di raccolta dati, la separazione dei compiti permette di organizzare il codice suddiviso in livelli (o strati). Infatti il progetto è stato suddiviso in moduli autonomi che risolvono problemi specifici come l'interfacciamento con il database, la business logic, il controllo degli accessi, l'autenticazione, la generazione di eventi e la loro notifica. Tutto ciò a vantaggio della semplicità e della manutenibilità del codice, che sono parametri importanti della qualità di un software. Di seguito si analizzerà ciascuno di questi moduli.
La persistenza dei dati, oltre ad essere uno dei livelli dell'architettura, rappresenta la componente *model* nel pattern MVC. In questo progetto viene utilizzata una tecnica chiamata ORM (*Object-Relational Mapping*) che permette di interagire con il database mediante un'interfaccia orientata agli oggetti, si rimanda al capitolo 4 per una discussione più approfondita.
Parte della logica di business (in inglese *business logic*) è contenuta nei controllori. Il codice in comune viene separato in servizi, per chi ha familiarità con la programmazione ad oggetti si tratta di classi che svolgono attività comuni a più controller come, per esempio, la verifica del token e l'invio di notifiche. Ogni modello ha un suo controllore dedicato che permette di separare la definizione delle entità coinvolte nell'applicazione dal modo con cui interagiscono nel sistema.
Con il controllo degli accessi si intende la gestione dei permessi per ciascuna risorsa esposta dal server, infatti alcune di queste sono disponibili ad ogni utente, mentre l'accesso ad altre necessità

privilegi particolari. Per esempio, l'accesso ai file statici è senza restrizione, invece per richiedere i dati di un particolare kit è necessario che l'utente sia anche il legittimo proprietario del kit. Nell'ambito di un'app SailsJS, la gestione degli accessi viene attuata tramite il concetto di politiche (dall'inglese *policies*). Quest'ultime sono delle parti di codice che, sulla base dei dati inviati dal client, hanno il compito di decidere se accettare o rifiutare la richiesta. Più politiche possono essere concatenate insieme, in questo modo la richiesta viene accettata soltanto se tutte le condizioni sono verificate contemporaneamente. Se la richiesta viene accettata, allora viene passata al controller che restituirà il dato al client.
Una delle policies più importanti è quella che si occupa di autenticare gli utenti che accedono all'API del server. Nelle moderne applicazione web si sta abbandonando il meccanismo di autenticazione basato su sessione a favore di quello basato su Json Web Token (JWT). Questo standard, abbastanza recente, è uscito all'inizio del 2015 e consente al server di incapsulare le informazioni relative all'utente in un messaggio che viene firmato digitalmente. Un token è costituito da 3 parti:

1. Header: è un oggetto JSON in cui sono racchiuse le informazioni principali dell'algoritmo utilizzato per per preparare la firma criptata e includerla nel token stesso.
2. Payload: è un oggetto JSON che rappresenta le informazioni dell'utente autenticato.
3. Signature: qui c'è la parte più interessante, ovvero la firma con cui il server certifica che questo messaggio è stato scritto da lui e che non è stato modificato da nessun altro.

Per autenticare ogni chiamata, il client deve aggiungere un particolare header HTTP chiamato *Authorization* contenente un token valido. A quel punto, il server dovrà soltanto verificare la firma del token. D'altra parte l'autenticazione basata su sessione porta un carico di lavoro maggiore sul lato server, che dovrà memorizzare e gestire opportunamente diverse sessioni contemporaneamente, e aumenta la complessità del codice se vengono utilizzati più server per bilanciare il carico del sito.
Da come si può notare dal capitolo 2, è compito del gateway di generare alcuni degli eventi riguardanti la persona monitorata nell'abitazione. Ma anche il server di raccolta dati genera eventi e si occupa della notifica di quest'ultimi tramite messaggi email, SMS e Telegram. Questa distribuzione del carico di lavoro serve a decentralizzare, anche se in parte, l'elaborazione dei dati per ridurre i costi di gestione del server. Questa suddivisione dei compiti e l'elenco completo delle varie notifiche viene affrontata in modo più esaustivo nel capitolo 5.
Nella figura 3.1 è presente uno schema di riepilogo che elenca i moduli appena descritti e il modo con cui comunicano tra di loro.

Figura 3.1 - Scherma dell'architettura del server di raccolta dati



3.1.2 RESTful API

La descrizione dettagliata dell'API RESTful esposta dal server è stata generata con il tool apiDoc versione 6.9.0, è possibile consultare la documentazione nell'appendice A. Grazie a questo strumento, tramite opportuni commenti nel codice chiamati annotazioni, viene generata una pagina web che documenta l'API. Per generare la documentazione si deve aver installato il modulo di NodeJS apiDoc tramite NPM (Node Package Manager) con il comando shell `npm install@6.9.0 apidoc -g`. Il passo successivo consiste nella generazione dei file che descrivono l'API tramite il comando `apidoc -i api -o apidoc/`.

3.2 Frontend

3.2.1 Concetto di Single Page Application

Un'applicazione single-page è una applicazione web che interagisce con l'utente modificando dinamicamente la pagina corrente piuttosto che caricare nuove pagine del server, da qui il suo nome abbreviato con SPA (Single Page Application). Questo approccio rende la fruizione dei contenuti più fluida, simulando il comportamento di una applicazione desktop nativa. In una SPA tutto il codice necessario (HTML, JavaScript e CSS) viene scaricato al momento del caricamento della pagina e può essere conservato localmente dal web browser per garantire tempi di avvio più veloci. L'interazione con l'applicazione single-page molto spesso è accompagnata da chiamate HTTP al web server e da l'uso di canali di comunicazione full-duplex basati su TCP come la tecnologia WebSocket.

Un componente fondamentale di una applicazione di questo tipo è il router. Il suo compito è quello di interpretare correttamente le URL, caricare il *layout* opportuno e ripristinare lo stato dell'applicazione.
Una SPA è un'applicazione multipiattaforma e questo rappresenta un vantaggio significativo rispetto alle applicazione native. Con un solo codice è possibile distribuire la propria applicazione su qualsiasi dispositivo dotato di web browser. In aggiunta, per la pubblicazione di un'app single-page non è necessario affidarsi a servizi di distribuzione di software come Google Play, Windows Store o Apple Store, bensì l'applicazione è disponibile semplicemente seguendo un link. Anche il rilascio di aggiornamenti è immediato perché le SPA non devono essere installate sul dispositivo ma interpretate dal motore del browser. Risulta evidente che la compatibilità dell'applicazione riveste un ruolo importante durante la fase di sviluppo. Ed è buona norma testare l'applicazione su diversi browser prima del rilascio definitivo.
L'evoluzione dell'app single-page è chiamata PWA, acronimo di Progressive Web App. Il termine, coniato da Google, sta ad indicare applicazioni web che hanno un comportamento simile alle applicazioni native: sono un ibrido tra le normali pagine web e le applicazioni mobili. La differenza sostanziale tra una PWA e una SPA sta nel fatto che la prima sfrutta le nuove funzionalità dei moderni browser web, come Service Workers[21] e Web App Manifests[22], che permettono al sistema operativo di trattare queste web app come se fossero delle applicazioni native. Ma entrambe condividono i vantaggi precedentemente elencati.

3.2.2 Struttura del progetto

L'interfaccia utente, chiamata anche frontend, è stata implementata utilizzando AngularJS versione 1.5.8, si tratta di una versione datata perché lo sviluppo è iniziato nel 2015. In base alle caratteristiche descritte nel sottoparagrafo 3.2.1, il progetto è a tutti gli effetti un'app single-page. Il codice sorgente è sotto controllo di versionamento GIT e si trova nella stessa repository del backend raggiungibile all'indirizzo https://bitbucket.org/elitefabriano/el-care/src/master/. Nella tabella 3.1 viene descritto come è stato organizzato il codice sorgente del progetto, mentre di seguito vengono descritte le varie funzionalità offerte dall'app.
Dopo aver fatto il login tramite una finestra modale, l'utente si troverà una app che rispecchia il design pattern del *master-detail*, con una menu di navigazione laterale sempre presente, posizionato sulla parte sinistra dello schermo, mentre dall'altra parte è visualizzata l'interfaccia corrispondente alla voce selezionata. Le voci del menu di navigazione sono raggruppate in due sezioni: nella prima sezione troviamo le pagine riguardanti i dati della persona monitorata suddivisi per tematica mentre nella seconda sezione sono elencate le varie stanze della casa. Più in avanti verranno elencate le varie voci del menu con una breve didascalia dell'interfaccia corrispondente e, per dare una panoramica completa della UI, sono inclusi degli screenshot (figure 3.2 e 3.3).

PANORAMICA
In questa sezione l'utente può consultare un grafico a linea spezzata che mostra l'andamento della temperatura nelle varie stanza, e una breve panoramica sul riposo notturno e pomeridiano dell'anziano nonché le informazioni sull'eventuale presenza di ospiti autenticati all'interno dell'abitazione. In aggiunta è presente anche una grafico a torta che mostra come è distribuita la presenza della persona nelle varie stanze.

[21] Un Service Worker è un proxy di rete implementato in JavaScript nel browser web per gestire da programma le richieste HTTP.

[22] Il manifesto della web app è una specifica del W3C in formato JSON che fornisce agli sviluppatori il posto dove mettere i metadata associati con l'applicazione web facilitando l'indicizzazione dei motori di ricerca.

AVVISI
La pagina degli avvisi mostra semplicemente una lista degli allarmi ordinata per data.

STATISTICHE
In questa schermata si trovano un istogramma a barre con il numero delle ore di sonno e un grafico degli accessi in cucina, entrambi con dati raggruppati per giorni della settimana.

SONNO NOTTURNO
Come dice il titolo, la sezione mostra tutti le informazioni raccolte durante il sonno notturno. Qui possiamo trovare l'orario di inizio del sonno e quello in cui l'anziano si è alzato dal letto. Inoltre sono elencati parametri come la durata complessiva del sonno, la luminosità media nella stanza, eventuali interruzioni del sonno durante la notte, le stanze visitate durante le interruzioni e un'indicazione sulla regolarità del sonno.

TEMPO FUORI CASA
In questa sezione si trova un grafico che mostra l'assenza della persona nelle ultime 24 ore e un diagramma a barre con le ore trascorse fuori casa giorno per giorno nell'arco di una settimana.

ATTIVITÀ GIORNALIERA
La schermata raggruppa vari diagrammi che mostrano i dati dei sensori di movimento correlati con la presenza della persona nelle stanze. Lo scopo è quello di fornire informazioni sull'attività motoria dell'anziano con indicazioni sullo spostamento all'interno dell'abitazione.

ATTIVITÀ SOCIALE
La pagina contiene un grafico che mostra le apertura della porta d'ingresso nelle ultime 24 ore correlate con le autenticazioni degli ospiti.

PAGINA RELATIVA ALLA STANZA
Per ogni stanza della casa è presente una sezione dedicata. Qui si trovano le informazioni sul comfort percepito nella stanza, rilevate dal sensore di temperatura e umidità, nonché i dati captati dal sensore di movimento.

Figura 3.2 - Schermata dell'applicazione web con le informazioni riguardanti il sonno.

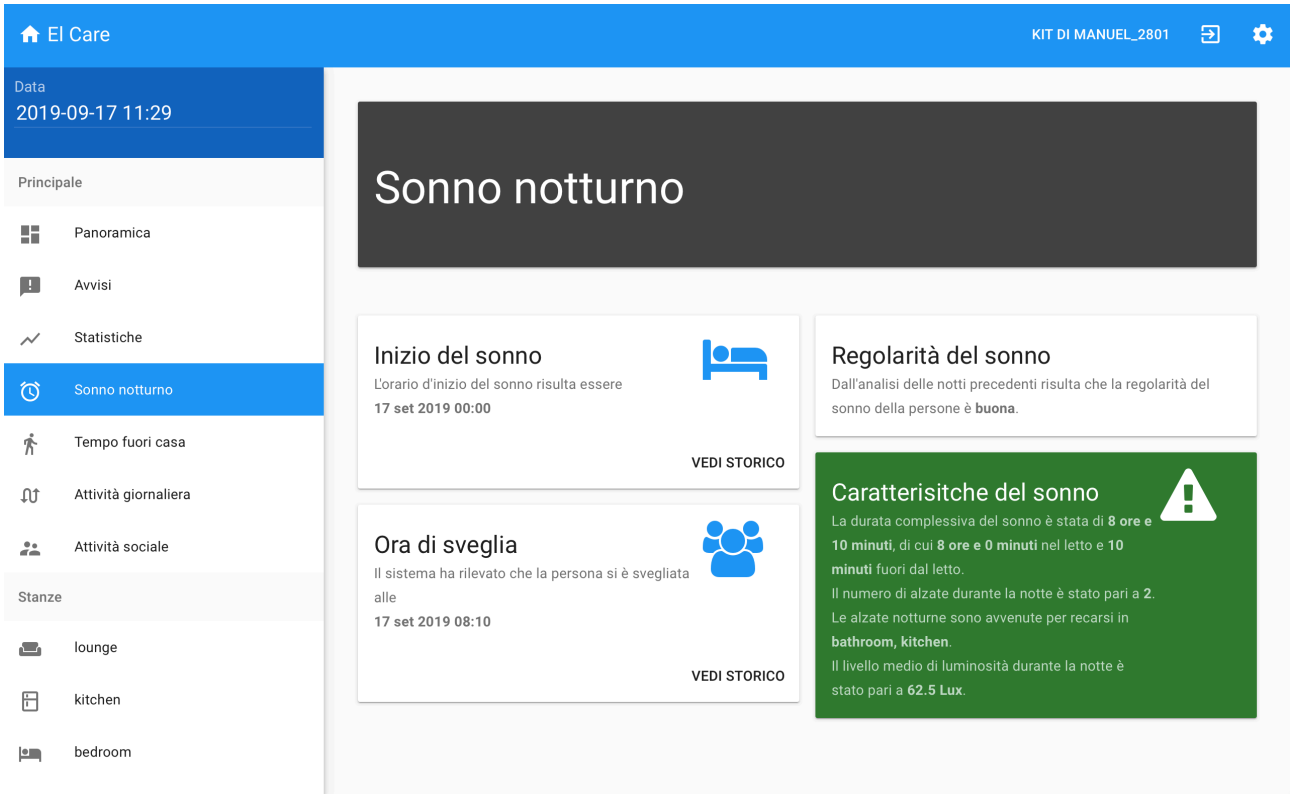


Figura 3.3 - Schermata dell'applicazione web con dati statistici settimanali.

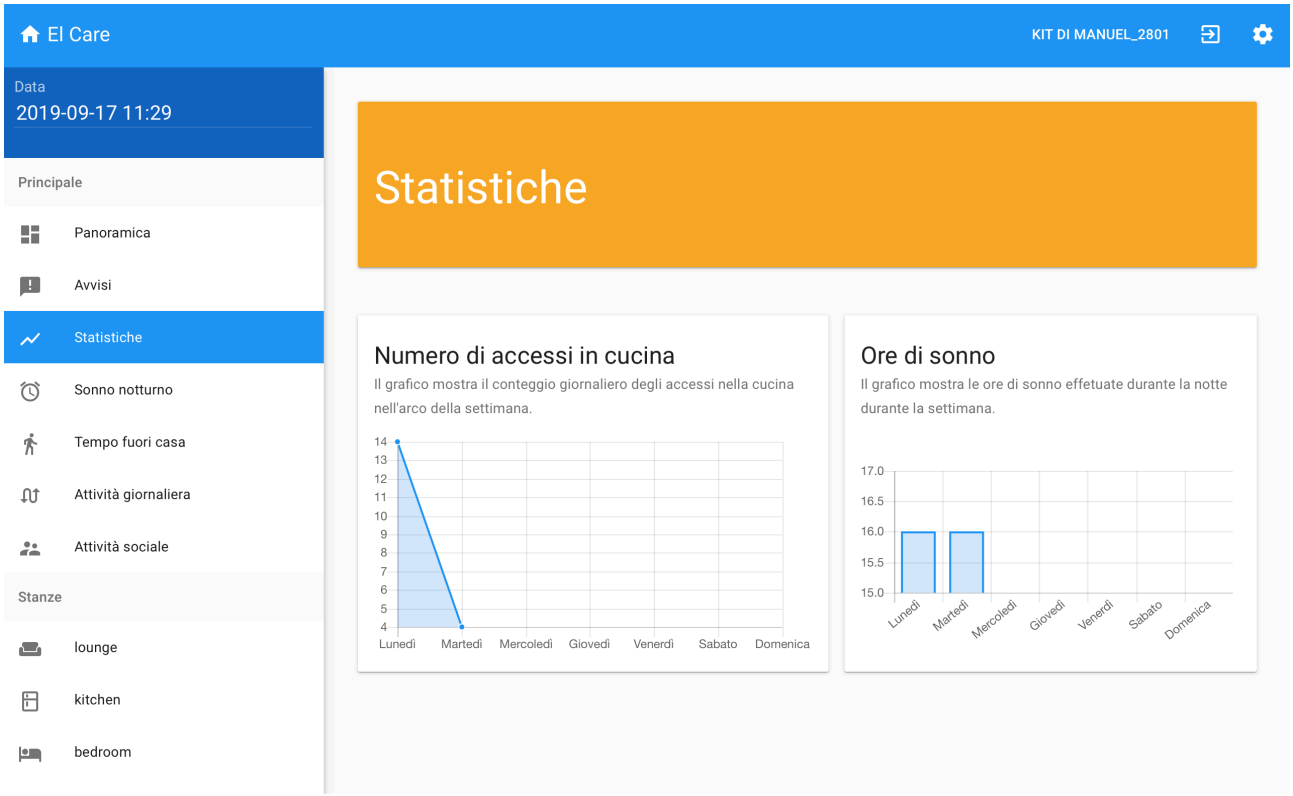


Tabella 3.1 - Struttura della directory contenente il codice dell’applicazione frontend

Nome	Descrizione
email_templates	Cartella contenente modelli di email, ovvero file HTML con espressioni per il template engine Handlebars.
images	Cartella contenente immagini.
partials	Cartella che contiene file HTML che rappresentano componenti della UI come la barra di navigazione e il menu laterale.
js	Cartella che contiene tutti i file JavaScript del frontend. Qui possiamo trovare i controllers, i services, la dichiarazione dell’app AngularJS e la struttura degli stati dell’app.
libs	Cartella per i moduli del frontend.
styles	Cartella per i file CSS.
templates	Cartella che contiene file HTML che rappresentano vari stati dell’applicazione.

Capitolo 4
DATABASE

Nel capitolo numero quattro si parla del database correlato al progetto del server di raccolta dati. Prima di elencare la struttura delle tabelle che compongono la base di dati, nel paragrafo 4.1 viene introdotto il concetto di adattatore e la tecnica usata per mappare le tabelle con gli oggetti JSON nativi del linguaggio JavaScript. I paragrafi 4.2 e 4.3 descrivono le varie entità e relazioni coinvolte nel database partendo dall'analisi delle singole tabelle fino allo schema ER.

4.1 Database adapter e ORM

Per interfacciare l'applicazione con una base di dati il framework SailsJS fa uso del concetto di *adapter* (in italiano adattatore) e ORM, acronimo di *Object-Relational Mapping*. Come già accennato nel sottoparagrafo 3.1.1, la persistenza dei dati è una delle componenti del pattern MVC, in particolare la componente chiamata *model*. Nel progetto sono state dichiarate particolari classi che servono a rappresentare i dati salvati nel database e a modellare la loro struttura. Si può dire che tali classi sono dei modelli di tabelle del database e sono scritte in linguaggio Javascript. La loro funzione principale è quella di permettere allo sviluppatore di gestire i dati contenuti nelle tabelle come se fossero degli oggetti, svincolandolo dalla conoscenza specifica dei vari comandi da passare al database. Ogni riga della tabella viene convertita in modo automatico in un oggetto JSON che rispecchia il suo modello, questo processo è chiamato mappatura. Un altro vantaggio di questa tecnica è il fatto che permette di scrivere codice completamente agnostico rispetto al tipo di database creando un livello di astrazione trasparente per lo sviluppatore. La tecnica ORM oggi viene impiegata in molti contesti, ad esempio nella programmazione Android di recente è stata rilasciata una libreria chiamata Room[23] che semplifica notevolmente l'accesso a database SQLite[24]. A differenza della libreria citata come esempio, che supporta soltanto una tipologia di database, SailsJS è capace di colloquiare con vari tipi di database grazie all'uso di adattatori appositi.
Un adattatore è definito come un dizionario (noto anche come oggetto JavaScript) con metodi utili per creare, leggere, aggiornare e eliminare elementi dal database. In base ai metodi implementati e alla loro completezza, si dice che gli adattatori implementano uno o più livelli dell'interfaccia di accesso alla base di dati. Ogni livello di interfaccia implica un contratto per implementare determinate funzionalità. Gli adattatori si concentrano principalmente sulla fornitura di metodi CRUD (*Create-Read-Update-Delete*) contestualizzati dal modello. Quelli ufficialmente supportati da SailsJS sono per database MySQL, PostgreSQL e MongoDB.
Finiamo questo paragrafo con una interessante funzionalità messa a disposizione dalla libreria che gestisce la persistenza dei dati, ovvero la migrazione. Nel corso dello sviluppo di un'app, è quasi sempre necessario apportare almeno una o più modifiche alla struttura del database. Tale modifica può avere un impatto più o meno significativo in base al database utilizzato. Ad esempio, se si aggiunge un nuovo attributo a una delle definizioni di un modello. Se quel modello è configurato per usare MongoDB, allora questo non è un grosso problema. Ma se quel modello è

---

[23] Room è una libreria per la persistenza dei dati usata in ambiente Android, maggiori informazioni sul sito https://developer.android.com/topic/libraries/architecture/room

[24] SQLite è una libreria molto leggera che implementa un DBMS di tipo SQL e può essere integrata in applicazioni per dispositivi mobili.

configurato per usare MySQL, allora c'è un passaggio in più: una colonna deve essere aggiunta alla tabella corrispondente (altrimenti alcuni metodi del modello smetteranno di funzionare). Pertanto, per un modello che utilizza MySQL, l'aggiunta di un attributo rappresenta una modifica sostanziale allo schema del database. Per sopperire a questo problema, SailsJS supporta l'auto-migrazione della base di dati e cerca di modificare opportunamente le tabelle senza rompere l'integrità dei dati. Quando questo non è possibile, si deve procedere manualmente modificando le singole tabelle.

4.2 Descrizione delle tabelle

Per questo progetto è stato scelto come RDBMS (acronimo di *Relational Database Management System*) il noto RDBMS MySQL. In particolare è stata installata la versione 5.7.28 per Linux a 64 bit. Nel paragrafo verranno elencate le varie tabelle, o per meglio dire entità, che compongono la base di dati. Per finire con l'analisi del database, nel paragrafo successivo (paragrafo 4.3) viene proposto lo schema entità-relazione, spesso abbreviato con schema o modello ER, usato nella fase di progettazione del database.
Il database è costituito dalle seguenti 7 tabelle.

TABELLA USER
L'entità User rappresenta l'utente che si è iscritto alla piattaforma durante la procedura di installazione del kit nella casa dell'anziano. Un utente può essere proprietario di più kit, da cui la relazione tra l'entità User e Kit è uno a molti. Per maggiori dettagli consultare la tabella 4.1 e la figura 4.1 che rappresenta il modello ER.

Tabella 4.1 - Descrizione degli attributi dell'entità User

Campo	Tipo	Descrizione
Uuid	UUID	Id univoco dell'utente.
CreatedAt	Date	Data di creazione del record sul database.
UpdatedAt	Date	Data di aggiornamento del record sul database.
Name	String	Nome dell'utente.
Surname	String	Cognome dell'utente.
Email	String	Email dell'utente.
Password	String	Hash della password.

TABELLA KIT
Questa tabella serve a memorizzare i dati di ogni kit e le informazioni della persona dato che un kit è in grado di monitorare soltanto un individuo. Sono state incluse in questa entità anche le liste contenenti i contatti da avvisare nel caso di eventi d'allarme. Per maggiori dettagli consultare la tabella 4.2 e la figura 4.1.

Tabella 4.2 - Descrizione degli attributi dell'entità Kit

Campo	Tipo	Descrizione
Uuid	UUID	Id univoco del kit.
CreatedAt	Date	Data di creazione del record sul database.

Campo	Tipo	Descrizione
UpdatedAt	Date	Data di aggiornamento del record sul database.
PersonName	String	Nome della persona monitorata.
PersonBirthdate	Date	Data di nascita della persona monitorata.
NotificationEmailList	Array	Lista di email per le notifiche.
NotificationPhoneList	Array	Lista di numeri telefonici per le notifiche Telegram e SMS.

TABELLA EVENTTYPE
Questa tabella conterrà tutti i tipi di allarme gestiti da un singolo kit, pertanto tra l'entità Kit e EventType vi è una relazione uno a molti. Sono presenti campi booleani per poter selezionare il meccanismo di notifica, infatti ogni tipo di evento può essere notificato in vari modi: via messaggio SMS, Telegram oppure tramite email. Per maggiori dettagli consultare la tabella 4.3 e la figura 4.1.

Tabella 4.3 - Descrizione degli attributi dell'entità EventType

Campo	Tipo	Descrizione
Uuid	UUID	Id univoco del tipo di evento.
CreatedAt	Date	Data di creazione del record sul database.
UpdatedAt	Date	Data di aggiornamento del record sul database.
Code	Integer	Codice del tipo di evento.
Description	String	Descrizione del tipo di evento.
IsSMSEnabled	Boolean	Indica se il tipo di evento deve essere notificato via SMS.
IsTelegramEnabled	Boolean	Indica se il tipo di evento deve essere notificato via Telegram.
IsEmailEnabled	Boolean	Indica se il tipo di evento deve essere notificato via email.

TABELLA EVENT
In questa tabella verrano salvati gli allarmi, chiamati anche eventi, che sono stati lanciati dal kit. Vi è un forte legame con la tabella EventType in quanto l'entità Event rappresenta il verificarsi di un tipo di allarme che viene specificato dalla tabella EventType. Per maggiori dettagli consultare la tabella 4.4 e la figura 4.1.

Tabella 4.4 - Descrizione degli attributi dell'entità Event

Campo	Tipo	Descrizione
Uuid	UUID	Id univoco dell'evento.
CreatedAt	Date	Data di creazione del record sul database.
UpdatedAt	Date	Data di aggiornamento del record sul database.
Timestamp	Date	Data dell'evento.

TABELLA DEVICE
L'entità Device rappresenta i dispositivi fisici e virtuali facenti parte di un kit. Un dispositivo può avere molteplici sensori che sono contenuti nella tabella apposita, ma deve per appartenere obbligatoriamente ad un unico kit. Per maggiori dettagli consultare la tabella 4.5 e la figura 4.1.

Tabella 4.5 - Descrizione degli attributi dell'entità Device

Campo	Tipo	Descrizione
Uuid	UUID	Id univoco del dispositivo.
CreatedAt	Date	Data di creazione del record sul database.
UpdatedAt	Date	Data di aggiornamento del record sul database.
LocalId	String	Identificativo generato dal Z-Way Home Automation.
NodeId	Integer	Identificativo del nodo nella rete Z-wave.
ManufacturerId	String	Identificativo del produttore del device.
ManufacturerProductId	String	Identificativo del prodotto.
Position	String	Posizione del dispositivo all'interno dell'abitazione. Le posizioni definite sono: lounge, kitchen, bedroom, bathroom, entrance, corridor.
GivenName	String	Friendly name dato al device (generato dal Z-Way Home Automation).
IsActive	Boolean	Indica se il dispositivo è funzionante oppure no.

TABELLA SENSOR
Questa tabella serve a memorizzare i vari sensori che fanno parte di uno stesso dispositivo fisico o virtuale. Per maggiori dettagli consultare la tabella 4.6 e la figura 4.1.

Tabella 4.6 - Descrizione degli attributi dell'entità Sensor

Campo	Tipo	Descrizione
Uuid	UUID	Id univoco del sensore.
CreatedAt	Date	Data di creazione del record sul database.
UpdatedAt	Date	Data di aggiornamento del record sul database.
LocalId	String	Identificativo generato dal Z-Way Home Automation.
Type	String	Tipo del sensore, può essere binary oppure multilevel.
Category	String	Categorie del sensore, valori possibili sono: temperature, luminosity, door, bed, location, power, cumulated_energy, energy, time, repetition, presence, meter, switch, alarm, battery, motion, virtual.

TABELLA DATA

L’entità Data rappresenta i dati provenienti dai sensori, nonché le misure. Risulta evidente che tale entità è legata all’entità Sensor tramite la un relazione di tipo molti-a-uno: che è equivalente a dire che una misura può appartenere ad un sensore ed un sensore può avere più misure. Per maggiori dettagli consultare la tabella 4.7 e la figura 4.1.

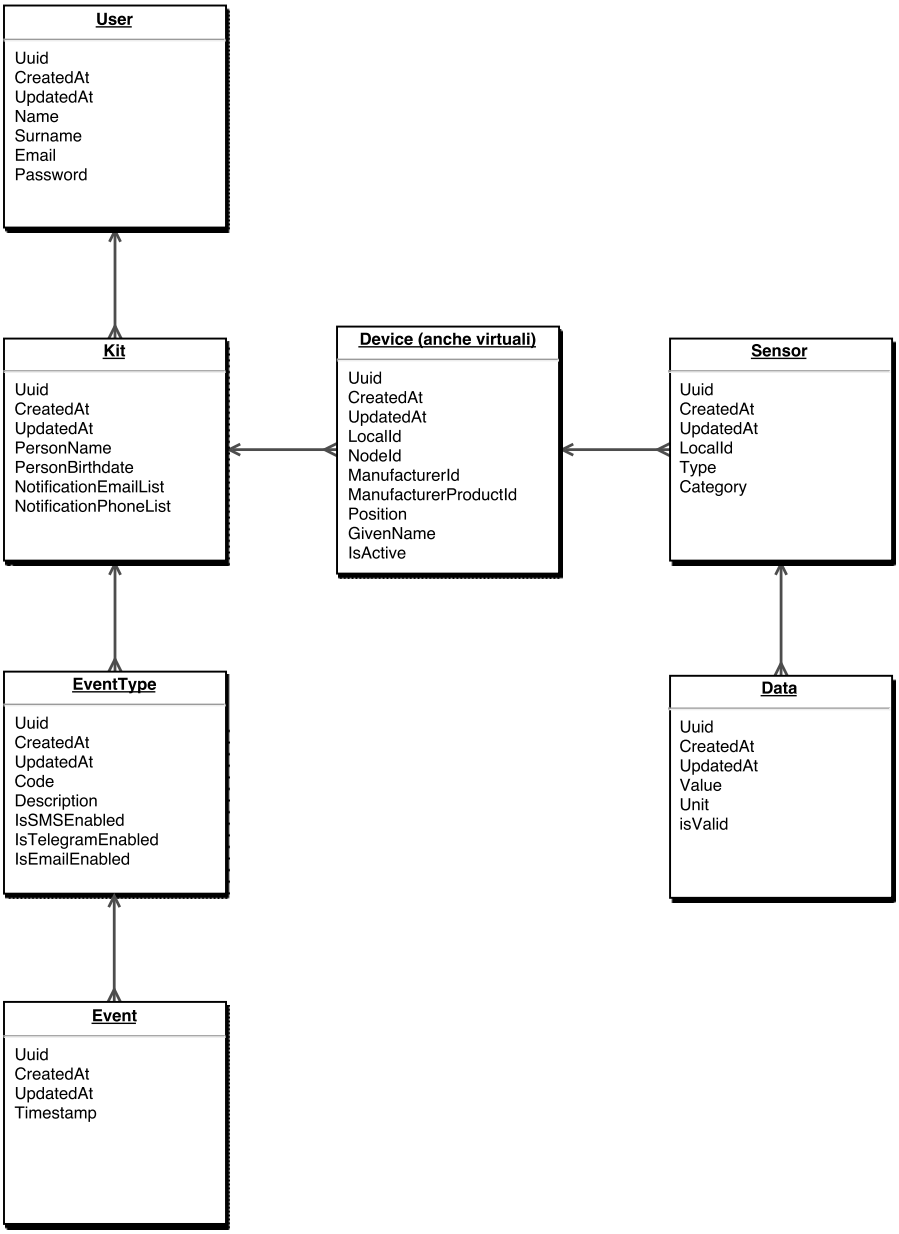
Tabella 4.7 - Descrizione degli attributi dell’entità Data

Campo	Tipo	Descrizione
Uuid	UUID	Id univoco del dato.
CreatedAt	Date	Data di creazione del record sul database.
UpdatedAt	Date	Data di aggiornamento del record sul database.
Value	String	String che rappresenta il valore del dato.
Unit	String	L’unità di misura del dato.
IsValid	Boolean	Indica se il dato è valido, quindi se deve essere incluso dai calcoli e durante la valutazione degli eventi.

4.3 Schema ER

La figura 4.1 è una rappresentazione diagrammatica che evidenzia le entità, le relazioni tra entità e gli attributi di entrambi. Questo tipo di diagramma è particolarmente adatto a database di tipo relazionale come MySQL perché permette di modellare in modo efficace complesse relazioni tra le entità. A partire da questo schema sono stati creati i modelli in linguaggio Javascript e, mediante l’adapter MySQL introdotto nel paragrafo 4.1, è stato generato in modo automatico un database apposito.

Figura 4.1 - Schema entità-relazione del database



Capitolo 5
SISTEMA DI NOTIFICHE

In questo capitolo viene tratto con più dettaglio l'argomento degli allarmi, o per meglio dire quello della notifica del verificarsi di eventi anomali. Il sistema monitora costantemente l'anziano e l'ambiente domestico, e in aggiunta a ciò, ogni misura provenenti dai sensori fisici e virtuali viene validata e verificata da un sistema di regole. Se una regola non è rispettata allora si genera un evento, da cui scaturirà una notifica a uno o più dei contatti registrati. Attualmente sono stati previsti tre canali di notifica: via email, via SMS e via messaggio Telegram, ma il supporto ad altri metodi, come notifiche push e feed RSS, può essere esteso facilmente. Mentre la validazione delle regole, nonché l'eventuale generazione di eventi, è distribuita tra il gateway presente nell'abitazione e il server di raccolta dati, il compito della notificazione degli eventi è riservato al server. Nel paragrafo 5.1 sono elencate le motivazioni alla base della suddivisione dei compiti tra il gateway e il server di raccolta dati in merito alla generazione degli eventi, inoltre è inclusa una tabella con tutti gli eventi che il sistema è capace di rilevare. Nel secondo paragrafo di questo capitolo, cioè il paragrafo 5.2, si parla dei sistemi utilizzati per integrare i 3 metodi di notifica.

5.1 Generazione di allarmi

Spesso si tende a confondere il concetto di evento con quello di notifica. Dal capitolo che tratta la struttura del database (capitolo numero 4) risulta chiaro che l'evento, rappresentato dall'omonima tabella, è un dato che viene salvato allo stesso modo di una misurazione provenenti da un sensore. Però, a differenza di quest'ultima, un evento viene generato da un sistema di regole distribuito tra il server di raccolta dati e il modulo data-ending e non da un sensore fisico o virtuale. Seguendo il principio del *edge computing*, la verifica delle regole è stata distribuita per il 70% sul Raspberry Pi, mentre il restante 30% viene valutato sul server di raccolta dati. Alla base di questa suddivisione vi è il fatto che il modulo data-ending non dispone di un database interno. Infatti per determinate regole è necessario l'accesso a misure registrate anche 15 giorni prima, e il data-sending è capace soltanto di valutare regole che coinvolgono misure istantanee. Nel prossimo aggiornamento del software è prevista l'implementazione di una database locale per svincolare il server di raccolta dati dalla verifica delle regole. Nella tabella 5.2 è presente un elenco dettagliato con gli eventi gestiti attualmente dal sistema.
Molto spesso il costo dell'affitto di una macchina virtuale, dove poter eseguire il codice del server, si basa sulla capacità di calcolo e sul tempo effettivo di utilizzo. E' difficile ridurre il tempo di utilizzo di un'istanza visto che il servizio deve essere sempre disponibile, quindi l'unico parametro realmente variabile è la capacità di calcolo. La tipologia di processore e la quantità di memoria RAM spesso sono indicatori base della capacità di calcolo, e la maggior parte dei provider di servizi web è solita suddividere la propria offerta in livelli. In genere, al primo livello troviamo le configurazioni con capacità di elaborazione ridotte, man mano che si sale troviamo macchine con caratteristiche più elevate ma costi di utilizzo maggiori. Ad esempio, Amazon Web Services, chiamato anche AWS, è un provider di servizi web che propone ben 275 configurazioni diverse di macchine virtuali EC2 (Elastic Compute Cloud)[25]. Quindi il beneficio maggiore del *edge computing* è la riduzione del carico di lavoro sul server a favore di una riduzione del costo necessario a mantenere attivo il servizio.

[25] L'offerta di AWS è in continua evoluzione, per maggiori informazioni visitare il sito https://aws.amazon.com/it/ec2/

Tabella 5.2 - Descrizione degli allarmi gestiti dal sistema

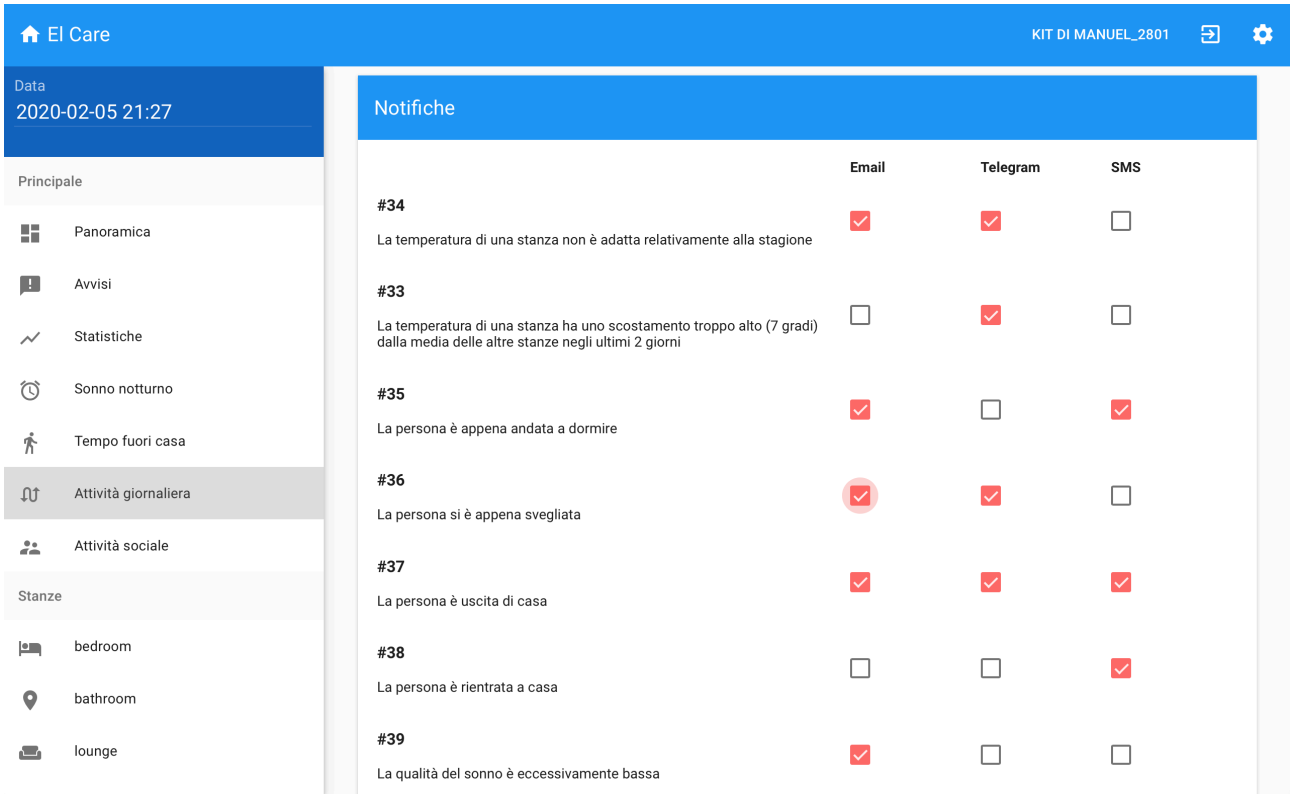
Id evento	Valutazione	Descrizione
33	Server	La temperatura di una stanza superiore di almeno 7°C dalla media delle altre stanza. La media è calcolata in base ai dati degli ultimi 2 giorni.
34	Server	La temperatura di una stanza ha una valore fuori norma tenendo conto della stagione corrente.
35	Client	La persona si è coricata a letto.
36	Client	L’anziano si è alzato dal letto
37	Client	L’anziano ha lasciato l’abitazione
38	Client	La persona monitorata è rientrata a casa dopo un periodo di assenza.
39	Server	La qualità del sonno è eccessivamente bassa.
40	Client	L’anziano è fuori casa da più di 5 ore consecutive.
41	Client	La persona non è rientrata prima del coprifuoco. Per ogni kit è possibile stabile un coprifuoco per ogni giorno della settimana.
42	Client	L'anziano è uscito dopo il coprifuoco e non è rientrato entro 10 minuti.
43	Client	La persona è a letto per più di 4 ore consecutive durante il giorno.
44	Server	La persona è sul letto durante il giorno per più di 2 ore rispetto alla media dei riposi diurni negli ultimi 15 giorni. Rimanere a letto è inteso cumulativo. La notifica viene ripetuta dopo ogni ora;
45	Client	L’anziano rimane troppo tempo all’interno della stessa stanza: 4 ore nel bagno, 6 ore nel soggiorno soggiorno, 9 ore in cucina, 2 ore e mezzo continuative in bagno. La notifica viene ripetuta dopo ogni mezz’ora.
46	Server	L’indice di movimento della persona all’interno della casa è molto più basso della media rispetto agli ultimi 15 giorni (quando la persona si trova fuori casa, viene considerato come se si stesse muovendo).
47	Client	La persona monitorata non entra in cucina negli orari dei pasti (11.00-14.00 per almeno 20 min e 17.00-21.00 per 20min). La valutazione dell’evento avviene alle 21:00.
48	Client	Uso insufficiente del bagno durante la giornata, conteggio delle presenze nella stanza da bagno della durata di almeno 1 minuto inferiore a 3 volte al giorno. La valutazione dell’evento avviene a mezzanotte.
49	Client	La porta è stata aperta per più di una volta nell’arco di un intervallo definito di 10 minuti.
50	Client	Un ospite si è autenticato.
51	Client	Un ospite è uscito dall’abitazione.
52	Client	Il numero di aperture della porta, senza autenticazione degli ospiti, è maggiore o uguale a 2 negli ultimi 15 minuti e la presenza della persona rimane in casa.
53	Client	La porta d’ingresso è rimasta aperta per più di 30 minuti e la presenza della persona è in casa.
54	Client	La porta è aperta per più di 15 minuti e la presenza della persona è non in casa.

Id evento	Valutazione	Descrizione
55	Client	La porta è stata aperta, nessuno si è autenticato nell'arco di 6 minuti e c'è presenza in casa.
56	Server	La temperatura di una stanza superiore di almeno 7°C dalla media delle altre stanza. La media è calcolata in base ai dati degli ultimi 2 giorni.
57	Server	La temperatura di una stanza è troppo bassa o troppo alta per 2 ore consecutive rispetto a valori scelti dall'utente in fase di configurazione.
58	Server	Il kit dell'anziano non è connesso alla rete.

5.2 Metodi di notifica

Ogni evento, dopo essere stato registrato sul database, viene notificato tramite i 3 canali citati all'inizio del capitolo: email, SMS e Telegram. Durante la fase di installazione del sistema, ma anche successivamente nella pagina delle impostazioni, è possibile registrare i contatti che riceveranno le notifiche. Infine, nella sezione notifiche della pagina di configurazione, l'utente può scegliere quali eventi devono essere notificati e il metodo di notifica. Nella figura 5.1 è mostrata la relativa interfaccia utente.

Figura 5.1 - Schermata dell'applicazione web con le informazioni riguardanti il sonno.

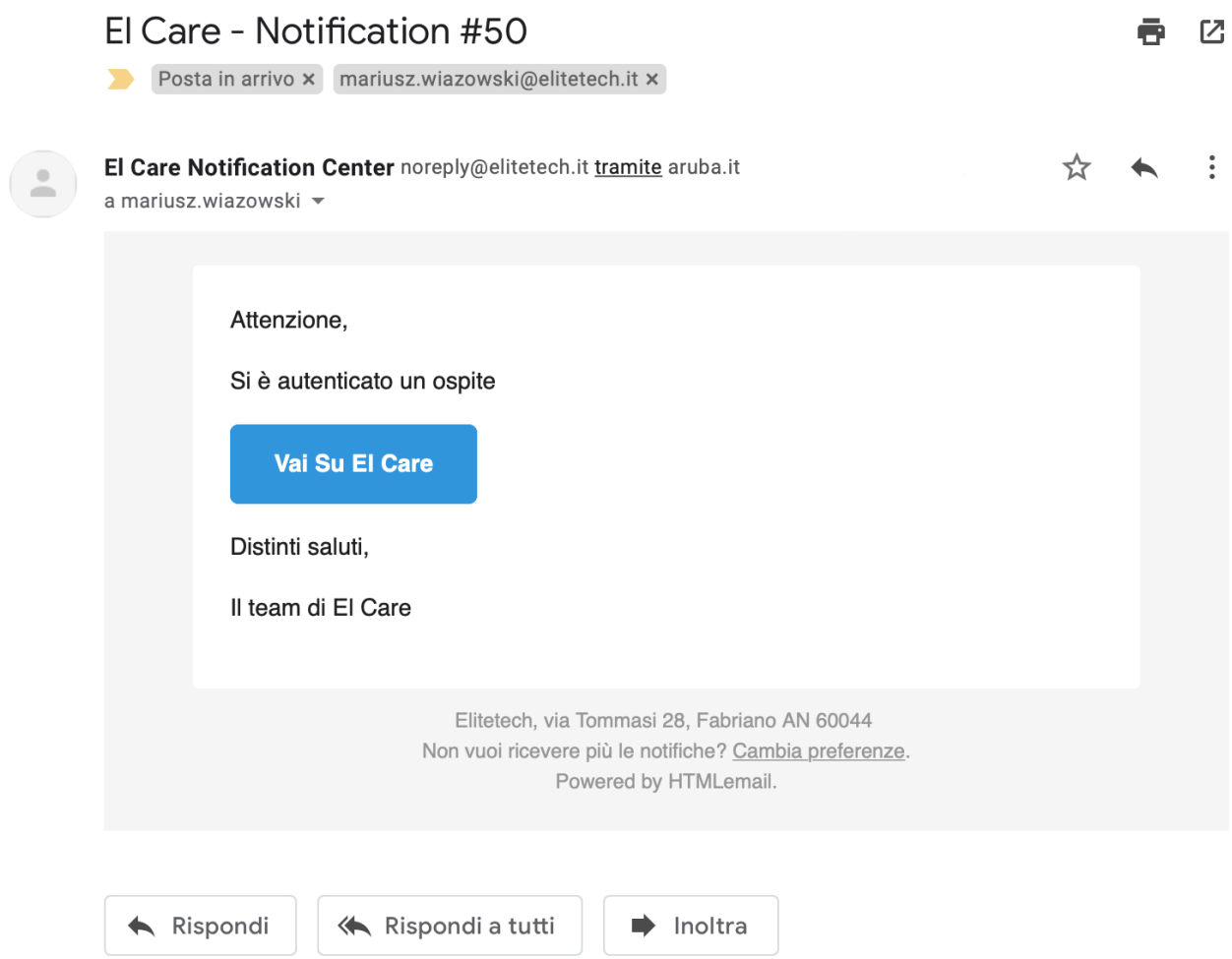


Indifferentemente dal canale utilizzato, ogni notifica presenta la medesima struttura:

- Titolo: contiene il l'identificativo dell'evento;
- Corpo: presenta una descrizione dell'allarme verificatosi;
- Link: è un collegamento diretto alla relativa pagina dell'applicazione web.

Per l'invio delle notifiche via email è stato utilizzato il modulo NodeJS chiamato nodemailer che permette l'invio di email contenenti messaggi in formato HTML. Infatti, per questo tipo di notifiche è stato utilizzato un template HTML con lo scopo di offrire un'esperienza utente più completa, un esempio è mostrato in figura 5.2. L'invio di notifiche SMS è possibile grazie a Twilio[26], un provider di servizi di telefonia, SMS e VoIP integrabili con software moderni tramite una web API. Invece, l'integrazione con Telegram si basa sul concetto di bot, cioè un programma che cerca di emulare il comportante di un utente reale della chat. La persona dovrà aggiungere tra i propri contatti questo bot in modo da ricevere le notifiche da esso tramite semplici messaggi in chat.

Figura 5.2 - Esempio di notifica email relativa all'evento di autenticazione di un ospite



[26] https://www.twilio.com/sms - Maggiori informazioni sul servizio di SMS offerto da Twilio.

Capitolo 6
CONCLUSIONI E SVILUPPI FUTURI

Lo sviluppo del sistema Nexty ha dimostrato come, a partire da un problema sociale legato all'invecchiamento della popolazione, si possano fornire informazioni sulle abitudini della persona monitorata e migliorarne la qualità della vita attraverso l'integrazione di semplici sensori all'interno di un'abitazione. La sfida più grande è stata quella di realizzare un sistema domotico realmente utile ed economico per le persone più anziane e fragili. Per questo motivo si è scelto di partire da un progetto domotico open-source e modulare, lo Z-Way Home Automation. Inoltre la scelta dei sensori è stata motivata anche dal costo, oltre che dalle specifiche di progetto.
Con l'installazione di due esemplari del kit si è conclusa la fase di sperimentazione del prototipo durata 2 mesi. Ciò ha permesso di perfezionare il sistema e comprendere le sue limitazioni, ma ha fornito anche spunto per l'aggiunta di nuove funzionalità come l'integrazione dei comandi per i dispositivi Z-Wave che li supportano e l'implementazione di notifiche push. A ulteriore dimostrazione del fatto che il sistema di notifiche è un elemento chiave del kit.
La più grande limitazione del sistema Nexty è l'impossibilità di monitorare più di una persona all'interno dell'ambiente domestico, ma questo è il risultato del compromesso dell'uso di sensori ad infrarossi passivi (sensori PIR) per il rilevamento della presenza. Per sopperire a questa limitazione si dovrebbe effettuare un nuovo studio di fattibilità del progetto e, per rilevare la presenza di più persone contemporaneamente, si dovrebbero adoperare soluzioni tecnologiche più avanzate come per esempio la visione artificiale. E' importante notare che questa scelta renderebbe sicuramente il sistema più invasivo e meno immediato nell'installazione, visto che ad oggi è difficile trovare telecamere alimentate a batteria.
In merito agli sviluppi futuri del progetto, è in atto una seconda fase di sperimentazione che coinvolgerà 10 installazioni presso una casa di cura. Si tratta di una cooperazione con un istituto sanitario con l'obiettivo di integrare il controllo remoto del kit e implementare l'autenticazione multiutente. In questo modo più utenti, ad esempio il familiare e l'operatore sanitario, disporranno di account sperati con privilegi diversi per la consultazione dei dati.

Bibliografia e Sitografia

Forum ufficiali Raspberry Pi https://www.raspberrypi.org/forums/
Community italiana Raspberry Pi https://forum.raspberryitaly.com
Portale di progetti DIY https://www.instructables.com
Modulo Razberry https://z-wave.me/products/razberry/
Documentazione Z-Way https://www.generationrobots.com/media/Domotique/Z-Way-developers-documentation.pdf
Sails.js https://sailsjs.com
Architettura ARM https://it.wikipedia.org/wiki/Architettura_ARM
Z-Wave https://it.wikipedia.org/wiki/Z-Wave
HID https://en.wikipedia.org/wiki/Human_interface_device
RFID https://it.wikipedia.org/wiki/Radio-frequency_identification
SIM800 https://www.itead.cc/wiki/RPI_SIM800_GSM/GPRS_ADD-ON_V2.0
Hayes command set https://en.wikipedia.org/wiki/Hayes_command_set#GSM
GPIO https://it.wikipedia.org/wiki/General_Purpose_Input/Output
Separazione dei concetti https://en.wikipedia.org/wiki/Separation_of_concerns
MongoDB https://www.mongodb.com/what-is-mongodb

Ringraziamenti

Vorrei ringraziare ogni persona che ha creduto in me e che mi ha dato la forza e il coraggio per non abbandonare gli studi. In modo speciale la mia fidanzata Anna e mia madre che mi hanno supportato in questo lungo percorso quando più ne avevo bisogno.

Appendice A

Nexty

Nexty API

Data

DataCreate - Data create

Is used to create new data. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/kit/:kit/sensor/:sensor/data
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Data-DataCreate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`value`	String	String value representing data.
`unit`	String	Data unit.
`isValid`	Boolean	Indicates if data is valid an has to be included in event evaluation.

- Success-Response-Example: [#success-examples-Data-DataCreate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "value": "32",
                        "unit": "°C",
                        "isValid": true,
                        "id": 3247,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
```

```
}
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Data-DataCreate-0_0_0-0]
- Error-Response: [#error-examples-Data-DataCreate-0_0_0-1]
- Error-Response: [#error-examples-Data-DataCreate-0_0_0-2]
- Error-Response: [#error-examples-Data-DataCreate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{

  "error": "No permission"
}
```

DataDestroy - Data destroy

Is used to delete data.

DELETE

```
/kit/:kit/data/:data
```

Permessi: kitOwner ☐

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Data-DataDestroy-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Data-DataDestroy-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "value": "32",
                        "unit": "°C",
                        "isValid": true,
                        "id": 3247,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Data-DataDestroy-0_0_0-0]
- Error-Response: [#error-examples-Data-DataDestroy-0_0_0-1]
- Error-Response: [#error-examples-Data-DataDestroy-0_0_0-2]
- Error-Response: [#error-examples-Data-DataDestroy-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
```

```
  "error": "No permission"
}
```

DataFind - Data find

Is used to find kit data that matches search criteria.

GET

```
/kit/:kit/data
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Data-DataFind-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

URL Parameters

Campo	Tipo	Descrizione
`skip`	Integer	The number of records to skip (useful for pagination).
`limit`	Integer	The maximum number of records to send back (useful for pagination). Defaults to 30.
`where`	String	Instead of filtering based on a specific attribute, you may instead choose to provide a where parameter with a Waterline WHERE criteria object, encoded as a JSON string.
`sort`	String	The sort order. By default, returned records are sorted by primary key value in ascending order.

- Request-Example: [#parameter-examples-Data-DataFind-0_0_0-0]

```
GET /kit/8/data?sort=createdAt DESC&limit=30&where={"createdAt":{">=":"2017-07-29T10
```

- Success-Response-Example: [#success-examples-Data-DataFind-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
       [
                       {
                                "value": "32",
```

```
                                "unit": "°C",
                                "isValid": true,
                                "id": 3247,
                                "createdAt": "2017-06-29T10:31:07.000Z",
                                "updatedAt": "2017-06-29T10:31:07.000Z"
                                }
                        ]
                }
```


Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Data-DataFind-0_0_0-0]
- Error-Response: [#error-examples-Data-DataFind-0_0_0-1]
- Error-Response: [#error-examples-Data-DataFind-0_0_0-2]
- Error-Response: [#error-examples-Data-DataFind-0_0_0-3]


```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```


DataUpdate - Data update

Is used to update data. Requires a JSON POST body and Content-Type header to be set to application/json.

PUT

```
/kit/:kit/data/:data
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Data-DataUpdate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`value`	String	String value representing data.
`unit`	String	Data unit.
`isValid`	Boolean	Indicates if data is valid an has to be included in event evaluation.

- Success-Response-Example: [#success-examples-Data-DataUpdate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                      "value": "32",
                      "unit": "°C",
                      "isValid": true,
                      "id": 3247,
                      "createdAt": "2017-06-29T10:31:07.000Z",
                      "updatedAt": "2017-06-29T10:31:07.000Z"
               }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Data-DataUpdate-0_0_0-0]
- Error-Response: [#error-examples-Data-DataUpdate-0_0_0-1]

- Error-Response: [#error-examples-Data-DataUpdate-0_0_0-2]
- Error-Response: [#error-examples-Data-DataUpdate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

Device

DeviceCreate - Device create

Is used to create a new device. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/kit/:kit/device
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Device-DeviceCreate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`localId`	String	

		Hardware id of device generated locally by Razberry.
`nodeId`	Number	Zwave node id.
`manufacturerId`	Number	Manufacturer id.
`manufacturerProductId`	Number	Product id.
`position`	String	Device position.
`givenName`	String	Device friendly name.
`isActive`	Boolean	Indicates if the device is working.

- Success-Response-Example: [#success-examples-Device-DeviceCreate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "uuidKit": 8,
                        "localId": "",
                        "nodeId": 10,
                        "manufacturerId": 271,
                        "manufacturerProductId": 4097,
                        "position": "kitchen",
                        "givenName": "Fibaro Motion Series kitchen",
                        "isActive": true,
                        "id": 47,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Device-DeviceCreate-0_0_0-0]
- Error-Response: [#error-examples-Device-DeviceCreate-0_0_0-1]
- Error-Response: [#error-examples-Device-DeviceCreate-0_0_0-2]
- Error-Response: [#error-examples-Device-DeviceCreate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
```

```
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

DeviceDestroy - Device destroy

Is used to delete a device.

DELETE

```
/kit/:kit/device/:device
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Device-DeviceDestroy-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Device-DeviceDestroy-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "uuidKit": 8,
                        "localId": "",
                        "nodeId": 10,
                        "manufacturerId": 271,
                        "manufacturerProductId": 4097,
                        "position": "kitchen",
                        "givenName": "Fibaro Motion Series kitchen",
                        "isActive": true,
                        "id": 47,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Device-DeviceDestroy-0_0_0-0]
- Error-Response: [#error-examples-Device-DeviceDestroy-0_0_0-1]
- Error-Response: [#error-examples-Device-DeviceDestroy-0_0_0-2]
- Error-Response: [#error-examples-Device-DeviceDestroy-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

DeviceUpdate - Device update

Is used to update a device. Requires a JSON POST body and Content-Type header to be set to application/json.

PUT

```
/kit/:kit/device/:device
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	

		Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Device-DeviceUpdate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`localId`	String	Hardware id of device generated locally by Razberry.
`nodeId`	Number	Zwave node id.
`manufacturerId`	Number	Manufacturer id.
`manufacturerProductId`	Number	Product id.
`position`	String	Device position.
`givenName`	String	Device friendly name.
`isActive`	Boolean	Indicates if the device is working.

- Success-Response-Example: [#success-examples-Device-DeviceUpdate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "uuidKit": 8,
                        "localId": "",
                        "nodeId": 10,
                        "manufacturerId": 271,
                        "manufacturerProductId": 4097,
                        "position": "lounge",
                        "givenName": "Fibaro Motion Series lounge",
                        "isActive": true,
                        "id": 47,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-07-13T14:55:07.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Device-DeviceUpdate-0_0_0-0]
- Error-Response: [#error-examples-Device-DeviceUpdate-0_0_0-1]
- Error-Response: [#error-examples-Device-DeviceUpdate-0_0_0-2]
- Error-Response: [#error-examples-Device-DeviceUpdate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

Event

EventCreate - Event create

Is used to create a new event. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/kit/:kit/event
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Event-EventCreate-0_0_0-0]

- Request Header Example: [#header-examples-Event-EventCreate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`timestamp`	Date	Event timestamp.

- Success-Response-Example: [#success-examples-Event-EventCreate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "timestamp": "2017-06-29T10:29:38.000Z",
                        "id": 43,
                        "createdAt": "2017-06-29T10:29:38.000Z",
                        "updatedAt": "2017-06-29T10:29:38.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Event-EventCreate-0_0_0-0]
- Error-Response: [#error-examples-Event-EventCreate-0_0_0-1]
- Error-Response: [#error-examples-Event-EventCreate-0_0_0-2]
- Error-Response: [#error-examples-Event-EventCreate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
```

```
  "error": "No permission"
}
```

EventDestroy - Event destroy

Is used to delete an event.

DELETE

```
/kit/:kit/event/:id
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Event-EventDestroy-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Event-EventDestroy-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "timestamp": "2017-06-29T10:29:38.000Z",
                        "id": 43,
                        "createdAt": "2017-06-29T10:29:38.000Z",
                        "updatedAt": "2017-06-29T10:29:38.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Event-EventDestroy-0_0_0-0]
- Error-Response: [#error-examples-Event-EventDestroy-0_0_0-1]
- Error-Response: [#error-examples-Event-EventDestroy-0_0_0-2]
- Error-Response: [#error-examples-Event-EventDestroy-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
```

```
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

EventFind - Event find

Is used to find events that matches search criteria.

GET

```
/kit/:kit/event
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Event-EventFind-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

URL Parameters

Campo	Tipo	Descrizione
`skip`	Integer	The number of records to skip (useful for pagination).
`limit`	Integer	The maximum number of records to send back (useful for pagination). Defaults to 30.
`where`	String	Instead of filtering based on a specific attribute, you may instead choose to provide a where parameter with a Waterline WHERE criteria object, encoded as a JSON string.

sort	String	The sort order. By default, returned records are sorted by primary key value in ascending order.

- Request-Example: [#parameter-examples-Event-EventFind-0_0_0-0]

```
GET /kit/8/event?sort=createdAt DESC&limit=30&where={"createdAt":{">=":"2017-07-29T1
```

- Success-Response-Example: [#success-examples-Event-EventFind-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                [
                        {
                                "timestamp": "2017-06-29T10:29:38.000Z",
                                "id": 43,
                                "uuidEventType": {
                                        "uuidKit": 8,
                                        "code": 33,
                                        "description": "La temperatura di una stanza
                                        "isSMSEnabled": false,
                                        "isTelegramEnabled": false,
                                        "isEmailEnabled": true,
                                        "id": 43,
                                        "createdAt": "2017-06-29T10:29:38.000Z",
                                        "updatedAt": "2017-06-29T10:29:38.000Z"
                                },
                                "createdAt": "2017-06-29T10:29:38.000Z",
                                "updatedAt": "2017-06-29T10:29:38.000Z"
                                }
                        ]
                }
```

Error 4xx

Nome	Descrizione
WrongTokenFormat	Format is 'Authorization: Bearer [token]'
NoAuthHeader	No Authorization header was found
InvalidToken	Invalid Token!
NoPermission	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Event-EventFind-0_0_0-0]
- Error-Response: [#error-examples-Event-EventFind-0_0_0-1]
- Error-Response: [#error-examples-Event-EventFind-0_0_0-2]
- Error-Response: [#error-examples-Event-EventFind-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
```

```
  error : "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

EventType

EventTypeCreate - EventType create

Is used to create a new type of event. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/kit/:kit/eventtype
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-EventType-EventTypeCreate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`code`	Number	Event type code.
`description`	String	Event type description.
`isSMSEnabled`	Boolean	Indicates if SMS notification is active.
`isTelegramEnabled`		

	Boolean	Indicates if Telegram notification is active.
isEmailEnabled	Boolean	Indicates if email notification is active.

- Success-Response-Example: [#success-examples-EventType-EventTypeCreate-0_0_0-0]

```
    HTTP/1.1 200 Ok
 {
                        "uuidKit": 8,
                        "code": 33,
                        "description": "La temperatura di una stanza ha uno scostame
                        "isSMSEnabled": false,
                        "isTelegramEnabled": false,
                        "isEmailEnabled": true,
                        "id": 43,
                        "createdAt": "2017-06-29T10:29:38.000Z",
                        "updatedAt": "2017-06-29T10:29:38.000Z"
                }
```

Error 4xx

Nome	Descrizione
WrongTokenFormat	Format is 'Authorization: Bearer [token]'
NoAuthHeader	No Authorization header was found
InvalidToken	Invalid Token!
NoPermission	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-EventType-EventTypeCreate-0_0_0-0]
- Error-Response: [#error-examples-EventType-EventTypeCreate-0_0_0-1]
- Error-Response: [#error-examples-EventType-EventTypeCreate-0_0_0-2]
- Error-Response: [#error-examples-EventType-EventTypeCreate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

```
}
```

EventTypeDelete - EventType delete

Is used to delete an event type.

DELETE

```
/kit/:kit/eventtype/:eventType
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-EventType-EventTypeDelete-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-EventType-EventTypeDelete-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "uuidKit": 8,
                        "code": 33,
                        "description": "La temperatura di una stanza ha uno scostame
                        "isSMSEnabled": false,
                        "isTelegramEnabled": false,
                        "isEmailEnabled": true,
                        "id": 43,
                        "createdAt": "2017-06-29T10:29:38.000Z",
                        "updatedAt": "2017-06-29T10:29:38.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-EventType-EventTypeDelete-0_0_0-0]
- Error-Response: [#error-examples-EventType-EventTypeDelete-0_0_0-1]
- Error-Response: [#error-examples-EventType-EventTypeDelete-0_0_0-2]
- Error-Response: [#error-examples-EventType-EventTypeDelete-0_0_0-3]

- Error-Response: [#error-examples-EventType-EventTypeDelete-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

EventTypeFind - EventType find

Is used to find kit event types that matches search criteria.

GET

```
/kit/:kit/eventtype
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-EventType-EventTypeFind-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

URL Parameters

Campo	Tipo	Descrizione
`skip`	Integer	The number of records to skip (useful for pagination).
`limit`	Integer	The maximum number of records to send back (useful for pagination). Defaults to 30.
`where`	String	Instead of filtering based on a specific attribute, you may instead choose to provide a where parameter with a

		instead choose to provide a where parameter with a Waterline WHERE criteria object, encoded as a JSON string.
`sort`	String	The sort order. By default, returned records are sorted by primary key value in ascending order.

- Request-Example: [#parameter-examples-EventType-EventTypeFind-0_0_0-0]

```
GET /kit/8/eventtype?sort=createdAt DESC&limit=30&where={"code":33}
```

- Success-Response-Example: [#success-examples-EventType-EventTypeFind-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
        [
                        {
                                "uuidKit": 8,
                                "code": 33,
                                "description": "La temperatura di una stanza ha uno
                                "isSMSEnabled": false,
                                "isTelegramEnabled": false,
                                "isEmailEnabled": true,
                                "id": 43,

                                "createdAt": "2017-06-29T10:29:38.000Z",
                                "updatedAt": "2017-06-29T10:29:38.000Z"
                                }
                        ]
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-EventType-EventTypeFind-0_0_0-0]
- Error-Response: [#error-examples-EventType-EventTypeFind-0_0_0-1]
- Error-Response: [#error-examples-EventType-EventTypeFind-0_0_0-2]
- Error-Response: [#error-examples-EventType-EventTypeFind-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

EventTypeUpdate - EventType update

Is used to update a type of event. Requires a JSON POST body and Content-Type header to be set to application/json.

PUT

```
/kit/:kit/eventtype/:eventType
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-EventType-EventTypeUpdate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`code`	Number	Event type code.
`description`	String	Event type description.
`isSMSEnabled`	Boolean	Indicates if SMS notification is active.
`isTelegramEnabled`	Boolean	Indicates if Telegram notification is active.
`isEmailEnabled`	Boolean	Indicates if email notification is active.

- Success-Response-Example: [#success-examples-EventType-EventTypeUpdate-0_0_0-0]

```
    HTTP/1.1 200 Ok
 {
                        "uuidKit": 8,
                        "code": 33,
                        "description": "La temperatura di una stanza ha uno scostame
                        "isSMSEnabled": false,
                        "isTelegramEnabled": false,
                        "isEmailEnabled": true,
                        "id": 43,
                        "createdAt": "2017-06-29T10:29:38.000Z",
                        "updatedAt": "2017-06-29T10:29:38.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-EventType-EventTypeUpdate-0_0_0-0]
- Error-Response: [#error-examples-EventType-EventTypeUpdate-0_0_0-1]
- Error-Response: [#error-examples-EventType-EventTypeUpdate-0_0_0-2]
- Error-Response: [#error-examples-EventType-EventTypeUpdate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

Kit

KitCreate - Kit create

Is used to create a new kit. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/kit
```

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Kit-KitCreate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`personName`	String	Observed person name.
`personBirthdate`	Date	Observed person birthdate.
`notificationEmailList`	Array	Array of email addresses.
`notificationPhoneList`	Array	Array of phone numbers.

- Success-Response-Example: [#success-examples-Kit-KitCreate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "personName": "Maria",
                        "personBirthdate": "1943-03-23T0:0:0.000Z",
                        "notificationEmailList": ["luigi@gmail.com"],

                        "notificationPhoneList": ["luigi@gmail.com"],
                        "createdAt": "2017-06-01T11:10:35.000Z",
                        "updatedAt": "2017-06-02T22:20:21.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'

`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

- Error-Response: [#error-examples-Kit-KitCreate-0_0_0-0]
- Error-Response: [#error-examples-Kit-KitCreate-0_0_0-1]
- Error-Response: [#error-examples-Kit-KitCreate-0_0_0-2]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

KitDestroy - Kit destroy

Is used to destroy a kit.

DELETE

```
/kit/:kit
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Kit-KitDestroy-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Kit-KitDestroy-0_0_0-0]

```
HTTP/1.1 204 No Content
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	

	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Kit-KitDestroy-0_0_0-0]
- Error-Response: [#error-examples-Kit-KitDestroy-0_0_0-1]
- Error-Response: [#error-examples-Kit-KitDestroy-0_0_0-2]
- Error-Response: [#error-examples-Kit-KitDestroy-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

KitDevices - Kit devices

Is used to get a list of devices within a kit.

GET

```
/kit/:kit/devices
```

Permessi: kitOwner ☐

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Kit-KitDevices-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Kit-KitDevices-0_0_0-0]


```
   HTTP/1.1 200 Ok
[
                        {
                                "uuidKit": 8,
                                "localId": "",
                                "nodeId": 10,
                                "manufacturerId": 271,
                                "manufacturerProductId": 4097,
                                "position": "kitchen",
                                "givenName": "Fibaro Motion Series kitchen",
                                "isActive": true,
                                "id": 47,
                                "createdAt": "2017-06-29T10:31:07.000Z",
                                "updatedAt": "2017-06-29T10:31:07.000Z"
                        },
                        {
                                "uuidKit": 8,
                                "localId": "99789bd95661c2b7b5789c6dfe71bd62",
                                "nodeId": 1,
                                "manufacturerId": 1,
                                "manufacturerProductId": 1,
                                "position": "lounge",
                                "givenName": "Main controller",
                                "isActive": true,
                                "id": 48,
                                "createdAt": "2017-06-29T10:31:07.000Z",
                                "updatedAt": "2017-06-29T10:31:07.000Z"
                        },
                        {
                                "uuidKit": 8,
                                "localId": "",
                                "nodeId": 12,
                                "manufacturerId": 151,
                                "manufacturerProductId": 17665,
                                "position": "entrance",
                                "givenName": "Zipato Keypad entrance",
                                "isActive": true,
                                "id": 49,
                                "createdAt": "2017-06-29T10:31:07.000Z",
                                "updatedAt": "2017-06-29T10:31:07.000Z"
                        },
                        {
                                "uuidKit": 8,
                                "localId": "",
                                "nodeId": 2,
                                "manufacturerId": 0,
                                "manufacturerProductId": 0,
                                "position": "entrance",
                                "givenName": "entrance",
                                "isActive": true,
                                "id": 50,
                                "createdAt": "2017-06-29T10:31:07.000Z",
                                "updatedAt": "2017-06-29T10:31:07.000Z"
                        },
                        {
                                "uuidKit": 8,
                                "localId": "",
                                "nodeId": 3,
                                "manufacturerId": 271,
                                "manufacturerProductId": 4096,
                                "position": "entrance",
                                "givenName": "Fibaro Door/Window Sensor bathroom",
                                "isActive": true,
                                "id": 51,
                                "createdAt": "2017-06-29T10:31:07.000Z",
                                "updatedAt": "2017-06-29T10:31:07.000Z"
                        },
```

```
                {
                        "uuidKit": 8,
                        "localId": "",
                        "nodeId": 9,
                        "manufacturerId": 0,
                        "manufacturerProductId": 0,
                        "position": "entrance",
                        "givenName": "entrance",
                        "isActive": true,
                        "id": 52,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
                },
                {
                        "uuidKit": 8,
                        "localId": "",
                        "nodeId": 7,
                        "manufacturerId": 0,
                        "manufacturerProductId": 0,
                        "position": "entrance",
                        "givenName": "entrance",
                        "isActive": true,
                        "id": 53,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
                },
                {
                        "uuidKit": 8,
                        "localId": "",
                        "nodeId": 6,
                        "manufacturerId": 271,
                        "manufacturerProductId": 4096,
                        "position": "bedroom",
                        "givenName": "Fibaro Door/Window Sensor entrance",
                        "isActive": true,
                        "id": 54,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
                },
                {
                        "uuidKit": 8,
                        "localId": "",
                        "nodeId": 11,
                        "manufacturerId": 271,
                        "manufacturerProductId": 4097,
                        "position": "bedroom",
                        "givenName": "Fibaro Motion Series bedroom",
                        "isActive": true,
                        "id": 55,
                        "createdAt": "2017-06-29T10:31:07.000Z",
                        "updatedAt": "2017-06-29T10:31:07.000Z"
                }
        ]
```


Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Kit-KitDevices-0_0_0-0]

- Error-Response: [#error-examples-Kit-KitDevices-0_0_0-1]
- Error-Response: [#error-examples-Kit-KitDevices-0_0_0-2]
- Error-Response: [#error-examples-Kit-KitDevices-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

KitFind - Kit find

Is used to find a kit by id.

GET

```
/kit/:kit
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Kit-KitFind-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Kit-KitFind-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                      "personName": "Maria",
                      "personBirthdate": "1943-03-23T0:0:0.000Z",
                      "notificationEmailList": ["luigi@gmail.com"],
                      "notificationPhoneList": ["luigi@gmail.com"],
```

```
                    "createdAt": "2017-06-01T11:10:35.000Z",
                    "updatedAt": "2017-06-02T22:20:21.000Z"
                }
```


Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Kit-KitFind-0_0_0-0]
- Error-Response: [#error-examples-Kit-KitFind-0_0_0-1]
- Error-Response: [#error-examples-Kit-KitFind-0_0_0-2]
- Error-Response: [#error-examples-Kit-KitFind-0_0_0-3]


```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```


KitMyKits - Kit get my kits

Is used to get a list of user kits.

GET

```
/kit
```

Header

Campo	Tipo	Descrizione

`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Kit-KitMyKits-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Kit-KitMyKits-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                [
                        {
                                "personName": "Maria",
                                "personBirthdate": "1943-03-23T0:0:0.000Z",
                                "notificationEmailList": ["luigi@gmail.com"],
                                "notificationPhoneList": ["luigi@gmail.com"],
                                "createdAt": "2017-06-01T11:10:35.000Z",
                                "updatedAt": "2017-06-02T22:20:21.000Z"
                                }
                        ]
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

- Error-Response: [#error-examples-Kit-KitMyKits-0_0_0-0]
- Error-Response: [#error-examples-Kit-KitMyKits-0_0_0-1]
- Error-Response: [#error-examples-Kit-KitMyKits-0_0_0-2]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

KitUpdate - Kit update

Is used to update a kit. Requires a JSON POST body and Content-Type header to be set to application/json.

PUT

```
/kit/:kit
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Kit-KitUpdate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`personName`	String	Observed person name.
`personBirthdate`	Date	Observed person birthdate.
`notificationEmailList`	Array	Array of email addresses.
`notificationPhoneList`	Array	Array of phone numbers.

- Success-Response-Example: [#success-examples-Kit-KitUpdate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                    "personName": "Maria",
                    "personBirthdate": "1943-03-23T0:0:0.000Z",
                    "notificationEmailList": ["luigi@gmail.com"],
                    "notificationPhoneList": ["luigi@gmail.com"],
                    "createdAt": "2017-06-01T11:10:35.000Z",
                    "updatedAt": "2017-06-02T22:20:21.000Z"
               }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Kit-KitUpdate-0_0_0-0]
- Error-Response: [#error-examples-Kit-KitUpdate-0_0_0-1]
- Error-Response: [#error-examples-Kit-KitUpdate-0_0_0-2]
- Error-Response: [#error-examples-Kit-KitUpdate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

Notification

SendTestEmail - Send test email

Sends test email.

POST

```
/notification/email
```

JSON Body

Campo	Tipo	Descrizione
`to`	Array	List of receivers.
`subject`	String	Subject line.
`text`	String	Plaintext body.

- Success-Response-Example: [#success-examples-Notification-SendTestEmail-0_0_0-0]

```
HTTP/1.1 200 OK
```

Sensor

SensorCreate - Sensor create

Is used to create a new sensor. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/kit/:kit/device/:device/sensor
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Sensor-SensorCreate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`localId`	String	Hardware id of sensor generated locally by Razberry.
`type`	String	Sensor type: binary or multilevel.
`category`	String	Sensor category: temperature, luminostiy, door, bed...

- Success-Response-Example: [#success-examples-Sensor-SensorCreate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "uuidDevice": 54,
                        "uuidKit": 8,
                        "localId": "ZWayVDev_zway_Remote_6-0-0-B",
                        "type": "binary",
                        "category": "virtual",
                        "id": 52,
```

```
                    "createdAt": "2017-06-29T10:31:08.000Z",
                    "updatedAt": "2017-06-29T10:31:08.000Z"
                }
```


Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Sensor-SensorCreate-0_0_0-0]
- Error-Response: [#error-examples-Sensor-SensorCreate-0_0_0-1]
- Error-Response: [#error-examples-Sensor-SensorCreate-0_0_0-2]
- Error-Response: [#error-examples-Sensor-SensorCreate-0_0_0-3]


```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```


SensorDestroy - Sensor destroy

Is used to delete a sensor.

DELETE

```
/kit/:kit/sensor/:sensor
```

Permessi: kitOwner ☐

Header

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Sensor-SensorDestroy-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-Sensor-SensorDestroy-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                  "uuidDevice": 54,
                  "uuidKit": 8,
                  "localId": "ZWayVDev_zway_Remote_6-0-0-B",
                  "type": "binary",
                  "category": "virtual",
                  "id": 52,
                  "createdAt": "2017-06-29T10:31:08.000Z",
                  "updatedAt": "2017-06-29T10:31:08.000Z"
                  }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Sensor-SensorDestroy-0_0_0-0]

- Error-Response: [#error-examples-Sensor-SensorDestroy-0_0_0-1]
- Error-Response: [#error-examples-Sensor-SensorDestroy-0_0_0-2]
- Error-Response: [#error-examples-Sensor-SensorDestroy-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

SensorFind - Sensor find

Is used to find kit sensors matching search criteria.

GET

```
/kit/:kit/sensor
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Sensor-SensorFind-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

URL Parameters

Campo	Tipo	Descrizione
`skip`	Integer	The number of records to skip (useful for pagination).
`limit`	Integer	The maximum number of records to send back (useful for pagination). Defaults to 30.
`where`	String	Instead of filtering based on a specific attribute, you may instead choose to provide a where parameter with a Waterline WHERE criteria object, encoded as a JSON string.
`sort`	String	The sort order. By default, returned records are sorted by primary key value in ascending order.

- Request-Example: [#parameter-examples-Sensor-SensorFind-0_0_0-0]

```
GET /kit/8/sensor?sort=createdAt DESC&limit=30&where={"localId":{"contains":"door"}}
```

- Success-Response-Example: [#success-examples-Sensor-SensorFind-0_0_0-0]

```
    HTTP/1.1 200 Ok
```

```
 {
                [
                        {
                                "uuidDevice": 54,
                                "uuidKit": 8,
                                "localId": "ZWayVDev_zway_Remote_6-0-0-B",
                                "type": "binary",
                                "category": "virtual",
                                "id": 52,
                                "createdAt": "2017-06-29T10:31:08.000Z",
                                "updatedAt": "2017-06-29T10:31:08.000Z"
                                }
                        ]
                }
```


Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!
`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Sensor-SensorFind-0_0_0-0]
- Error-Response: [#error-examples-Sensor-SensorFind-0_0_0-1]
- Error-Response: [#error-examples-Sensor-SensorFind-0_0_0-2]
- Error-Response: [#error-examples-Sensor-SensorFind-0_0_0-3]


```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```


SensorUpdate - Sensor update

Is used to update a sensor. Requires a JSON POST body and Content-Type header to be set to application/json.

PUT

```
/kit/:kit/sensor/:sensor
```

Permessi: kitOwner

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-Sensor-SensorUpdate-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`localId`	String	Hardware id of sensor generated locally by Razberry.
`type`	String	Sensor type: binary or multilevel.
`category`	String	Sensor category: temperature, luminostiy, door, bed...

- Success-Response-Example: [#success-examples-Sensor-SensorUpdate-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                        "uuidDevice": 54,
                        "uuidKit": 8,
                        "localId": "ZWayVDev_zway_Remote_6-0-0-B",
                        "type": "binary",
                        "category": "virtual",
                        "id": 52,
                        "createdAt": "2017-06-29T10:31:08.000Z",
                        "updatedAt": "2017-06-29T10:31:08.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

`NoPermission`	You are not the author of this resource, you can not update it.

- Error-Response: [#error-examples-Sensor-SensorUpdate-0_0_0-0]
- Error-Response: [#error-examples-Sensor-SensorUpdate-0_0_0-1]
- Error-Response: [#error-examples-Sensor-SensorUpdate-0_0_0-2]
- Error-Response: [#error-examples-Sensor-SensorUpdate-0_0_0-3]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "No permission"
}
```

User

UserDeleteCurrent - User delete

Is used to delete current user.

DELETE

```
/me
```

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-User-UserDeleteCurrent-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-User-UserDeleteCurrent-0_0_0-0]

```
HTTP/1.1 204 No Content
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

- Error-Response: [#error-examples-User-UserDeleteCurrent-0_0_0-0]
- Error-Response: [#error-examples-User-UserDeleteCurrent-0_0_0-1]
- Error-Response: [#error-examples-User-UserDeleteCurrent-0_0_0-2]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

UserGetCurrent - User get current

Is used to get current user.

GET

```
/me
```

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-User-UserGetCurrent-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

- Success-Response-Example: [#success-examples-User-UserGetCurrent-0_0_0-0]

```
   HTTP/1.1 200 OK
{
                        "name": "Mario",
                        "surname": "Rossi",
                        "email": "mario@gmail.com",
                        "createdAt": "2017-06-01T11:10:35.000Z",
                        "updatedAt": "2017-06-01T11:10:35.000Z"
                }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

- Error-Response: [#error-examples-User-UserGetCurrent-0_0_0-0]
- Error-Response: [#error-examples-User-UserGetCurrent-0_0_0-1]
- Error-Response: [#error-examples-User-UserGetCurrent-0_0_0-2]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```

UserLogin - User login

Is used to login an user and get the authorization token. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/login
```

JSON Body

Campo	Tipo	Descrizione
`email`	String	User email

		User email.
password	String	User password.

- Success-Response-Example: [#success-examples-User-UserLogin-0_0_0-0]

```
    HTTP/1.1 200 OK
{
                "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxN
                "user": {
                        "name": "Mario",
                        "surname": "Rossi",
                        "email": "mario@gmail.com",
                        "createdAt": "2017-06-01T11:10:35.000Z",
                        "updatedAt": "2017-06-01T11:10:35.000Z"
                        }
                }
```

UserRegister - User register

Is used to register an user. Requires a JSON POST body and Content-Type header to be set to application/json.

POST

```
/user
```

JSON Body

Campo	Tipo	Descrizione
name	String	User name.
surname	String	User surname.
email	String	User email.

- Success-Response-Example: [#success-examples-User-UserRegister-0_0_0-0]

```
    HTTP/1.1 201 OK
 {
                        "name": "Mario",
                        "surname": "Rossi",
                        "email": "mario@gmail.com",
                        "createdAt": "2017-06-01T11:10:35.000Z",
                        "updatedAt": "2017-06-01T11:10:35.000Z"
                }
```

UserUpdateCurrent - User update

Is used to update current user. Requires a JSON POST body and Content-Type header to be set to application/json.

to application/json.

PUT

```
/me
```

Header

Campo	Tipo	Descrizione
`Authorization`	String	Authentication token, format is 'Bearer [token]'.

- Request-Header-Example: [#header-examples-User-UserUpdateCurrent-0_0_0-0]

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImNhdmFsb
```

JSON Body

Campo	Tipo	Descrizione
`name`	String	User name.
`surname`	String	User surname.
`email`	String	User email.

- Success-Response-Example: [#success-examples-User-UserUpdateCurrent-0_0_0-0]

```
   HTTP/1.1 200 Ok
{
                      "name": "Luigi",
                      "surname": "Rossi",
                      "email": "luigi@gmail.com",
                      "createdAt": "2017-06-01T11:10:35.000Z",
                      "updatedAt": "2017-06-02T22:20:21.000Z"
               }
```

Error 4xx

Nome	Descrizione
`WrongTokenFormat`	Format is 'Authorization: Bearer [token]'
`NoAuthHeader`	No Authorization header was found
`InvalidToken`	Invalid Token!

- Error-Response: [#error-examples-User-UserUpdateCurrent-0_0_0-0]
- Error-Response: [#error-examples-User-UserUpdateCurrent-0_0_0-1]
- Error-Response: [#error-examples-User-UserUpdateCurrent-0_0_0-2]

```
HTTP/1.1 401 Bad Request
{
  "error": "WrongTokenFormat"
```

```
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "NoAuthHeader"
}
```

```
HTTP/1.1 401 Bad Request
{
  "error": "InvalidToken"
}
```