



**SAPIENZA**  
UNIVERSITÀ DI ROMA

## **Internet of Things: Progettazione e sviluppo di interfacce per Dashboard Integrative**

**Facoltà di Ingegneria dell'Informazione, Informatica e Statistica**  
**Corso di laurea magistrale in Informatica**

**Candidato**  
**Rocco Musolino**  
**1628080**

**Relatore**  
**Emanuele Panizzi**

**A/A 2015/2016**



# Indice

1. Introduzione.....	4
1.1 - Internet of Things.....	4
1.2 - Tecnologie Software impiegate.....	7
1.2.1 - Javascript.....	7
1.2.2 - Node JS .....	8
1.2.3 - MQTT .....	9
2. Chrome MQTT Dashboard .....	10
2.1 - Introduzione.....	10
2.1.1 - Chrome App e App native .....	12
2.2 - Hardware .....	13
2.2.1 - ESP8266.....	13
2.2.2 - NodeMCU & Wemos .....	16
2.2.3 - Altre Board .....	18
2.2.4 - Quale board scegliere? .....	20
2.2.5 - Sensori e attuatori .....	20
2.3 - Software.....	22
2.3.1 - Firmware.....	23
2.3.2 - Idea di Base .....	31
2.3.3 - Perchè MQTT? .....	32
2.3.4 - HTTP o MQTT? .....	34
2.3.5 - XMPP o MQTT?.....	36
2.3.6 - Broker: Mosca o Mosquito.....	37
2.3.7 - Clients e Sketch MQTT .....	40
2.3.8 - Chrome APP .....	46
2.3.9 - GCM - Google Cloud Messaging .....	49
2.4 - Handshaking .....	51
2.5 - Altri sensori e collegamenti .....	53
3. IoT-433Mhz .....	57
3.1 - Introduzione.....	57
3.1.1 - A chi è destinato il sistema? .....	59
3.1.2 - Caratteristiche principali .....	59

3.2 - Hardware Necessario .....	60
3.2.1 - Raspberry Pi .....	61
3.2.2 - Arduino .....	62
3.2.3 - Moduli radio .....	64
3.2.4 - Antenna ideale .....	67
3.2.5 - Altri componenti .....	68
3.3 - Software .....	70
3.3.1 - Flusso d'esecuzione .....	70
3.3.2 - Avviamo iot-433mhz .....	71
3.3.3 - Interfaccia grafica .....	73
3.4 - Utilizzo .....	76
3.4.1 - Come rileviamo i codici? .....	79
3.4.2 - Aggiungere una nuova scheda dispositivo .....	80
3.4.3 - Bot Telegram .....	81
3.5 - Scelte Implementative .....	83
3.5.1 - Connessione dei moduli radio direttamente ai GPIO del RPi .....	83
3.5.2 - rc-switch .....	84
3.5.3 - Autenticazione .....	84
3.5.4 - Notifiche Telegram .....	85
3.5.5 - Dipendenze e moduli creati appositamente .....	87
3.5.6 - 433Mhz o Wifi 2.4Ghz ? .....	88
3.5.7 - Svantaggi della comunicazione 433mhz .....	89
3.6 - API & WebHooks .....	89
3.7 - Alternative a IoT-433mhz .....	93
3.8 - Miglioramenti per le prossime release .....	96
4. Sismometro SeismoCloud .....	99
4.1 - Costruire un sismometro digitale basato su esp8266 .....	99
4.1.1 - Hardware necessario .....	99
4.1.2 - Assemblaggio .....	101
4.1.3 - Software .....	104
5. Conclusioni .....	106
Bibliografia .....	109



# 1. Introduzione

---

## 1.1 - Internet of Things

L'internet of things è un argomento di discussione sempre più in voga, in rapida evoluzione. Nuovi standard e tecnologie appaiono quotidianamente e non ci sono ancora segni tangibili di un unico protocollo che permetta la comunicazione fra dispositivi, servizi e applicazioni, senza compromessi. Gli standard del web e gli strumenti già esistenti forniscono il substrato ideale per lo scambio di dati fra dispositivi perennemente connessi.

Internet è un chiaro esempio di un network globale scalabile, di computer che interoperano attraverso piattaforme hardware e software eterogenee.

Il web in cima a internet illustra bene come un insieme di standard aperti e relativamente semplici possano essere usati per costruire sistemi flessibili preservando efficienza e scalabilità.

Lo sviluppo di applicazioni composite sul Web, assieme alla loro disponibilità su una vasta gamma di dispositivi, desktop, laptop, tablet, smartphone, console etc. rendono il web un miscuglio di risorse e interfacce utili alla creazione di nuove soluzioni distribuite cloud-based.

Secondo un'analisi di Gartner, Inc. Nel 2016 avremo 6.4 miliardi di dispositivi connessi. Il 30% in più rispetto al 2015, con 5.5 milioni di nuovi dispositivi aggiunti ogni giorno e una previsione di 20.8 miliardi per il 2020, quasi 3 volte l'intera popolazione mondiale. Questo comporta la generazione di un'enorme mole di dati, il dato diverrà il vero nuovo prodotto e sempre più le aziende dovranno focalizzarsi sulla progettazione dell'esperienza utente in un mercato che vale 235 miliardi di dollari, con un incremento del 22% rispetto al 2015.

La diffusione di dispositivi sempre più piccoli, assieme alla crescente miniaturizzazione dei componenti e alla diffusione di dispositivi di controllo perennemente connessi hanno dato il via alle nuove tecnologie immersive su scala globale.

Questo è l'internet of things – *un network globale di dispositivi, oggetti e persone, connesse.*

L'Internet of Things (IoT) ha permesso a internet di espandersi all'infuori del browser. Queste "things" di cui si parla sono dispositivi elettronici capaci di interagire con il mondo fisico attraverso sensori che raccolgono e inviano dati. Attualmente questi dispositivi sono prodotti, progettati con uno scopo preciso in mente, un esempio tipico può essere un semplice bracciale che traccia l'attività fisica. Esso monitora e invia le informazioni grezze ad un'app che è in grado poi di elaborarle e analizzarle per fornire suggerimenti, stampare grafici e invogliare l'utente a proseguire nelle sue attività fisiche.

Quando si costruiscono dispositivi IoT, il processo è tipicamente suddiviso in due ruoli: un ingegnere progetta e crea il dispositivo fisico e uno sviluppatore l'ecosistema. Tuttavia, questo non è sempre vero. Nel caso di Javascript, la sua natura isomorfa permette ad un unico linguaggio di essere usato su più piattaforme differenti - incluso hardware.

Realizzare prototipi hardware e dispositivi connessi a internet è sempre stato qualcosa di complesso che solamente qualche ingegnere elettrico avrebbe mai potuto realizzare. Questa visione è cambiata completamente con l'introduzione di nuove board di sviluppo come Arduino, Particle, Raspberry Pi, Onion, Chip e molte altre.

Queste board di sviluppo mimano la scheda madre di un computer. Hanno canali di input e output, USB e alimentazione integrata, pin e altre estensioni che permettono l'aggiunta di componenti esterni. Il chip microcontrollore agisce come un processore, eseguendo il codice dell'applicazione e comunicando con input e output. Il microcontrollore è relativamente lento, specificatamente pensato per eseguire semplici task come la lettura di dati dai sensori.

Questo nuovo mercato ha aperto le porte ai molti appassionati di fai da te, elettronica e innovazione. Nasce proprio in questi anni il movimento dei "Maker", una parola abbastanza recente. Nel passato, chiunque facesse innovazione nella propria cantina veniva chiamato "bricoleur" o semplicemente inventore. Il movimento dei maker riguarda principalmente l'idea di poter pensare, progettare e realizzare qualcosa, rompendo le barriere dei costi, degli strumenti o della necessità di grosse catene di produzione. Stampanti 3D, computer in miniatura e altri strumenti hi-tech a buon mercato hanno reso questo mondo accessibile a chiunque.

Lo stesso movimento che ha guadagnato molta popolarità negli ultimi anni ha allargato il mercato delle board di sviluppo, e adesso c'è un'ampia offerta, ognuna con le proprie caratteristiche. La competizione ha fatto sì che ci si concentrasse su alcune feature punti di forza della board, come il reparto comunicazione: wifi e bluetooth, la dimensione, la capienza della batteria o ancora, il linguaggio di programmazione

che è possibile eseguire sulla board stessa. Tutte caratteristiche importanti da tenere in considerazione quando si intraprende un nuovo progetto di sviluppo.

Tra gli scopi di questo progetto, presentare delle applicazioni reali che si collocano nell'ambito dell'Internet of Things. Usare le tecnologie web per ascoltare e connettere il mondo reale. Capire come integrare protocolli di comunicazione come MQTT o WebSocket nel Web. Imparare ad usare alcune delle piattaforme, dei protocolli e degli strumenti più gettonati nel mondo dell'IoT. Estendere le architetture più gettonate e allinearne i requisiti a quelli dell'utente moderno. Mostrare e discutere i problemi che più comunemente si dovranno affrontare lavorando con queste tecnologie.

La trattazione è suddivisa in 3 sezioni principali.

- 1) Si introduce e spiegano i vari concetti legati all'IoT. Le piattaforme Hardware, i sensori e gli attuatori più comuni. I Firmware disponibili per le piattaforme hardware, i protocolli disponibili. Infine si mostra l'implementazione di una Dashboard di controllo che sfrutta queste tecnologie per realizzare un esempio di rete di dispositivi interconnessa tramite MQTT.
- 2) In quest'altra sezione a seguire si mostra un sistema completo per l'automazione e l'internet of things attraverso dispositivi e protocolli che funzionano sulla frequenza radio 433Mhz. Dall'installazione, fino all'esecuzione finale, accennando alle scelte implementative che sono state effettuate al fine di realizzare un applicativo stabile, utilizzabile in produzione, modulare e di facile estensione attraverso API e webHooks.
- 3) Infine si realizza un sismometro digitale compatto, come esempio di sensore basato su piattaforma NodeMCU e connesso alla rete di allerta per terremoti (Earthquake Early Warning – EEW) SeismoCloud, un progetto italiano nato all'università la Sapienza. Si passerà dalla prototipazione dell'hardware alla scrittura del software necessario per comunicare con i server SeismoCloud.

Mettendo assieme questi concetti si realizza una rete interoperabile di dispositivi intelligenti, rapida, facile da estendere, in grado di assottigliare ulteriormente il gap che esiste fra il mondo virtuale e quello reale.



## 1.2 - Tecnologie Software impiegate

Mostriamo alcune delle tecnologie software usate nella realizzazione di questo progetto di tesi.

### 1.2.1 - Javascript



*Figura 1 - Javascript Logo*

Oggigiorno, qualunque web app si realizzi, si hanno dozzine di decisioni architetturelle da tenere in considerazione. Si tenta di usare le tecnologie che permettano uno sviluppo rapido, un'iterazione costante, l'efficienza massima, la velocità, la robustezza e molto altro. Si cercano tecnologie in grado di massimizzare la produttività a lungo termine.

Fino a qualche anno fa, la regola è sempre stata quella di usare linguaggi come Perl, PHP, ASP, JSP, .NET, Ruby, Python per le applicazioni lato server, e Javascript per la parte client.

Gli sviluppatori hanno iniziato a realizzare che usare due linguaggi differenti per la parte client e l'ambiente server spesso complica le cose.

Javascript è un linguaggio di programmazione lato-client (e da qualche anno lato-server nella forma di Node.js), debolmente tipizzato, interpretato, usato per rendere le pagine web interattive o costruire vere e proprie applicazioni web. Si discosta dal classico modello orientato agli oggetti e segue invece il modello prototipale.

E' stato standardizzato nelle specifiche del linguaggio ECMAScript.

Assieme a HTML e CSS, è una delle tre tecnologie essenziali per la visualizzazione dei contenuti nel World Wide Web. Tutti i browser moderni supportano nativamente Javascript.

Javascript è stato pensato per Internet. Esiste fin dall'inizio ed è molto difficile sparisca. Tutti i tentativi che sono stati fatti per cercare di annientarlo sono falliti: dalle applet Java, a VBscript fino al più recente

Adobe Flash. Sostituire Javascript senza inibire il funzionamento di milioni di pagine web è letteralmente impossibile. E' una tecnologia che può soltanto migliorare e mai esser messa da parte.

### 1.2.2 - Node JS



Figura 2 - Node.js Logo

Node.js è un tentativo di portare il dialetto Javascript fuori dal suo normale ambiente d'esecuzione, il browser. Node.js integra il motore d'esecuzione di JS (V8) presente in Google Chrome per l'interpretazione del codice lato server e non più lato client. E' particolarmente adatto in ambienti asincroni, come lettura e scrittura di file o gestione di connessioni di rete.

In quanto a framework basato sugli eventi asincroni, Node.js è progettato per realizzare applicazioni di rete scalabili. In contrasto con il modello di concorrenza oggi più comune e adottato nei sistemi operativi dove si usano i Thread. Il networking basato sui Thread è relativamente inefficiente e difficile da usare. Gli utenti di Node sono liberi da preoccupazioni quali mandare in dead-lock i processi, non ci sono lock in node. Quasi nessuna funzione in Node.js effettua direttamente operazioni di I/O, dunque il processo non si bloccherà mai. Node da questo punto di vista è molto simile e influenzato da sistemi come la Event Machine di Ruby o Twisted in Python. Node modifica leggermente il suo modello a eventi, presentando l'evento di Loop come un costrutto del linguaggio piuttosto che come una libreria. In altri sistemi c'è sempre una chiamata bloccante che avvia l'event-loop. Semplicemente si entra nel ciclo di eventi dopo l'esecuzione dello script di input, e si esce dal ciclo quando non ci saranno più callbacks da eseguire.

Questo comportamento è in linea con l'esecuzione di Javascript nel Browser, con un ciclo di eventi nascosti all'utente.

Solo perchè Node sia stato progettato senza thread, non vuol dire che non si possano sfruttare i multi-core del nostro ambiente di lavoro. E' possibile creare processi figli attraverso l'API "`child_process.fork()`", che semplifica anche la comunicazione fra processi. Costruito sulla stessa interfaccia abbiamo il modulo "`cluster`", che permette di condividere i socket fra i processi per permettere il load balancing sui vari core.

Nella trattazione corrente Node.js verrà usato come framework per lo sviluppo di javascript Server-Side. Tra le scelte che mi hanno spinto ad adottare Node.js come base per questo progetto, in primis troviamo l'elevata portabilità. Infatti Node.js è disponibile per una moltitudine di sistemi operativi e piattaforme hardware, questo garantisce portabilità del codice.

Node.js come già visto, utilizza anche un modello event-driven, non-blocking per gestire i propri task. Questo lo rende efficiente in ambienti asincroni.

Infine npm (Node Package Manager), il gestore di pacchetti di Node, è una delle repository di codice open source più grandi al mondo, con estrema facilità chiunque abbia npm installato può gestire moduli e dipendenze, e proprio su npm sono presenti i progetti che vedremo man mano nella trattazione.

### 1.2.3 - MQTT



Figura 3 - MQTT Logo

MQTT sta per MQ Telemetry Transport.

E' un protocollo di messaggistica leggero, basato sul concetto di publish/subscribe. Progettato per dispositivi dalle scarse capacità hardware, in reti con una larghezza di banda bassa, ad alta latenza e non affidabili.

I principi di progettazione sono quelli di minimizzare il traffico di rete e le risorse dispositivo, cercando di garantire affidabilità e un certo grado di garanzia di consegna. Questi principi si rivelano essere ideali per l'emergente mondo del M2M (Machine-to-Machine) o Internet of Things, dove ogni dispositivo è connesso alla rete e richiede bassi consumi in termini di banda e batteria.

- Chi ha inventato MQTT?

Mqtt è stato inventato nel 1999 dal Dr. Andy Stanford-Clark di IBM, e Arlen Nipper di Arcom (adesso Eurotech).

- MQTT è uno Standard?

Da Marzo 2013, MQTT è sotto il processo di standardizzazione presso OASIS. Le specifiche del protocollo sono state pubblicate apertamente, senza costi di licenza. La porta TCP/IP standard per MQTT è la 1883, mentre per MQTT con SSL si usa la 8883. Dalla versione 3.1 MQTT inoltre supporta l'autenticazione tramite Username e Password.

MQTT è un protocollo usato nell'industria spaziale o in compagnie del calibro di Facebook o Amazon. E' infatti alla base dell'app Facebook Messenger, usata da milioni di persone ogni giorno, gli stessi ingegneri per far fronte a problemi di latenza e battery drain hanno optato per l'utilizzo di MQTT, si cita il post di Lucy Zhang, Software Engineer presso Facebook:

*"One of the problems we experienced was long latency when sending a message. The method we were using to send was reliable but slow, and there were limitations on how much we could improve it. With just a few weeks until launch, we ended up building a new mechanism that maintains a persistent connection to our servers. To do this without killing battery life, we used a protocol called MQTT that we had experimented with in Beluga. MQTT is specifically designed for applications like sending telemetry data to and from space probes, so it is designed to use bandwidth and batteries sparingly. By maintaining an MQTT connection and routing messages through our chat pipeline, we were able to often achieve phone-to-phone delivery in the hundreds of milliseconds, rather than multiple seconds."*

## 2. Chrome MQTT Dashboard

---

### 2.1 - Introduzione

In questa prima sezione della tesi, si sviluppa una Dashboard di base, per il controllo e il monitoraggio dei dispositivi connessi a un broker MQTT.

I sistemi M2M e IoT devono affrontare spesso problematiche relative all'interruzione della rete, sbalzi di connessione o alta latenza. I costi in termini di dati trasmessi sono cruciali su network in cui si hanno migliaia se non milioni o miliardi di dispositivi connessi.

Dispositivi o server ai "bordi della rete" hanno risorse di calcolo limitate. Dispositivi basati su esp8266 compresi. Questo è il motivo principe per l'utilizzo di MQTT sui dispositivi embedded. Data la forte domanda, moltissime librerie open source sono nate per supportare l'implementazione di MQTT su tutta una serie di architetture embedded. Esistono librerie in C, Java, Lua, Python, C++, Javascript e molti altri linguaggi.

Se il sistema scala e dunque il numero di dispositivi connessi aumenta, si richiede obbligatoriamente una gestione centralizzata, avanzata e più coesa di tutti i dispositivi connessi al network mqtt. Per questo motivo, una dashboard di controllo diventa necessaria, se non indispensabile. Essa garantisce, sfruttando le ultime tecnologie in ambito web, un controllo funzionale dei dispositivi attivi e passivi connessi al broker, con una pratica e comoda interfaccia.

La scelta di implementare una chrome App di controllo garantisce una certa flessibilità d'esecuzione su tutti i sistemi operativi moderni oltre che una chiara praticità di sviluppo. Adesso più che mai, le tecnologie web brillano di luce propria, sono state infatti altamente rivalutate e impiegate per la costruzione di applicativi anche complessi. Javascript in primis, intraprendendo un cammino totalmente opposto al mero utilizzo per il restyling dinamico di pagine web.

Di seguito le Funzionalità implementate nell'app e la topologia di rete:

- Connessione a broker MQTT locali o Remoti.
- Protocollo di HandShaking e rilevamento Broker-indipendent.
- Material-Design UI per elencare topic disponibili e dispositivi associati al broker.
- Invio di comandi ai DEVICES (attuatori - subscribers).
- Lettura valori sensori (sensori - publishers).
- Notifiche (di 2 tipi, sia in-app, che a livello di OS con API di chrome).
- Possibilità di impostare Triggers.
- Possibilità di inviare comandi in una data specifica, prefissata (API alarms di Chrome).
- Notifiche push interpretate nel layout a schede.
- Sincronizzazione di dati con lo storage in cloud Google.

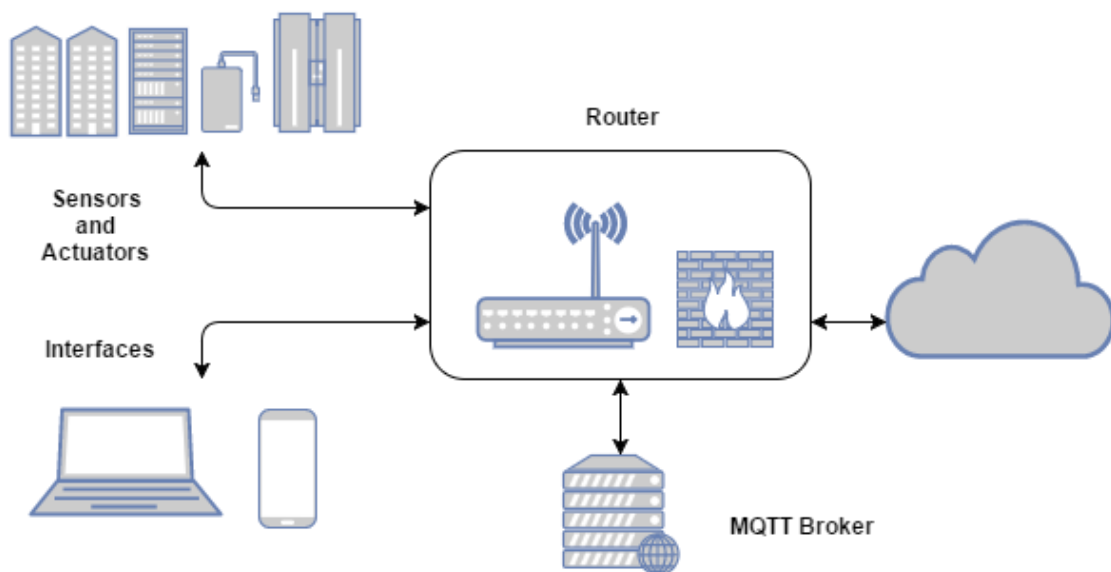


Figura 4 - Topologia di Rete

### 2.1.1 - Chrome App e App native

Le Chrome App permettono ad uno sviluppatore di usare linguaggi web per sviluppare app in grado di garantire un'esperienza utente al pari di un'applicazione nativa. Una Chrome app si integra perfettamente nell'ambiente Desktop e si comporta proprio come una vera applicazione nativa.

Quando si scrive una Chrome app ci si astrae ulteriormente dall'OS sottostante, ogni chrome app infatti gira come fosse un componente di Google Chrome. Immediatamente si rende disponibile l'app a un pubblico molto più vasto, di contro con un'app nativa saremmo costretti a riscrivere l'app per ogni sistema operativo supportato. Piuttosto che scrivere e mantenere un'applicazione separata per Windows, OS X o Linux, con un'unica applicazione scritta, utilizzando linguaggi web, siamo in grado di supportare ogni OS dove Chrome stesso può essere installato.

Con un'applicazione basata su Chrome si avrà una web application eseguibile su qualsiasi piattaforma dotata di web browser Chrome. Ma non solo, in quanto a web application si potrebbe pensare a tutta una serie di limitazioni tipiche degli applicativi online, Google per garantire una flessibilità maggiore e permettere un'implementazione più robusta e dinamica delle proprie app, fornisce tutta una serie di interfacce ausiliarie (API) per accedere a funzionalità di base, tipiche di un'app nativa, altrimenti inaccessibili per una pagina web. Ad esempio si ha la

possibilità di Accedere all'USB, al Bluetooth o al file system stesso nonché avere un'app che funzioni anche offline.

## 2.2 - Hardware

### 2.2.1 - ESP8266

*"L'ESP8266 è un modulo Wifi dotato di Input/Output General Purpose e processore ARM. Compatibile con arduino e attualmente tra i più economici e piccoli sul mercato. E' molto utilizzato attualmente nel campo dell'Internet of Things."*

Sono disponibili una gran varietà di board costruite sulla base di questo modulo.

Le tipologie di ESP8266 attualmente esistenti sono mostrate nella tabella che segue, tuttavia nella trattazione avremo a che fare solamente con la versione ESP-01 ed ESP-12-E.

Board ID	pins	pitch	form factor	LEDs	Antenna	Ant.Socket	Shielded	dimensions mm
ESP-01	8	.1"	2x4 DIL	Yes	Etched-on PCB	No	No	14.3 x 24.8
ESP-02	8	.1"	2x4 notch	No?	None	Yes	No	14.2 x 14.2
ESP-03	14	2mm	2x7 notch	No	Ceramic	No	No	17.3 x 12.1
ESP-04	14	2mm	2x4 notch	No?	None	No	No	14.7 x 12.1
ESP-05	5	.1"	1x5 SIL	No	None	Yes	No	14.2 x 14.2
ESP-06	12+GND	misc	4x3 dice	No	None	No	Yes	?
ESP-07	16	2mm	2x8 pinhole	Yes	Ceramic	Yes	Yes	20.0 x 16.0
ESP-08	14	2mm	2x7 notch	No	None	No	Yes	17.0 x 16.0
ESP-09	12+GND	misc	4x3 dice	No	None	No	No	10.0 x 10.0
ESP-10	5	2mmm?	1x5 notch	No	None	No	No	14.2 x 10.0
ESP-11	8	1.27mm	1x8 pinhole	No?	Ceramic	No	No	17.3 x 12.1
ESP-12	16	2mm	2x8 notch	Yes	Etched-on PCB	No	Yes	24.0 x 16.0
ESP-12-E	22	2mm	2x8 notch	Yes	Etched-on PCB	No	Yes	24.0 x 16.0
ESP-13	18	1.5mm	2x9	?	Etched-on PCB	No	Yes	? x ?
WROOM-02	18	1.5mm	2x9	No	Etched on PCB	No	Yes	20.0 x 18.0

La differenza fra una board e l'altra è essenzialmente il quantitativo di memoria interna, il numero di GPIO e l'antenna talvolta stampata sulla stessa PCB.

Come accennato, nella trattazione faremo riferimento solamente a due di questi modelli. L'ESP-01 e l'ESP-12-E, utilizzati nel loro form factor più comune, mostrato in figura, oppure integrati sopra altre PCB per facilitarne la prototipazione.



Figura 5 - ESP8266-01



Figura 6 - ESP8266-12

L'ESP-01 è forse il più popolare dei moduli, tuttavia non sempre il più conveniente da utilizzare. Con le sue dimensioni (24.75mm x 14.5 mm) entra benissimo in qualsiasi tipo di contenitore. E' dotata di due pin GPIO che possono essere utilizzati per controllare delle periferiche. Con un cablaggio adeguato e l'adattatore serial-to-usb è possibile appunto caricarci su dei firmware alternativi.

Di default è consegnato con una delle differenti versioni del firmware AT che permette di usarlo anche in combinazione con Arduino. Uno dei più grandi problemi di questa board è forse la disposizione dei pin che rendono impossibile un piazzamento diretto sulla breadboard per la prototipazione. Le due file di pin infatti sono troppo vicine fra di loro. In ogni caso è possibile ugualmente usare il modulo su breadboard utilizzando degli appositi cavetti jumper female-to-male. Queste le caratteristiche tecniche:

- E' un SoC Wireless (2.4Ghz, 802.11 b/g/n) con supporto WPA/WPA2



- Possiede GPIO, I2C, ADC, SPI, PWM e altri.
- Lavora a 80MHz
- 64KBytes di instruction RAM
- 96KBytes di data RAM
- 64KBytes boot ROM
- Winbond W25Q40BVNIG SPI flash
- Architettura RISC
- Il core è un 106micro Diamond Standard core (LX3) prodotto da Tensilica
- ESP8266 è prodotto da Espressif

L'ESP8266 ha un assorbimento medio di 50mA circa. Quando però c'è uso di GPIO ed elevata attività sul Wifi, si può tranquillamente arrivare a 200mA. Talvolta questi picchi di trasmissione possono far resettare la board, poichè l'assorbimento si alza di colpo. Dunque è consigliabile per evitare problemi di questo tipo, saldare un condensatore tra i 100uF e i 470uF fra il GND e 3.3v, quanto più possibile vicino ai pin dell'ESP, esso avrà il ruolo di fornire una carica aggiuntiva immediata per far fronte ai picchi durante un uso intensivo ed evitare dunque il reset della board.



*Figura 7 - ESP8266-01 e Condensatore*

L'ESP-12 possiede 11 pin GPIO, un convertitore analogico-a-digitale (ADC) con una risoluzione di 10 bit. Permette di configurare facilmente la modalità sonno-profondo che (secondo le specifiche) permette di alimentare il modulo per 3 anni con sole 2 batterie AA.

Di contro: non è breadboard-friendly affatto. Come per i moduli precedenti l'antenna è stampata sulla PCB. Per poterli prototipare è necessario costruirsi un "adattatore" per poterlo collegare facilmente alla breadboard oppure acquistarne uno dei tanti già in commercio.

Un'altra soluzione è quella di acquisitare direttamente delle board di prototipazione basate sull'ESP-12, che adesso andremo a vedere.

### 2.2.2 - NodeMCU & Wemos

Le due board basate su ESP-12 in assoluto più conosciute sono NodeMCU devkit, e Wemos D1.

Il devkit NodeMCU è nato da un progetto di firmware open source per ESP8266 scritto in Lua. E' forse la migliore piattaforma low cost, open source per l'Internet of Things sul mercato. Una singola board costa in media 4 euro. Essendo basata su ESP8266 anch'essa può essere flashata con diversi firmware, per essere programmata con altrettanti diversi linguaggi di programmazione.

Questo modulo è in qualche modo abbastanza differente dai moduli descritti in precedenza. Ha già tutto l'occorrente necessario per iniziare a lavorarci. Possiede un adattatore serial-to-usb con connessione micro USB, da cui viene alimentato e programmato.

Due interruttori posti sulla board inoltre permettono il reset del modulo e il booting nella FlashMode per poter cambiare firmware. Forse è la miglior scelta per chi vuole iniziare a lavorare con ESP8266. Specie per chi realizza prototipi. Una volta terminato il prototipo infatti si può pensare di salvare il software, assemblare i componenti hardware esterni e passare ai moduli più economici e compatti.

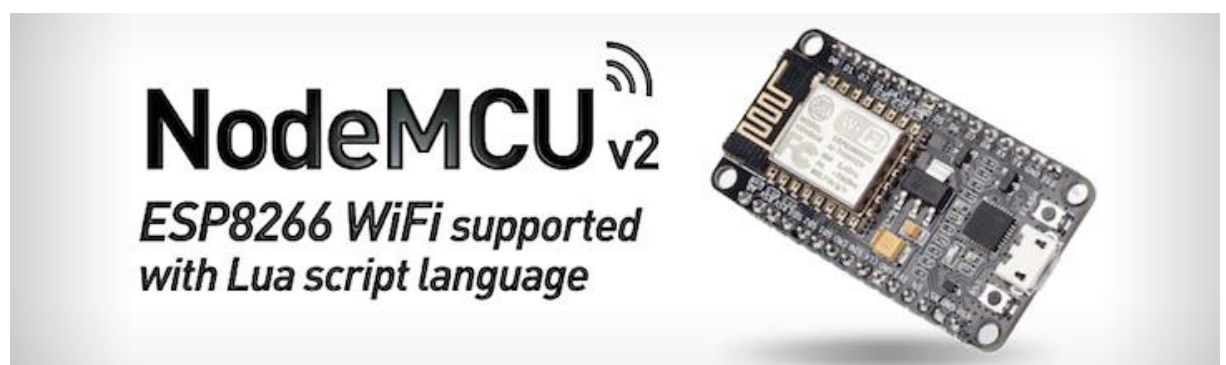


Figura 8 - Board NodeMCU Devkit 1.0

Con il relativo PinOut:

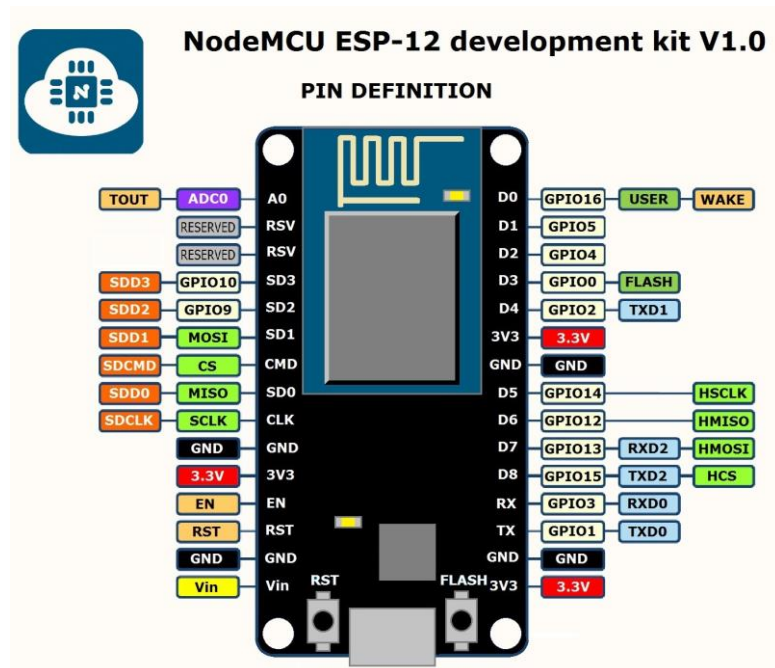


Figura 9 - NodeMCU devkit PinOut

L'altra board è la Wemos D1 Mini, sicuramente meno diffusa del NodeMCU ma altrettanto performante e più piccola nella forma.



Figura 10 - WeMos D1 Mini

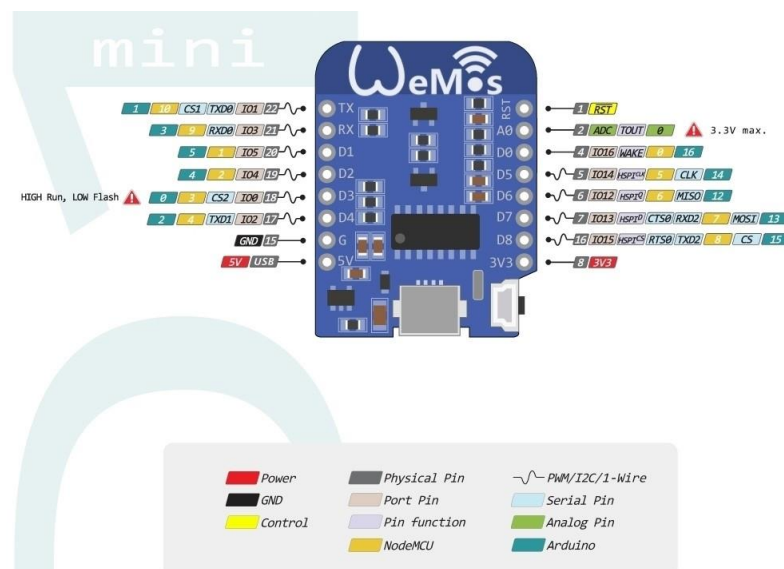




Figura 12 - Onion Omega

## # ESP32

E' il fratello maggiore dell'esp8266. Annunciato da Espressif per sopperire alle mancanze dell'ormai diffusissimo esp8266. Esso non è ancora disponibile alla vendita.

L'esp32 non sostituisce l'esp8266, ma lo migliora sotto certi aspetti. Non solo ha il wifi integrato, ma persino il bluetooth 4.2, due core, 128KB di ROM e 416KB di SRAM. Questo lo rendono più versatile e potente, sicuramente una valida aggiunta al catalogo dei microcontrollori per l'Internet of Things.

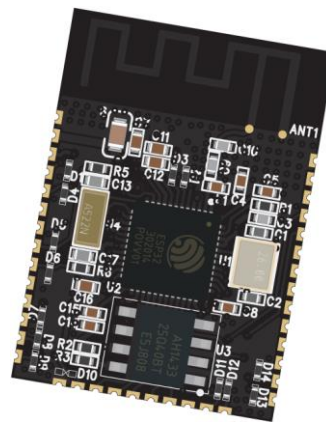


Figura 13 - Espressif ESP-32

## # C.H.I.P

Next Thing Co. con una campagna su Kickstarter iniziata l'8 Maggio 2015 e con oltre 2 milioni di finanziamento raccolti, ha annunciato il loro mini computer da 9 dollari, C.H.I.P.

Chip è un computer a tutti gli effetti, e può far girare software scritti con qualunque linguaggio. Python, Java, Ruby, PHP, C++, JS, Assembly e molti altri.

Ha delle caratteristiche tecniche che non lo collocano nella categoria dei microcontrollori, piuttosto nella categoria micro-computer. Dato il

prezzo di vendita e le caratteristiche di connettività, è l'ideale per applicativi Internet of Things.

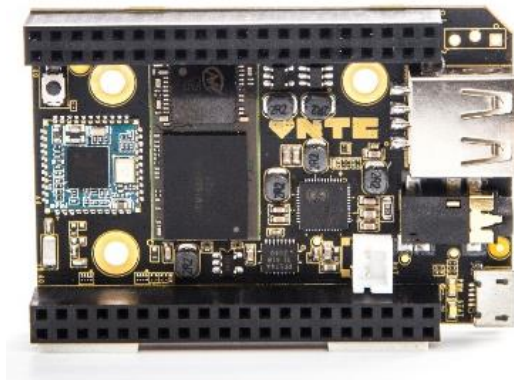


Figura 14 - Next Thing Co. CHIP

#### 2.2.4 - Quale board scegliere?

Ogni board si interfaccia facilmente al sistema purchè si abbia la possibilità di caricare lo sketch adatto, che faccia uso di una libreria per la comunicazione con protocollo MQTT. Quale modulo è il più indicato dipende sicuramente dal tipo di sensore o attuatore che si vuole realizzare. Se il prezzo e le dimensioni contano in ciò che si sta realizzando, e 2 pin sono sufficienti allora l'ESP-01 è il giusto candidato. Mentre l'ESP-12 è sicuramente la scelta più azzeccata se si ha bisogno di un gran numero di GPIO o se si desidera comunicare con periferiche tramite SPI o I2C.

Raspberry Pi, Onion, Chip et simili date le caratteristiche tecniche importanti, possono collocarsi nello scenario assumendo il ruolo di Broker, un'entità il cui scopo è offrire un unico punto di diramazione per giostrare le risorse connesse alla rete e interagire coi clients.

#### 2.2.5 - Sensori e attuatori

Alcuni dei sensori e attuatori più comuni che è possibile collegare ad un ESP8266 e interfacciare al network MQTT.



Figura 15 - Sensore Piroelettrico di Movimento

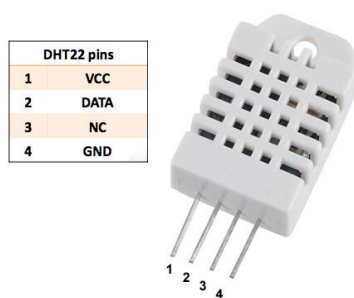


Figura 16 - Sensore di temperatura digitale DHT22



Figura 17 - Sensore magnetico per porte e finestre





Figura 18 - Relay allo stato solido

Per il controllo di un relay con esp8266, siccome all'avvio i pin sono HIGH ed uno di essi è collegato al Solid State Relay, si entrerà nella FlashMode. Dunque non verrà eseguito correttamente lo script all'interno della board. E' necessario invertire anodo e catodo nella connessione del relay al pin gpio per evitare questo effetto e invertire anche il comportamento logico del relay.

All'avvio i pin dell'esp8266 saranno HIGH, l'inversione dei poli farà sì che il relay rimanga spento (come desiderato). Portando a DOWN il gpio di controllo del relay, si avrà l'effetto di attivare il relay stesso. La connessione è mostrata in figura.

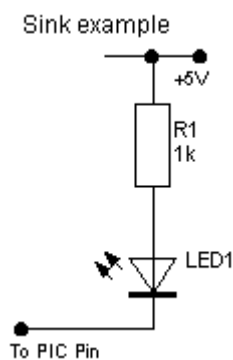


Figura 19 - Circuito Sink per Relay

## 2.3 - Software



### 2.3.1 - Firmware

Programmare il chip dell'esp8266 nella versione ESP-01 richiede l'utilizzo di un connettore per la comunicazione seriale (USB to UART). Nella board NodeMCU questo come già visto non è necessario, perchè già integrato sopra la board.

L'FTDI è attualmente il leader del mercato nella produzione di chip per la comunicazione seriale.

Il voltaggio di funzionamento di ESP-01 è 3.3v, quindi è importante assicurarsi che il piedino di switch nell'adattatore FTDI sia disposto per fornire i 3.3v necessari.

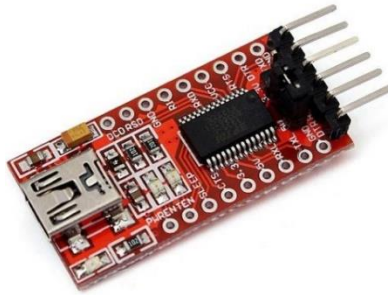


Figura 20 - FTDI USB Adapter

Per connettere l'esp all'unità seriale bisogna accertarsi che prima venga connessa quest'ultima al PC e poi si colleghi l'esp. Se connesso direttamente ci possono essere alcuni sbalzi di tensione che non permettono una corretta comunicazione con l'esp8266.

E' sufficiente dunque connettere l'unità seriale e predisporre la connessione del chip, con il cavetto del GND (Ground) scollegato. Una volta che la board seriale è attiva colleghiamo il cavetto del ground all'esp8266 e possiamo procedere correttamente. E' importante controllare, se si usa una breadboard, di non usare cavetti troppo lunghi e di aver un collegamento stabile (un tester è l'ideale per verificare i collegamenti).

Il Baud Rate di funzionamento consigliato è 9600 o 115200.

Tenendo presente la mappa dei pin dell'ESP-01:

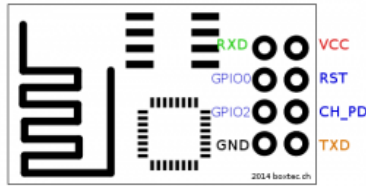


Figura 21 - ESP8266-01 PinOut

Il collegamento che si va ad effettuare è questo:

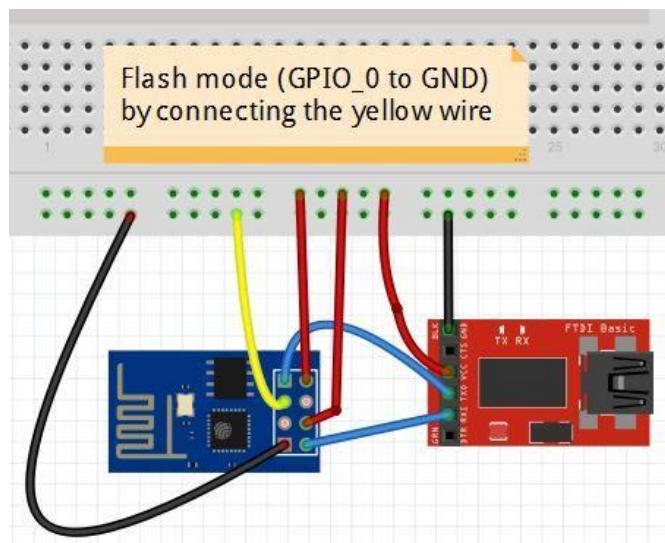


Figura 22 - Flashing ESP8266-01 Firmware

Per poter flashare il firmware è necessario portare a LOW il pin GPIO0 dell'esp8266. Come nel collegamento mostrato in foto dal cavetto giallo.

Una volta che il collegamento è stato effettuato, Flashare il firmware è abbastanza facile, grazie ad un'app software appositamente concepita per questa procedura.

Il software in questione è chiamato **NodeMCU Flasher**. Reperibile gratuitamente nella versione 32 bit o 64 bit dalla [Repository GitHub ufficiale](#).

Scaricato il firmware (.bin) e scaricato NodeMCU Flasher, possiamo collegare il modulo ESP al PC tramite l'adattatore seriale e iniziare con la procedura, seguendo quanto mostrato sotto:

- Assicurarsi che il GPIO-0 sia connesso al GND
- Apriamo NodeMCU Flasher e verifichiamo che la board venga riconosciuta sulla porta seriale (COM3 nel mio caso):

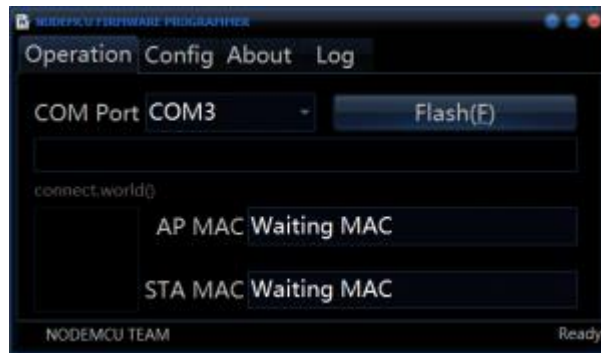


Figura 23 - Scelta della porta seriale

- Nella scheda **Config** andiamo a selezionare come prima voce il file .bin del firmware scaricato in precedenza (location: 0x00000).

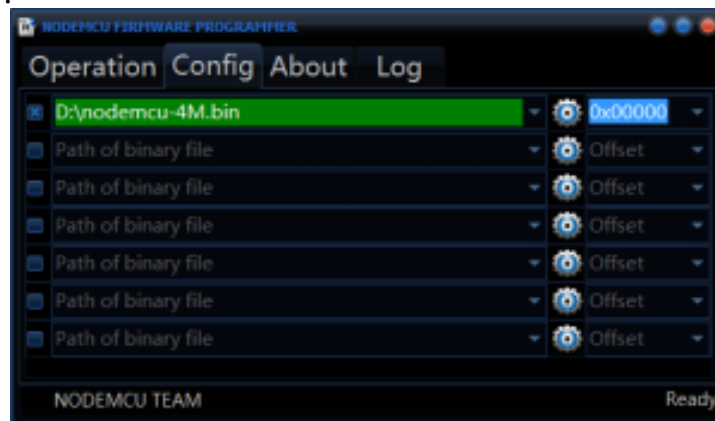


Figura 24 - Selezione del firmware

- Torniamo nella scheda Operation e avviamo la procedura facendo click su **Flash (F)**
- Aspettiamo che la barra di completamento arrivi al 100%. La procedura perchè vada a buon fine deve arrivare al 100%, se si interrompe prima qualcosa non va, sarebbe meglio ricontrollare i collegamenti.
- Se tutto è andato per il meglio vedremo una spunta verde in basso a sinistra nella finestra:

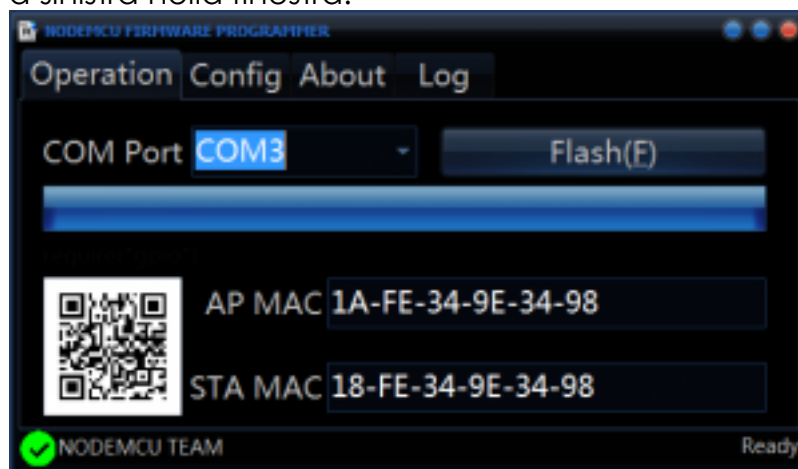


Figura 25 - Flashing completato

Se il modulo non viene riconosciuto dopo un minuto o più, probabilmente c'è un problema con la connessione o con i driver UART (TTL).

Online esistono servizi che permettono di scaricare build del firmware personalizzate, rimuovendo i moduli che non sono necessari alla nostra applicazione e dunque salvando preziosa memoria:

[frightanic.com/nodemcu-custom-build/](http://frightanic.com/nodemcu-custom-build/)

Da alcuni test effettuati, l'ultimo firmware disponibile risulta essere di 520kb. All'avvio del sistema la memoria ram disponibile è di circa 21kb. Con un firmware personalizzato, rimuovendo moduli non utilizzati, si può abbassare la dimensione del firmware a 300kb con una memoria RAM disponibile all'avvio di 36kb. Un bel guadagno.

Dopo che la procedura di flashing è stata portata a termine, si può disconnettere il pin GPIO-0 dal GND e provare a connettersi al modulo ESP usando Putty e caricare degli sketch di prova.

Nel caso volessimo usare il firmware NodeMCU Lua, per comunicare con l'esp8266 non possiamo usare l'IDE di Arduino. Piuttosto siamo costretti a cambiare IDE.

Oltre a Putty, in questo caso, l'IDE di riferimento è **ESPlorer** (<http://esp8266.ru/esplorer/>). Scritto interamente in java permette di eseguire comandi in run-time direttamente sulla board e salvare script all'interno del chip.

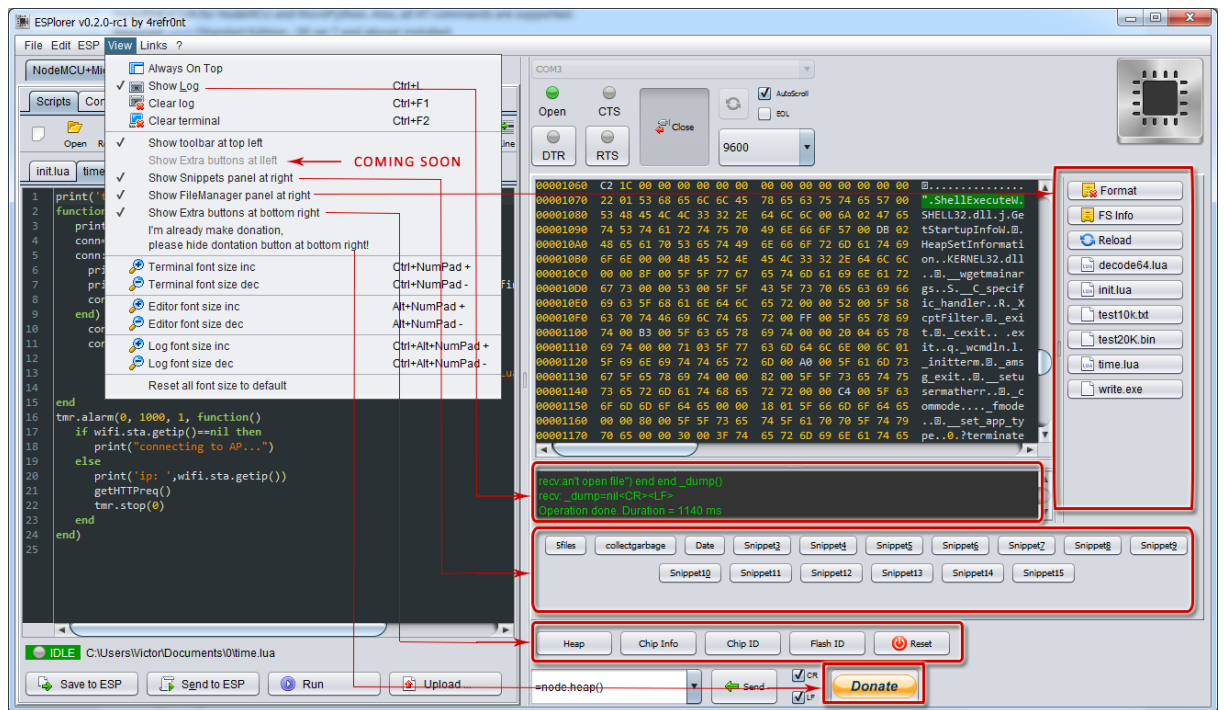


Figura 26 - ESPlorer IDE

Durante la connessione con ESPlorer o in generale con un terminale seriale, è possibile che la board non venga riconosciuta. Talvolta è sufficiente disconnettere e riconnettere l'unità seriale. Per evitare questi problemi qualcuno suggerisce di usare un'alimentazione ausiliaria per alimentare il modulo e non lo stesso adattatore seriale.

Personalmente consiglio l'utilizzo di un normale alimentatore da muro, 5V 1A dovrebbe essere sufficiente, e poi uno stepper down, modello LM2596, che serve a regolare la tensione a 3.3V (siamo noi a scegliere il voltaggio d'uscita, quindi è utile munirsi di tester e di cacciavite per regolare il potenziometro).

L'uso di uno stepper down non è obbligatorio ma legato al fatto che trovare a buon mercato alimentatori da parete che erogino direttamente 3.3V 1A è abbastanza difficile. Alimentare il modulo a 5V equivale a danneggiarlo irrimediabilmente.

Altri firmware alternativi oltre al già nominato NodeMCU Lua, sono:

- esp8266/Arduino
- Micropython
- Espruino



Figura 27 - Arduino Logo

[github.com/esp8266/Arduino](https://github.com/esp8266/Arduino)

Questo progetto open source mira a garantire il pieno supporto del chip ESP8266 all'interno dell'ambiente Arduino. Permette la scrittura di sketch usando funzioni e librerie di arduino con cui siamo già familiari.

L'esecuzione avviene direttamente su ESP8266 senza la necessità di altri microcontrollori esterni.

Questo core ESP8266-Arduino è dotato di apposite librerie per garantire la comunicazione tramite WiFi usando TCP, UDP. Possiede molteplici funzionalità: HTTP, mDNS, SSDP, DNS Server, Aggiornamenti OTA, gestione dei file su SD, controllo di servi e periferiche SPI, I2C etc.

A partire dalla versione di Arduino IDE 1.6.4, è possibile infatti installare dal Boards Manager di Arduino delle piattaforme di terze parti. Questa piattaforma è disponibile sia per Windows, che MAC OS o Linux.

La procedura d'installazione, grazie al gestore delle board è rapida e qui brevemente riassunta:

- Si installa regolarmente una versione di Arduino pari o superiore ad Arduino 1.6.5 dal sito ufficiale ([arduino.cc](http://arduino.cc)).
- Si avvia l'app e si va nella finestra "Preferences".
- Si aggiunge nel campo "Additional Board manager URLs" questa url:  
`http://arduino.esp8266.com/stable/package\_esp8266com\_index.json`
- Da Tools > Board Menu, si apre il Boards Manager e si installa la piattaforma esp8266 che appare.

A questo punto dal menù delle board, avremo delle voci in più che indicheranno tutte le board basate su esp8266, supportate dal core appena installato. Selezionata quella che fa al caso nostro saremo in grado di caricare lo sketch desiderato. Con questo metodo non c'è bisogno di flashare preventivamente il firmware nella board. Ad ogni upload dall'IDE di Arduino infatti, automaticamente sarà scritto il firmware Arduino. Da questo momento in poi nella trattazione, si darà per scontato l'utilizzo di questo firmware e dell'IDE di Arduino per l'upload degli sketch.

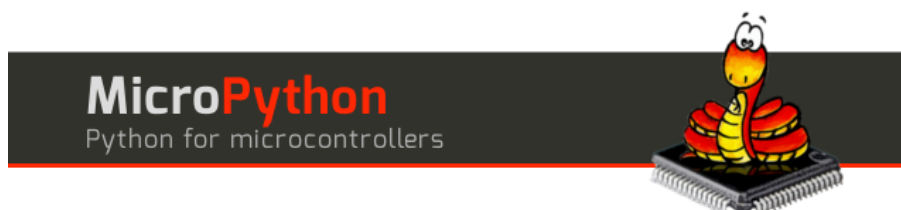


Figura 28 - MicroPython Logo

MicroPython è una leggera e veloce implementazione del linguaggio di programmazione Python 3, ottimizzata per girare sui microcontrollori. Da questo progetto nasce anche un microcontrollore che nativamente installa questo firmware.

Micropython è stato finanziato attraverso un progetto su kickstarter. Il software è open source e fornisce un sistema operativo di basso livello in python, utile per controllare parti elettroniche.

Il beneficio di usare Micropython risiede nei benefici di un linguaggio di programmazione d'alto livello, ben conosciuto, semplice e potente, tutto all'interno di un microcontrollore. Usare Micropython su ESP8266 permette di creare dispositivi per l'Internet of Things in modo molto più rapido di quanto potremmo fare usando C o altri linguaggi compilati.

Sul sito [micropython.org](http://micropython.org) è possibile scaricare i firmware aggiornati, comprare le board pre-configurate con micropython e consultare la documentazione dell'intero firmware.



Figura 29 - Espruino Logo

Come riporta il sito ufficiale [espruino.com](http://espruino.com), Espruino oltre ad essere il nome della board venduta e finanziata tramite crowdsourcing su kickstarter è anche il nome dato al firmware che da ottobre 2015 è disponibile in versione beta per l'installazione su ESP8266, è un progetto totalmente open source, sia nel firmware, nella documentazione che nei tool per upload degli script. Con repository dedicate su GitHub: [github.com/espruino](https://github.com/espruino)

Il firmware Espruino permette la programmazione in Javascript di applicativi. Espone molte delle funzionalità e dei metodi tipici dell'interprete Javascript, più molti altri, necessari per la comunicazione con i pin hardware sottostanti.

La documentazione completa ufficiale è reperibile su:  
[espruino.com/Reference](http://espruino.com/Reference)

L'upload degli script avviene tramite la **Espruino Web IDE**, realizzata come Estensione per Google Chrome (dunque è liberamente scaricabile dallo stesso Chrome Web Store).

E' presente anche una libreria che permette di implementare un client MQTT: [espruino.com/MQTT](http://espruino.com/MQTT) . Tuttavia Espruino su ESP8266 è un porting relativamente recente, e molti moduli ancora non funzionano portando ad errori di 'Out Of Memory', dovuti alla scarsa ottimizzazione del firmware e della scarsa gestione della memoria RAM. Tra i moduli non funzionanti, MQTT è uno di questi. Un modulo correttamente supportato invece è quello Web Socket.

Espruino non supportando ancora il modulo MQTT dovrebbe utilizzare una tecnologia di fallback quale quella basata su Web Socket, come accade già con i client web MQTT. In qualità di client web, essi utilizzano puramente il protocollo HTTP, e non MQTT. Per operare a livello di MQTT, si utilizzano i Web Socket, e i broker solitamente espongono oltre al classico server MQTT, anche un server Web Socket (MQTT over Web Socket).

Si potrebbe dunque immaginare uno scenario in cui le board basate su Espruino, al pari dei client web, comunichino tramite Web Socket e non MQTT.

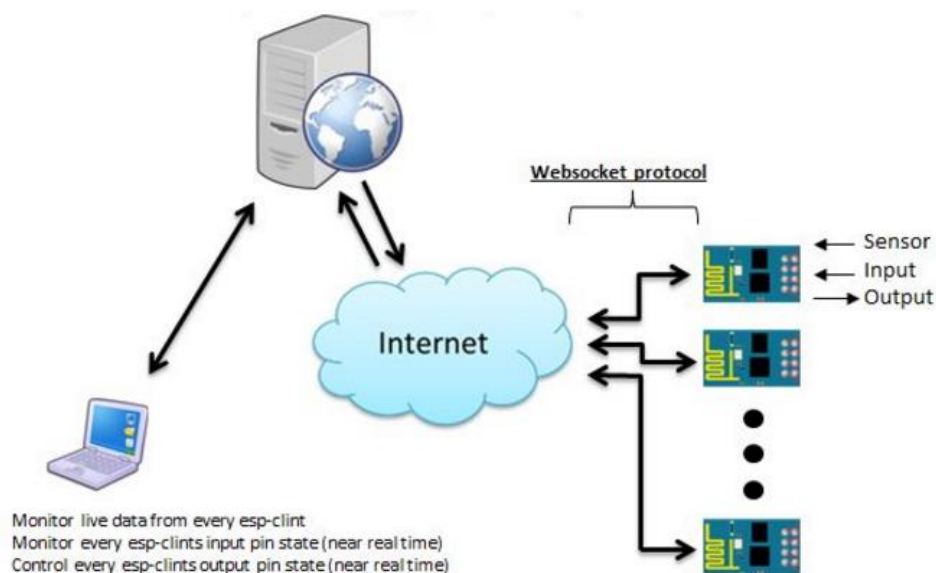


Figura 30 - Topologia basata su WebSocket



### 2.3.2 - Idea di Base

L'idea di base è di permettere una comunicazione bidirezionale fra i microcontrollori e la dashboard di controllo.

L'ideale sarebbe avere uno stack che permetta ai dispositivi microcontrollori di ricevere e inviare dati usando l'architettura standard TCP/IP.

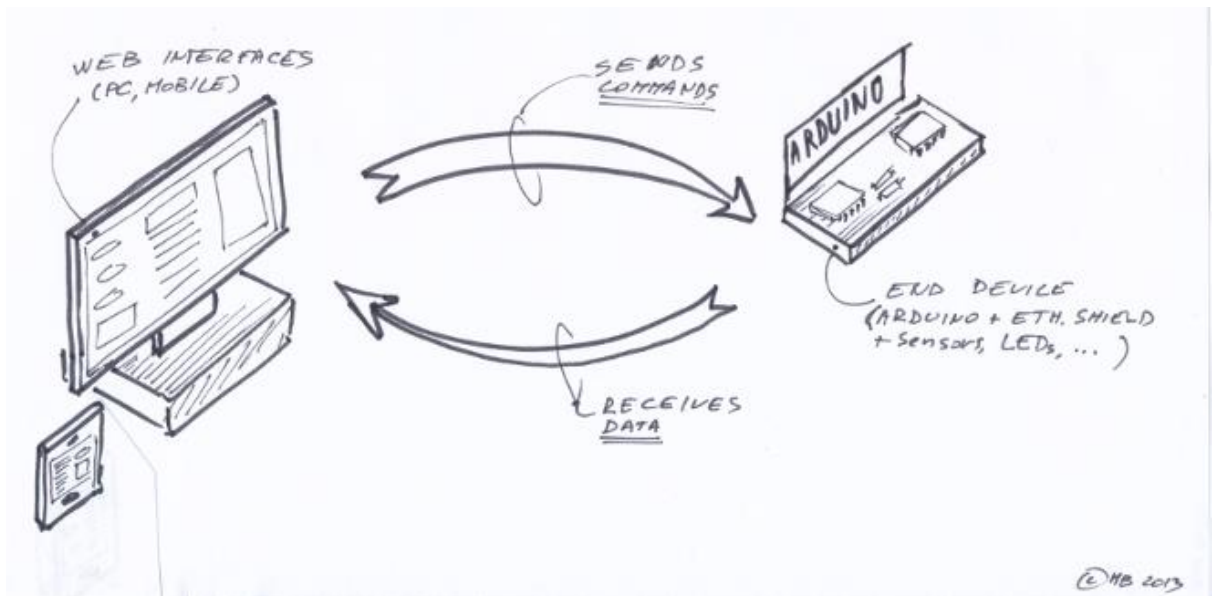


Figura 31 - Comunicazione Full-Duplex TCP/IP

La prima idea che può venire in mente è di usare delle API REST (REpresentational State Transfer), usando un applicativo lato server che ospiti le pagine dell'interfaccia web, rispondendo alle richieste provenienti dai dispositivi. La cosa sarebbe adatta alla ricezione di dati presupponendo che i dispositivi effettuino delle chiamate asincrone periodiche o quando un dato evento si verifica. Il client a questo punto può leggere questi dati direttamente dalla web GUI o usare un insieme di API differenti.

Il vero problema in quest'ottica di funzionamento è che non c'è una soluzione di tipo 'push', bensì una 'pull'. Ogniquale volta il client richiede un nuovo dato, è necessario aprire una nuova connessione http, senza sapere se troverà qualcosa di nuovo o meno. La frequenza di richiesta deve essere definita in anticipo: se c'è un incremento nel tasso di aggiornamento dei dati si può perdere molto di quest'ultimi nella parte client se non è previsto un sistema di buffering sul server.

Anche l'invio di comandi al dispositivo diventa problematico. Il dispositivo infatti per sapere se ci sono nuovi comandi deve interpellare il

server, dunque aprire una connessione http e verificare la presenza di nuovi comandi.

Una soluzione parziale potrebbe essere quella di aumentare il numero di richieste (sia dal client che dai dispositivi, verso il server). Di contro questa soluzione aggiunge overhead computazionale e un maggior consumo di banda.

### 2.3.3 - Perché MQTT?

MQTT risolve in modo elegante le problematiche sopra accennate. Da un'alta prospettiva, MQTT è un protocollo di messaggistica client-server che include un meccanismo di publish/subscribe: un client pubblica un messaggio su una coda specifica chiamata topic, ospitata su un broker. Un altro client può ricevere il messaggio registrandosi al topic. Ogni client è in grado di pubblicare e ricevere messaggi nello stesso tempo, e la comunicazione fra ogni client avviene attraverso il message broker. Due client non sarebbero in grado di scambiarsi messaggi senza un broker al centro.

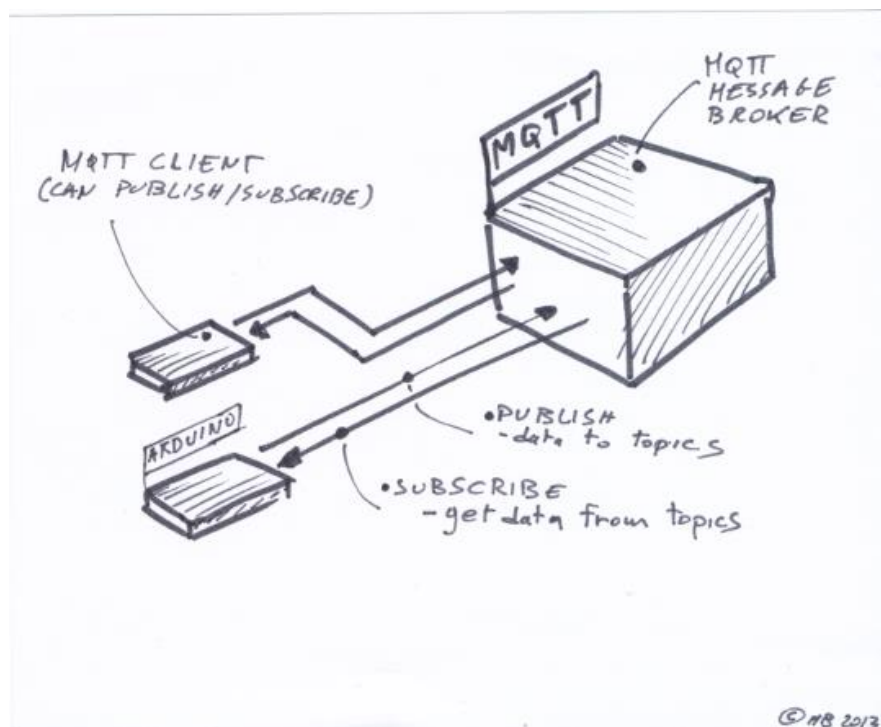


Figura 32 - Topologia MQTT

La differenza più rilevante se confrontato con HTTP (il protocollo usualmente dietro l'implementazione di REST) è che MQTT stabilisce una connessione a stati permettendo una comunicazione quasi real-time.

MQTT risolve molti dei problemi legati alla connettività, aprendo un nuovo mondo di possibilità per gli sviluppatori, al fine di creare applicazioni perennemente connesse.

E' pensato come complemento ai sistemi di messaggistica enterprise, per coprire il gap fra i numerosi dispositivi connessi e le applicazioni che possono pubblicare e consumare questi dati.

In pratica MQTT offre la possibilità di scrivere applicativi che pubblicano i propri dati come messaggi destinati al network MQTT, senza preoccuparsi di come altre applicazioni ricevano questi dati.

I messaggi sono pubblicati nei "topic", stringhe che possono richiamare i percorsi di sistema o la logica applicativa. Il messaggio – payload – è tipicamente un piccolo pacchetto, ma può arrivare anche ad occupare fino a 256MB.

I topic sul broker sono definiti come: "dominio/categoria/oggetto", dove dominio è un identificatore univoco legato a un ambiente specifico. Categoria è un nome all'interno di un dominio che può identificare un insieme di dispositivi correlati, e l'oggetto rappresenta il singolo dispositivo o sensore, l'ultimo pezzo non ulteriormente scomponibile nella gerarchia.

Tutti i messaggi mqtt scambiati sul broker possono essere di qualunque formato, nel nostro caso andremo a utilizzare il formato standard JSON.

Il protocollo MQTT standard è agnostico rispetto a queste scelte stilistiche. Sia i topic che i messaggi possono assumere il formato che più ci aggrada. Queste sono soltanto scelte dettate dalle convenzioni.

Altri concetti base legati a MQTT sono:

- 1) **Livelli di Qualità del servizio (QoS):** MQTT definisce 3 livelli di quality of service (QoS) per la consegna dei messaggi. Ogni livello indica un impegno maggiore da parte del server per assicurarsi che il messaggio venga consegnato. QoS più alti indicano una maggiore affidabilità nella consegna dei messaggi ma un impiego maggiore in termini di banda.
- 2) **Messaggi Trattenuiti (Retained Messages):** Con MQTT quando il server riceve un retained message su uno specifico topic, esso viene inoltrato a tutti i subscribers e in quanto retained message, salvato sul server e inviato a ogni subscriber futuro.

- 3) **Testamenti (Wills):** Quando un client si connette a un server, può informare il server di possedere un "testamento", ovvero un messaggio da pubblicare su uno o più topic specificati, in caso di disconnessione improvvisa. Un messaggio di testamento è particolarmente utile in sistemi di sicurezza dove i gestori devono sapere immediatamente quando un sensore remoto perde contatto con la rete.

#### 2.3.4 - HTTP o MQTT?

La comparazione con HTTP è d'obbligo per le seguenti ragioni:

- 1) HTTP è il protocollo più usato e diffuso. Praticamente qualunque dispositivo con uno stack TCP/IP lo usa.
- 2) Il protocollo HTTP usa un modello richiesta/risposta, che è il protocollo per lo scambio di messaggi più comune. MQTT usa un pattern publish/subscribe.

	<b>MQTT</b>	<b>HTTP</b>
<b>Orientamento</b>	Incentrato sul Dato	Incentrato sul Documento
<b>Pattern</b>	Publish/Subscribe	Request/Response
<b>Complessità</b>	Semplice	Media
<b>Dimensione messaggi</b>	Piccola, con un header binario di soli 2 byte.	Grande, i dettagli sullo status sono text-based
<b>Livelli di servizio</b>	3 QoS	Tutti i messaggi hanno lo stesso livello di servizio
<b>Libreria Extra</b>	Librerie per C (30KB) e Java (100KB)	Dipende dall'app (JSON, XML, etc.)
<b>Distribuzione dei dati</b>	Supporta 1 a 0, 1 a 1, 1 a N	Soltanto 1 a 1

- Orientamento: MQTT è incentrato sul dato, trasferisce i dati come fossero un array di byte. Ignorando il contenuto. HTTP è incentrato sul documento, supporta lo standard MIME che definisce il tipo di contenuto.
- MQTT usando il pattern Publish/Subscribe si dice sia scarsamente accoppiato, dunque i client non sono a conoscenza dell'esistenza di altri dispositivi. In HTTP invece col modello request/response è necessario che il client conosca l'indirizzo del dispositivo a cui si sta per connettere.
- La complessità di HTTP risiede nelle sue specifiche. Esistono molti codici restituiti (200, 400, 404, 500 etc.) e molti metodi (POST, PUT, GET, DELETE, HEAD, TRACE, CONNECT etc.). Su MQTT esistono pochi tipi di messaggi realmente utili allo sviluppatore (CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, DISCONNECT).
- Le dimensioni del messaggio più piccolo possibile su MQTT sono di 2 bytes. Tutto è codificato in binario. Su HTTP si usa un formato testuale, facilmente leggibile e che in parte ha contribuito alla sua popolarità. Tuttavia, per dispositivi dalle risorse computazionali esigue e reti con una scarsa larghezza di banda questo è tutt'altro che desiderabile.
- La qualità del servizio come già accennato, su MQTT raggiunge 3 livelli. Gli sviluppatori non devono pensare a realizzare ulteriori feature per garantire la consegna dei messaggi. Su HTTP non si hanno livelli di servizio, nè dunque la capacità di "ritentare" la consegna. Se questa è una feature necessaria all'applicativo gli sviluppatori dovranno implementarla per conto proprio.
- Infine per quanto riguarda la distribuzione dei dati, HTTP essendo point-to-point non prevede questa possibilità.

Anche dal punto di vista dei consumi MQTT risulta più parsimonioso, specie se la comunicazione client – server è frequente.

Keep Alive (Seconds)	% Battery / Hour			
	3G		Wifi	
	HTTPS	MQTT	HTTPS	MQTT
60	1.11553	0.72465	0.15839	0.01055
120	0.48697	0.32041	0.08774	0.00478
240	0.33277	0.16027	0.02897	0.00230
480	0.08263	0.07991	0.00824	0.00112

Figura 33 - Tabella dei consumi HTTPS e MQTT

### 2.3.5 - XMPP o MQTT?

XMPP è un protocollo di messaggistica precedentemente noto come Jabber basato su XML. XMPP sta per Extensible Messaging and Presence Protocol. Le sue specifiche sono talmente tante che non esiste un'implementazione completa di esso. La più nota è forse "ejabberd", un server XMPP open source. Siccome XMPP è stato progettato nella dotcom era, potrebbe non sembrare una scelta azzeccata per l'era moderna, tuttavia il 90% delle compagnie di messaggistica usa delle versioni di ejabberd, anche se pesantemente modificate (WhatsApp, APNs - Apple Push Notification server). Persino Google con il suo servizio di notifiche push, Google Cloud Messaging (GCM), permette l'utilizzo di XMPP.

Messo a confronto con MQTT, la prima cosa che notiamo è che quest'ultimo genera un volume di traffico inferiore, anche grazie al peso irrisorio dei pacchetti di keep-alive e alla codifica binaria. XMPP al contrario invia contenuti XML e anche l'overhead computazionale generato sui dispositivi è alto considerando la necessità di un parser XML. Inoltre, essendo MQTT un protocollo di trasporto, non definisce il formato dei messaggi. Sta all'utente decidere la tipologia di messaggi che più gli è consona. XMPP essendo un protocollo di messaggistica, definisce con attenzione i formati dei messaggi e richiede espressamente che siano in XML.

Ma la dimensione dei messaggi non è l'unica cosa che conta quando dobbiamo scegliere un protocollo. XMPP infatti offre molte feature di cui MQTT è completamente sprovvisto, come il concetto di identità o

federazione, assieme a molte estensioni che lo rendono privilegiato in ambito corporate.

### 2.3.6 - Broker: Mosca o Mosquitto

Esistono implementazioni di Broker MQTT per qualsiasi linguaggio di programmazione. Standalone, pronti all'esecuzione o integrabili in altre app esterne.



Figura 34 - Logo Mosquitto

Mosquitto è uno di questi. E' un progetto open source portato avanti dalla Eclipse foundation. Il broker Mosquitto implementa la versione 3.1.1 del protocollo MQTT. Il progetto mette a disposizione dei broker pubblicamente utilizzabili dalla community per eseguire test e realizzare applicazioni di prova, raggiungibili a questo indirizzo: [test.mosquitto.org](http://test.mosquitto.org)

Altri broker pubblicamente disponibili, offerti dai principali sostenitori di questo protocollo sono:

[iot.eclipse.org](http://iot.eclipse.org), [dev.rabbitmq.com](http://dev.rabbitmq.com), [broker.mqttdashboard.com](http://broker.mqttdashboard.com), [q.m2m.io](http://q.m2m.io), [mq.thingmq.com](http://mq.thingmq.com).

Mosca è un altro broker, open source e sviluppato in Node.js.

Si comporta più o meno come Mosquitto, ma dà la possibilità di essere facilmente integrato in nuove app Node-based.

Su di esso è basato l'intero progetto. La scelta è ricaduta su Mosca per l'ovvia facilità di personalizzazione, la possibilità di avviare oltre che a un server MQTT un server WebSocket di FallBack e in particolare per una gestione autonoma dell'autenticazione dei client mqtt.

Mosca così come da specifica del protocollo, supporta anche i Retained Message.

Il client che si sottoscrive a un topic non deve necessariamente conoscere l'esatto topic. Esso riceverà un messaggio conservato anche se si sottoscrive a un topic usando dei caratteri jolly (wildcards). Per esempio un client A che pubblica un messaggio di tipo retained su

`myhove/livingroom/temperature` e un client B che si sottoscrive successivamente a `myhome/#`, riceverà ugualmente il retained message dopo la sottoscrizione. Il client B è in grado di capire se ciò che ha ricevuto è un messaggio di tipo retained perchè il broker nel messaggio inviato avrà appositamente settato un flag retained a `true`.

I Retained message come visto aiutano i nuovi client a ottenere un aggiornamento di stato subito dopo la sottoscrizione a un topic, senza dover aspettare qualche altro client che pubblichi un nuovo messaggio. In altre parole, messaggi di questo tipo rappresentano l'ultimo valore utile conosciuto.

Per consegnare i retained Message, col Broker Mosca, è necessario aggiungere un sistema di persistenza, un DB o una delle strutture dati supportate per la memorizzazione dei messaggi e il successivo invio. Nella versione attuale del Broker, 4 sono i sistemi di persistenza utilizzabili: Redis, MongoDB, LevelUp, Memory.

Personalmente avendo a che fare con un numero di sensori relativamente basso ho preferito impiegare il sistema di persistenza più leggero e pratico possibile: LevelUp.

```
var mosca = require('mosca');
var server = new mosca.Server(settings);
var db = new mosca.persistence.LevelUp({ path: './db_levelup' }); // DB path
db.wire(server);
```

L'altra configurazione attuata, è quella di implementare un sistema d'autenticazione e autorizzazione per i client.

Su Mosca, 3 sono i metodi che possono essere usati per restringere l'accesso ai topic:

- `authenticate`
- `authorizePublish`
- `authorizeSubscribe`

Questi metodi vanno implementati per definire le regole d'accesso volute.

Nella Dashboard MQTT un file è stato creato affinché contenga credenziali e autorizzazioni, relative ai client e alle varie istanze della dashboard:

```
{
  "auth_required": true,
```



```

"users": {
  "alice": "pass123",
  "bob": "ciao123",
  "frankenstein": "123456",
  "chrome": "secret123"
},
"pub_authorizations": {
  "test/topic": ["alice", "bob", "frankenstein", "chrome"],
  "all": ["alice", "bob", "frankenstein", "chrome"],
  "esp8266/alice": ["alice", "chrome"],
  "esp8266/frankenstein": ["frankenstein", "chrome"]
},
"sub_authorizations": {
  "priv8": ["alice", "chrome"]
}
}

```

L'utente "chrome" rappresenta tutte le istanze della dashboard MQTT. Ogni dashboard infatti è trattata come fosse un client con "accessi privilegiati".

Questo file, abbinato ai metodi sopra visti detta le regole d'accesso, in questo modo:

- Un client innanzitutto per connettersi al broker deve prima autenticarsi. L'autenticazione consiste nell'invio di username e password, che devono corrispondere ad una voce dell'oggetto "users" nel file delle credenziali.
- L'authorizePublish è così gestita: solo gli utenti presenti nel file json sotto il topic specifico sono autorizzati a pubblicare in quel topic. Se non esiste il topic, allora è pubblico, non ha restrizioni. Ad esempio, dal file visto sopra sappiamo che solo alice e chrome sono in grado di pubblicare nel topic "esp8266/alice". Mentre nel topic "all" tutti gli utenti sono in grado di pubblicare.
- Allo stesso modo la sottoscrizione è gestita dall'oggetto "sub\_authorizations". Se un client tenta di sottoscrivere ad un topic con accesso ristretto, la funzione authorizeSubscribe verifica che l'username del client sia presente nella lista di client autorizzati. Nel file sopra l'unico topic con restrizioni per la sottoscrizione è "priv8", soltanto alice e chrome potranno ricevere aggiornamenti da quel topic. Ciò nonostante esso non richiede autorizzazioni speciali per la pubblicazione, chiunque può pubblicare sul topic "priv8".

Infine la semplicità di Mosca risiede soprattutto nella gestione degli eventi del Broker. Riprendendo un pò la sintassi event-driven di Node.js si gestiscono così gli eventi tipici del protocollo MQTT:

```
// eseguito quando un client si connette
server.on('clientConnected', function(client) {
    // ...
});

// eseguito quando un client si disconnette
server.on('clientDisconnected', function(client) {
    // ...
});

// eseguito quando un messaggio è ricevuto
server.on('published', function(packet, client) {
    // ...
});

// eseguito quando un client si sottoscrive a un topic
server.on('subscribed', function(topic, client) {
    // ...
});

// eseguito quando un client si disiscrive da un topic
server.on('unsubscribed', function(topic, client) {
    // ...
});
```

### 2.3.7 – Clients e Sketch MQTT

Ci sono tanti client MQTT disponibili per i sistemi più disparati.

Quello più diffuso per Node.js è MQTT.js reperibile da [npmjs.com/package/mqtt](https://npmjs.com/package/mqtt) .

Altre librerie per altri linguaggi di programmazione sono realizzate dal progetto Paho di Eclipse: [eclipse.org/paho/](https://eclipse.org/paho/)

(per realizzare client multipiattaforma sono l'ideale, più avanti in particolare viene usata la libreria in Javascript per realizzare il client da Web Browser, sfruttando i WebSocket).

Ancora, altre librerie sono disponibili per l'utilizzo su dispositivi embedded. Arduino, NodeMCU, Esp8266, etc. Ognuno rappresenta un endpoint all'interno della nostra rete MQTT, in grado di connettersi a un broker, autenticarsi, pubblicare e sottoscrivere ai topic.

Per il firmware Arduino la libreria più diffusa è 'arduino-client-for-mqtt' scritta da Nick O'Leary ([github.com/knolleary/pubsubclient/](https://github.com/knolleary/pubsubclient/)), con cui possiamo:

1. Come sottoscrittore di un topic, restare in attesa di comandi nella forma di messaggi MQTT json.
2. Come publisher inviare dati rilevati dai sensori su uno o più topic.

Nota. Per permettere l'invio di messaggi di lunghezza superiore a 128byte è necessario incrementare in PubSubClient.h la dimensione dei pacchetti. Da 128 a 256 per esempio, riferendoci alla riga: `#define MQTT_MAX_PACKET_SIZE 128`

Segue un esempio di client MQTT generico su firmware Arduino, eseguibile su piattaforme basate su esp8266 e dunque connesse alla rete WiFi:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Credenziali per l'accesso alla rete WiFi

const char* ssid = ".....";
const char* password = ".....";
const char* mqtt_server = "broker.mqtt-dashboard.com";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;

void setup_wifi() {

  delay(10);
  // Connessione al network WiFi
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
```

```

Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // LED on se si riceve '1' come primo carattere.
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW); // LED on quando il voltaggio è LOW
  } else {
    digitalWrite(BUILTIN_LED, HIGH); // LED off
  }
}

void reconnect() {
  // Ritentiamo finchè non siamo connessi
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Tentativo di connessione
    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
      // Una volta connessi pubblichiamo un messaggio
      client.publish("outTopic", "hello world");
      // ... e ci sottoscriviamo nuovamente ai topic
      client.subscribe("inTopic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // 5 secondi d'attesa prima di un nuovo tentativo
      delay(5000);
    }
  }
}

void setup() {
  pinMode(BUILTIN_LED, OUTPUT); // Inizializziamo il BUILTIN_LED come un output
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;
    ++value;
    snprintf (msg, 75, "hello world #%ld", value);
    Serial.print("Publish message: ");
    Serial.println(msg);
  }
}

```

```
client.publish("outTopic", msg);  
}  
}
```

Questo sketch può essere usato come base per l'invio di dati provenienti dai sensori o per la ricezione di comandi attraverso il network MQTT. L'esecuzione è banale, una volta caricato sulla board non farà altro che tentare una connessione wifi a ripetizione finché non risulterà connessa. A quel punto la board tenterà di connettersi al server MQTT indicato, effettuando un tentativo ogni 5 secondi. Una volta stabilita la connessione col server MQTT ogni 2 secondi invierà un messaggio con un numero, incrementato di volta in volta, sul topic chiamato 'outTopic'.

Consultando la documentazione della libreria usata, facilmente possiamo migliorare lo sketch e introdurre ad esempio l'autenticazione, in questo modo:

```
if (client.connect("arduinoClient", "testuser", "testpass")) {  
  client.publish("outTopic", "hello world");  
  client.subscribe("inTopic");  
}
```

L'idea a questo punto è quella di avere delle board basate su esp8266, con un firmware Arduino, che possano comunicare sul network MQTT ed essere facilmente configurate.

Per questa seconda parte, introduco una nuova libreria. Nata per facilitare la configurazione dei dispositivi embedded quali l'esp8266 e le board da esso derivate.

Nel caso in cui si dovesse ad esempio cambiare credenziali d'accesso al router o trasportare i moduli in altre reti, risulterebbe alquanto scomodo dover riscrivere il firmware per ogni device, per aggiornare le credenziali d'accesso wifi scritte brutalmente nel codice sorgente. Anche nel caso in cui si volessero aggiornare le credenziali d'accesso del broker MQTT, risulterebbe scomodo collegare fisicamente i moduli in seriale, aggiornare broker, host, username e password nel codice sorgente e caricare nuovamente il tutto sulla board.

Fortunatamente la libreria open source, **WiFiManager** ([github.com/tzapu/WiFiManager](https://github.com/tzapu/WiFiManager)), semplifica il processo di configurazione remota. Infatti nessuna credenziale viene esplicitamente scritta nello sketch, ma viene salvata su EEPROM, rimanendo disponibile anche al successivo riavvio e proponendo anche una soluzione d'acquisizione più logica ed efficace sotto ogni punto di vista:

- All'avvio dell'ESP si entra in Station Mode, cercando di connettersi all'Access Point precedentemente salvato.
- Se non ci sono network salvati o il collegamento fallisce si entra in Access Point Mode, avviando DNS e WebServer (su un ip di default 192.168.4.1).
- Usando qualsiasi dispositivo dotato di wifi e browser, ci si può connettere all'access point generato dall'ESP.
- Grazie al Captive Portal e al server DNS qualunque sito si tenterà di visitare dall'access point si verrà rimandati alla pagina di configurazione.
- Dalla pagina di configurazione verranno elencati gli access point disponibili alla connessione, basterà dunque selezionarne uno e inserire la password ed eventualmente inserire gli altri parametri opzionali richiesti.
- L'ESP si riavvierà tentando la connessione. Se la connessione ha successo il controllo passa alla logica implementata nella nostra app, altrimenti si avvia nuovamente un access point in attesa di una nuova configurazione.

Dalle ultime release è stata introdotto la possibilità di salvare anche parametri addizionali, oltre al semplice SSID e password, anche l'host MQTT e la porta, nonché le stesse credenziali d'accesso al network MQTT.

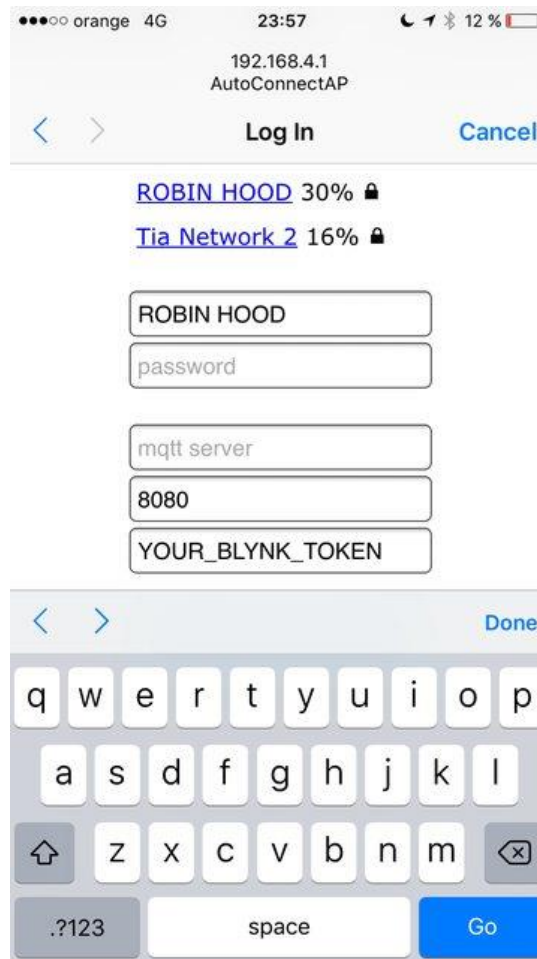


Figura 35 - WiFi Manager Captive Portal

Lo sketch visto è pensato per firmware Arduino, ma anche su firmware Lua è possibile interagire ovviamente con il protocollo MQTT, usando l'IDE *ESPlorer*.

Tuttavia essendo il firmware pensato per caricare sketch interpretati, non è raro che si arrivi durante l'esecuzione spesso a reset improvvisi o a errori di saturazione della memoria. Motivo per cui gli sketch della trattazione vanno riscritti e ottimizzati con un occhio di riguardo alla semplicità e al consumo della RAM. In questo caso i file da caricare sulla memoria flash dell'esp8266 diverrebbero 2:

- init.lua (di default è lo script eseguito all'avvio dal firmware)
- server.lua (contiene il programma vero e proprio)

Quest'ultimo dev'essere caricato sulla memoria, dopodichè per ottimizzare il consumo di RAM, compilato inviando tramite seriale alla board il comando:

```
node.compile('server.lua')
```

a questo punto viene creato il file `server.lc` (bytecode compilato) e possiamo rimuovere il file `server.lua` e liberare dunque spazio in memoria:

```
file.remove('server.lua')
```

Se il caricamento è avvenuto con successo, possiamo riavviare la board (`node.restart()`) e in seriale ci verranno stampate le informazioni di connessione alla rete, se tutto è andato per il meglio.

Per ogni nuovo chip che programmiamo, se vogliamo impostare manualmente un IP Locale, dobbiamo ricordare di modificare nel sorgente di `server.lua` la variabile `IPADR`, ovvero l'ip locale che la board andrà a richiedere per se al server DHCP del router. Assicuriamoci che l'ip non sia richiesto da altri dispositivi e che non sia già stato assegnato all'interno della rete locale.

Un approccio differente consisterebbe nell'uso di una libreria per la creazione di interfacce REST su protocollo HTTP, come ad esempio: **aREST**, essa crea un server e fornisce degli endpoint per interagire velocemente con i pin GPIO della board ([github.com/marcoschwartz/aREST-ESP8266](https://github.com/marcoschwartz/aREST-ESP8266)), con tutte le limitazioni descritte all'inizio. E' possibile testare gli endpoint delle API tramite Postman. Da cui osserviamo oltre agli header di risposta, anche il tempo di risposta, che varia dai 70ms ai 240ms.

### 2.3.8 - Chrome APP

La parte relativa all'interfaccia web pone delle problematiche rilevanti a causa della tecnologia sottostante in un browser HTTP o web app basata su chrome, come nel nostro caso. In primis abbiamo l'impossibilità di configurare un socket TCP, ovvero una connessione persistente, all'interno di una pagina web. Questo è un requisito di base per lavorare con MQTT.

Grazie all'introduzione nel 2011 delle specifiche `WebSocket`, questo non è più un grande problema, dal momento che quasi tutti i browser moderni adesso sono in grado di mantenere una connessione persistente. Un `WebSocket` è un canale di comunicazione full duplex su una singola connessione TCP, basato su un protocollo specifico, differente dall'HTTP (l'unica parte che hanno in comune è il processo di handshaking interpretato dal server come una 'upgrade request'). In ogni caso, un `WebSocket` è esattamente ciò che è necessario per implementare un'interfaccia web basata su MQTT.



L'architettura che ne consegue è qualcosa del genere:

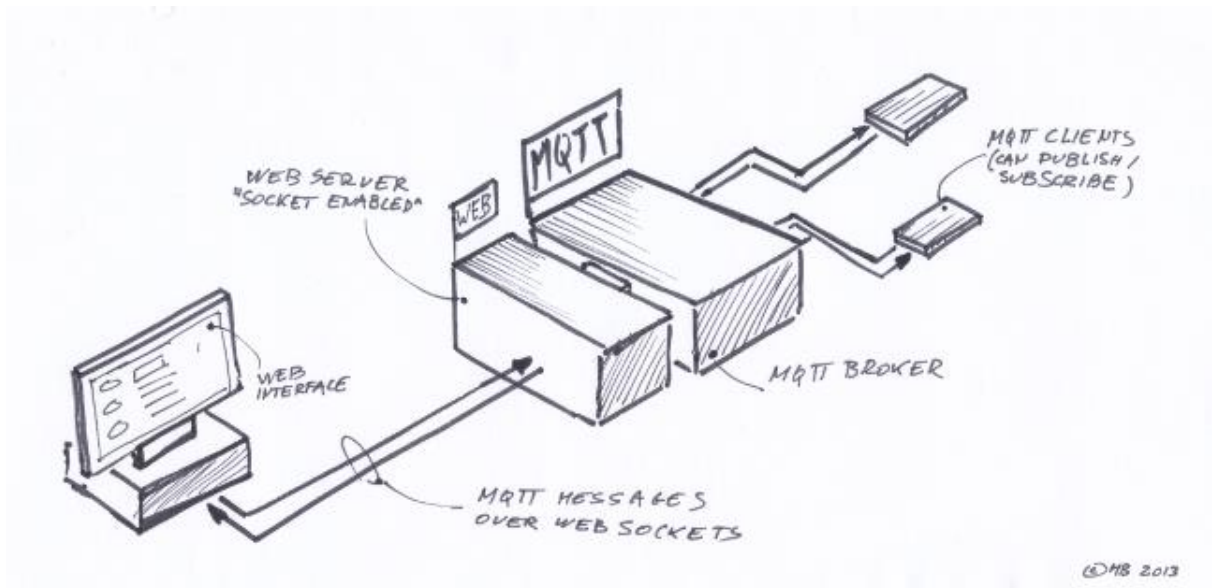


Figura 36 - MQTT e WebSocket

L'Interfaccia web viene gestita con questo meccanismo attraverso la libreria javascript prodotta dal progetto Paho di Eclipse Foundation e reperibile a questo link: [eclipse.org/paho/clients/js/](http://eclipse.org/paho/clients/js/) . Essa è stata opportunamente modificata per utilizzare lo storage di chrome (chrome.storage.local) e non il 'localStorage' tipico dei Web Browser. Nel riscrivere la libreria le difficoltà maggiori si hanno quando si passa da un sistema di storage sincrono (localStorage), ad un sistema di storage asincrono (chrome.storage.local).

Il server Mosca impiegato nel progetto prevede la possibilità di utilizzare MQTT da web browser e dunque implementa un server WebSocket.

La Chrome App viene gestita come fosse un client Web con privilegi particolari e condivide la stessa interfaccia web che verrà mostrata successivamente quando si parlerà del progetto iot-433mhz.

Di seguito si mostrano API esposte e sfruttate dalla Dashboard, Classi e rispettive descrizioni:

<p><code>view.render.template(whatToRender, whereToRender, dataObj)</code> - Metodo per renderizzare un template in una specifica posizione nella pagina.</p> <p><code>view.render.templateFile(whatToRender, whereToRender, dataObj, loaded_callback)</code> - Metodo per renderizzare in una pagina un template preso da un file esterno.</p> <p><code>tools.tts(text, language, rate)</code> - Metodo di sintesi vocale che sfrutta le API di Text To Speech esposte da Chrome.</p>
--

`tools.notify(options, callback)` – Notifica di Sistema con la possibilità di includere un'immagine.

`tools.storage.put(key, ObjToSave, callback)` – Inserimento di un nuovo oggetto nello Storage in Cloud, Sincronizzato da Chrome.

`tools.storage.get(key, callback)` – Metodo per ottenere un dato oggetto presente nello storage.

`tools.storage.getAll(callback)` – Fetch di ogni oggetto nello Storage.

`tools.storage.set(obj, callback)` – Modifica di un oggetto in Storage.

`tools.storage.remove(key, callback)` - Rimozione di un oggetto in Storage.

`tools.storage.removeAll()` – Rimozione di tutti gli oggetti salvati in Storage.

`tools.storage.getRemainingSpace()` – Restituite in Kb, lo spazio disponibile nello Storage.

`tools.storage.addListener(key, callback)` – Aggiungiamo un listener per un dato oggetto in Storage, così che una volta modificato venga eseguita la funzione specificata come parametro di callback.

`tools.storage.startGlobalListener()` – Listener per ogni oggetto in Storage.

`app.broker.isConfigured(callback)` – Restituisce true o false a seconda se le impostazioni del Broker siano state configurate.

`app.broker.loadSettings(callback)` – Carichiamo le impostazioni di configurazione del Broker.

`app.broker.saveSettings(data, callback)` – Salviamo le impostazioni di configurazione del Broker.

`app.config.get(key, callback)` – Otteniamo una specifica opzione generica di configurazione per l'app.

Le 2 classi definite sono invece:

`Page(templateFile, dom_element)` – Classe che definisce una nuova pagina

- `#setDomElement(_dom)` – Metodo per settare l'elemento del dom che identifica la nuova pagina
- `#setListeners(callback)` – listener generici sulla pagina
- `#render(obj)` – Renderizziamo la nuova pagina secondo i dati ricevuti come parametro

`MQTTClient(settings_obj)` – Classe per instanziare il client MQTT della Dashboard

- `#_constructor()` – Costruttore che inizializza le impostazioni iniziali.
- `#getHost()` – Otteniamo l'host Broker
- `#getPort()` – Porta usata per la connessione

`#setBroker(host, port, id_client)` – Configuriamo un nuovo broker, host, porta e id client.

`#onConnectionLost(callback)` – Funzione chiamata quando la connessione viene persa

`#onConnectionError(callback)` – Funzione chiamata quando c'è un errore di connessione.

`#onConnectionEstablished(callback)` – Funzione chiamata quando la connessione è stabilita.

`#onMessageArrived(callback <message>)` – Funzione chiamata all'arrivo di un nuovo messaggio.

`#connect()` – Connessione al Broker configurator.

`#publish(topic, payload, qos)` – Pubblicazione di un messaggio sul topic specificato.

`#Subscribe(topic, callback_success, callback_failure, qos)` – Sottoscrizione al topic specificato

### 2.3.9 - GCM - Google Cloud Messaging

Google Cloud Messaging (GCM) è un servizio gratuito che permette agli sviluppatori di inviare messaggi fra server e app client. In entrambe le direzioni, dai server ai client o dai client ai server.

Esso è utile ad esempio per informare un client della presenza di nuovi dati sul server, così che possano essere scaricati. Un messaggio GCM può trasportare fino a 4Kb di dati. Il servizio gestisce tutti gli aspetti necessari al funzionamento, dalla coda dei messaggi fino alla consegna a destinazione.

Implementa quelle che vengono chiamate più comunemente, notifiche Push.

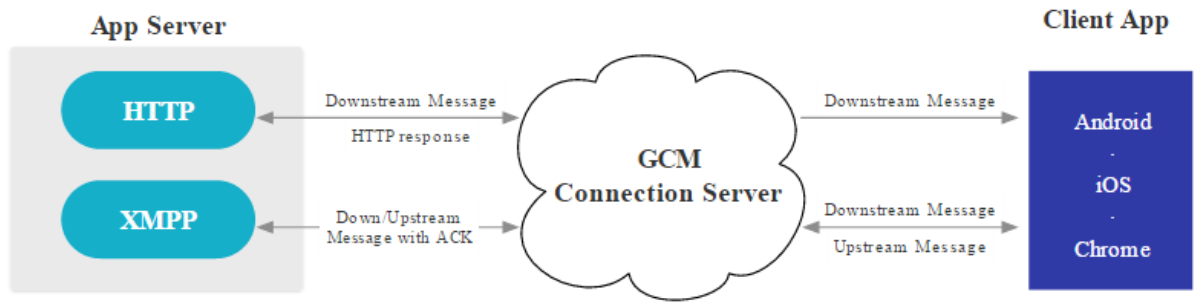


Figura 37 - Google Cloud Messaging

Nella nostra visione piu' ampia, l'idea è quella di avere la possibilita' di controllare e annunciare prossime e nuove funzionalita' della stessa dashboard tramite avvisi diretti (push notification) sui computer di chi installa la chrome app.

Queste le 'tipologie' di notifiche push implementate dall'app: card, voice, message. E' possibile inviare con una singola notifica push, piu' dati nel formato *chiave* → *valore* e dunque avere la possibilita' di invocare contemporaneamente piu' tipologie di notifica, message, card, vocale, combinate assieme.

Una notifica push di tipo voice viene interpretata con il motore di sintesi vocale.

Non è stata ancora introdotta la GCM topic notification nelle chrome-app; essa fa si che i client possano registrarsi ai topic e l'admin possa mandare loro notifiche push tramite gcm su quei topic specifici. Il sistema di 'fallback' attualmente implementato consiste nell'adesione da parte del client (inteso come dashboard mqtt) ad un 'gruppo'. Il server lato back-end simula la 'notifica tramite topic', mantenendo alcune informazioni sul client, tra cui, il nome del client (nome del pc su cui è installata la dashboard), il token, e il gruppo a cui aderisce.

Avendo informazioni sul gruppo è possibile inviare notifiche push globali. (esempi di gruppi sono: linux-os, win-os, mac-os, it-lang, en-lang).

Il server che dal lato back-end gestisce le registrazioni dei client è basato su questo modulo Node.js: [npmjs.com/package/node-pushserver](https://npmjs.com/package/node-pushserver) (permette sia ricezione dei tokens che l'invio di notifiche push).

L'altro tipo di notifica push è quella definita come 'card':

Gestire le card che arrivano tramite notifiche push, pone alcuni compromessi nel workflow dell'applicazione.

Le cards sono infatti gestite ad un livello superiore. La pagina *background.js* è caricata all'avvio dell'app (come da ciclo di vita delle chrome-app), dunque nel suo scope non è presente l'oggetto 'window' necessario per renderizzare i contenuti sull'interfaccia.

Allo stesso modo, i design pattern suggerirebbero di emettere un evento in caso di arrivo di una 'card'. Ma il problema persiste in quanto l'evento andrebbe catturato da uno script in grado di modificare direttamente il DOM e questo sappiamo non essere possibile, poichè *background.js* è isolato dal normale flusso applicativo.

Lo stratagemma usato per garantire una comunicazione fra il GCM gestito da *background.js* e i moduli di rendering è tanto semplice quanto efficace: usare le API *chrome.storage.local* per memorizzare le cards (direttamente dai listener del GCM all'arrivo di una card) e impostare dall'altro lato, un listener sullo storage, per catturare in tempo reale ogni aggiunta o modifica dello storage, e dunque renderizzarne il contenuto.

Infine l'ultima tipologia di notifica push è quella di tipo 'message':

Che identifica la notifica push classica, con un pop up a comparsa in basso a destra nella schermata del sistema. Eventualmente può essere accompagnata da un'immagine.

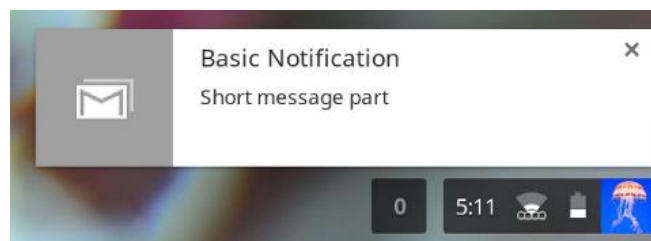


Figura 38 - Esempio di notifica Push Desktop

## 2.4 - Handshaking

Per Handshaking si intende la fase in cui una Dashboard si connette al broker e ha bisogno di conoscere tutti i dispositivi connessi ad esso e i relativi topic.

La Dashboard volutamente non si appoggia nelle fasi di HandShaking a funzioni scritte all'interno del Broker. Questo garantisce un "carattere" più universale e si limita quanto più possibile l'interdipendenza al broker stesso, rimanendo così meramente preposto alla gestione delle

connessioni, dei topic e all'inoltro dei messaggi. L'handshaking è una fase preliminare, la più importante in assoluto poichè mette in relazione clients e broker.

- Com'è implementata la fase di HandShaking?

1) I Client connessi al Broker si sottoscrivono al 'topic globale': `devices/` e anche ad un proprio topic personale: `devices/<clientUsername>/`

Rimanendo dunque in ascolto per comandi su entrambi i topic.

2) La dashboard è autorizzata a pubblicare sul topic globale `devices/`.

Quando lo fa, gli altri client leggono il comando presente all'interno del payload inviato dalla dashboard e leggendo 'self\_advertise' sanno di doversi annunciare sul topic `dashboard/handshake/`

In alternativa: si potrebbe comunicare il topic su cui la dashboard si aspetta un messaggio di presentazione da parte di tutti gli altri client. Quindi pubblicare ad esempio in `/handshaking`:

```
{"name":"<DashboardClientID>", "topic": "/dashboard"}
```

3) I client in ascolto su `/devices` inoltrano le loro specifiche sul topic della dashboard, `dashboard/handshake/` sia inizialmente quando connessi per la prima volta al broker, sia quando richiesto loro dalla Dashboard MQTT.

Un esempio di specifica è:

```
{"board_name": "name-xyz", "local_ip": "",  
  "topics": [{ "topic": "devices/actuator/bedroom/", "type": "on_off",  
    "size": "large/medium/small", "img": "", "title": "Bedroom - Abat Jour",  
    "body_content": "brief description...", "background_color": "#c1c8cc"},  
    { "topic": "devices/actuator/bedroom2/", "type": "on_off",  
      "size": "large/medium/small", "img": "", "title": "Bedroom - lamp",  
      "body_content": "other description...", "background_color": "#E2E26D"}] }
```

che denota per l'appunto alcune informazioni tipiche della board e i topics che sfrutta. Una board può comunicare alla dashboard più topics, questo significa che può aver collegati più relay (attuatori), oppure altri tipi di sensori (attivi o passivi).

Ogni topic è accompagnato da un tipo, che può essere: *on\_off*, *info*, *graph*, *custom*.

Ogni topic comunicato dal dispositivo, viene gestito dalla dashboard come una scheda separata.

Scheda che verrà interpretata e inserita nella sezione 'Devices' della Dashboard a seconda della tipologia.

3) La Dashboard MQTT è iscritta al topic `graveyard/` e ogni client mqtt imposta alla connessione un will-topic e un will-message, ovvero un topic di testamento con un relativo messaggio di testamento. Quando il client perde la connessione con il broker o semplicemente va offline, il broker recapita i messaggi di testamento sui topic specificati. In questo caso il protocollo di handshaking prevede che ogni client come topic di testamento definisca `graveyard/` e come messaggio di testamento, semplicemente il suo `clientId/clientUsername`. Un esempio è:

`<'graveyard/', 'pluto'>`

## 2.5 - Altri sensori e collegamenti

Altri esempi di sketch vengono di seguito proposti, assieme al relativo schema di collegamento.

- Led di Notifica

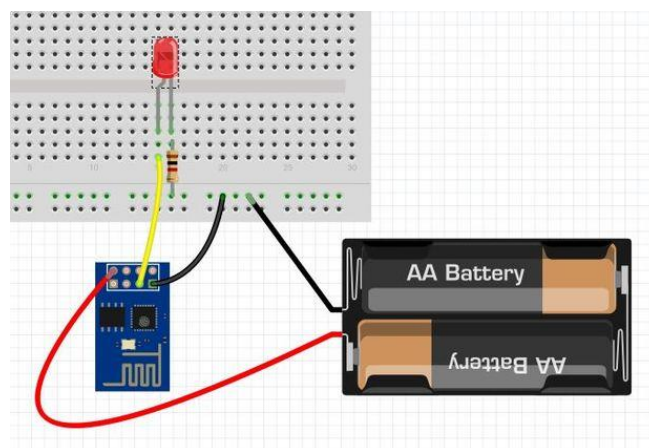


Figura 39 - ESP8266-01 e Led di notifica

- Esempio di Sensore DHT 22 ed esp8266 per l'invio della temperatura con MQTT

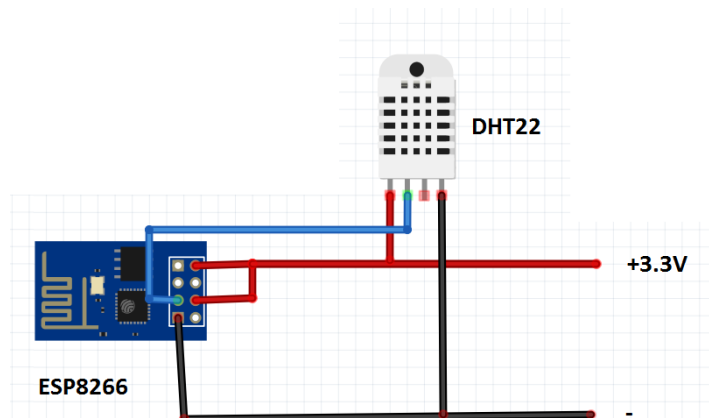


Figura 40 - ESP8266-01 e DHT22

O in alternativa usando NodeMCU:

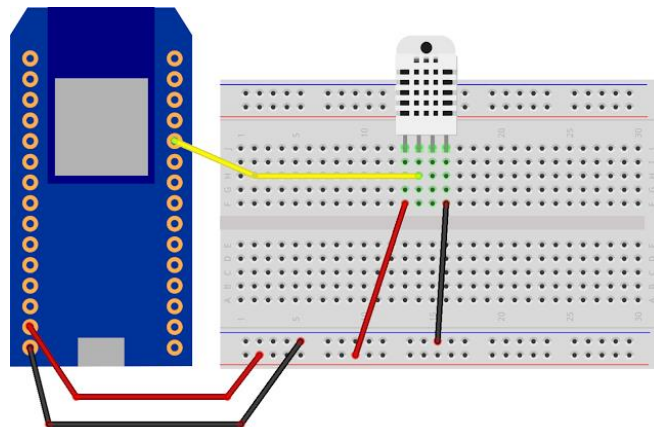


Figura 41 - NodeMCU Devkit e DHT22

- Ancora possiamo pensare di voler collegare un piccolo OLED per mostrare dati provenienti dal network MQTT in tempo reale:

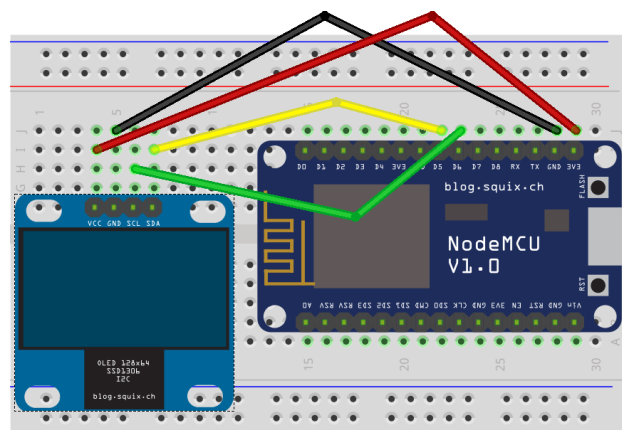


Figura 42 - NodeMCU devkit e OLED 0.96"

- Oppure utilizzare un sensore di movimento:



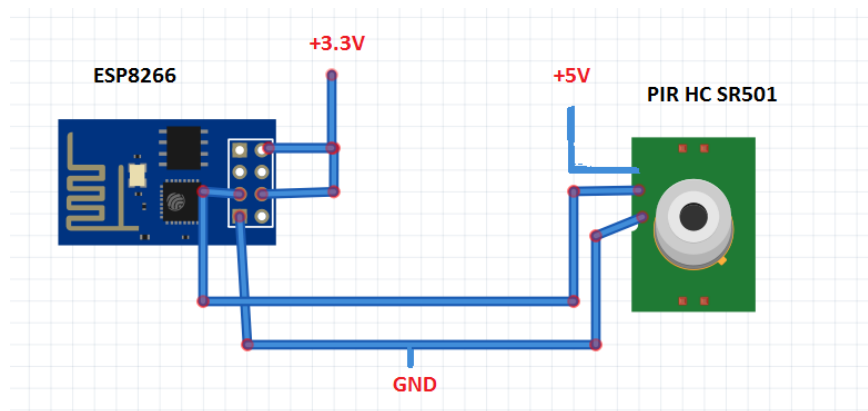


Figura 43 - ESP8266-01 e sensore PIR

MQTT è un protocollo utilizzato e consigliato anche da Amazon AWS. Amazon stessa offre delle alternative al costruirsi un Broker in casa. Infatti nella sua piattaforma, AWS IoT, viene proposto un gateway di dispositivo per MQTT, in grado di connettere miliardi di dispositivi e inviare migliaia di miliardi di messaggi. Citando lo stesso Amazon:

*“Con il gateway di dispositivo, i messaggi vengono scambiati utilizzando un modello di pubblicazione/iscrizione che consente comunicazioni sia univoche e sia verso più dispositivi. Con quest'ultimo modello di comunicazioni, AWS IoT consente a un dispositivo connesso di trasmettere dati a diversi dispositivi iscritti al servizio su un determinato argomento. Il gateway di dispositivo supporta i protocolli MQTT e HTTP 1.1 e offre l'implementazione del supporto per protocolli proprietari e protocolli legacy. Il gateway di dispositivo ridimensiona automaticamente le risorse per supportare fino a un miliardo di dispositivi senza dover effettuare il provisioning di un'infrastruttura.”*

...

*“Gli starter kit di AWS IoT consentono di passare rapidamente dall'idea al prototipo. Questi kit fisici sono sviluppati per accelerare lo sviluppo di prototipi forniti nel cloud di dispositivi collegati e di connettersi ad AWS IoT in modo sicuro. Questi kit includono microprocessori di sviluppo, schede di sviluppo, sensori e attuatori”.*

Tra gli starter kit proposti da Amazon per iniziare a sviluppare i propri dispositivi IoT troviamo alcune board abbastanza famose: Intel Edison, Seeduino (compatibile con Arduino), Microchip e BeagleBone. Con il

Device SDK di AWS IoT, rigorosamente scritto usando C e Node.js, interfacciarsi alla piattaforma IoT di Amazon è un processo rapidissimo. (si veda: <http://aws.amazon.com/it/iot/getting-started/> ).

Ecco alcuni scenari che meglio danno l'idea della portata di queste soluzioni, rapportate al listino prezzi di Amazon AWS IoT su base Mensile:

# Messaggi pubblicati	Cadenza di pubblicazione	# Dispositivi ricevanti	Traffico Mensile	Importo Mensile
1	ogni ora	5	3600 messaggi	0,022 USD
2	ogni minuto	nessuno	86.400 messaggi	0,432 USD
100	ogni minuto	1	4.300.000 messaggi	21,50 USD

(Fonte: <http://aws.amazon.com/it/iot/pricing/>)

## 3. IoT-433Mhz

---

Internet of Things for 433mhz-based devices

### 3.1 - Introduzione

Il nome che ho dato a questo secondo modulo è tanto semplice quanto descrittivo. Internet of Things for 433mhz-based devices.

Questo progetto è nato con l'idea di offrire una piattaforma, dashboard o se vogliamo "centralina" di controllo per dispositivi radio che comunicano sulla frequenza radio 433mhz.

Si compone sia di una parte hardware che di una parte software.

Sono molte le centraline d'allarme disponibili sul mercato e sempre più le necessità degli utenti spingono le case costruttive a realizzarne di nuovi. Tra la tipologia più comune spicca sicuramente il classico kit d'allarme con sensori radio a 433mhz. Quel che forse avremo notato è che qualunque produttore, quando si lavora con sistemi d'antifurto di questo tipo, spesso ripropone la solita minestra, pur cambiando qualche dettaglio, la centralina rimane pressochè invariata nelle funzionalità, qualunque sia la marca.

Un malloppo di sensori radio e una centralina che il più delle volte copre quasi interamente il costo dell'intero kit è la formula riproposta da ogni venditore, senza alcun valore aggiunto.

Gli svantaggi più comuni in un sistema d'allarme radio come questi sono:

1. Impossibilità di personalizzare la centralina come vorremmo.
2. Rapporto qualità prezzo decisamente elevato.
3. Notifiche non più al passo coi tempi, alcune centraline propongono esclusivamente la chiamata ad un numero prefissato.
4. Scarsa modularità.
5. Impossibilità di evolvere nel tempo e aggiornarsi.

Data la breve premessa è facile intuire quali siano gli scopi di questo nuovo progetto open source:

1. Avere un sistema di controllo flessibile e personalizzabile. Adatto non solo all'ambito domestico, ma a qualunque tipo di scenario in cui si voglia aver anche la praticità di estendere ulteriormente il sistema e di integrarlo a sistemi già esistenti.

2. Facilità d'esecuzione: Avere un sistema che sia facile da installare e semplice da usare.
3. Fornire sistemi di notifica immediati.
4. Abbattere i costi.
5. Aggiornarsi ed evolvere nel tempo.
6. Supportare una più ampia gamma di dispositivi radio, astraendosi dal mero impiego a sistema d'allarme.

Ed è sul punto 6 che spenderò qualche parola in più.

Il sistema è stato pensato per non essere relegato all'utilizzo esclusivo come sistema d'allarme o nello specifico centralina d'allarme classica, come si potrebbe di fatti erroneamente pensare.

Ma è pensato per interagire con un vasto ecosistema di dispositivi radio funzionanti alla frequenza di 433mhz. In particolare si pensi alle prese telecomandate, i sensori di gas/fumo, i sismografi radio o i telecomandi radio.

Da questa marcia in più ne deriva il nome, *iot-433mhz*, Internet of Things for 433mhz-based devices.

Le possibilità d'estensione sono molteplici. Avendo già visto il funzionamento di MQTT e dei moduli ESP8266, un ponte 2.4Ghz - 433Mhz è facilmente immaginabile come intermediario o ripetitore di segnale.

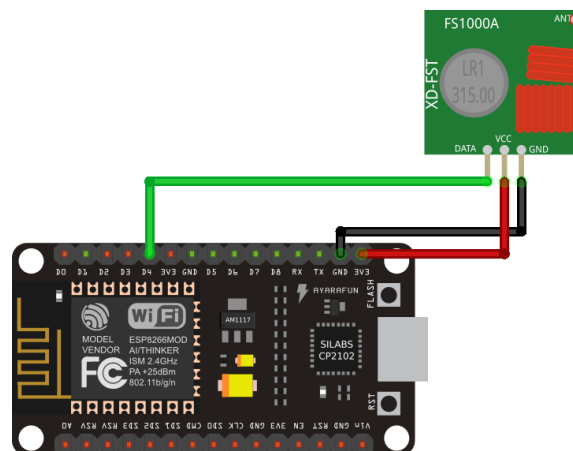


Figura 44 - NodeMCU devkit e trasmettitore 433mhz

In uno scenario del genere, la macchina che fa girare il broker MQTT e *iot-433mhz* fungerebbe al contempo anche da client MQTT. Il middleware *iot-433mhz* in particolare assumerebbe il ruolo di client MQTT. Nulla vieterebbe infatti di far girare sul raspberry pi, sia il broker che più istanze di client mqtt.

Avendo dunque un'interfaccia al broker, è facile per *iot-433mhz* sottoscrivere ai topic adibiti ai segnali radio e regolarsi di conseguenza. Allo stesso tempo per un ripetitore di segnale basato su ESP8266 è facile

ricevere segnali radio e rigirarli al broker sugli appositi topic, una volta connessi alla rete.

### *3.1.1 - A chi è destinato il sistema?*

L'elenco potrebbe essere infinito, ecco finalità e casi d'uso più comuni:

1. Appassionato di domotica.
2. Sicurezza domestica.
3. Automazione domestica.
4. Riutilizzo di sensori PIR e sensori magnetici per porte e finestre.
5. Scopi didattici.

### *3.1.2 - Caratteristiche principali*

- Multi-piattaforma (Windows, Mac OS X, Linux).
- Autenticazione di base.
- API intuitive e webHooks per costruire la propria interfaccia.
- Template basato su schede in material design.
- Interfaccia utente aggiornata in tempo reale.
- Rilevamento di codici radio per frequenza 433mhz.
- Possibilità di creare nuove schede e associarle alle stanze della casa.
- Controllo di sensori PIR, sensori magnetici e prese telecomandate.
- Bot telegram per le notifiche d'allarme.
- Completamente open source & open hardware.

Prima di passare in rassegna i component Hardware necessari, mostriamo la topologia di rete.

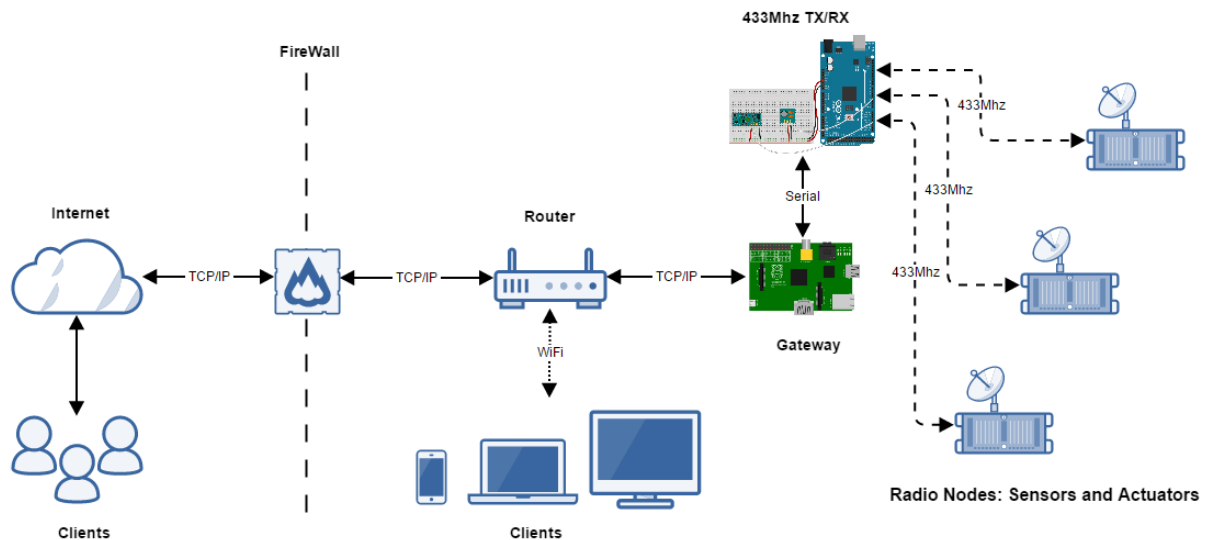


Figura 45 - Topologia iot-433Mhz

Vari client possono interagire con il gateway radio attraverso la rete locale o dall'esterno, con apposite regole di routing configurate nel dispositivo di rete.

I protocolli e le comunicazioni avvengono su diversi fronti. Il gateway comunica coi client su stack TCP/IP (principalmente protocolli HTTP e Web Socket). A sua volta interagisce tramite connessione seriale con il microcontrollore che gestisce i moduli d'invio e ricezione dei segnali radio. E proprio con i segnali radio si dialoga con sensori e attuatori.

I Clients non devono preoccuparsi di come la comunicazione radio venga gestita, il gateway astrae questo concetto di trasmissione dell'informazione grezza ed espone delle interfacce di alto livello. Per definizione stessa di gateway è possibile estendere l'eterogeneità della rete, integrando nel gateway quanto visto nel capitolo precedente, ovvero un Broker MQTT. L'idea è che un unico dispositivo possa fungere da gateway sia per le trasmissioni radio a 433 Mhz sia per le trasmissioni WiFi a 2.4 Ghz.

## 3.2 - Hardware Necessario

Ecco un elenco dell'hardware strettamente necessario per l'esecuzione del progetto:

- Un computer casalingo (dove girerà il nostro software iot-433mhz), il consiglio è di utilizzare un Raspberry Pi.
- Un ricevitore e un trasmettitore radio per RF 433mhz.
- Cavetti jumper per effettuare i collegamenti.
- Un arduino Nano.

L'hardware necessario per iniziare fin da subito è conosciutissimo e facilmente reperibile in rete per pochi euro. Passiamo ad analizzare ogni singolo componente hardware necessario.

### 3.2.1 - Raspberry Pi

A seconda di dove vogliamo far girare il sistema possiamo pensare di prendere o meno un Raspberry Pi, oppure usare un computer domestico.

Il Raspberry Pi non è altro che un computer dalle dimensioni di una carta di credito, realizzato in UK e disponibile in differenti versioni. Nel progetto fungerà da Gateway radio. Quella mostrata in figura è la versione 2.

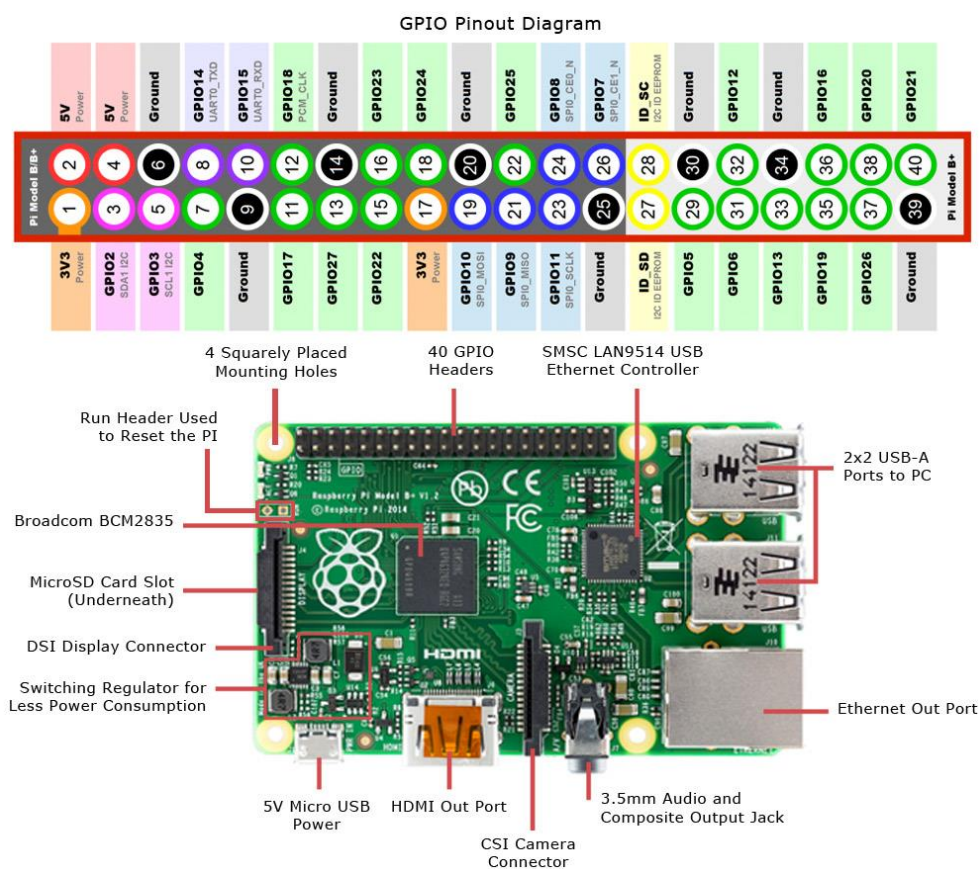


Figura 46 - Raspberry Pi 2

Le caratteristiche della versione 2 sono qui riportate:

- Broadcom 900 MHz BCM2836 ARMv7 Quad Core Processor SoC
- Broadcom VideoCore IV GPU
- 1 GB RAM
- 4 x USB2.0 con output fino a 1.2A
- Expanded 40-pin GPIO Header

- Video/Audio Out attraverso connettore 4-pole 3.5mm, HDMI, o Raw LCD (DSI)
- Storage: microSD
- 10/100 Ethernet (RJ45)
- Periferiche basso livello:
  - 27 x GPIO
  - UART
  - I2C bus
  - SPI bus
  - +3.3V
  - +5V
  - Ground
- Power Requirements: 5V @ 600 mA via MicroUSB o GPIO Header
- Supporta Windows 10, Debian GNU/Linux, Fedora, Arch Linux, RISC OS.

Raspberry Pi può far girare distribuzioni Windows/Linux compatibili con architettura ARM. Per questo progetto viene usata la distribuzione Raspbian. Da poco è stata annunciato il nuovo modello Raspberry Pi 3, che integrerà Wi-Fi e Bluetooth, con un occhio di riguardo all'Internet of Things.

### 3.2.2 - Arduino

Arduino è una piattaforma che non ha bisogno di presentazioni: Arduino Uno, Arduino Mega, Arduino Nano o persino NodeMCU Devkit basato su esp8266. Sono tutte board e vanno più che bene per poterci collegare i moduli radio.



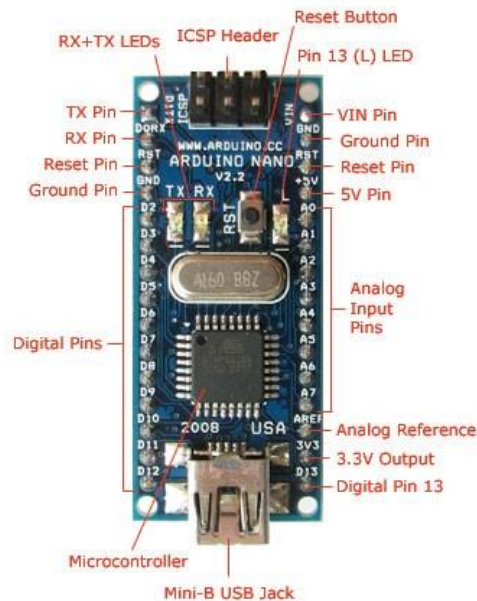


Figura 47 - Arduino Nano

Sul microcontrollore scelto andremo dunque a caricare tramite l'IDE di Arduino lo sketch che ci servirà per la ricezione e l'invio dei codici radio da e verso i dispositivi radio stessi.

Per quest'ultimo in realtà ci sono due soluzioni disponibili:

1. La prima prevede l'utilizzo di un arduino nano connesso al ricevitore e al trasmettitore radio.
2. La seconda prevede una connessione diretta del ricevitore e trasmettitore radio ai pin GPIO del Raspberry Pi. Dunque senza utilizzare un arduino Nano.

Se stiamo utilizzando come sistema di controllo un computer casalingo, adottiamo la soluzione numero 1, che consiste nell'utilizzo di un arduino nano collegato direttamente alla porta seriale USB.

Se volessimo invece collegare il ricevitore e il trasmettitore radio, direttamente ai GPIO del Raspberry Pi, adatteremmo la soluzione 2. E' anche possibile con RPi, delegare il compito di ricezione e trasmissione all'arduino, connesso tramite USB allo stesso RPi. Si tenga inoltre presente che con quest'ultima soluzione, essendo ricevitore e trasmettitore radio a 5V, mentre i GPIO a 3.3V, la necessità di un *Logic Level Converter* è d'obbligo.

Per i collegamenti si può usare una breadboard, o per comodità usare dei jumper femmina ed evitare al contempo "saldature" al volo.

Il consiglio, è quello di adottare in ogni caso un Arduino (Uno, Nano etc.) dal momento che la ricezione e la trasmissione radio richiedono anche



Passiamo alla scelta dei moduli radio.  
 Sul mercato son disponibili una vasta gamma di moduli radio.  
 I più diffusi sono sicuramente i **XY-MK-5V** mostrati in foto:

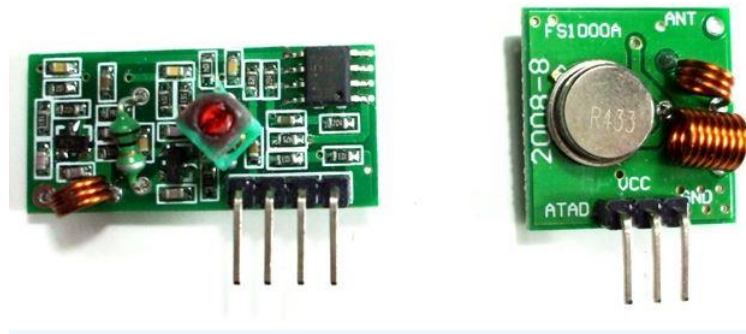


Figura 50 - Moduli Radio XY-MK-5V

Reperibili per pochi euro in rete non sono esattamente il massimo. La ricezione è debole perchè trattasi di un semplice risuonatore analogico con un amplificatore operazionale. Di seguito il datasheet:

- Trasmettitore:
  - Working voltage: 3V - 12V for max. power use 12V
  - Working current: max Less than 40mA max, and min 9mA
  - Resonance mode: (SAW)
  - Modulation mode: ASK
  - Working frequency: Eve 315MHz Or 433MHz
  - Transmission power: 25mW (315MHz at 12V)
  - Frequency error: + 150kHz (max)
  - Velocity: less than 10Kbps
- Ricevitore:
  - Working voltage: 5.0VDC +0.5V
  - Working current:  $\leq 5.5\text{mA}$  max
  - Working method: OOK/ASK
  - Working frequency: 315MHz-433.92MHz
  - Bandwidth: 2MHz
  - Sensitivity: excel  $-100\text{dBm}$  ( $50\Omega$ )
  - Transmitting velocity:  $< 9.6\text{Kbps}$  (at 315MHz and  $-95\text{dBm}$ )

E' possibile acquistare anche modelli nettamente superiori in quanto a prestazioni. Una soluzione senz'altro più valida è l'**RXB6 superheterodyne** (chiamato anche 3400RF):

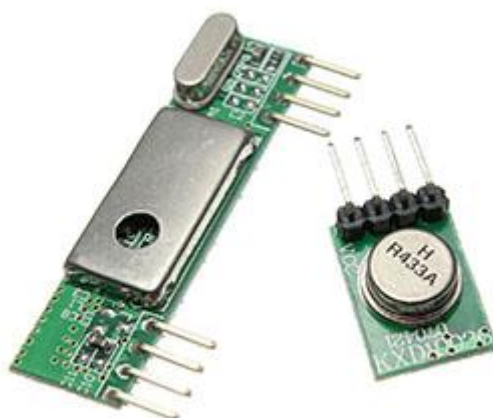


Figura 51 - Moduli radio RXB6 superheterodyne

Leggermente più costoso ma una spanna sopra al più economico XY-MK-5V. Garantisce una portata maggiore e un miglior filtro per il rumore radio. Il range si attesta attorno ai 30 metri.

#### Electrical Characteristics :

Parameter	Specification			Unit	Condition
	Min	Typ	Max		
Frequency Range	300	315/433.92	450	MHz	
Receiver Sensitivity	-114		-110	dBm	
Data Rate	0.058		10	KBaud	
Supply Voltage, VDD	3.0		5.5	V	DC
Current	5.7		7.3	mA	
Operating Temperature	-40		+85	°C	

#### Size :

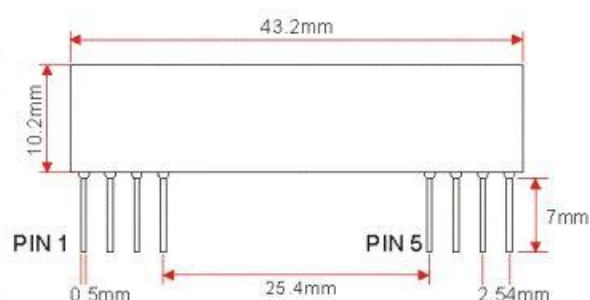


Figura 52 - RXB6 Datasheet

Utilizza la tecnologia supereterodina che converte il segnale che si desidera ricevere in un nuovo segnale con frequenza fissa, stessa modulazione ma indipendente dal segnale d'ingresso.

### 3.2.4 - Antenna ideale

L'uso di un'antenna esterna è opzionale, ma raccomandato se si vuole estendere il raggio di funzionamento. Ancora per far fronte ai picchi di tensione durante la ricezione o l'invio di un segnale radio, è consigliabile immettere un condensatore in parallelo con le linee GND e 5V (la capacità del condensatore è trascurabile, 100uF sono sufficienti).

- Qual è la lunghezza di un'antenna ideale?

Le antenne, sono obbligatorie per garantire la diffusione del segnale in maniera efficace. Costruire un'antenna per circuiti elettronici è possibile anche con materiali di scarto, come antenne si possono utilizzare fili di rame, riciclati da cavi telefonici o di rete (smaltati o meno, poco cambia, se smaltati avranno soltanto una maggiore resistenza all'ossidazione, che non compromette né incide sul segnale). La domanda che persiste è, quanto devono essere lunghe le antenne?

Esiste un modo per calcolare la lunghezza ottimale:

Quando si realizza un'antenna per un dispositivo RF il massimo segnale lo si ottiene con un'antenna lunga esattamente quanto la lunghezza d'onda ( $\lambda$ ) da trasmettere/ricevere. La formula per calcolare tale lunghezza d'onda è la seguente:

$$\lambda = v / f$$

Dove  $v$  è la velocità di propagazione nel mezzo di trasmissione, che nell'etere è pari alla velocità della luce, 300.000 Km/s, espressa in metri, ed  $f$  è la lunghezza d'onda, espressa nell'unità di misura Hertz.

Quindi per 433 MHz si ha:

$$300.000.000 / 433.000.000 = 0,6928 \text{ m} \rightarrow 69,28 \text{ cm}$$

Spesso, non è praticabile fare l'antenna di pari lunghezza di  $\lambda$  e si ricorre alle frazioni. Si usano  $\lambda$  mezzi,  $\lambda$  quarti e così via. Un quarto di  $\lambda$  nel nostro caso è:

$$69,28 / 4 = 17,32 \text{ cm}$$

Una frazione della lunghezza dell'antenna ideale per la nostra frequenza di trasmissione è dunque 17,32 cm (1/4 di  $\lambda$ ). Tuttavia, molti circuiti a 433Mhz hanno già un'antenna integrata, sottoforma di spirale di rame saldata sulla board stessa. Una spirale che compie circa 3 giri per un diametro di 5mm. Dunque  $5 \text{ mm} \times 3 \times \pi = 47,25 \text{ mm}$  ovvero circa 5 cm. Da questo si evince che l'antenna ideale esterna da saldare dev'essere di 12 cm circa, perchè già è presente un'antenna da 5cm saldata sulla PCB.

Ancora, per migliorare la portata è possibile pensare di effettuare un'induttanza alla base dell'antenna, come mostrato in figura.



Figura 53 - Antenna per 433Mhz con Induttanza

### 3.2.5 - Altri componenti

Altri componenti hardware richiesti sono i sensori e gli attuatori con cui il sistema è in grado di interagire:

- Prese wifi.
- Sensore PIR di movimento.
- Sensore magnetico per porte e finestre.

I 'wifi outlet switches', o semplicemente prese telecomandate che si trovano in commercio, sono generalmente di tre tipi, differenziati in base al modo in cui si codifica il codice identificativo per la comunicazione radio.

Il funzionamento è basilare, un relay viene commutato quando opportunamente indicato da un ricevitore radio.

I sensori piroelettrici, detti anche sensori ad infrarosso passivo, misurano i raggi infrarossi irradiati dagli oggetti nel campo di visuale.

Tutti gli oggetti con temperatura superiore allo zero assoluto emettono energia sotto forma di radiazioni luminose. La maggior parte delle volte queste radiazioni sono invisibili all'occhio umano, poiché a frequenza inferiore a quella della luce dello spettro visibile, ma possono essere rilevate tramite specifici dispositivi elettronici progettati a tal scopo.

Il termine passivo si riferisce al fatto che i PIR non emettono energia in nessuna forma ma lavorano esclusivamente rilevando l'energia sprigionata dagli oggetti.

Sono sensori molto usati come rilevatori di movimento.

I sensori di prossimità magnetici funzionano rilevando il campo magnetico generato da un magnete permanente montato appositamente sull'oggetto da rilevare. Questi sensori si basano sul principio dei contatti Reed o sull'effetto Hall.

I modelli realizzati con contatti Reed hanno una velocità di commutazione bassa (fino 50 Hz), ma i modelli realizzati con sensori ad effetto Hall possono commutare a velocità elevate (anche migliaia di Hz).

Una particolarità di questi sensori è che le portate nominali dipendono dalla potenza del campo generato dal magnete, più che dalle



caratteristiche del sensore, e pertanto, usando un grosso magnete, possono essere elevate (fino a 100 mm). Di contro l'oggetto da rilevare deve essere preparato montando l'opportuno magnete permanente. Per ovvi motivi questi sensori non possono essere utilizzati in prossimità di grosse fonti elettromagnetiche.



*Figura 54 - Sensore PIR 433Mhz*



*Figura 55 - Sensore magnetico 433Mhz*



*Figura 56 - Prese telecomandate a 433Mhz*

## 3.3 - Software

### 3.3.1 – Flusso d'esecuzione

Il flusso d'esecuzione rappresenta le fasi che vanno dall'inizializzazione del software iot-433mhz, fino all'avvio e dunque alla possibilità di utilizzarlo:

- 1) La prima fase consiste nell'elaborazione del Logo che verrà stampato nella console, sottoforma di ascii-art.
- 2) Si istanziano i Database, i WebHooks e le utility interne.
- 3) Successivamente si inizializza la piattaforma d'esecuzione. Il software identifica il sistema operativo su cui viene eseguito e seguendo le direttive del file di configurazione richiede o meno all'utente, la porta seriale a cui è connesso il microcontrollore con i moduli radio (se siamo su RPi e se specificato in *config.json* si può scegliere di inizializzare i moduli radio direttamente usando i GPIO, senza microcontrollore).
- 4) Registrata la scelta dell'utente, si istanzia l'oggetto adibito alla comunicazione radio.
- 5) La fase successiva inizializza il Database che conterrà le schede dispositivo. Se il database non esiste, verrà popolato da alcune schede di default. Altrimenti si caricheranno i contenuti già presenti nel DB.
- 6) Dall'oggetto precedentemente istanziato si inizializza la comunicazione seriale.
- 7) Si avvia il Web Server, per esporre API, Dashboard di controllo e Web Socket.
- 8) Si registrano gli event handler in attesa di input dall'utente o ricezione dei codici radio.
- 9) Il sistema arrivato a quest'ultima fase è operativo e pronto per essere utilizzato.



### 3.3.2 - Avviamo *iot-433mhz*



Figura 57 - Logo *iot-433mhz*

Se abbiamo reperito l'hardware necessario possiamo passare all'installazione del software.

Il primo step consiste nel caricare lo sketch di arduino. Esso interagisce con i moduli radio e funge da ponte fra i dispositivi radio e il software di controllo che andremo ad installare nel Raspberry Pi o nel nostro computer di casa.

Lo sketch da caricare sull'arduino si trova qua:

[github.com/roccomuso/iot-433mhz/tree/master/hardware-layer/single-arduino-tx-rx](https://github.com/roccomuso/iot-433mhz/tree/master/hardware-layer/single-arduino-tx-rx)

Le uniche dipendenze necessarie sono le librerie [RC-Switch](#) e [ArduinoJSON](#).

Il secondo step consiste nell'installazione di **iot-433mhz** all'interno del nostro sistema (RPI o computer casalingo che sia). Nel caso di windows o mac è sufficiente usare l'installer presente sul sito ufficiale di [Node.js](#). Se siamo su Raspberry Pi procederemo in questo modo per l'installazione di Node ed npm:

1. Aggiungiamo la repository di adafruit al nostro file `/etc/apt/sources.list`, con questo comando: `curl -sLS https://apt.adafruit.com/add | sudo bash`
2. Installiamo l'ultima versione di node.js usando apt-get: `sudo apt-get install node`
3. Se tutto è andato per il meglio possiamo controllare la versione di node e npm con i comandi: `npm -v` e `node -v`

Ora non ci resta che scaricare e installare globalmente nel sistema il modulo *iot-433mhz*, direttamente dalla repository di npm dove è hostato:

- Se siamo su Windows, assicuriamoci prima di installare le dipendenze necessarie, [Python 2.7](#) e [Microsoft Visual Studio Express 2013](#), poi potremo lanciare da terminale il comando:

```
npm install iot-433mhz -g
```

- Se siamo su Mac non dovremo aver bisogno di alcuna dipendenza esterna (Node.js e npm a parte si intende). Si può lanciare il comando così come visto sopra:

```
npm install iot-433mhz -g
```

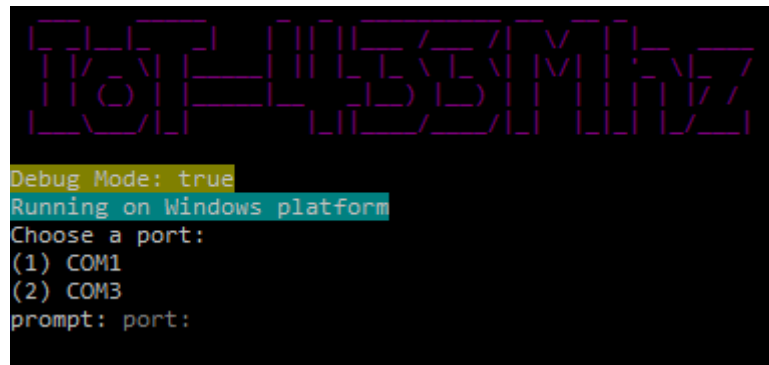
- Su Raspberry Pi qualora effettuassimo un collegamento diretto con i GPIO e il ricevitore/trasmittitore radio dovremmo installare [wiringPi](#). Altrimenti possiamo proseguire con l'installazione, lanciando nello specifico questo comando:

```
sudo npm install iot-433mhz -g --unsafe-perm
```

Se l'installazione viene completata senza errori possiamo procedere, collegando l'arduino tramite USB e lanciando da console il comando:

```
iot-433mhz
```

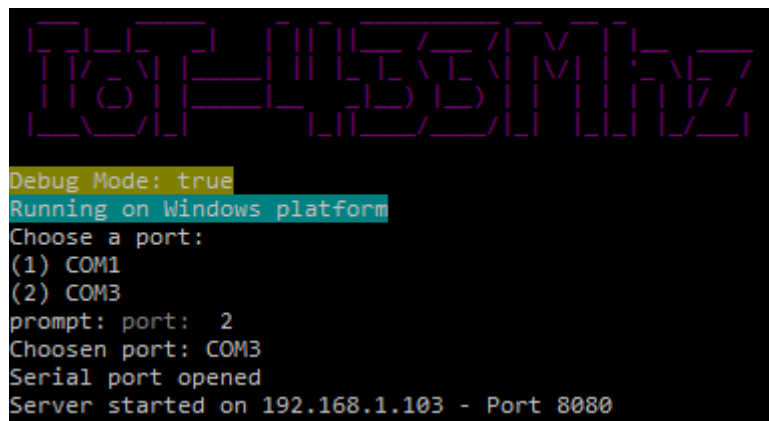
Quando viene lanciato, il normale flusso d'esecuzione ci chiederà di scegliere la porta seriale a cui abbiām collegato l'arduino.



```
IoT-ESM
Debug Mode: true
Running on Windows platform
Choose a port:
(1) COM1
(2) COM3
prompt: port:
```

Figura 58 - Scelta porta seriale

Una volta selezionata, l'applicativo sarà in grado di comunicare con l'arduino, inviare e ricevere codici radio, avviare il web server per l'interfaccia grafica ed esporre le API realizzate.



```
IoT-ESM
Debug Mode: true
Running on Windows platform
Choose a port:
(1) COM1
(2) COM3
prompt: port: 2
Choosen port: COM3
Serial port opened
Server started on 192.168.1.103 - Port 8080
```

Figura 59 - Avvio del server

### 3.3.3 - Interfaccia grafica

L'applicativo ad ogni avvio apre un server locale per esporre una pratica interfaccia realizzata seguendo le linee guida del Material Design.

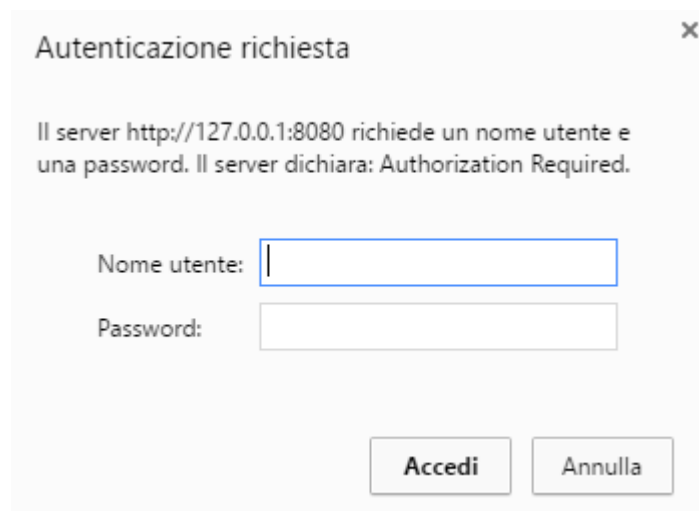
Le linee guida hanno il compito di rendere l'esperienza utente il più uniforme possibile fra i diversi tipi di app. Il Material design fa uso di griglie, animazioni responsive, transizioni, spaziature, luci e ombre. Che meglio danno il senso di ciò che è possibile cliccare e cosa non lo è.

Per costituire l'interfaccia web si è fatto uso di apposite librerie open source come *bootstrap-material-design* (paragonabile al più diffuso ed eccellente *Materialize*).

L'interfaccia è responsive, utilizzabile anche da smartphone. È raggiungibile a questo indirizzo:

`http://[indirizzo_server]:8080`

dove *indirizzo\_server* non è altro che l'ip della macchina locale su cui abbiamo avviato l'istanza corrente di **iot-433mhz**. Ci viene comunicato dopo la scelta della porta seriale. 8080 è invece la porta di default che è possibile cambiare dal file di configurazione, *config.json*. Navigando da browser all'indirizzo dell'interfaccia grafica ci verrà chiesto di autenticarci:



Autenticazione richiesta

Il server `http://127.0.0.1:8080` richiede un nome utente e una password. Il server dichiara: Authorization Required.

Nome utente:

Password:

Le credenziali d'accesso di default sono:

- Username: root
- Password: root

C'è da sottolineare che nella release corrente (1.0.20) non è possibile fornire parametri aggiuntivi da console quando lanciamo **iot-433mhz**. I parametri di configurazione, come la porta su cui avviare il server o le credenziali d'accesso sono definite nel file *config.json* presente all'interno della cartella d'installazione. La posizione di quest'ultima varia a seconda dell'OS. Nel momento in cui scriviamo è già in lavorazione la nuova release 1.1.0. Che introdurrà nuove funzionalità e in particolare la possibilità di modificare i valori di default direttamente da linea di comando. E' possibile tener traccia dello sviluppo direttamente dalla pagina "Milestones" della repo ufficiale su GitHub.

Se dovessimo vedere una nuova versione, tenendo bene a mente che l'aggiornamento cancella e ricostituisce da zero il DataBase e l'intero contenuto, potremmo aggiornare il pacchetto semplicemente lanciando il comando:

```
npm update -g
```

NB. Su Raspberry Pi per aggiornare si consiglia di lanciare nuovamente il comando:

```
sudo npm install iot-433mhz -g --unsafe-perm
```

Dopo l'autenticazione l'interfaccia che ci si presenta è questa mostrata in figura:

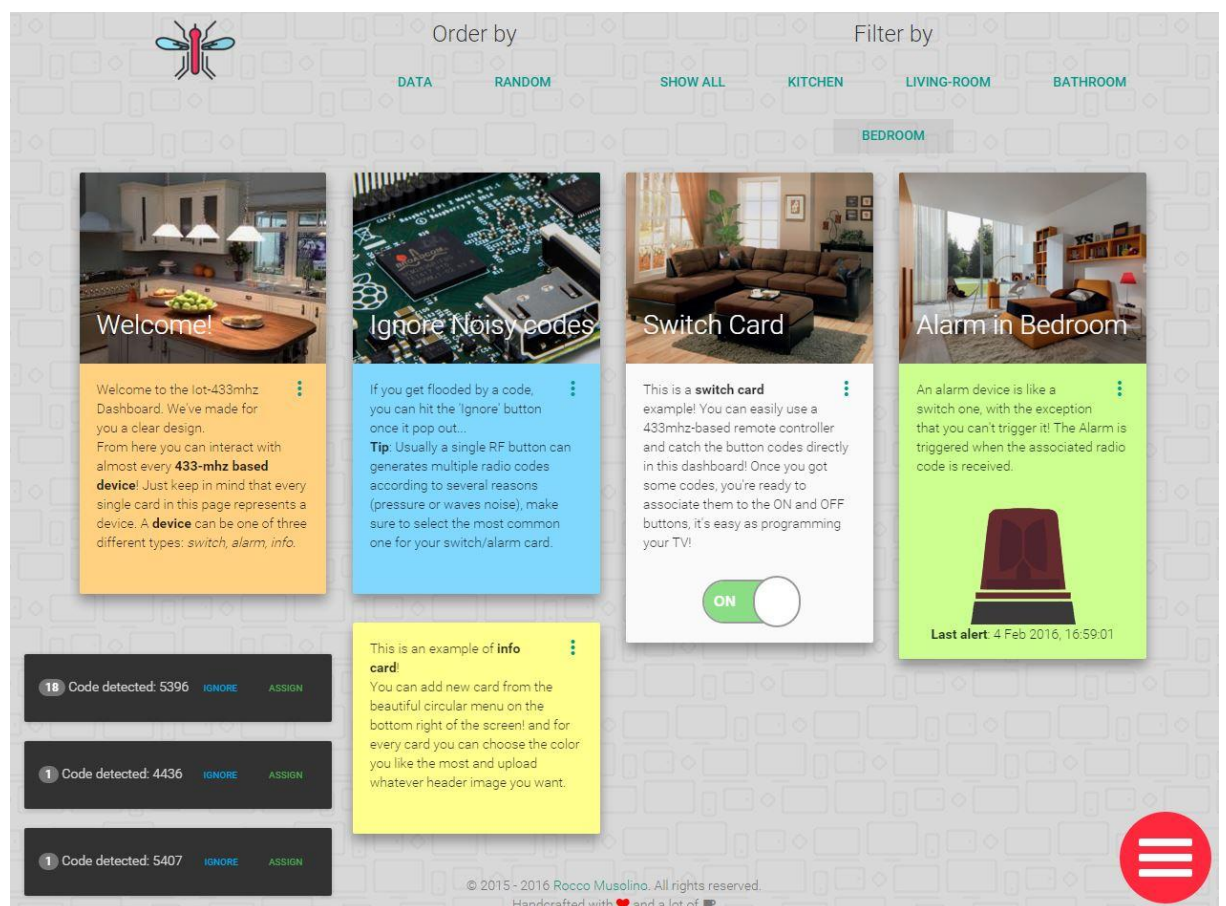


Figura 60 - Interfaccia grafica Dashboard

Con delle schede di default che spiegano brevemente come utilizzare il sistema e la tipologia di dispositivi che possiamo associare ad una scheda.

### 3.4 - Utilizzo

Dalla nostra interfaccia grafica possiamo individuare alcuni elementi chiave.

La dashboard con le schede, essa contiene le **schede dispositivo**. Una scheda può essere di 3 tipi: Allarme, Interruttore, Informativa. Ogni dispositivo in iot-433mhz è gestito come una scheda. La scheda informativa è a solo scopo informativo, le altre invece hanno un ruolo fondamentale per l'interazione con i dispositivi a 433mhz. Un altro elemento della UI è la sezione di ordinamento e filtraggio delle schede.

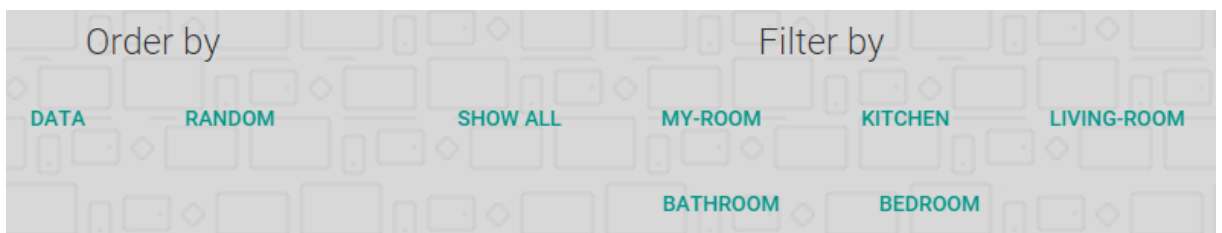


Figura 61 - Filtro e ordinamento schede

Tralasciando la scheda informativa passiamo in rassegna le altre 2 tipologie attualmente disponibili nella release corrente:

La scheda d'allarme è una scheda dispositivo che permette di associare un codice radio ad un'istanza di notifica. La notifica d'allarme avviene sia sull'interfaccia grafica attraverso l'apposito segnale acustico, sia attraverso bot Telegram (se configurato), e attraverso web Hooks, se impostati. Il segnale acustico può essere disattivato dal menù a tendina presente sulla scheda corrispondente. Anche le notifiche Telegram, webHooks ed Email (non presenti nella release attuali) possono essere disattivate disarmando l'allarme.



Figura 62 - Scheda d'allarme

Queste le voci del menu a tendina:

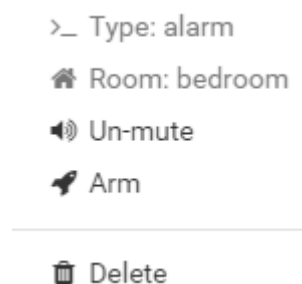
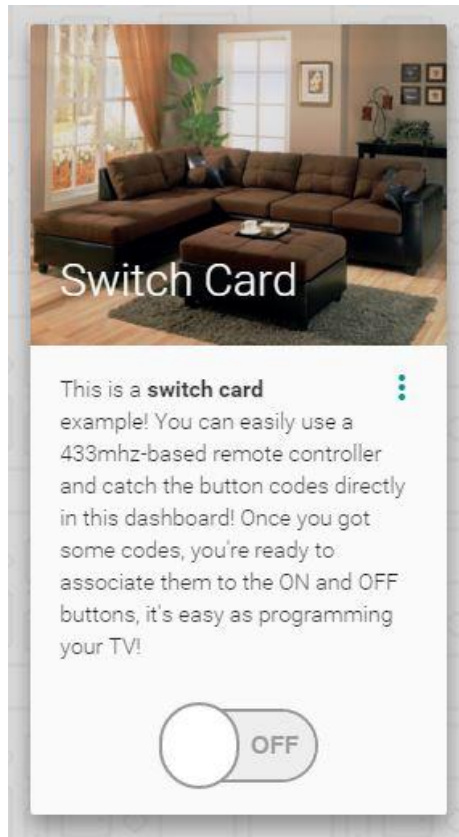


Figura 63 - Menù contestuale

Ogni voce rispettivamente ci indica, la tipologia di scheda, la stanza a cui è stata associata. La possibilità di attivare o disattivare la notifica acustica, la possibilità di armare e disarmare l'allarme, così da evitare notifiche di altro tipo. Infine la voce per l'eliminazione della scheda.

L'altra tipologia di scheda è quella interruttore. E' una scheda dispositivo che permette di associare 2 codici radio per commutare un

interruttore/attuatore/relay. Uno dei codici da associare corrisponde al codice d'accensione ON, e un altro al codice di spegnimento OFF.



*Figura 64 - Scheda interruttore*

Ovunque sia aperta l'interfaccia web, smartphone, tablet o PC, e qualunque numero di client la mantengano aperta, riceveranno gli aggiornamenti in tempo reale. La comunicazione tra la UI e il server avviene infatti grazie ai WebSocket e ad un misto di chiamate API.

Sempre nell'interfaccia web, troviamo in basso a destra troviamo il menù circolare dalla quale è possibile tornare alla Home, visionare i codici ignorati, aggiungere un nuovo dispositivo, visualizzare le informazioni relative al progetto e all'autore ed infine accedere alle Impostazioni di notifica.



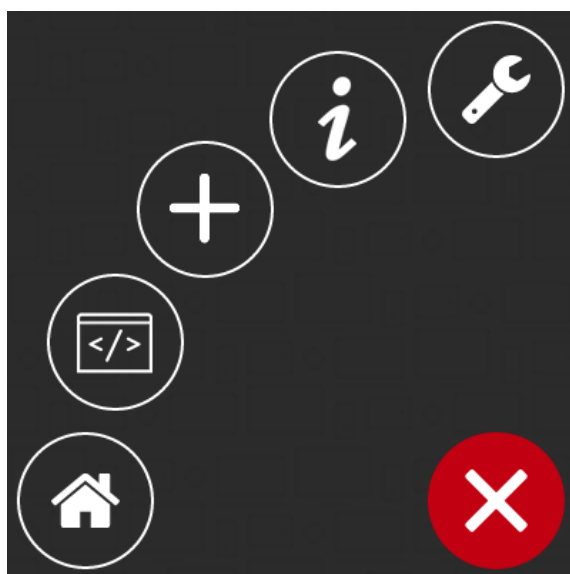


Figura 65 - Dashboard Menù

### 3.4.1 - Come rileviamo i codici?

Una volta che il sistema è avviato, rilevare i codici è un processo abbastanza semplice. Basta infatti avvicinare il dispositivo radio al ricevitore (neanche troppo perchè si presume che il raggio sia ottimizzato per coprire l'intera abitazione, in tal senso delle prove di portata sarebbero uno step fondamentale), e aspettare che il codice radio venga identificato dal sistema.

Quando un codice radio viene identificato uno snackbar appare in basso a sinistra sull'interfaccia grafica, e allo stesso tempo un log d'acquisizione viene stampato sulla console d'avvio del server node.

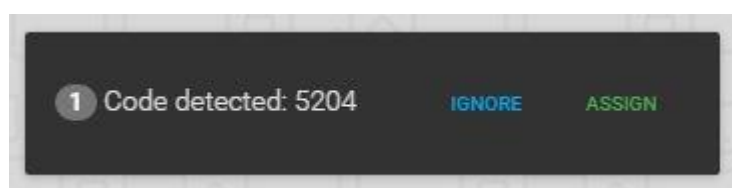


Figura 66 - Snackbar di rilevamento

In ogni snackbar oltre al pulsante “assegna” è presente un tasto “ignora”, questo perchè spesso lavorando con i codici radio è facile ricevere codici che in realtà non corrispondono a nulla perchè puro “rumore” di fondo. Ignorandoli saremo sicuri che non appariranno nuovamente se ricevuti.

A questo punto l'idea di base è abbastanza chiara: **Per le prese telecomandate**, con il telecomando, schiacciando i pulsanti per

accendere le prese, si potranno registrare i codici radio sul sistema, i singoli codici potranno così essere replicati dal sistema e il telecomando diverrà superfluo. Si potranno pilotare le prese direttamente dalla pratica interfaccia web, sia da PC che da smartphone o tablet.

**Per i sensori pir o magnetici** il discorso è simile, una volta fatto innescare il sensore, si cattura il codice radio univoco che, associato ad una scheda dispositivo allarme sul sistema, notificherà tramite telegram, email, webHook o allarme sonoro sull'interfaccia web, ogniqualvolta il sensore rileverà qualcosa.

Altre feature secondarie includono ad esempio la possibilità di aprire l'interfaccia su più dispositivi, mobile o desktop che siano e avere in tempo reale l'aggiornamento di ogni attuatore su tutte le interfacce. Ancora da mobile in particolare è possibile mescolare le schede, semplicemente scuotendo il dispositivo.

### *3.4.2 - Aggiungere una nuova scheda dispositivo*

Quando il sistema rileva dei codici radio, ognuno di questi codici radio può essere utilizzato per creare una scheda dispositivo interruttore o allarme.

Si attraverso il tasto d'assegnazione 'ASSIGN', sia dall'icona + raggiungibile dal Menù circolare in basso a destra nell'interfaccia grafica. In entrambi i casi ciò che ci appare è il form per l'aggiunta di una nuova scheda dispositivo, mostrato nella figura che segue.

New Card ×

Fill the form below to add a new Device Card

Headline

Card Headline

Short name

Max 15 characters

Description

Room name

Room name

Card color

☒

☐

☐

☐

☐

☐

☐

Select Type

Switch

RF Codes

-- choose --

-- choose --

Image

Put an Image for your card header...

CANCEL

SUBMIT

Figura 67 - Form aggiunta nuova scheda

### 3.4.3 - Bot Telegram

L'attivazione delle notifiche Telegram avviene dalla pagina Settings Raggiungibile dal Menù e rappresentata in figura 68.

## Settings:



▶ START  TELEGRAM NOTIFICATION

▶ START  EMAIL NOTIFICATION

### How to configure Telegram Notification

1. Get a UID:  [Generate](#)
2. Go to [web.telegram.com](https://web.telegram.com) and start talking with the bot **@my\_iot\_bot**
3. You need to associate your *IoT System UID* to your Telegram user; type and send this to the bot:  
**`/register 86c7e7eb1588f3dcf5bd312ce0bfe4c64890c1dd`**
4. You're good to go! Every time an armed alarm is triggered you'll get a message from the bot.
5. **Remember:** whoever got your UID and do the above procedure will get notifications message from your IoT System too.
6. If you run the IoT System on a different Server, you'll get a different UID. Whenever you get a new UID, you'll have to do again the whole telegram registration procedure.

✓ CLOSE

Figura 68 - Impostazioni Bot Telegram

Il codice univoco di 40 caratteri (IoT ID) viene generato in modo casuale dal sistema *iot-433mhz*, esso è necessario per completare il processo di registrazione. Il processo di registrazione vede coinvolto un altro sistema, di back-end. Quest'ultimo gira su un server remoto in attesa di chiamate da parte dei webHooks di telegram o notifiche d'allarme dalle varie istanze di *iot-433mhz*.

La registrazione presso il bot telegram con il codice univoco (IoT ID) fa sì che il nostro codice univoco venga associato al nostro Chat ID. Il chat ID identifica il canale di comunicazione privato instaurato con il bot.

Quando un'allarme armata viene fatta scattare, il sistema invia una notifica d'allarme al back-end, allegando l'ID univoco del sistema IOT, quello usato appunto durante la registrazione. In questo modo il back-end può associare quella notifica d'allarme al Chat ID

precedentemente associato. Dunque spedire il messaggio di notifica verso il canale di comunicazione che identifica l'utente in questione.

Naturalmente ad un unico IoT ID possono essere associati più Chat ID, in questo modo più utenti possono ricevere le notifiche d'allarme dalla stessa istanza di `iot-433mhz`.

In figura un esempio di registrazione presso il bot telegram con successiva ricezione di notifica d'allarme.

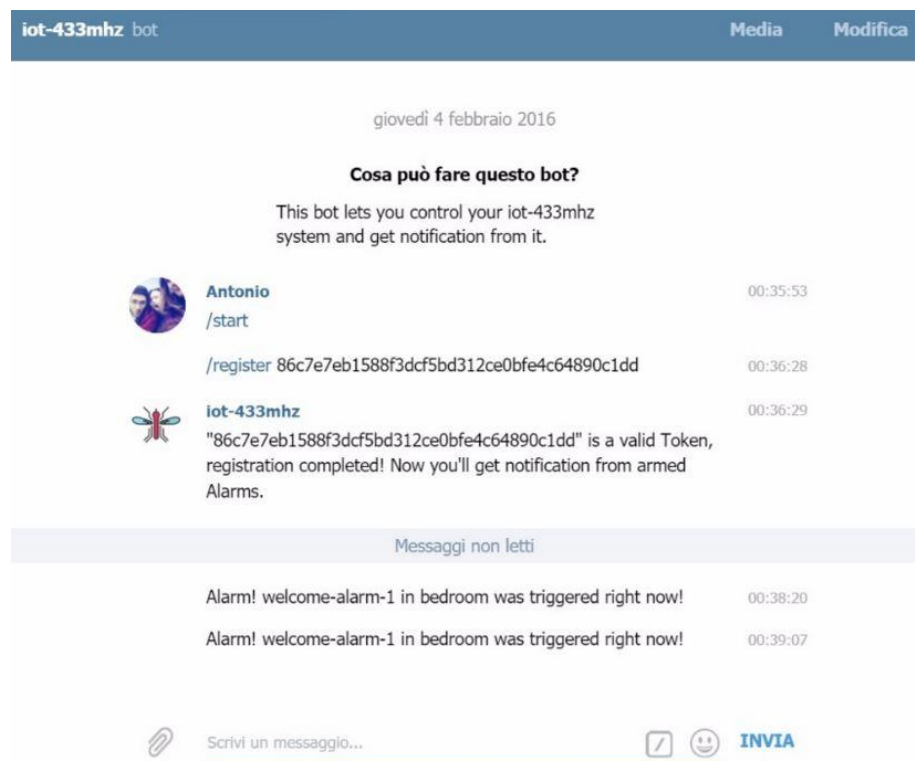


Figura 69 - Esempio registrazione e notifiche telegram

## 3.5 - Scelte Implementative

### 3.5.1 - Connessione dei moduli radio direttamente ai GPIO del RPi

Per realizzare il core su RPi con Node.js e avere una connessione diretta dei moduli radio usiamo: [github.com/eroak/rpi-433](https://github.com/eroak/rpi-433)

Quest'ultimo oltre all'invio garantisce il listening dei segnali radio, poichè spawna un nuovo processo di RCSniffer che all'arrivo di dati li trasmette al wrapper superiore.

L'applicativo sopra è un'interfaccia per Node.js che presuppone l'utilizzo di un Raspberry Pi sfruttando `rcswitch-pi` (un porting dell'originale

rcswitch realizzato per Arduino), e come dipendenza quest'ultimo necessita di "wiringPi" per comunicare con i GPIO del Raspberry.

### 3.5.2 - rc-switch

rc-switch è una libreria open source per la decodifica di protocolli radio. Tra le prime sviluppate per Arduino, supporta i protocolli più comuni usati dai dispositivi radio low cost. Da questa libreria ne derivano molte altre. La libreria è limitata a protocolli fissi e non è in grado di decodificare protocolli rolling codes.

E' reperibile a questo link: [github.com/sui77/rc-switch](https://github.com/sui77/rc-switch)

### 3.5.3 - Autenticazione

Il modulo server utilizzato nel progetto è un derivato del modulo http di base di Node.js, chiamato Express.js. Quest'ultimo è giunto alla versione 4.0, ma già dalle prime versioni continua a mantenere una filosofia di gestione delle richieste basata su Middleware: software intermedi che elaborano progressivamente le richieste provenienti dai clients.

Un middleware che torna molto utile lavorando con Express è chiamato "basic-auth". Questo middleware gestisce un'autenticazione di tipo basic, fra client e server.

L'autenticazione HTTP di base è una tecnica che non richiede pagine di login, cookie o identificatori di sessione; Ma utilizza un campo standard nell'header HTTP, ovviando anche al problema dell'handshaking.

Di contro, questo tipo d'autenticazione non fornisce confidenzialità per le credenziali trasmesse. Esse viaggiano codificate in base64 senza alcun tipo di crittografia, motivo per cui questo tipo d'autenticazione è preferita in abbinamento con HTTPS.

Inoltre siccome il campo d'autenticazione dev'essere inviato nell'header di ogni richiesta HTTP, il web browser deve conservare le credenziali per un periodo ragionevole per evitare che il server richieda continuamente le credenziali d'accesso all'utente.

In `iot-433mhz` le credenziali d'accesso utente sono uniche. Salvate e modificabili dal file `config.json`. Di default root, root.

### 3.5.4 - Notifiche Telegram

La domanda che verrebbe da porsi è: perchè telegram per la gestione delle notifiche?

Telegram nel corso degli anni è stato fortemente rivalutato. Non solo per essere un client di messaggistica gratuito e molto valido, ma per l'attenzione riposta nei confronti degli sviluppatori. E' un progetto open source e agli sviluppatori viene offerta molta libertà nell'interazione con telegram stesso.

E' il primo attualmente a offrire la possibilità di creare bot automatici. E proprio i bot rappresentano il punto di giuntura fra l'applicazione iot-433mhz e gli utenti telegram, ansiosi di ricevere le notifiche d'allarme.

In generale per interagire con un bot su telegram ci sono vari modi. Per l'utente è sufficiente utilizzare una delle interfacce messe a disposizione da Telegram: interfaccia web, smartphone o desktop. Per il server, che gestisce il comportamento del bot, è un tantino differente l'interazione, è possibile ad esempio inviare direttamente una richiesta alle API tramite GET/POST (dunque usare l'entrypoint che permette la scrittura di un messaggio dal bot, verso l'utente selezionato). Noi interagiamo con l'utente su telegram esclusivamente tramite il bot. I bot su telegram si creano incamerando una piacevole discussione con il "botFather", ovvero il padre di tutti i bot che contattato su telegram e seguendo i comandi da lui proposti, ci guida nella creazione di un bot. La cosa si fa più delicata quando dobbiamo non inviare dati all'utente tramite il bot, ma ricevere i dati che l'utente invia al bot.

Telegram gestisce questa interazione implementando due modelli. Il primo chiamato **Long Polling**, il secondo **WebHooks**.

Nel long polling, noi, in quanto server, inviamo a telegram una richiesta prolungata che espressamente viene mantenuta aperta e mai mandata in timeout. In questo modo, quando son disponibili dei messaggi perchè inviati al bot da un qualche utente, vengono rigirati sulla richiesta aperta fra il nostro server e i server di telegram.

Questo ha delle ovvie implicazioni a livello di performance. Bisogna mantenere una connessione perennemente aperta, con tutti gli svantaggi del caso.

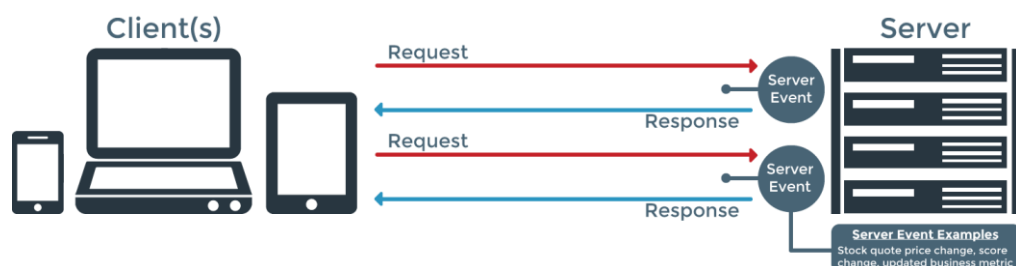


Figura 70 - Long Polling

L'altro modello che sviluppa telegram sono i webhooks. Con i webhooks si effettua una singola richiesta antecedente ad ogni comunicazione, in cui si indica un URL di recapito qualora un messaggio sia stato ricevuto dal bot (dunque dai server telegram). Quando questo avviene, il server telegram che ha memorizzato il nostro url di ritorno, effettua una chiamata HTTPS POST verso quel link con i dati che identificano mittente e messaggio inviato al bot.

Quel che il nostro server deve fare è rimanere in ascolto di una chiamata su una data url di entrata (entrypoint). L'entrypoint suggerito dalla stessa telegram deve rimanere possibilmente segreto. Questo garantirebbe altrimenti a malintenzionati la possibilità di effettuare richieste facendo credere al nostro server che un utente abbia inviato un dato messaggio al nostro bot.

Motivo per cui l'entrypoint suggerito da telegram è lo stesso token d'autorizzazione che è una sorta di password che identifica il nostro bot in modo univoco ed è generato esclusivamente dal botFather.

Come detto in precedenza la richiesta effettuata dai server telegram verso il nostro webHook è di tipo POST, e richiede espressamente un'URL HTTPS.

Per avere un sito cifrato con SSL è necessario che venga fornito un certificato valido.

Fortunatamente telegram permette e accetta anche certificati self-signed. Se così non fosse si dovrebbe acquistare un certificato firmato da un'autorità di certificazione per poter dunque hostare il back-end di `iot-433mhz`.

Un modo per generare rapidamente un certificato self-signed è usare openssl:

```
openssl req -newkey rsa:2048 -sha256 -nodes -keyout YOURPRIVATE.key -x509 -days 365 -out YOURPUBLIC.pem -subj "/C=YOURCOUNTRY/ST=YOURSTATE/L=YOURCITY/O=YOURORGANISATION/CN=YOURDOMAIN.EXAMPLE"
```

Assicurandoci che i rispettivi campi vengano sostituiti dagli appositi valori.

Nei test che ho effettuato, inizialmente senza ricorrere ad un certificato self-signed ho optato per il deploy dell'applicativo Node.js di backend su Heroku. Un noto servizio di PaaS che fornisce durante il deploy un sottodominio con un certificato valido. Lo svantaggio di una soluzione di questo tipo è che avendo a che fare con soluzioni gratuite, si va incontro ad una serie di limitazioni. La prima è il non poter aprire per un singolo applicativo (dyno nel gergo di Heroku) più di una porta. La seconda è quella di aver a che fare con dyno, entità dinamiche istanziate solo nel momento del bisogno e dunque con un file system



NON permanente. Di conseguenza lo stesso DB implementato usando nedb sarebbe stato azzerato ogniqualvolta l'istanza fosse stata avviata.

### 3.5.5 - Dipendenze e moduli creati appositamente

#### - Dipendenze lato Server:

- 1) Async: E' un modulo che permette l'esecuzione di flussi di codice asincroni in modo sincrono, attraverso vari pattern.
- 2) Express.js: Framework per la gestione semplificata di server con Node.js.
- 3) Basic-auth: Middleware per express che fornisce l'autenticazione HTTP di base.
- 4) Body-parser: Middleware per express per l'elaborazione del contenuto passato dai form html.
- 5) Chalk: Colorazione delle scritte su console.
- 6) Socket.io: Gestione dei WebSocket su Node.js.
- 7) Console-mirroring: E' un modulo scritto appositamente per questo progetto. E' utile per effettuare debugging e avere la console e i relativi log "specchiati" su una finestra del browser. L'aggiornamento avviene in tempo reale grazie ai WebSocket.
- 8) Nedb: Database persistente su file JSON.
- 9) Prompt: Acquisizione di input da tastiera. Usato per permettere la scelta di una porta seriale all'utente.
- 10) Request: Modulo per effettuare richieste HTTP di ogni genere.
- 11) Serialport: Gestione della comunicazione con le porte seriali USB.
- 12) Validator: Modulo per la validazione dei parametri passati attraverso form HTML.
- 13) Figlet: Creazione di una scritta in ASCII ART partendo da un testo.
- 14) Multer: Middleware per la gestione di contenuti multipart/form-data, primariamente utilizzati per l'upload di files.
- 15) Node-webhooks: Modulo creato appositamente per questo progetto. Fornisce la possibilità di realizzare dei webhooks. Sfrutta nedb per salvare i nomi dei webhooks e le URL associate, mentre request per l'invio della richiesta HTTP POST.

#### - Dipendenze lato Client:

- 1) jQuery: Fornisce una pratica interazione con il DOM dell'interfaccia web.
- 2) Socket.io.js: Gestione della connessione WebSocket.
- 3) circleMenu: Realizza il menù circolare presente nella Dashboard.
- 4) MixItUp: Espone un sistema di manipolazione grafica per lo spostamento e il riordinamento dei contenuti configurati. In iot-433mhz viene utilizzata per filtrare e riordinare le schede dispositivo.

- 5) Bootstrap: Framework grafico per la prototipazione rapida di interfacce web.
- 6) Bootstrap-Material: Framework che sullo stile di Bootstrap offre componenti grafici che seguono le linee guida del Material Design di Google.
- 7) Notie.js: Gestione dei pop up di notifica.
- 8) Ion.Sound: Script per l'esecuzione di suoni sulla pagina web.
- 9) SnackBar.js: Plugin per bootstrap-material, permette l'apparizione dinamica di SnackBar in basso a sinistra nell'interfaccia grafica. Viene usato come mezzo di notifica per l'arrivo di segnali radio.
- 10) Mustache.js: Sistema di templating, per la separazione delle viste dai modelli e i contenuti generati dinamicamente.
- 11) EventEmitter: Script per la gestione interna degli eventi.
- 12) Shake.js: Libreria per rilevare lo scuotimento del dispositivo (funziona da smartphone e tablet).
- 13) Hover.css: Libreria grafica per l'applicazione di semplici animazioni ai componenti del DOM.

### 3.5.6 - 433Mhz o Wifi 2.4Ghz ?

A seconda del progetto che si sta realizzando, è importante considerare la tecnologia di trasmissione più adatta. E' un progetto data-intensive? Si cerca un'ampia portata o sicurezza nelle comunicazioni?

Lo scopo del progetto iot-433mhz è quello di realizzare una dashboard di controllo per sensori e attuatori radio, un'alternativa per le centraline d'allarme, di conseguenza la scelta non può che esser ricaduta sulla frequenza radio usata dai produttori di tali dispositivi, 433 Mhz.

Elenchiamo i vantaggi di entrambe le frequenze domestiche più diffuse:

I vantaggi della comunicazione Wifi 2.4Ghz:

- Velocità di trasferimento alte
- Dispositivi di ricezione/invio economici
- Frequenza standard e altamente in uso fra gli utenti
- Portata limitata (possibilità di usare dei repeater di segnale o antenne migliori)
- Cifratura delle informazioni semplice da attuare
- Mesh network facili da realizzare
- Buona gestione delle interferenze

I vantaggi della comunicazione 433Mhz includono:

- Buon range e copertura

- Il segnale penetra bene nei muri
- Dispositivi di ricezione/invio economici
- Basso consumo energetico
- Alta diffusione nei sistemi d'allarme domestici

### 3.5.7 - Svantaggi della comunicazione 433mhz

Gli svantaggi sono molteplici.

La comunicazione radio in generale è facilmente inibita da jammer e dispositivi in grado di disturbare le frequenze. L'interferenza è alta, le velocità di trasferimento basse.

Questo tipo di frequenze non presentano meccanismi di codifica e cifratura robusti. Inoltre la maggior parte dei dispositivi basati su trasmissione a 433mhz non garantiscono alcun tipo di feedback di avvenuta ricezione/invio.

Oltre a non avere un feedback sull'avvenuto invio e conseguente ricezione di un comando da parte ad esempio di una presa telecomandata, non si ha neanche un feedback di tipo "keepalive", ovvero se un dispositivo va giù, il sistema non ha modo di accorgersene. L'implementazione di un sistema di keepalive, che a intervalli regolari notifichi al sistema se il dispositivo è ancora attivo, sarebbe abbastanza complessa e onerosa da realizzare usando tecnologie radio di questo tipo.

## 3.6 - API & WebHooks

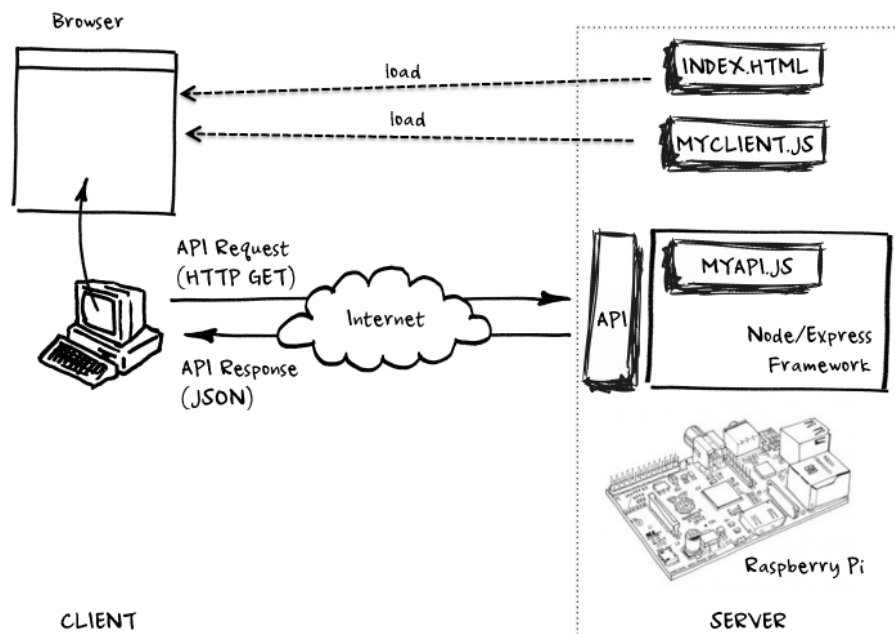


Figura 71 – Scenario API

Le molteplici API offerte dall'applicativo permettono di espandere il sistema, offrire un'esperienza utente quanto più completa possibile, così da avere la possibilità di realizzare applicativi paralleli per soddisfare altre esigenze specifiche.

- GET /api/settings/get – Restituisce le impostazioni correnti. Utile per capire ad esempio se le notifiche sono attive.
- GET /api/system/get/uid – Restituisce l'UID univoco del Sistema IoT.
- GET /api/system/new/uid – Genera un nuovo UID di Sistema.
- GET /api/system/telegram/enable – Abilita le notifiche Telegram.
- GET /api/system/telegram/disable – Disabilita le notifiche Telegram.
- GET /api/system/email/enable – Abilita le notifiche tramite Email.
- GET /api/system/email/disable – Disabilita le notifiche tramite Email.
- GET /api/code/send/[RFcode] – Invia il codice radio specificato. L'oggetto json restituito dalla chiamata è del tipo: {"status": "ok"} or {"status": "error", "error": "error description.."}.
- GET /api/codes/ignored – Restituisce una lista dei codici ignorati presenti nel DB.
- GET /api/codes/all – Restituisce tutti i codici registrati e salvati nel DB.

- GET /api/codes/available – Restituisce tutti i codici disponibili salvati nel DB. I codici disponibili possono essere assegnati a una nuova scheda dispositivo.
- GET /api/cards/all – Restituisce tutte le schede salvate nel DB.
- GET /api/cards/get/[shortname] – Restituisce una singola scheda col nome passato, se esiste.
- POST /api/cards/new – Permette l’inserimento di una nuova scheda dispositivo, di seguito i form-data richiesti:
  - headline – breve titolo.
  - shortname – shortname, minuscolo e senza spazi.
  - card\_body – breve descrizione, html permesso.
  - room – stanza di appartenenza, scritto in minuscolo e senza spazi.
  - type – uno dei 3 tipi supportati: switch/alarm/info
  - device – se la tipologia scelta è ‘switch’ deve avere 2 parametri, on\_code e off\_code. Se di tipo ‘alarm’ soltanto il parametron trigger\_code.

Altri parametri opzionali sono: card\_img, background\_color (dev’essere un colore espresso in esadecimale). Un esempio di risposta Json è: {"done": true, "newCard": ...} dove newCard è la scheda appena inserita.

- GET /api/cards/delete/[shortname] – Cancella la scheda con il nome specificato, restituisce in caso di successo {"status": "ok", cards\_deleted: 1} oppure {"status": "error", "error": "error description.."}.
- GET /api/alarm/[shortname]/arm – Arma la scheda dispositivo di tipo ‘alarm’ specificata. Se armata verranno chiamati i webHooks e le notifiche per questa scheda inviate.
- GET /api/alarm/[shortname]/disarm – Disarma la scheda dispositivo specificata.
- GET /api/switch/[shortname]/on – Accende l’interruttore specificato.
- GET /api/switch/[shortname]/off – Spegne l’interruttore specificato.
- GET /api/switch/[shortname]/toggle – Commuta l’interruttore specificato.

Ogni chiamata API richiede un campo nell’header della richiesta per l’autenticazione di base. Così composto:

Authorization: Basic cm9vdDpyb290

Dove l’ultima stringa è una codifica base64 della concatenazione di username e password in questo modo:

*base64( root:root )*

I Web Hooks, espongono anch'essi delle API, utili per la registrazione ad eventi del sistema iot-433mhz. Quando uno di questi eventi si verifica viene effettuata una richiesta HTTP POST con un payload ben preciso all'URL configurata. Per questo progetto è stato realizzato un modulo separato (node-webhooks) adibito alla sola gestione dei webHooks e al richiamo delle URL di callback, aggiunto come dipendenza del progetto principale.

Gli eventi che è possibile catturare con i webHooks sono attualmente 5:

- Alarm triggered event
- New card event
- Card deleted event
- New code detected event
- Switch toggle event

Di seguito le API per interagire con i Web Hooks:

- GET /api/webhook/get – Restituisce l'intero DB dei WebHook.
- GET /api/webhook/get/[WebHookShortname] – Restituisce il webHook specificato.
- GET /api/webhook/delete/[WebHookShortname] – Rimuove tutte le URL di callback per il webHook specificato.
- POST /api/webhook/delete/[WebHookShortname] – Rimuove una singola URL per il webHook specificato se passata come parametron nel body JSON: { "url": "http://..." }
- POST /api/webhook/trigger/[WebHookShortname] – Simula l'evento del webHook. Richiede un body JSON che verrà rigirato all'URL specificata per quel determinato webHook.
- POST /api/webhook/add/[WebHookShortname] – Aggiungere una nuova URL di callback per il webHook selezionato. I parametri obbligatori per questa richiesta sono:
  - webHookShortname: Fornito in URL, dev'essere uno di questi eventi: alarmTriggered, newCard, cardDeleted, newCode, switchToggle.
  - url: l'URL a cui verrà effettuata la richiesta HTTP POST quando un evento si verifica.

## 3.7 - Alternative a IoT-433mhz

- **MySensors**



Figura 72 - MySensors Logo

MySensors è un progetto open source, guidato da una community di appassionati di automazione domestica. Fornisce facili istruzioni per l'assemblaggio di hardware e codici sorgenti pronti all'uso, che aprono a un mondo fatto di sensori e attuatori a basso costo. Il progetto similmente a iot-433mhz permette di controllare attuatori e ricevere allarmi dai sensori preposti. Inoltre da anche la possibilità di collezionare dati dai sensori e visualizzarli in grafici. I Sensori di MySensors creano automaticamente un network di comunicazione strettamente interconnesso, in quella che viene chiamata, topologia di rete a mesh. Questo è possibile usando dei nodi che fungano da ripetitori.

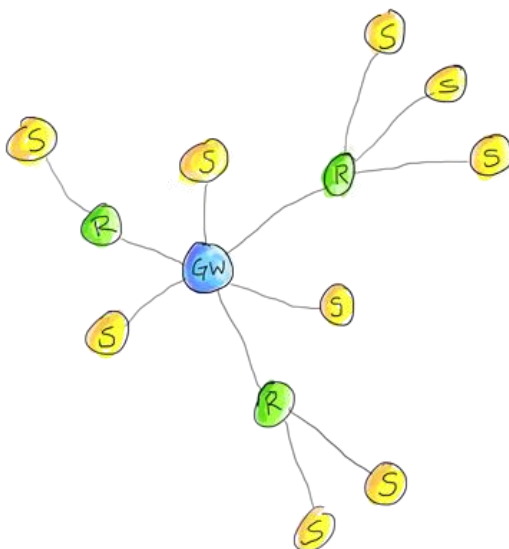
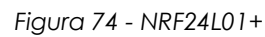


Figura 73 - MySensors Mesh Network

I nodi etichettati con S rappresentano i sensori, che inviano i propri dati ai gateway, etichettati con GW. I nodi R rappresentano i dispositivi ripetitori. Ovvero dei dispositivi che possono comportarsi come sensori, inviando dati ai gateway, ma allo stesso tempo fungere da repeater, realizzando un effetto "eco" per i segnali ricevuti. I Gateway solitamente sono connessi a dei controller, il cui scopo è quello di inviare parametri di configurazione ai sensori (orario corrente, id etc.), tener traccia dei dati più recenti ricevuti e soprattutto fornire

Il sensore tipo proposto da MySensor fa uso di Arduino e del modulo transceiver NRF24L01+. Un modulo a basso costo che permette la comunicazione radio sulla frequenza 2.4Ghz a lunghe distanze (20-60m) e con un consumo di batteria minimo. E' anche possibile introdurre nel sistema, gateway differenti che facciano uso di altre tecnologie di comunicazione. Di recente è stato scritto un gateway per integrare nel sistema MySensors, dispositivi wifi con esp8266 e comunicazione MQTT.



A collection of various electronic components and modules, including a large LCD screen, a microcontroller board, a motor, a sensor, and various smaller modules, arranged on a white background.

94



## - Pimatic



Figura 76 - Pimatic Logo

Pimatic ([github.com/pimatic](https://github.com/pimatic)) è un framework open source per l'automazione domestica che gira su Node.js. Fornisce una piattaforma estensibile per il controllo domestico e l'automazione di task. E' pensato per girare su Raspberry Pi.

Definisce un'interfaccia uniforme per diversi attuatori e sensori, così che possano essere controllati e monitorati da un'unica interfaccia universale.

E' un framework estensibile, molti plugin sono stati già rilasciati per estenderne le funzionalità. Similmente a [iot-433mhz](#) il core è sviluppato in node.js e il server implementato da Express.js.

I task d'automazione possono essere definiti da regole nella forma "if this then that" e rappresentano un punto di forza del sistema.

Pimatic può essere usato con diversi dispositivi a 433Mhz, usando un gateway radio tramite arduino e moduli radio (come per [iot-433mhz](#)) oppure sfruttando il gateway wifi, dunque con dispositivi esp8266 e sketch ad hoc che permettono il supporto nativo per una moltitudine di sensori.

La libreria radio utilizzata sul microcontrollore arduino non decodifica on board direttamente i dati, ma vengono girati al modulo pimatic-Homedeino e decodificati dalla piattaforma ([github.com/pimatic/rfcontroljs](https://github.com/pimatic/rfcontroljs)).

## - ThingSpeak



Figura 77 - ThingSpeak Logo

Piuttosto che costruirsi il proprio cloud da zero, è possibile optare per l'utilizzo di piattaforme alternative già ampiamente collaudate.

ThingSpeak è una piattaforma applicativa per l'Internet of Things. Permette di costruire un'applicazione che giri attorno ai dati collezionati dai sensori. Tra le feature di ThingSpeak in particolare abbiamo la possibilità di collezionare dati in real-time, processarli, visualizzarli su varie interfacce, impostare dei task automatici, nonché la possibilità di installare plug-in che ne estendano le funzionalità.

Al cuore di ThingSpeak abbiamo il canale ThingSpeak. Un canale dove si inviano i dati affinché vengano salvati. Ogni canale include 8 campi per ogni tipologia di dato. Una volta creato un canale ci si può pubblicare all'interno dati e fare in modo che vengano estrapolati e utilizzati da altre app.

Esistono librerie ufficiali per piattaforme come Arduino, Raspberry Pi, Particle Photon, Electric Imp.

## 3.8 - Miglioramenti per le prossime release

Le prossime release del sistema prevedono una serie di miglioramenti, atti soprattutto al supporto di un numero di protocolli e dispositivi radio maggiori, all'introduzione di nuovi meccanismi di notifica e alla possibilità di immettere parametri di configurazione direttamente da linea di comando all'avvio del server di controllo.

Per ulteriori informazioni sull'avanzamento dei lavori e le migliorie che verranno apportare si può far riferimento alle rispettive pagine sulla repository ufficiale, costantemente aggiornate:

- Milestones: [github.com/roccomuso/iot-433mhz/milestones](https://github.com/roccomuso/iot-433mhz/milestones)

- TODO e Issues: [github.com/roccomuso/iot-433mhz/issues](https://github.com/roccomuso/iot-433mhz/issues)

Di seguito i punti completati e mancanti al rilascio della prossima release v1.1:

## Core

---

- ☒ Use SerialPort to communicate with Arduino.
- ☒ Make adapter to expose one way to operate with HW layer.
- ☒ Multiplatform test (Windows, RPi, Mac, Linux).
- ☐ Test 433utils and WiringPi on RPi.
- ☒ Socket.io integration.
- ☒ Fixed version needed for the **package.json** dependencies.
- ☒ Use 'nedb' for DB handling.
- ☒ DB Compacting.
- ☒ Real-time console-mirroring, (/console.html).
- ☐ Console parameters parsing (yargs.js).

## Notifications

---

- ☐ Email notification.
- ☒ WebHooks.
- ☒ Telegram Bot.

## UI

---

- ☒ Add dependencies, Bootstrap, Bootstrap-material, jQuery, hover.css, notie.js etc..
- ☒ Add Logo.
- ☒ Add footer credits.
- ☒ Add to Home Screen button for Mobile devices.
- ☒ Web socket integration.
- ☒ Handle incoming RF codes with snackbars.
- ☒ Add Cards system based on 4 columns.
- ☒ Ignore/Assign code button on snackbar.
- ☒ Notification sound on initCards or codes received.
- ☒ Switch sound.
- ☒ Animated favicon (favico.js)

## Cards

- ✓ Implement **Info** Card type.
- ✓ Implement **Switch** Card type.
- ✓ Implement **Alarm** Card type [#8](#) .
- ✓ Add color background to the newCard form.
- ✓ Cards filtering and re-ordering with MixItUp.
- ✓ Handle Delete and Mute button in Dropdown menu.

## Menu

- ✓ Add Circular Menu.
- ✓ Add Home Button.
- ✓ Add Ignored Codes Page.
- ✓ Add new Card Page.
- ✓ Add About Page.
- ✓ Add Settings Page. ([#4](#))

## 4. Sismometro SeismoCloud

---

### 4.1 - Costruire un sismometro digitale basato su esp8266

In questa sezione mostro la realizzazione di un Sismometro connesso alla rete di sismometri digitali SeismoCloud, come esempio d'integrazione di un sensore NodeMCU all'interno dei sistemi domotici.

Quella che segue è una guida per la realizzazione del prototipo, tenendo bene in mente che è possibile optare per soluzioni differenti, in termini di forma, board di controllo o accelerometro.

#### 4.1.1 - Hardware necessario

Nello specifico I componenti hardware da me adottati sono elencati di seguito:

- accelerometro I2C (mpu-6050)
- modulo esp8266 (nodemcu devkit)
- alimentatore 5V 1A
- box contenitore di plastica

Il box di plastica è ricavato direttamente da una presa telecomandata. Svuotata delle componenti interne per utilizzare il case esterno di plastica.



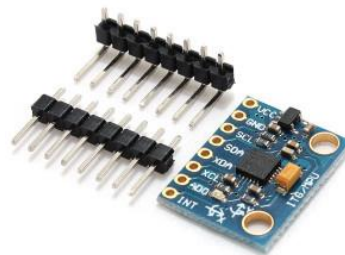
Figura 78 - Presa telecomandata

A seconda del microcontrollore e dei componenti elettronici che si vogliono usare è necessario considerare il voltaggio e l'ampereaggio necessario.

Nel caso di un accelerometro come l'MPU6050, l'alimentazione della board stessa è sufficiente. Per la board invece è possibile fornire un'alimentazione di 5v-20v direttamente tramite il pin VIN e GND, il regolatore di tensione integrato provvederà a regolare la tensione sui 3.3V. La peculiarità del devkit Nodemcu è che essendo basato su chip ESP8266 possiede una connessione wifi integrata. In alternativa al nodemcu si può utilizzare una board basata su arduino e il modulo ESP8266-01 separatamente connesso per fornire la connessione wifi.

Ci sono più tipi di breakout board che montano sopra l'MPU-6050.

Alcune di queste board non hanno regolatori di tensione interni, dunque vanno alimentati a 3.3v (ad esempio il SEN-11028), altri possiedono un regolatore interno e quindi possono essere alimentati a 5V (GY-52, GY-521, Drotek MPU-6050 etc.). Quello usato in questo progetto è il GY-521, mostrato in figura:



*Figura 79 - Accelerometro MPU-6050, GY-521*

Successivamente reperiamo un alimentatore che entri comodamente nello chassis designato.

E' importante che i pezzi scelti, che andranno a comporre il progetto finale, siano abbastanza piccoli da entrare comodamente all'interno dello chassis. Per procurarmi un alimentatore abbastanza piccolo da entrare nel box di plastica è stato necessario smontare un alimentatore usb da viaggio, piccolo compatto, uno di quelli realizzati da Samsung, Apple o simili.

Per aprirne uno è sufficiente fare leva con un taglierino per rimuovere la parte superiore, dopodiché il trasformatore interno può essere sfilato con una pinza dal suo alloggiamento. Nel caso fosse incollato alla base, si deve necessariamente con un pò di forza, rompere il case di plastica attorno, difficilmente potrà essere riutilizzato successivamente.



*Figura 80 - Trasformatore 5V riciclato*

I componenti hardware scelti sono molto diffusi e di facile reperibilità sul mercato, tant'è che il costo totale del progetto è estremamente basso.

Seguono adesso i procedimenti necessari a collegare insieme le parti elettroniche e inscatolare il tutto.

#### *4.1.2 - Assemblaggio*

Estratto il trasformatore, possiamo procedere alla collocazione all'interno della scatola di plastica ricavata dalla presa telecomandata già smontata e vista sopra. Trovata la posizione più idonea possiamo saldare i contatti 220V uscenti dal box di plastica con i contatti d'entrata del trasformatore USB. Non è un problema mantenere il connettore USB all'interno del box saldato, anche se inutilizzato, dissaldarlo infatti è un'inutile perdita di tempo, lo spazio ricavato è pressocchè nullo, dunque manteniamolo e procediamo a saldare i contatti:



*Figura 81 - Saldatura del trasformatore*

Completate le saldature dei cavi che porteranno la 220v dai morsetti della presa di casa al trasformatore a 5V, assicuriamoci che il tutto sia ben isolato, onde evitare contatti indesiderati e procediamo a testare quanto appena realizzato collegandolo alla rete elettrica di casa con un qualunque dispositivo usb collegato all'apposito connettore o usando un tester.

E' sempre consigliabile la supervisione di un esperto se non si è familiari con le alte tensioni (220v).

Il prossimo passo è quello di identificare le piste dei poli positivo e negativo uscenti dal trasformatore, così da poterli usare per alimentare la board NodeMCU.

Con un tester possiamo facilmente trovare i punti in cui andremo a catturar l'alimentazione che ci servirà per la nostra board.

Questo passo è critico se non si è sicuri di ciò che si sta facendo. E' raccomandato l'uso di un paio di guanti isolanti.

Senza richiudere il box di plastica colleghiamo alla presa elettrica il tutto. E rifacendoci alle figure sotto mostrare e seguendo le piste sul retro della PCB che vanno al connettore USB, cerchiamo di individuare i possibili poli + e - da 5V.



#### PC USB Connectors Pinout

- 1 = +5 Volts
- 2 = -Data
- 3 = +Data
- 4 = Ground



A questi poli saldiamo due cavetti, che poi andranno a loro volta connessi al pin VIN e GND della board NodeMCU

L'ultimo passo consiste nel collocare la board e l'accelerometro all'interno della scatola.

Personalmente ho trovato un pò di difficoltà nel far entrare la board di lungo dentro il case di plastica. Per farci stare il tutto è stato necessario rimuovere parti di plastica superflue e sciogliere e deformare



leggermente il contenitore. Inoltre per comodità ho voluto mantenere tutti i pin della board, ruotandoli di 90° per poi posizionare la board incastrata sul coperchio superiore.

Per collegare accelerometro e alimentazione al NodeMCU piuttosto che effettuare delle saldature permanenti è risultato più comodo prendere dei cavetti jumper femmina e rimuovere il cappuccio in plastica, dopodichè si possono inserire e isolare l'uno dall'altro con un pò di nastro isolante. Come si mostra in foto.



*Figura 82 - Jumper e collegamenti*

Lo schema di collegamento è il seguente:

<b>MPU6050</b>	<b>NodeMCU devkit</b>
VCC	3.3V
GND	GND
SDA	D2
SCL	D1

Nota. Nel codice arduino, il mapping dei pin non segue la serigrafia sulla board. D2 corrisponde a 4 e D1 corrisponde a 5.

Una volta posizionata la board NodeMCU ed MPU-6050. Possiamo assicurarci che il tutto funzioni collegando la presa elettrica. Se il tutto viene alimentato si è pronti all'upload dello sketch.

Se la board è alimentata e funziona come previsto. Passiamo a richiudere il tutto.

Le componenti hardware, una volta rinchiusi nel box, sono a stretto contatto. Questo non è l'ideale. Si potrebbe facilmente incorrere in cortocircuiti. Per evitare che succeda è sufficiente usare delle pellicole di plastica e stenderle sopra al modulo d'alimentazione, così che non possa entrare in contatto diretto con la board.



*Figura 83 - Pellicole isolanti*

Il prototipo è pronto per l'utilizzo.



*Figura 84 - Prototipo di sismometro*

#### 4.1.3 - Software

Il codice sorgente necessario è hostato sulla repository ufficiale del progetto SeismoCloud, direttamente su github:

[github.com/sapienzaapps/seismoclouddevice-nodemcu](https://github.com/sapienzaapps/seismoclouddevice-nodemcu)

Nella stessa repository si trova un breve How-To che spiega le librerie e gli step necessari a caricare il codice basato su Arduino.

La versione attuale, prevede che la posizione di latitudine e longitudine siano manualmente inserite all'interno del codice. Essendo un Sismometro digitale ad uso abitativo, è meglio accertarsi di aver inserito le coordinate in base al luogo d'effettivo utilizzo.

Le chiamate alle API di SeismoCloud sono già implementate all'interno del codice sorgente sulla repository. Se si apportano modifiche al codice è sempre bene effettuare dei test locali, a tal proposito sarebbe utile istanziare un server sulla propria macchina di sviluppo ed effettuare i dovuti test prima di effettuare chiamate alle API ufficiali SeismoCloud.

La comunicazione fra la board e l'accelerometro avviene attraverso il protocollo di comunicazione I2C, che rende semplice l'interazione.

Essendo il microcontrollore sprovvisto di alimentazione autonoma, non si hanno meccanismi interni per il mantenimento dell'orario di sistema allo spegnimento. NTP è la soluzione adottata. Ad ogni avvio viene richiesto a un server NTP l'orario corrente con un aggiornamento periodico, ogni 15 minuti.

Il firmware di arduino fornisce pratiche librerie per l'interazione con gli accelerometri I2C. Optando per il firmware Lua si rinuncia all'utilizzo di queste librerie. Si va incontro a una mole di lavoro non indifferente. E' necessario infatti studiare la documentazione dell'accelerometro per leggere i registri che contengono le posizioni degli assi prima di poter applicare il classico algoritmo di rilevamento della vibrazione.

Inoltre per via di limitazioni hardware e scarsa ottimizzazione del firmware interpretato, si aggrava la situazione. Alcune operazioni sui registri non possono essere effettuate facilmente per via di un consumo eccessivo di memoria.

La parte relativa alla configurazione della rete wireless è gestita tramite la libreria WifiManager, già vista in precedenza. La comodità risiede nella possibilità di poter aprire un Access Point di configurazione all'avvio. Dopodichè si possono inserire le credenziali d'accesso e altri dati ausiliari per la connessione alla rete Wifi, necessaria per poter comunicare il rilevamento di vibrazioni ai server SeismoCloud.

Stabilita la connessione wifi verrà effettuata la calibrazione dell'accelerometro, ripetuta ogni 30 secondi. Con la differenza dei quadrati rispetto alla misurazione corrente ed ai valori di calibrazione, si effettua una rilevazione, se il valore è superiore a una certa soglia allora viene considerata come vibrazione.

Sono le variazioni rispetto alla calibrazione iniziale che determinano la presenza di vibrazioni o meno.

## 5. Conclusioni

---

L'obiettivo della tesi è di mostrare l'implementazione di dashboard integrative per reti di sensori e attuatori.

Utilizzando differenti protocolli, tecnologie e metodi di trasmissione.

Ogni Dashboard ha in comune l'utilizzo di un unico ambiente di sviluppo, Node.js, basato interamente sull'utilizzo del linguaggio interpretato Javascript.

Questo garantisce l'esecuzione multiplatforma e la flessibilità necessaria per portare i software-server su differenti architetture hardware.

Il protocollo di comunicazione MQTT è il protocollo che più di tutti sta suscitando clamore nel settore IoT e viene usato come base per stabilire un dialogo fra i nodi della rete. Come visto, aziende del calibro di Facebook o Amazon si dotano di piattaforme compatibili con queste tecnologie.

L'obiettivo nella prima sezione della tesi è mostrare la semplicità con cui è possibile realizzare una rete MQTT, usando microcontrollori e sensori a basso costo, con canali di comunicazione affidabili, a bassa latenza e che garantiscano un consumo energetico minimo sui dispositivi embedded.

Dunque avere una Dashboard di monitoraggio e controllo, realizzata come Chrome-App.

I Sensori e i controllori utilizzati rappresentano quanto di meglio attualmente sul mercato in termini di rapporto qualità prezzo. Si è visto quali sono le board più utilizzate dai Maker, il chip ESP8266 di Espressif, i firmware disponibili, Arduino, Espruino, MicroPython, NodeMCU Lua. Tutte queste tecnologie adottate sono open source e open hardware.

Nella seconda sezione, si è visto un sistema di controllo completo basato su trasmissione radio (433 Mhz). Questo progetto in particolare è stato rilasciato come software open source, pronto per l'utilizzo e ad oggi conta centinaia di download con una community attiva, pronta al rilascio di una prossima release. Contrariamente alla Chrome-app della sezione precedente, l'interfaccia viene sviluppata su web browser,

esposta direttamente dal gateway di comunicazione radio (Raspberry Pi). Nella trattazione si mostra ogni fase per l'esecuzione del sistema iot-433mhz. Dal reperimento dell'hardware, gli schemi di collegamento per i moduli radio, all'installazione del software, l'interfaccia web e le API per l'estensione del sistema. Successivamente vengono discusse le scelte implementative e la topologia di rete. Nonchè i sistemi già esistenti a cui il progetto si ispira sotto certi aspetti e i miglioramenti che è possibile apportare, realizzando ad esempio una rete più eterogenea, multi-gateway e multi-protocollo.

Infine nella terza e ultima sezione la riprova di quanto un microcontrollore come NodeMCU possa essere adatto a molteplici scenari. Si costruisce per l'appunto, un prototipo compatto e funzionante di sismografo digitale, basato su accelerometro MPU-6050, connesso al sistema di allerta per terremoti, SeismoCloud, mostrando passo passo le procedure d'assemblaggio.

L'internet of Things (IoT) ha per molto tempo favorito l'interconnettività di dispositivi ed ecosistemi altrimenti separati. I dispositivi connessi sempre più fanno parte della nostra quotidianità. Migliorano il nostro stile di vita e portano produttività, intrattenimento e sicurezza nelle nostre case. Con la diffusione di internet e i costi in discesa, sempre più dispositivi e sensori vengono creati con capacità di connessione alla rete. La connessione di oggetti fisici alla rete permette l'accesso remoto ai dati e il controllo del mondo fisico a distanza.

L'internet of Things (IoT) a detta degli esperti è la prossima rivoluzione. La verità è che ci sono infinite applicazioni per l'Internet of Things. Che sia un frigorifero che ci avvisi della scadenza degli alimenti, un sistema d'irrigazione automatica o un termostato intelligente che regoli la temperatura della nostra abitazione, l'unico limite è la nostra immaginazione.

Ci sono soluzioni IoT per qualunque settore. In medicina esistono già sensori sottoforma di pillole o bottiglie con reminder incorporati. In agricoltura gli agricoltori possono avere informazioni dettagliate sulle condizioni del terreno e le condizioni atmosferiche grazie a soluzioni IoT.

Molte sono le sfide da affrontare, come fare un uso intelligente dell'enorme mole di dati collezionati? come offrire soluzioni che possano facilmente essere integrate nelle nostre abitazioni da chiunque e non solo per ingegneri o hobbysti?

Nell'immediato futuro, i produttori di queste tecnologie dovranno assicurarsi che gli utenti non anneghino in un mare d'informazioni,

fornendo strumenti utili d'analisi, garantendo armonia fra le varie applicazioni, facilità nell'installazione ed un'esperienza utente all'avanguardia.

# Bibliografia

- Amazon Web Services. (2016, 02 21). *AWS IoT*. Retrieved from Amazon Web Services: <http://aws.amazon.com/it/iot/>
- Borioli. (2013, november 24). *Playing with MQTT*. Retrieved from The monday morning tech newsletter: <http://mmtn.borioli.net/?p=1342>
- Dominique D. Guinard, V. M. (2016). *Building the web of things*. Zurich: Manning.
- Eclipse. (2016, 01 01). *Open Source messaging for M2M*. Retrieved from Paho: <https://eclipse.org/paho/>
- Foundation, R. P. (2016, 2 12). *Raspberry Pi*. Retrieved from Raspberry Pi: <https://www.raspberrypi.org/>
- Gartner. (2015, 11 10). *6.4 billion connected "things" will be in use in 2016*. Retrieved from Gartner: <http://www.gartner.com/newsroom/id/3165317>
- Guinard, D. (2011, August 1). *A Web of Things Application Architecture -- Integrating the Real-World into the Web*. ETH Zurich, Zurich, Switzerland.
- IBM. (2014, 11 7). *MQTT*. Retrieved from MQTT: <http://mqtt.org/>
- Inc., A. (2016, 02 17). *What are Chrome apps?* Retrieved from Google Chrome Developer: [https://developer.chrome.com/apps/about\\_apps](https://developer.chrome.com/apps/about_apps)
- Joyent, I. (2016, 02 12). *About Node JS*. Retrieved from Node JS: <https://nodejs.org/en/about/>
- Lampkin, V., Leong, W. T., Olivera, L., Rawat, S., Subrahmanyam, N., & Xiang, R. (2012). *Building Smarter Planet solutions with MQTT and IBM WebSphere MQ Telemetry*. RedBooks.
- Miller, J., & Marschalko, M. (2016, February 08). *Hardware Hacking with Javascript*. Retrieved from smashing magazine: <https://www.smashingmagazine.com/2016/02/hardware-hacking-with-javascript-internet-of-things/>
- Musolino, R. (2015, november 28). *Chip o Pi Zero - quale mini computer scegliere*. Retrieved from Hackers Tribe: <http://www.hackerstribes.com/2015/c-h-i-p-o-pi-zero-quale-mini-computer-scegliere/>
- Musolino, R. (2015, 10 2). *esp8266 flashare il firmware nodemcu*. Retrieved from Hackers Tribe: <http://www.hackerstribes.com/2015/esp8266-flashare-il-firmware-nodemcu/>
- Musolino, R. (2015, 07 25). *ESP8266 flashare il firmware nodemcu*. Retrieved from Hackers Tribe: <http://www.hackerstribes.com/2015/esp8266-flashare-il-firmware-nodemcu/>

- Musolino, R. (2016, 1 12). *Costruire un Sismometro Digitale con esp8266 NodeMCU e MPU-6050*. Retrieved from Hackers Tribe:  
<http://www.hackerstribes.com/2016/costruire-un-sismometro-digitale-con-esp8266-nodemcu-e-mpu-6050/>
- Musolino, R. (2016, 2 9). *Domotica e Internet of Things tramite sensori a 433mhz*. Tratto da Hackers Tribe: <http://www.hackerstribes.com/2016/domotica-e-internet-of-things-tramite-sensori-a-433mhz/>
- Musolino, R. (2016, 2 10). *IoT-433Mhz official Repo*. Retrieved from github:  
<https://github.com/roccomuso/iot-433mhz>
- Nicholas, S. (2012, May 31). *Power Profiling: HTTPS Long Polling vs MQTT with SSL*. Retrieved from Stephendnicholas.com:  
<http://stephendnicholas.com/archives/1217>
- NodeMCU-Team. (2014, 04 04). *NodeMCU - an Open-source firmware based on ESP8266 wifi-soc*. Retrieved from NodeMCU:  
[http://nodemcu.com/index\\_en.html](http://nodemcu.com/index_en.html)
- tzapu. (2016, 02 21). *ESP8266 WiFi Connection manager with web captive portal*. Retrieved from github.com/tzapu/WiFiManager:  
<https://github.com/tzapu/WiFiManager>