UNIVERSITA' DI PISA

FACOLTA' DI INGEGNERIA
CORSO DI LAUREA SPECIALISTICA IN
INGEGNERIA INFORMATICA PER LA GESTIONE D'AZIENDA

Progettazione e sviluppo di un'applicazione web per la configurazione e la visualizzazione di un sistema domotico basato su protocollo ZigBee

Relatori:
Prof. Ing. Giuseppe Iannaccone
Prof. Enzo Mingozzi
Dott. Ing. Elisa Spanò

Tesi di Laurea di:
Irene Bontà

May 9, 2013

Sommario

Indice delle figure

1. Introduzione

Il presente lavoro di tesi tratta della progettazione e dello sviluppo di un'applicazione web per la configurazione e la visualizzazione di un sistema domotico Wireless Sensor Network (WSN) basato su protocollo ZigBee.

Il lavoro si è incentrato sulla configurazione dell'interfaccia utente, a partire dallo strato più esterno, fino alla configurazione degli accessi, alla visualizzazione dei dati, all'invio di comandi e alla ricezione delle risposte provenienti dai nodi presenti sulla rete.

Nel secondo capitolo viene descritto il sistema nelle sue parti principali, seguendo un percorso che va dalle reti all'utente, passando per il server che gestisce la comunicazione con i dispositivi, ai database per la raccolta dei dati, al frontend.

Nel terzo capitolo vengono descritti gli strumenti utilizzati per la progettazione, lo sviluppo e la configurazione del sistema. Il percorso si snoda attraverso gli strumenti prettamente grafici, che gestiscono gli aspetti visivi del sistema. Si passa poi alla descrizione delle caratteristiche dei web server, che gestiscono la configurazione del sistema e la comunicazione tra client e reti di dispositivi. In questo capitolo vengono soprattutto messe in luce le caratteristiche di ogni strumento utilizzato, per evidenziare i vantaggi che hanno portato alla scelta dell'utilizzo degli stessi.

Nel quarto capitolo viene descritta nel dettaglio la fase di progettazione e di sviluppo del sistema. Con la presenza di numerose e significative parti di codice, si vuole mostrare quali sono state le scelte di implementazione dell'intero sistema e quali possono essere i possibili scenari futuri di sviluppo.

Infine, nel quinto capitolo, vengono presentati alcuni casi di test, che vogliono dimostrare le funzionalitá e le caratteristiche attuali del sistema, secondo quanto descritto nei capitoli precedenti.

Ho scelto tale argomento di tesi in quanto riassume molti degli argomenti appresi in questi cinque anni universitari, dalla progettazione allo sviluppo di reti, applicazioni web e sistemi complessi e dinamici, alla gestione delle risorse, dall'ottimizzazione della fase di scrittura del codice, all'apertura nell'ottica di un possibile e continuo sviluppo futuro.

Inoltre ho scelto questo argomento perché il campo della domotica è un settore in crescita ed in continua espansione, quindi l'esperienza acquisita con questo lavoro è facilmente spendibile eventualmente nel mondo del lavoro.

2. Sistema MetroPower

Il lavoro di tesi si inserisce nel progetto di ricerca MetroPower, attivo presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Pisa.

Il progetto consiste nello sviluppo di un'infrastruttura condivisa da reti di sensori, utenti e servizi, avente come interfaccia utente un'applicazione web.

Tramite questa interfaccia è possibile accedere remotamente ai singoli nodi delle reti, indipendementemente dal protocollo di comunicazione, riconfigurarli, visualizzarne i dati ed inviare ad essi dei comandi.

Nell'ambito del lavoro di tesi, il sistema MetroPower è stato utilizzato in un'applicazione domotica basata sul protocollo Wireless Sensor Network (WSN) ZigBee (Figura 1).

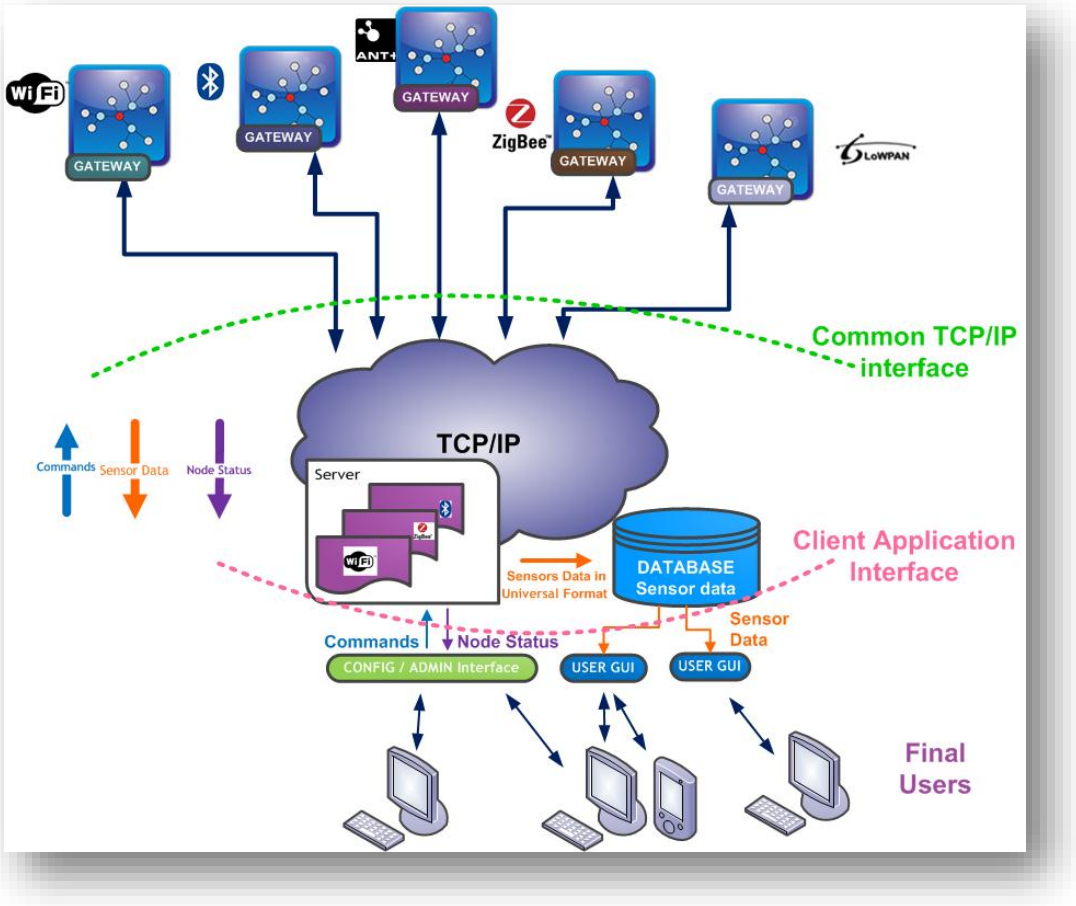


Figura 1 - Architettura WSN

Obiettivi del sistema MetroPower sono quelli di sviluppare, testare, ottimizzare e validare un prototipo completo di sistema wireless per la gestione intelligente dell'energia in edifici residenziali e commerciali, controllato attraverso l'accesso ad un servizio web.

Il sistema gestisce remotamente messaggi che vengono trasferiti in entrambe le direzioni: dal sensore all'utente (upstream) e dall'utente al sensore/attuatore (downstream).

Le caratteristiche principali offerte dal sistema all'utente finale sono:

- La possibilità di conoscere e modificare lo stato di ogni nodo della rete;
- La possibilità di riconfigurare e riprogrammare remotamente i nodi della rete;
- Un trasferimento sicuro dei dati;
- Il salvataggio dei dati letti dai sensori in un database a scorrimento;
- Un accesso sicuro all'informazione e alle reti solo agli utenti autorizzati.

2.1. Panoramica del sistema

Il sistema MetroPower si basa sull'architettura descritta in Figura 2.

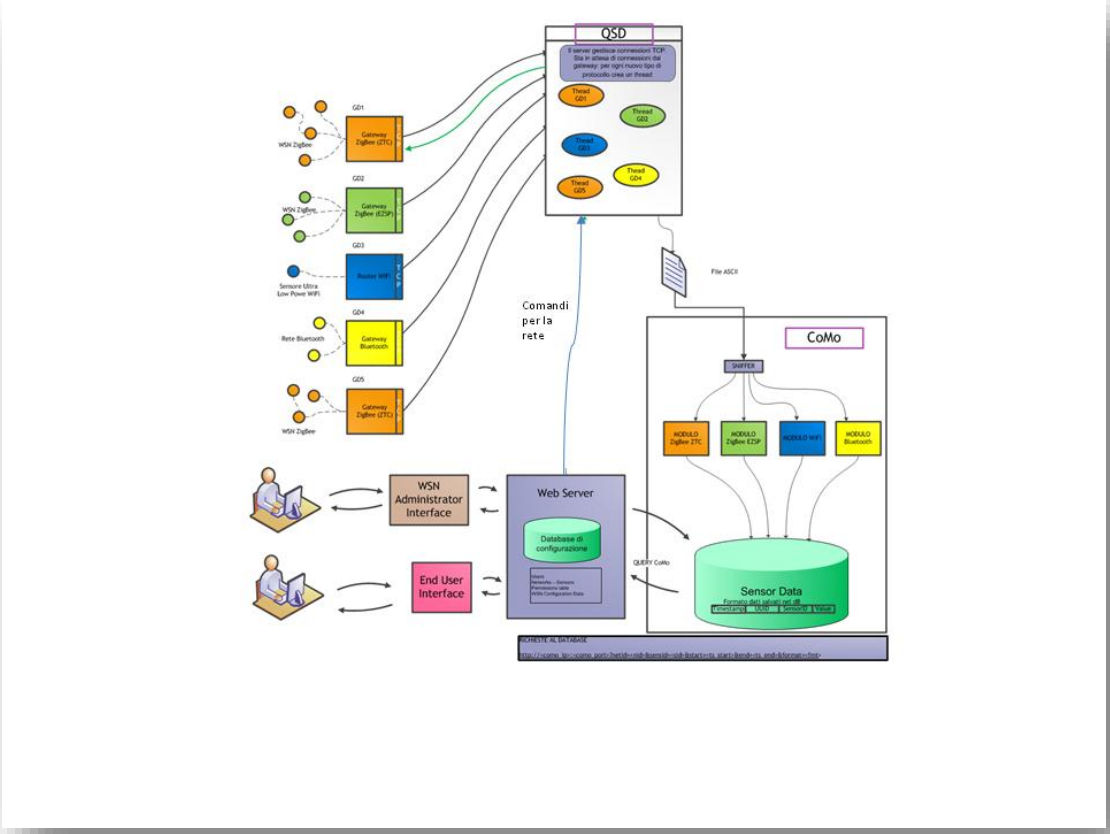


Figura 2- Schema del sistema MetroPower

I componenti principali dell'architettura sono:

- I sensori wireless e gli attuatori;
- Un gateway;
- Un server di comunicazione remota;
- Un software per la raccolta dei dati;
- Il software di front-end;
- L’interfaccia utente.

2.1.1. Sensori wireless e attuatori

Il sistema MetroPower può funzionare con qualsiasi tipo di rete di sensori, poiché è stato progettato in modo da poter cambiare protocollo a seconda delle esigenze. L'implementazione attuale è basata sull'utilizzo di reti ZigBee.

I nodi sensori presenti nell'attuale sistema MetroPower includono sensori per misurare la potenza assorbita da utenze elettriche (smart meters), temperatura, luce e presenza. Il sistema include anche nodi attuatori ed un coordinatore di rete che implementa le politiche di gestione della stessa.

Ogni nodo contiene, oltre ai sensori, l'hardware necessario al condizionamento del segnale, un microcontrollore, una piccola memoria, il trasmettitore ZigBee a un'antenna da 2,45GHz.

2.1.2. Gateway

Il gateway è l'elemento di unione tra la rete di sensori/attuatori e Internet. Il suo ruolo è quello di trasmettere i dati tra i nodi della rete e un host remoto: da una parte, raccoglie i dati ricevuti dai nodi della rete, li processa e li manda su un collegamento TCP/IP sicuro a un host remoto; dall'altra parte, è responsabile dell'invio ai nodi ricevitori di ogni comando proveniente dal lato server.

Il gateway è stato implementato su una scheda hardware che include un chip dedicato alla comunicazione di rete ZigBee, e un microcontrollore Cortex-M3-based, per la comunicazione Ethernet. Il firmware sul gateway implementa un micro-IP stack per la comunicazione con il server remoto e permette una comunicazione sicura (>AES128) a doppio senso tra i singoli sensori e il sistema. La connessione del gateway con il server remoto viene raggiunta, nella nostra implementazione, da un comune router ADSL.

2.1.3. Server per la comunicazione remota

Il server remoto è responsabile della gestione della comunicazione TCP/IP con ogni gateway connesso al sistema, senza preoccuparsi del tipo di rete collegata o del formato dei dati presente all'interno del payload del pacchetto. Le funzioni principali del server consistono nel:

- “Ascoltare” le connessioni provenienti dalla rete TCP;
- Accettare le richieste di connessione entranti provenienti dai nuovi gateway;
- Stabilire e mantenere connessioni sicure con più gateway contemporaneamente;
- Decriptare i pacchetti ricevuti dalle reti, indipendentemente dal formato dei dati, e passarli al software di back-end (software per la raccolta dei dati) per l'interpretazione del contenuto;
- Inviare al corretto gateway i comandi o i messaggi provenienti dal front-end e destinati ad una specifica rete.

Nel nostro sistema MetroPower il server remoto è intenzionalmente disegnato per essere trasparente alla rete ed ai tipi di dati. Esso deve preoccuparsi solo di tutte le comunicazioni di basso livello tra il gateway, e di conseguenza le reti, e il sistema remoto. Abbiamo definito un protocollo di comunicazione gateway-server su TCP/IP con l'idea di imporre vincoli minimi che tutte le reti possono incontrare con bassi requisiti hardware e software. Questo protocollo descrive un access point ben definito al sistema.

2.1.4. Software per la raccolta dei dati

Il software per la raccolta e l'archiviazione dei dati (CoMo) ha il compito di interpretare, processare e salvare i dati dei sensori per l'accessibilità da parte dell'utente finale di servizi web. È disegnato con l'idea di separare le fonti dalle informazioni che generano. Il software si basa su più moduli. Ogni modulo a valle delle reti (downstream) è capace di decifrare un tipo specifico di messaggio. I moduli ricevono in ingresso i pacchetti raccolti dal server e sono capaci di estrarre da essi i dati letti dai sensori. Dopo di che le letture dei sensori vengono formattate in un formato generale indipendente dalla rete e salvate in un database a scorrimento.

2.1.5. Software front-end

Il front-end (implementato dal web server Tornado) è la parte del sistema che gestisce l'accesso dell'utente al sistema. Grazie ad un database di configurazione interno contenente le tabelle di utenti, reti, sensori, chiavi e permessi, il front-end può fornire un accesso sicuro all'informazione e alle reti solamente agli utenti autorizzati.

2.1.6. Interfaccia utente

L’interfaccia permette all'utente di accedere al sistema attraverso un'applicazione client web-based. L'interfaccia sviluppata per il sistema MetroPower cerca di fornire un ambiente familiare all'utente e un insieme di funzionalità facilmente utilizzabili per interagire con i dispositivi remoti. Attraverso comuni pagine web anche utenti non esperti possono recuperare e visualizzare i dati, così come configurare alcuni sensori secondo le credenziali che essi hanno, utilizzando un qualsiasi computer o smartphone connesso a Internet.

3. Strumenti utilizzati

In questo capitolo vengono descritti gli strumenti utilizzati nella progettazione dell'intera interfaccia del sistema MetroPower.

Per quanto riguarda la parte prettamente grafica, la scelta è ricaduta sull'utilizzo di Twitter Bootstrap per la parte html e su Highcharts/Highstock per la visualizzazione dei grafici dei dati provenienti dai dispositivi della rete.

Per quanto riguarda invece i due web server, vengono descritti Tornado, utilizzato per la parte di front-end, e CoMo, utilizzato per la parte di back-end del sistema MetroPower.

3.1. Twitter Bootstrap

Twitter Bootstrap è un front-end framework open source, realizzato dai creatori di Twitter, utilizzabile per lo sviluppo di applicazioni e pagine web. Tale strumento contiene un'ampia collezione di elementi grafici (bottoni, form, typography, tabelle, griglie, etc.) e pronti per essere inclusi in un progetto web. L'ovvio vantaggio di questa impostazione è che lo sviluppo di un template (front-end o back-end) è molto più semplice e veloce.

E' il progetto più popolare su GitHub e viene usato tra gli altri da NASA e MSNBC.

3.1.1. Origine

Bootstrap è stato sviluppato da Mark Otto [1] e Jacob Thornton di Twitter come un framework per incoraggiare la consistenza attraverso strumenti interni. Prima di Bootstrap, venivano usate varie librerie per lo sviluppo dell'interfaccia, che portavano a inconsistenze e ad un alto costo di mantenimento. Secondo lo sviluppatore di Twitter Mark Otto, di fronte a tali problemi: *"...[Un] gruppo molto ristretto di sviluppatori ed io ci siamo riuniti per disegnare e costruire un nuovo strumento interno ed abbiamo visto un'opportunità per fare qualcosa di più. Attraverso quel processo, ci siamo accorti che stavamo costruendo qualcosa di ben più sostanziale che un altro strumento interno. Mesi dopo, ci siamo ritrovati con una prima versione di Bootstrap, ovvero uno strumento per documentare e condividere comuni modelli di progettazione ed attività all'interno dell'azienda*" [2].

La prima distribuzione del framework ci fu durante la prima Hackweek di Twitter. Mark Otto mostrò ad alcuni colleghi come velocizzare lo sviluppo dei propri progetti con l'aiuto del kit di strumenti. In seguito, dozzine di team sposarono l'uso del framework.

Ad Agosto 2011 Twitter rilasciò Bootstrap come open-source. A Febbraio 2012, era il progetto di sviluppo più popolare su GitHub.

3.1.2. Caratteristiche

Bootstrap ha un supporto relativamente incompleto per HTML5 e CSS 3, ma è compatibile con tutti i maggiori browser. Informazioni di base di compatibilità di siti web o applicazioni sono disponibili per tutti i dispositivi e browser.

C'è un concetto di parziale compatibilità che rende l'informazione base di un sito web disponibile per tutti i dispositivi e browser. Per esempio, le proprietà introdotte in CSS3 per angoli stondati, gradienti e ombre sono usate da Bootstrap a dispetto della mancanza di supporto su vecchi web browser. Queste estendono la funzionalità del kit di strumenti ma non sono richieste per il suo utilizzo.

L’utilizzo di un framework valido come Twitter Bootstrap permette di creare applicazioni e siti web con un’ottima compatibilità multipiattaforma. Ciò significa che un template realizzato con questo set di strumenti avrà un aspetto simile in tutti i browser (IE, Firefox, Chrome, Safari, etc.) evitando un ulteriore spreco di tempo da parte degli sviluppatori per scrivere codice “personalizzato” per uno specifico software di navigazione web.

Dalla versione 2.0 (l’attuale versione del framework è la 2.3.1), Twitter Bootstrap supporta anche il “responsive design”. Questo significa che il modello grafico delle pagine web si aggiusta dinamicamente, tenendo conto delle caratteristiche del dispositivo utilizzato (PC, tablet, cellulare).

Bootstrap è open source e disponibile su GitHub. Gli sviluppatori sono incoraggiati a partecipare nel progetto ed a fornire il proprio contributo alla piattaforma.

I componenti disponibili in Bootstrap sono elencati in Figura 3.

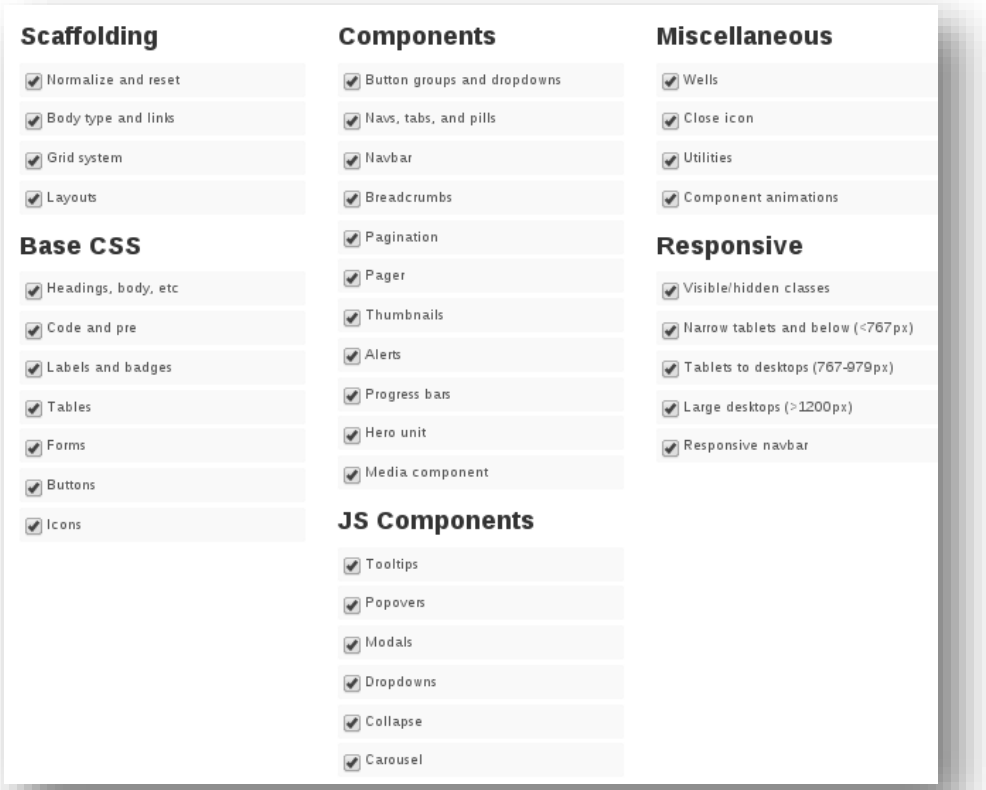


Figura 3 - Componenti di Twitter Bootstrap

3.1.3. Struttura e funzione

Bootstrap è modulare e consiste essenzialmente di una serie di fogli di stile LESS (Figura 4) che implementano i vari componenti del kit di strumenti. Un foglio di stile chiamato bootstrap.less include i fogli di stile componenti. Gli sviluppatori possono adattare il file di Bootstrap stesso, selezionando i componenti che vogliono usare nel proprio progetto.

Figura 4 - Struttura di Twitter Bootstrap

Gli aggiustamenti sono possibili in maniera limitata attraverso un foglio di stile di configurazione centrale. Modifiche più profonde possono essere fatte attraverso le dichiarazioni LESS.

L'uso di fogli di stile LESS di lingua permettono l'utilizzo di variabili, funzioni e operatori, selettori annidati, così come i cosiddetti mixin.

Dalla versione 2.0, la configurazione di Bootstrap ha anche una speciale opzione "Personalizza" nella documentazione. Inoltre, lo sviluppatore sceglie su un modulo i componenti che desidera e aggiusta, se necessario, i valori delle varie opzioni in base ai propri bisogni. Il pacchetto successivamente generato include già il foglio di stile CSS pre-costruito.

3.1.4. Sistema a griglia e responsive design

Bootstrap normalmente si presenta con un layout a griglia di 940 pixel di larghezza. Tale layout a griglia, detto “grid system”, è visibile in Figura 5.

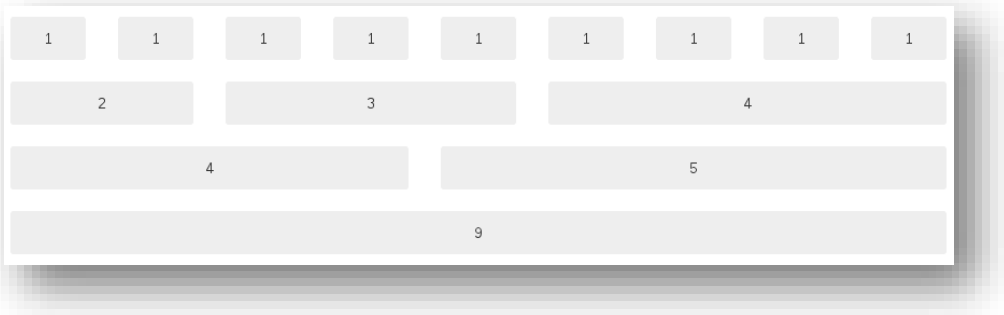


Figura 5 - Grid System di Bootstrap

Alternativamente, lo sviluppatore può utilizzare un layout a larghezza variabile, detto “Fluid Grid System”, visibile in Figura 6.

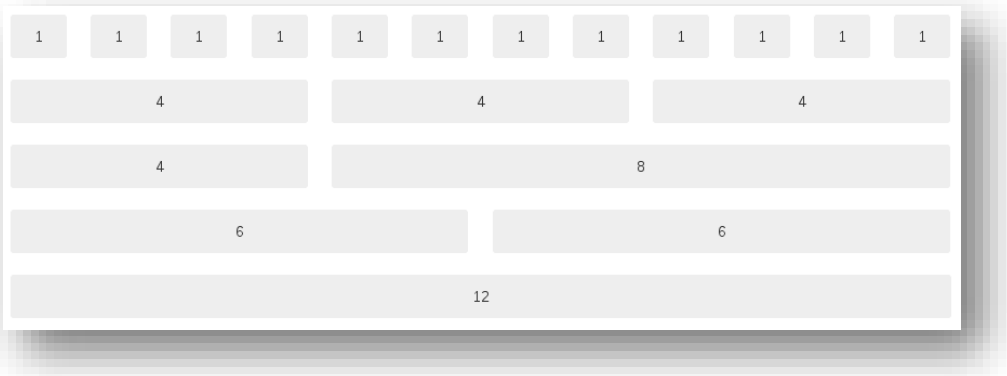
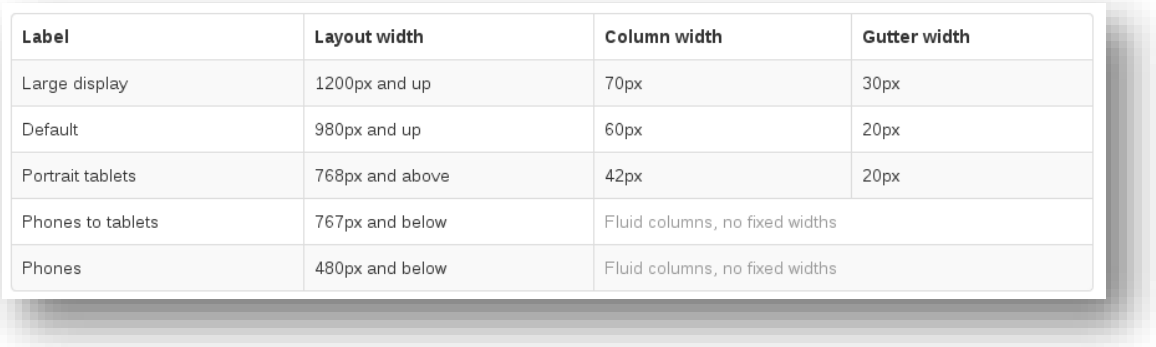


Figura 6 - Fluid Grid System di Bootstrap

In entrambi i casi, il kit di strumenti ha quattro varianti per fare uso di diverse risoluzioni e tipi di dispositivi: telefoni cellulari, formato ritratto e panoramica, Tablet e PC con bassa e alta risoluzione (wide screen).

Inoltre, come già evidenziato, Twitter Bootstrap offre un valido supporto per lo sviluppo "responsive". Ciò significa che un sito web creato con questo framework si adatterà alle risoluzione dello schermo dell'utente, che potrà visualizzarlo sia sul monitor di casa che su dispositivi mobili, quali smartphone, tablet, palmari, etc. (Figura 7).



Label	Layout width	Column width	Gutter width
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait tablets	768px and above	42px	20px
Phones to tablets	767px and below	Fluid columns, no fixed widths	
Phones	480px and below	Fluid columns, no fixed widths	

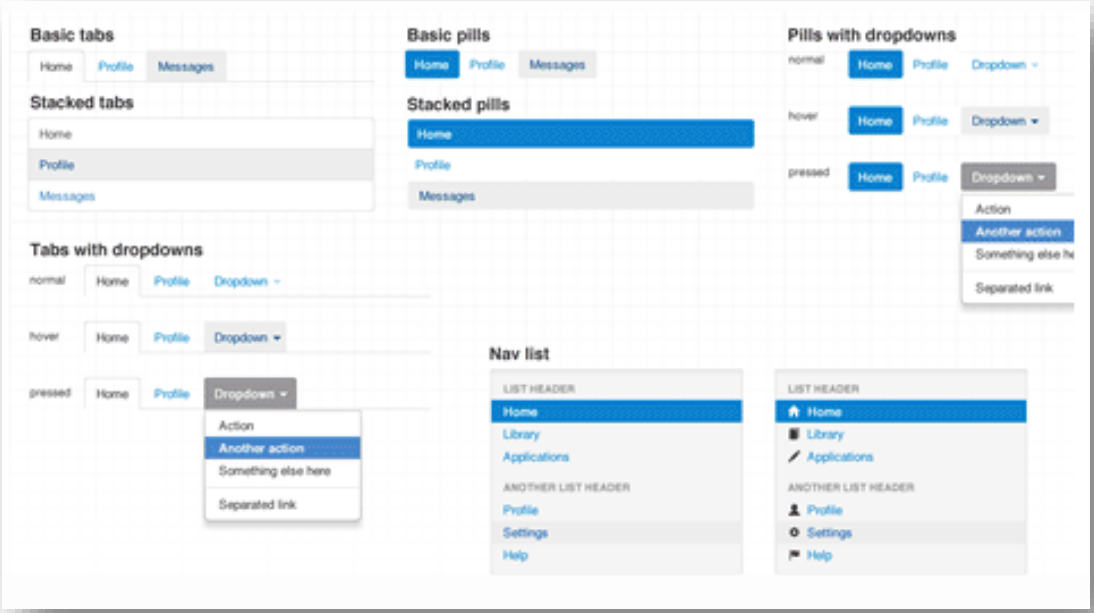
Figura 7 - Dispositivi supportati da Bootstrap

3.1.5. Fogli di stile CSS

Bootstrap fornisce una serie di fogli di stile (Cascading Style Sheets) che offrono definizioni di base di stile per tutte le componenti chiave HTML. Tali fogli di stile offrono un aspetto moderno, uniforme a livello del browser e di sistema per la formattazione del testo, delle tabelle e degli elementi dei moduli.

3.1.6. Componenti riutilizzabili

In aggiunta ai normali elementi HTML, Bootstrap contiene altri elementi di interfaccia comunemente usati. Questi includono bottoni con caratteristiche avanzate (ad es. raggruppamento di bottoni o bottoni con opzione di cascata, liste di navigazione, schede orizzontali e verticali, navigazione, paginazione, ecc.), etichette, miniature con avanzate capacità tipografiche, formattazione per messaggi di errore e barre del progresso (Figura 8).



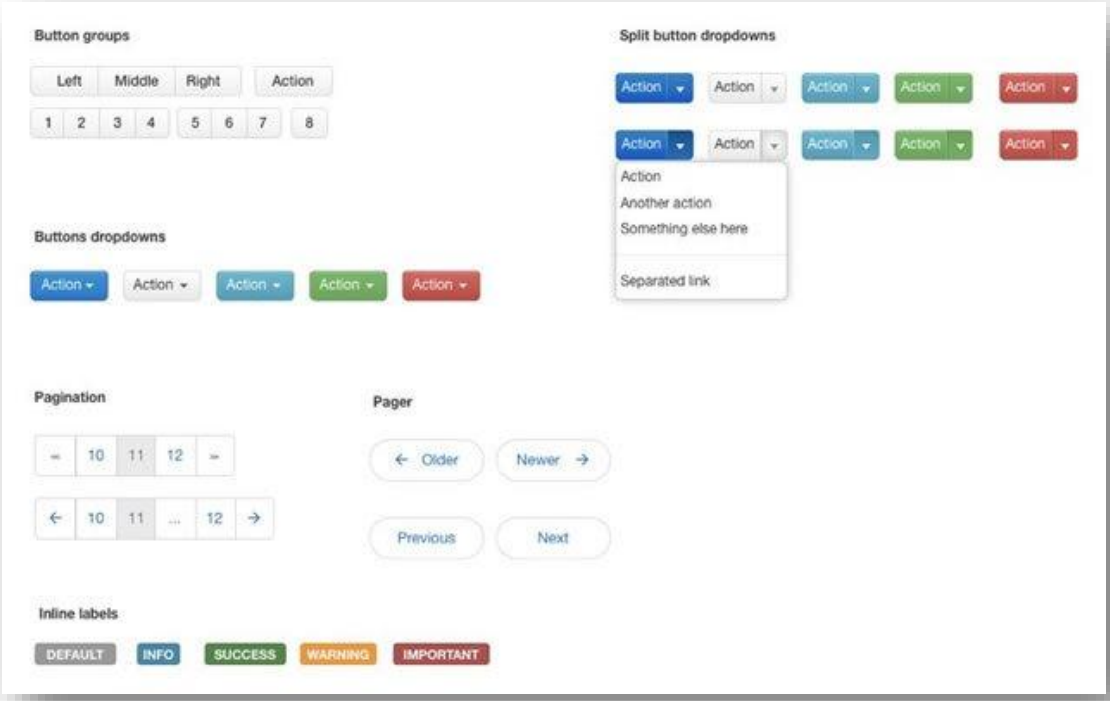


Figura 8 - Componenti di Bootstrap

3.1.7. Plug-in JavaScript

I componenti di Bootstrap si basano sulla libreria JavaScript jQuery. I plug-in si trovano quindi nei plug-in del kit di strumenti di jQuery. Essi forniscono elementi addizionali di interfaccia utente come finestre di dialogo, tooltip e caroselli. Inoltre estendono la funzionalità di alcuni elementi di interfaccia esistenti, inclusa per esempio una funzione di auto completamento per i campi di input. Nella versione 2.0, sono supportati i seguenti plug-in JavaScript: Modal, Dropdown, Scrollspy, Tab, Tooltip, Popover, Alert, Button, Collapse, Carousel and Typeahead (Figura 9).

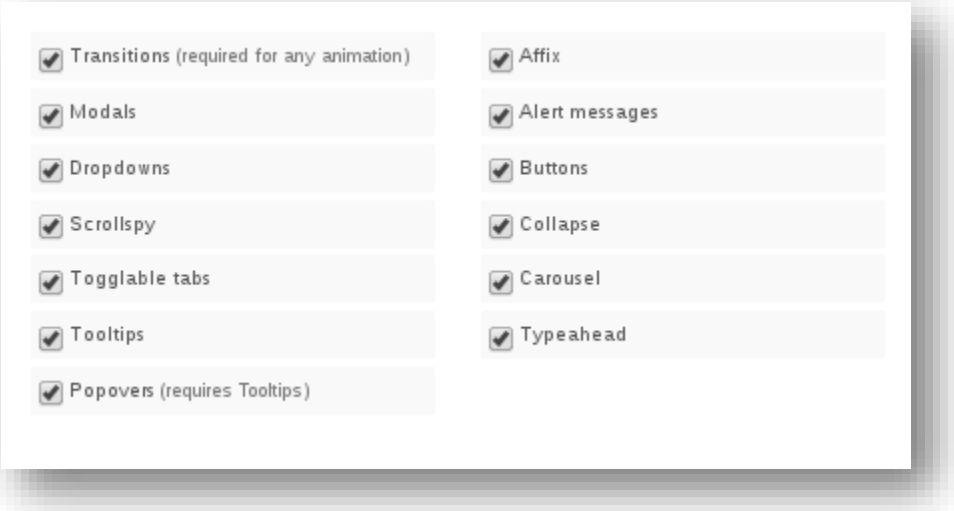


Figura 9 - jQuery plug-in di Bootstrap

E' disponibile anche un'implementazione di Twitter Bootstrap che utilizza il Dojo Toolkit. Si chiama Dojo Bootstrap e è un'esportazione dei plug-in di Twitter Bootstrap. Essa utilizza al 100% codice Dojo e ha il supporto per l'AMD (Asynchronous Module Definition).

3.1.8. Conclusioni

Considerati i vantaggi elencati, si è scelto di utilizzare il framework Twitter Bootstrap per realizzare l'interfaccia del sistema MetroPower così da realizzare un'interfaccia utente gradevole e funzionale, senza problemi di compatibilità multipiattaforma, aspetto questo fondamentale per un'applicazione web.

3.2. Tornado

3.2.1. Cosa è Tornado?

Tornado è un potente web server scalabile scritto in Python [3]. È abbastanza robusto per gestire grandi traffici web, ma è anche semplice da impostare e da scrivere e può essere usato per una gran varietà di applicazioni e utilità.

Tornado si basa sul framework di un web server che fu inizialmente sviluppato da Bret Taylor [4] ed altri per FriendFeed e successivamente reso open source da Facebook quando acquisì FriendFeed.

A differenza dei web server tradizionali che raggiungono un limite massimo intorno alle 10.000 connessioni simultanee, Tornado è stato scritto con una performance in mente, quella di riuscire a risolvere il problema C10K[1], perciò, per come è disegnato, è un framework per applicazioni web estremamente ad alte prestazioni. È inoltre impacchettato con strumenti per trattare la sicurezza e l'autenticazione degli utenti, i social network e l'interazione asincrona con servizi esterni come database e API web.

Dalla data del rilascio, il 10 Settembre 2009, Tornado ha raccolto un grande supporto dalla comunità ed è stato adottato per soddisfare una varietà di scopi. Oltre a FriendFeed e Facebook, una serie di aziende si sono rivolte a Tornado, incluse Quora, Turntable.fm, Bit.ly, Hipmunk e MyYearbook, per nominarne alcune.

[1] Il problema C10K si riferisce al problema di ottimizzare il software di un socket server per gestire un gran numero di client allo stesso tempo (da qui il nome C10K – diecimila connessioni concorrenti).

3.2.2. Vantaggi

I vantaggi principali che hanno fatto ricadere la scelta del web server su Tornado sono:

- Supporto di tutti i blocchi di base per la costruzione di un sito – Tornado si presenta con il supporto integrato per la maggior parte degli aspetti difficili e noiosi dello sviluppo web, inclusi template, cookie, autenticazione, localizzazione, caching aggressivo, protezione cross-site request forgery e autenticazione da terze parti come Facebook Connect.

- Servizi real-time – Tornado supporta un gran numero di connessioni concorrenti. È semplice scrivere servizi real-time attraverso long polling o streaming HTTP con Tornado. Ogni utente attivo mantiene una connessione aperta al server.

- Alte prestazioni – Tornado è abbastanza veloce in relazione alla maggior parte dei framework web scritti in Python. Nella Figura 10 è possibile vedere il confronto tra le prestazioni di Tornado e altri web server ad esso simili.

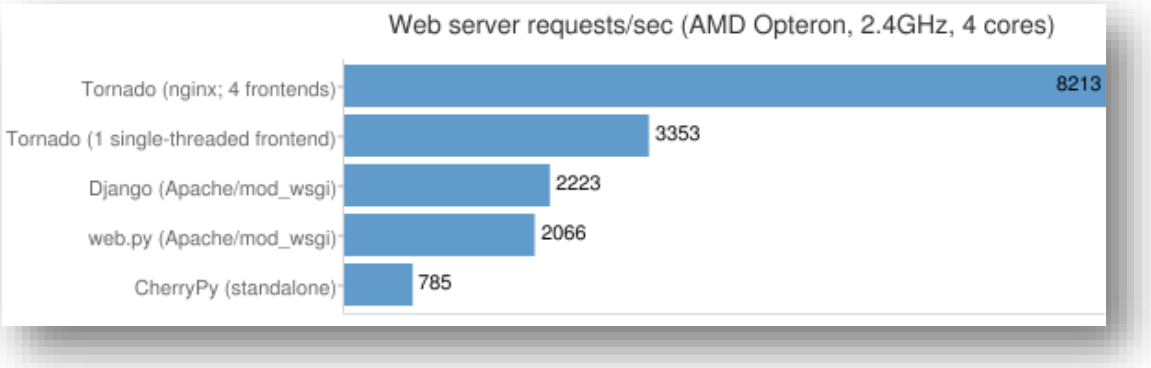


Figura 10 - Prestazioni di Tornado rispetto ad altri web server

3.2.3. La tecnologia dietro Tornado

Una volta installato, Tornado si presenta come una serie di moduli libreria utilizzabili per la realizzazione della propria applicazione [5]. Il modulo principale di Tornado è *tornado.web*, che implementa un leggero framework di sviluppo web. Il modulo è costruito su un server HTTP non bloccante e su moduli I/O di basso livello.

Un'applicazione web di Tornado mappa le URL o i pattern URL in sottoclassi di *tornado.web.RequestHandler*. Queste classi definiscono metodi get() o post() per gestire le richieste HTTP GET o POST a quella URL.

Tutte le caratteristiche aggiuntive di Tornado (come la localizzazione o i cookie) sono disegnate per essere utilizzate così "come da menu".

3.2.3.1. Richieste asincrone

La maggior parte delle applicazioni web sono bloccanti- questo significa che mentre viene gestita una richiesta, il processo rimane in attesa finché la richiesta non viene completata. Questo è un problema su larga scala.

Tornado fornisce però un modo per gestire la situazione. Invece di lasciare il processo in attesa mentre si aspetta che una richiesta termini, l'applicazione può avviare una richiesta e avvisare quando questa viene completata, lasciando l'I/O loop aperto per servire altri clienti mentre si attende che il primo processo sia completato.

La classe inclusa da Tornado è chiamata *AsynchHTTPClient* e consente di eseguire richieste in maniera asincrona. Essa è inclusa nel modulo *tornado.httpclient*.

Di default, quando un request handler viene eseguito, Tornado termina la richiesta automaticamente. Si può riscrivere questo comportamento predefinito per implementare connessioni streaming o pendenti, che sono comuni per servizi real-time. Se si vuole che una richiesta rimanga aperta dopo la fine del metodo request handler principale, basta usare il decoratore *tornado.web.asynchronous*.

Quando si usa questo decoratore, è propria responsabilità chiamare la funzione self.finish() per terminare la richiesta HTTP o il browser dell'utente rimarrà semplicemente in attesa.

3.2.3.2. Autenticazione da terze parti

Tornado si presenta con un supporto pre-caricato per l'autenticazione con Facebook Connect, Twitter, Google e FriendFeed in aggiunta a Oauth e OpenID.

Tutti i metodi di autenticazione supportano un'interfaccia relativamente uniforme così da poterli utilizzare senza dover conoscere tutte le complessità dei diversi protocolli di autenticazione e autorizzazione di cui si vuole usufruire sul proprio sito.

3.2.3.3 Utilizzo di template

Altra caratteristica interessante di Tornado è la possibilità di utilizzare i template per generare in maniera flessibile le pagine web della propria applicazione. I template sono file HTML che permettono di includere frammenti di codice Python al loro interno. Tornado mette a disposizione il modulo *tornado.template* che traduce i template in codice Python.

È possibile utilizzare un qualsiasi linguaggio a template supportato da Python, ma Tornado integra un proprio linguaggio a template molto più veloce e flessibile della maggior parte degli altri sistemi a template.

A differenza di molti altri sistemi a template, in Tornado non c'è alcuna restrizione sulle espressioni che si possono includere nei propri statement. I blocchi if e for vengono tradotti esattamente in Python.

Tradurre direttamente in Python significa che si possono applicare le funzioni alle espressioni facilmente. Si possono passare le funzioni nel proprio template così come ogni altra variabile.

Un template di Tornado, infatti, è semplicemente codice HTML con sequenze di controllo ed espressioni di Python integrate al suo interno:

```
<html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <ul>
      {% for item in items %}
      <li>{{ escape(item) }}</li>
      {% end %}
    </ul>
  </body>
</html>
```

I template di Tornado supportano statement di controllo ed espressioni. Gli statement di controllo sono racchiusi da *{%* e *%}.* Le espressioni sono racchiuse da *{{* e *}}*.

Gli statement di controllo mappano più o meno esattamente gli statement di Python. Supportano *if*, *for, while* e *try,* ognuno dei quali deve terminare con *{%* e *%}*. Inoltre è supportata l'ereditarietà dei template mediante l'utilizzo degli statement *extends* e *block*.

Le espressioni, invece, possono essere una qualsiasi espressione Python, incluse le chiamate di funzioni.

Dietro le quinte, i template di Tornado vengono tradotti direttamente in Python. Le espressioni incluse nel template vengono copiate alla lettera nella funzione Python che rappresenta il template. Di conseguenza, in presenza di errori nel codice Python del template, si ottengono errori solamente quando questo viene eseguito.

3.2.3.4. Codici di stato HTTP

Tornado è in grado di gestire i seguenti codici di stato HTTP.

404 Not Found – Tornado restituisce automaticamente un codice di risposta 404 se il path della richiesta HTTP non corrisponde a nessun pattern associato con la classe *RequestHandler*.

400 Bad Request – se il metodo *get_argument* viene chiamato senza un valore di default, e non viene trovato nessun argomento con il nome dato, Tornado ritorna automaticamente un codice di risposta 400.

405 Method Not Allowed – se una richiesta in ingresso utilizza un metodo HTTP che il corrispondente *RequestHandler* non definisce, Tornado restituisce un codice di risposta 405.

500 Internal Server Error – Tornado restituisce un codice 500 quando incontra errori non abbastanza gravi da causare l'uscita dal programma. Anche ogni eccezione non catturata, causa la restituzione di un codice di risposta 500.

200 OK – se la risposta è andata a buon fine e nessun codice di stato è stato definito, Tornado restituisce il codice di risposta 200 di default.

E' possibile ridefinire il metodo *write_error* per personalizzare il messaggio di errore che il server invia al client.

3.2.3.5. Sicurezza

Tornado è stato disegnato tenendo conto di una serie di considerazioni in merito alla sicurezza.

I *secure cookies* di Tornado utilizzano una firma crittografica per verificare che il valore di un cookie non sia stato modificato da nessun altro tranne che dal software del server. Poiché uno script maligno non conosce la chiave segreta, non può modificare un cookie senza che l'applicazione lo sappia.

Le funzioni di Tornado *set_secure_cookie()* e *get_secure_cookie()* inviano e recuperano i cookie del browser che sono protetti da modifiche maligne nel browser. Per usare queste funzioni, occorre specificare il parametro *cookie_secret* nel costruttore dell'applicazione.

Tornado codifica il valore del cookie come una stringa Base64 e aggiunge un timestamp e una firma HMAC ai contenuti del cookie. Se il timestamp del cookie è troppo vecchio (o del futuro) o se la firma non corrisponde con il valore aspettato, la funzione *get_secure_cookie()* assume che il cookie sia stato manomesse e ritorna *None*.

Per quanto riguarda invece le *vulnerabilità delle richieste*, una delle maggiori insidie da affrontare è il Cross-Site Request Forgery, abbreviato in CSRF o XSRF. Questa vulnerabilità sfrutta un bug nella sicurezza del browser che permette di iniettare codice in un sito vittima che fa richieste non autorizzate per conto di un utente loggato.

Una delle precauzioni in merito a ciò è l'uso del metodo HTTP POST. Inoltre Tornado consente di abilitare la protezione XSRF includendo il parametro *xsrf_cookies* nel costruttore dell'applicazione.

In questo modo Tornado rifiuta le richieste POST, PUT e DELETE che non contengono il corretto valore *_xsrf* come parametro della richiesta. Il token XSRF deve essere incluso nei form HTML così da autorizzare legittime richieste.

3.3. Highcharts/Highstock

Per quanto riguarda la visualizzazione dei dati provenienti dai dispositivi, la scelta è ricaduta sull'utilizzo della libreria grafica Highcharts/Highstock [6], dopo aver scartato l'utilizzo di Google Chart Tools, che sfrutta la tecnologia di Flash Player.

Di seguito vengono elencate le caratteristiche ed i vantaggi offerti da tali librerie, per capire i motivi della scelta fatta.

3.3.1. Cosa è Highcharts?

Highcharts (Javascript Charting Library) è una libreria per la realizzazione di grafici scritta in puro linguaggio JavaScript, che offre un modo semplice di aggiungere grafici interattivi ai propri siti o applicazioni web (Figura 11). Attualmente supporta i tipi di grafico a linea, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange e polar.

Figura 11 - Esempio di grafico realizzato con la libreria Highcharts

Caratteristiche di compatibilità

Funziona su tutti i moderni browser, inclusi iPhone/iPad ed Internet Explorer dalla versione 6.

Il web attualmente gira tutto intorno agli standard aperti e a tecnologie non proprietarie. La libreria Highcharts, che usa il linguaggio SVG, rientra perfettamente in tale contesto.

Purtroppo Internet Explorer 6 non supporta SVG. Dall'altro lato, però, IE6 supporta il linguaggio proprietario VML (Vector Markup Language).

Dunque Highcharts seleziona automaticamente la modalità di rendering giusta. In questo modo allontana dalle preoccupazioni dello sviluppatore tutte le questioni che riguardano i browser. Lo stesso codice Javascript creerà una bella grafica SVG o VML in base alle caratteristiche del browser. Il codice viene scritto una volta sola e funziona ovunque. Inoltre, come già detto, superando l'uso di Flash per la grafica vettoriale, possiamo distribuire il risultato su un numero maggiore di dispositivi, tra cui quelli equipaggiati con WebKit per il mobile e i dispositivi basati su iOS.

Codice aperto

Una delle caratteristiche chiave di Highcharts è che sotto certe licenze, free o meno, si è autorizzati a scaricare il codice sorgente e a fare le proprie modifiche.

L'uso della libreria è libero per siti personali o di organizzazioni no-profit. Questo permette modifiche personali ed una grande flessibilità.

Puro JavaScript

Highcharts è unicamente basato sulle native tecnologie dei browser e non richiede plug-in lato client come Flash o Java. Inoltre non c'è bisogno di installare niente sul proprio server. Niente PHP o ASP.NET. Highcharts ha bisogno solo di due file JS per girare: il cuore highcharts.js e entrambi i framework jQuery, MooTools o Prototype. Uno di questi framework è molto probabilmente già usato nella propria pagina web.

Numerosi tipi di grafico

Highcharts supporta i tipi di grafico line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange e polar. Molti di questi possono essere combinati in un grafico (Figura 12).

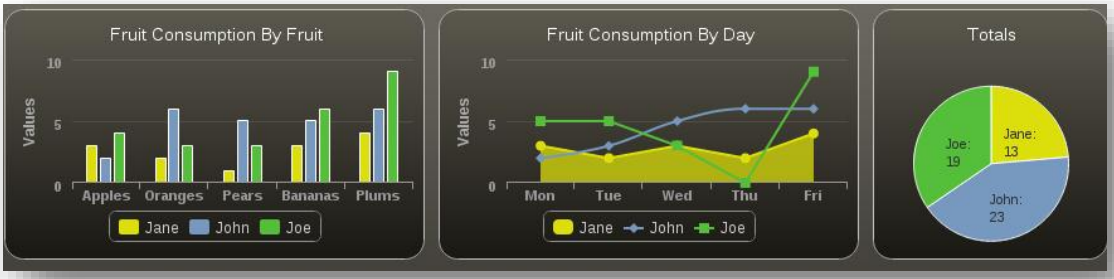


Figura 12 - Tipi di grafici creati con Highcharts

Semplice sintassi di configurazione

Impostare le opzioni di configurazione di Highcharts non richiede speciali abilità di programmazione. Le opzioni vengono date in una struttura con notazione di un oggetto JavaScript, che è di base una serie di chiavi e valori connessi da colonne, separate da virgole e raggruppate da parentesi graffe. Si veda ad esempio il codice seguente, che permette la visualizzazione di un grafico a linee in cui viene riportato l'andamento della temperatura nei vari mesi dell'anno in alcune città del mondo:

```
$(function(){

$('#container').highcharts({
chart:{
type:'line',
marginRight:130,
marginBottom:25
},
title:{
text:'Monthly Average Temperature',
x:-20//center
},
subtitle:{
text:'Source: WorldClimate.com',
x:-20
},
xAxis:{
categories:['Jan','Feb','Mar','Apr','May','Jun',
'Jul','Aug','Sep','Oct','Nov','Dec']
},
yAxis:{
```

```
title:{
text:'Temperature (°C)'
},
plotLines:[{
value:0,
width:1,
color:'#808080'
}]
},
tooltip:{
valueSuffix:'°C'
},
legend:{
layout:'vertical',
align:'right',
verticalAlign:'top',
x:-10,
y:100,
borderWidth:0
},
series:[{
name:'Tokyo',
data:[7.0,6.9,9.5,14.5,18.2,21.5,25.2,26.5,23.3,18.3,13.9,9.6]
},{
name:'New York',
data:[-0.2,0.8,5.7,11.3,17.0,22.0,24.8,24.1,20.1,14.1,8.6,2.5]
},{
name:'Berlin',
data:[-0.9,0.6,3.5,8.4,13.5,17.0,18.6,17.9,14.3,9.0,3.9,1.0]
},{
name:'London',
data:[3.9,4.2,5.7,8.5,11.9,15.2,17.0,16.6,14.2,10.3,6.6,4.8]
}]
});
});
```

Dinamicità

Attraverso l'intera API è possibile aggiungere, rimuovere e modificare serie e punti o modificare assi in ogni momento dopo la creazione di un grafico. Numerosi eventi forniscono strumenti per programmare i grafici. In combinazione con jQuery, MooTools o le API Ajax di Prototype, Highcharts consente di realizzare grafici dinamici costantemente aggiornati con valori provenienti da un server, da dati forniti dall'utente o altro.

Assi multipli

Volendo poter confrontare variabili che non sono sulla stessa scala - per esempio temperatura rispetto a precipitazioni e pressione dell'aria, Highcharts permette di assegnare un asse y per ogni serie - o un asse x se si vogliono confrontare serie di dati di categorie diverse. Ogni asse può essere posizionato a destra o a sinistra, sopra o sotto al grafico. Tutte le opzioni possono essere impostate individualmente, incluse inversione, stile e posizionamento.

Etichette per la descrizione dei comandi

Passando sopra ad un grafico Highcharts può mostrare un testo descrittivo con informazioni per ogni punto e serie. Il tooltip segue come l'utente muove il mouse sopra al grafico.

Asse Data e Ora

La libreria Highcharts è molto precisa riguardo ai valori temporali. Con l'unità dell'asse in millisecondi, Highcharts determina dove porre dei segni di riferimento, così che indichino sempre l'inizio del mese o della settimana, mezzanotte e mezzogiorno, un'ora, ecc.

Esportazione e stampa

La libreria Highcharts contiene anche un modulo precaricato per l'esportazione dei grafici visualizzati. Con tale modulo abilitato, gli utenti possono esportare il grafico nei formati PNG, JPG, PDF o SVG con un click di un bottone, oppure stampare il grafico direttamente dalla pagina web.

Zoom

Facendo l'ingrandimento di un grafico si può esaminare un parte specialmente interessante dei dati più da vicino (Figura 13). Lo zoom può essere fatto nella dimensione X o Y o entrambe.

Figura 13 - Zoom di un grafico con Highcharts

Caricamento di dati esterni

Highcharts prende I dati da un array JavaScript, che può essere definito nell'oggetto di configurazione locale, in un file separato o anche su un altro sito. Inoltre, I dati possono essere gestiti da Highcharts in qualsiasi forma e usare una funzione di callback per convertire I dati in un array.

3.3.2. Cos'è Highstock?

Le caratteristiche di Highstock (Timeline and Stock Charting) sono simili a quelle di Highcharts. Un esempio di grafico realizzato utilizzando la libreria Highstock è visibile in Figura 14.Qui di seguito vengono elencate quelle aggiuntive fornite da questa libreria grafica.

Figura 14 - Esempio di grafico realizzato con la libreria Highstock

Selettore di range

Quando si naviga in un'ampia serie di dati, peraltro con valori orari su un insieme di alcuni anni, è importante per la libreria grafica fornire un modo veloce per visualizzare un certo spazio di tempo. Highstock fornisce un selettore di range dove l'utente può zoomare in avanti su intervalli preselezionati di tempo come 1 minuto, 1 anno o il giorno precedente oppure, addirittura, aggiungere il range di date manualmente (Figura 14).

Scrollbar e Navigatori

Un altro modo più visuale di navigare la serie di dati è quello di trascinare la barra di scorrimento (Figura 14) o trascinare i bordi della finestra di navigazione fino all'area d'interesse.

Flag segnalatori di eventi

Attraverso una specifica serie di tipi chiamati "flags", si possono aggiungere segnalatori di eventi e annotazioni direttamente correlati a un'area interessata sulla serie (Figura 15). Inoltre è possibile aggiungere diversi stili di visualizzazione ai segnalatori per differenziarli tra di loro.

Figura 15 - Grafico ottenuto con Highstock in cui sono presenti i flag

3.4. CoMo

CoMo (Continuous Monitoring) è un'infrastruttura aperta per il monitoraggio di rete, progettato per chi vuole processare e condividere facilmente statistiche sul traffico di rete [7].

Gli obiettivi che CoMo si propone di raggiungere sono:

- Permettere che ogni generica metrica sul flusso di traffico in ingresso possa essere calcolata;
- Fornire garanzie di privacy e sicurezza al proprietario del collegamento monitorato, agli utenti della rete e agli utenti di CoMo;
- Essere robusto a fronte di schemi di traffico anomali.

CoMo è stato progettato per essere flessibile, scalabile e per funzionare su hardware di tipo consumer [8].

3.4.1. Architettura di alto livello

Il sistema è fatto di due principali componenti [9]. I processi *core* controllano il percorso dei dati attraverso il sistema CoMo, incluso la cattura del pacchetto, l'esportazione, il salvataggio, la gestione delle query e il controllo delle risorse. I moduli *plug-in* sono responsabili di varie trasformazioni sui dati (Figura 16).

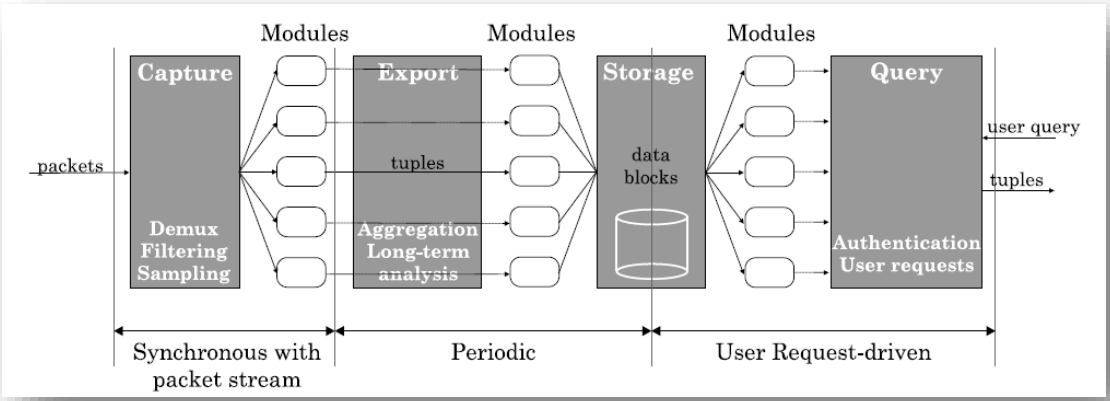


Figura 16 - Schema del server CoMo

Da una parte, CoMo raccoglie pacchetti (o sottoinsiemi di pacchetti) sul collegamento monitorato. Questi pacchetti vengono processati da una successione di processi core e memorizzati in hard disk. Dall'altra parte, i dati vengono recuperati dall'hard disk su richiesta dell'utente (attraverso richieste indirizzate al sistema CoMo). Prima di essere esportati all'utente, questi dati passano attraverso un'ulteriore fase di elaborazione.

I moduli eseguono specifici task sui dati. I processi core sono responsabili delle operazioni di "gestione", comuni a tutti i moduli (per es. cattura del pacchetto e filtraggio, salvataggio dei dati). È responsabilità del componente core:

- Gestire le risorse (per es. decidere quali moduli plug-in devono essere caricati ed avviati);
- Applicare le politiche per gestire i privilegi di accesso dei moduli;

- Gestire le richieste on-demand per schedulare e rispondere alle richieste degli utenti;
- Gestire delle eccezioni per fronteggiare situazioni di anomalie nel traffico e la possibile graduale riduzione delle prestazioni del sistema.

I moduli da un lato prendono i dati e dall'altro lato forniscono metriche di traffico definite dall'utente o dati di misura elaborati.

3.4.2. I processi core

I cinque processi principali che compongono il core di CoMo sono:

1. Il processo *capture*, responsabile di catturare pacchetti, di filtrare, campionare e mantenere le informazioni di stato per ogni modulo;
2. Il processo *export* permette analisi a lungo termine del traffico e fornisce accesso all'informazione di rete aggiuntiva (per es. tabelle di routing);
3. Il processo *storage* schedula e gestisce gli accessi ai dischi;
4. Il processo *query* riceve le richieste dell'utente, applica le query al traffico (o legge i risultati pre-calcolati) e restituisce i risultati;
5. Il processo *supervisor* è responsabile di gestire le eccezioni (per es. fallimento dei processi) e di decidere se caricare, avviare o terminare i moduli plug-in a seconda delle risorse disponibili o delle correnti politiche di accesso.

3.4.3. Moduli plug-in

Le metriche di traffico e le statistiche vengono calcolate con un set di moduli plug-in. I moduli possono essere visti come coppie *filter:function*, dove il filtro specifica i pacchetti sui quali la funzione deve essere eseguita [10].

Non tutti i moduli necessariamente calcolano statistiche. I moduli possono semplicemente trasformare il traffico in ingresso, come ad esempio trasformare un insieme di pacchetti in un record di flusso.

I processi core sono responsabili dell'esecuzione dei filtri di pacchetto e di comunicare con i moduli utilizzando un insieme di funzioni di callback.

4. Interfaccia di configurazione

In questo capitolo vengono descritti i passaggi effettuati nella realizzazione dell'interfaccia di configurazione del sistema MetroPower. L'nterfaccia ha lo scopo di consentire all'utente di comunicare con il sistema e con le reti di dispositivi. Lo sviluppo si basa sull'architettura del sistema MetroPower rappresentato in Figura 17.

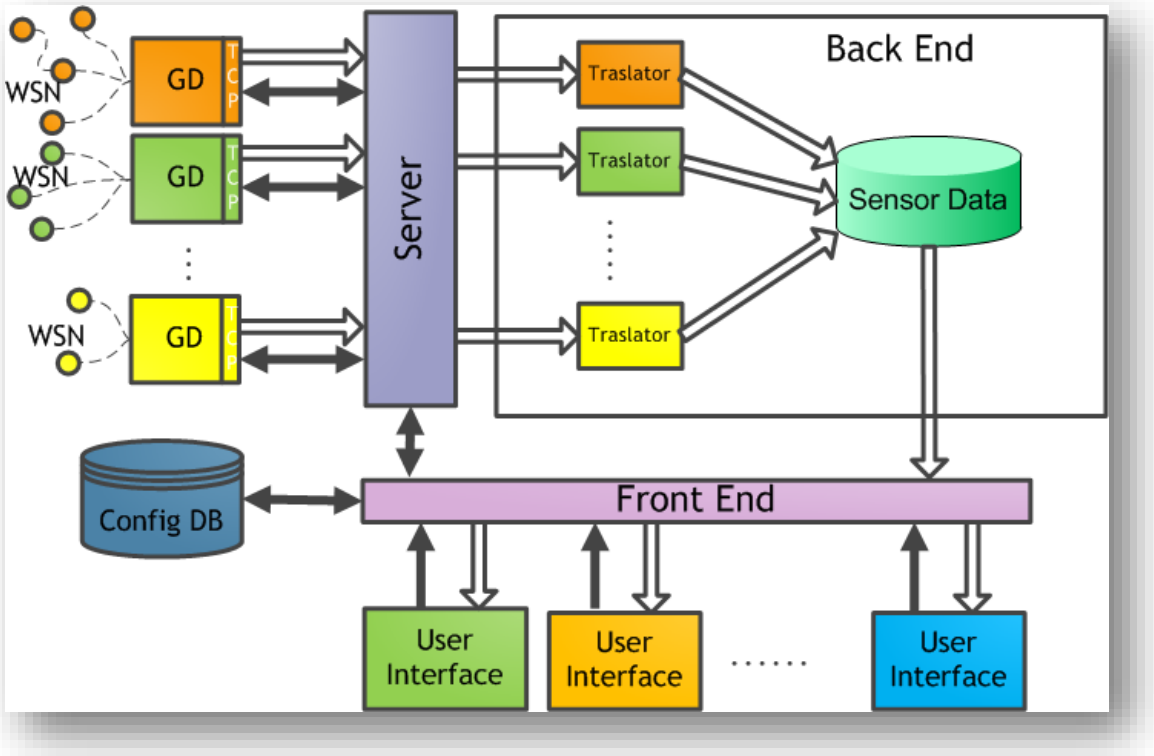


Figura 17 - Schema a blocchi dell'architettura del sistema MetroPower

Innanzitutto viene presentata l'interfaccia che si presenta all'utente che accede al sistema MetroPower. Dopo aver definito la struttura, e descritto il flusso di visualizzazione, verranno evidenziati gli aspetti implementativi e come sono stati sfruttati gli strumenti grafici nella sua realizzazione.

Viene poi descritta l'interfaccia di configurazione, ponendo in luce come siano state realizzate le strutture del database MySQL e del web server Tornado.

Infine viene presentata la parte di back-end, in particolare ponendo attenzione sulle modifiche apportate al modulo di CoMo dedicato alla gestione delle richieste fatte al sistema MetroPower da parte degli utenti.

4.1. Descrizione dell'interfaccia utente

Il sistema MetroPower si presenta all'utente come un normale sito web.

La navigazione tra le pagine segue il diagramma di flusso evidenziato in Figura 18.

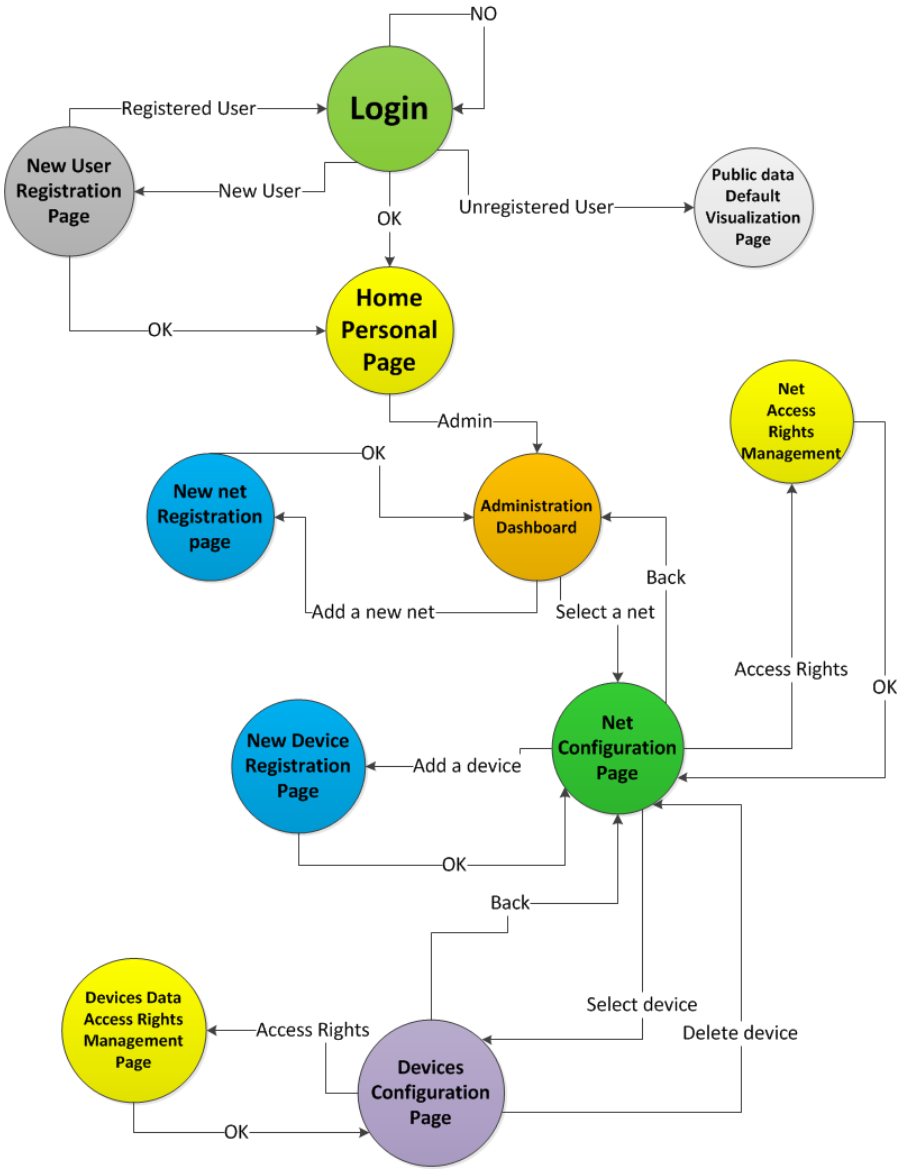


Figura 18 - Schema del flusso di visualizzazione dell'interfaccia utente

Quando un utente si collega al sistema MetroPower, viene visualizzata la homepage del progetto. La homepage e le informazioni pubbliche, quali gli autori e la descrizione del progetto, vengono visualizzate da tutti gli utenti. Registrandosi al sistema è possibile entrare nella pagina di configurazione, aggiungere, modificare o eliminare reti, aggiungere, modificare o eliminare sensori e gestire le opzioni di invio dei comandi e di accesso alle risorse.

Prima di entrare nel dettaglio della descrizione, occorre spiegare approfonditamente quali sono i profili utente che possono accedere al sistema MetroPower.

4.1.1. Profili Utente e privilegi

Il sistema supporta differenti profili utente:

- Amministratore del sistema: è unico ed è colui che ha accesso fisicamente ai server, ai database e alle reti di sensori.
- Amministratore di rete: È la persona che ha completo accesso alla rete e determina chi può accedere ai dispositivi ad essa connessi e in che modalità. I proprietari di reti, hanno di default i diritti di amministrazione delle proprie reti. Hanno cioè accesso a tutte le informazioni delle proprie reti e possono interagire con esse e gestire i diritti di accesso ai dati da parte di terzi. Il proprietario di una rete può decidere chi è autorizzato ad accedere ai dati dei dispositivi ad essa connessi. In particolare può:
    - Aggiungere o rimuovere utenti
    - Modificare i privilegi relativi ad un dato dispositivo/rete
    - Assegnare i diritti di invio comandi ad altri utenti
    - Modificare la visibilità dei propri dispositivi/reti

- Utente registrato: Persona autorizzata a leggere ed eventualmente inviare comandi ai dispositivi cui ha accesso. Gli utenti registrati hanno accesso alle reti e ai sensori per i quali sono stati eventualmente autorizzati con i privilegi definiti dall'amministratore di rete. Chiunque può registrarsi autenticandosi con il proprio account Google.
- Visitatore (utente non registrato): ha accesso alle sole informazioni di base del progetto, quali homepage, descrizione del progetto ed autori.

Privilegi di accesso

Solo gli utenti registrati possono creare e gestire reti e dispositivi.

Al momento della creazione di una propria rete, all'utente vengono automaticamente assegnati i privilegi di amministratore della stessa.

Tipi di privilegio

I principali privilegi sono:

- Nessun privilegio
- Amministratore
- Lettura
- Lettura + Invio Comandi

4.1.2. Pagine di amministrazione

Dopo aver effettuato il login gli utenti registrati hanno accesso alle pagine di ammisistrazione del sistema da cui possono gestire le retila base dei privilegi che posseggono. Per effettuare operazioni di amministrazione, come aggiungere, modificare o rimuovere utenti, è necessario aprire la relativa sezione nella pagina della propria rete o dispositivo. L'amministratore è autorizzato a modificare tutti gli utenti, mentre gli utenti senza privilegi amministrativi non hanno abilitata tale sezione relativa alla gestione degli accessi.

Pagina Principale di Amministrazione (Administration Dashboard)

In questa pagina sono presenti tutte le reti per le quali si hanno i diritti di lettura (Figura 19).

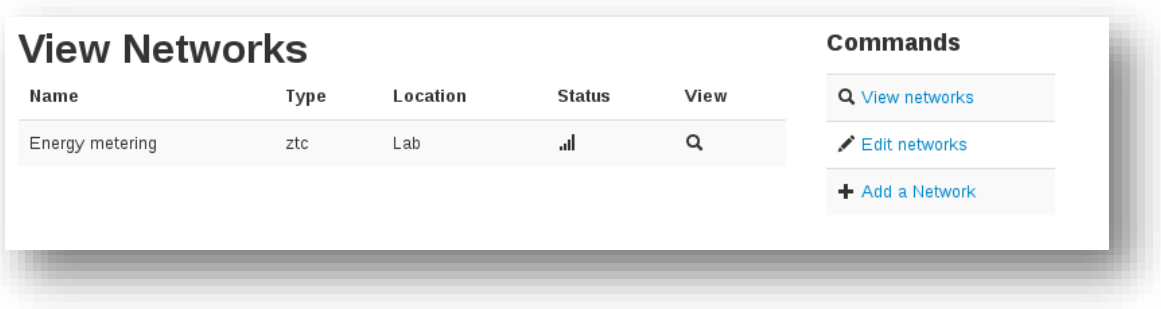


Figura 19 - Pagina principale di amministrazione

Da qui è possibile aggiungere nuove reti, o modificare/cancellare una rete esistente utilizzando i comandi a lato della pagina.

Il comando ‘edit’ consente di visualizzare e modificare le informazioni della rete. (Figura 20).

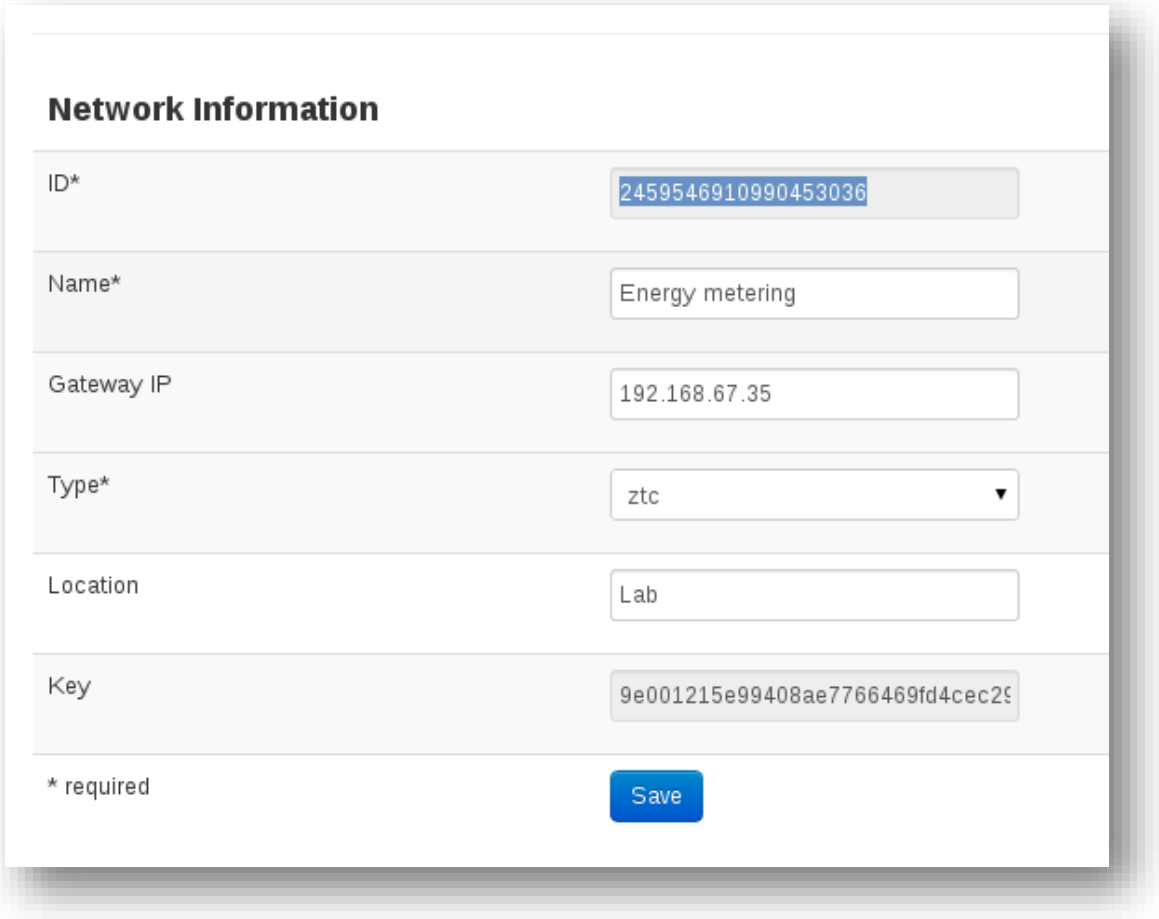


Figura 20 - Modifica di una rete esistente

Per ogni rete visibile all'utente, cliccando sul campo ‘status’, vengono visualizzate le seguenti informazioni (Figura 21):

- Nome (modificabile)
- Posizione fisica (modificabile)
- Tipo
- Indirizzo del gateway (modificabile)
- Descrizione (opzionale - modificabile)
- Id
- Chiave AES

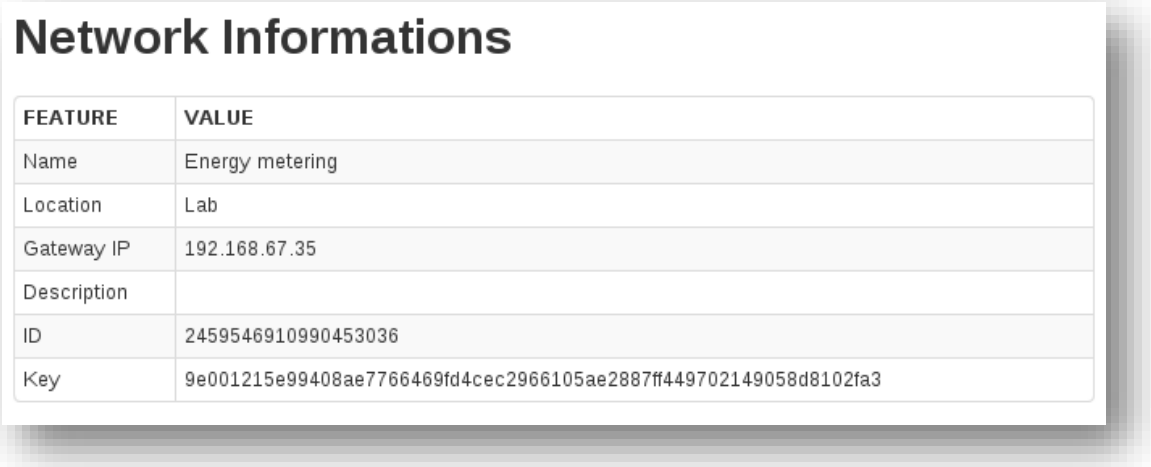


Figura 21 - Informazioni di rete

La pagina per l'aggiunta di una nuova rete si presenta come una serie di campi vuoti (Figura 22) nei quali l'utente inserisce le informazioni relative alla rete da creare:

1. Inserisce il MAC Address del gateway;
2. Definisce la tipologia di rete;
3. Assegna un nome alla rete;
4. Assegna una descrizione, una posizione, ecc.

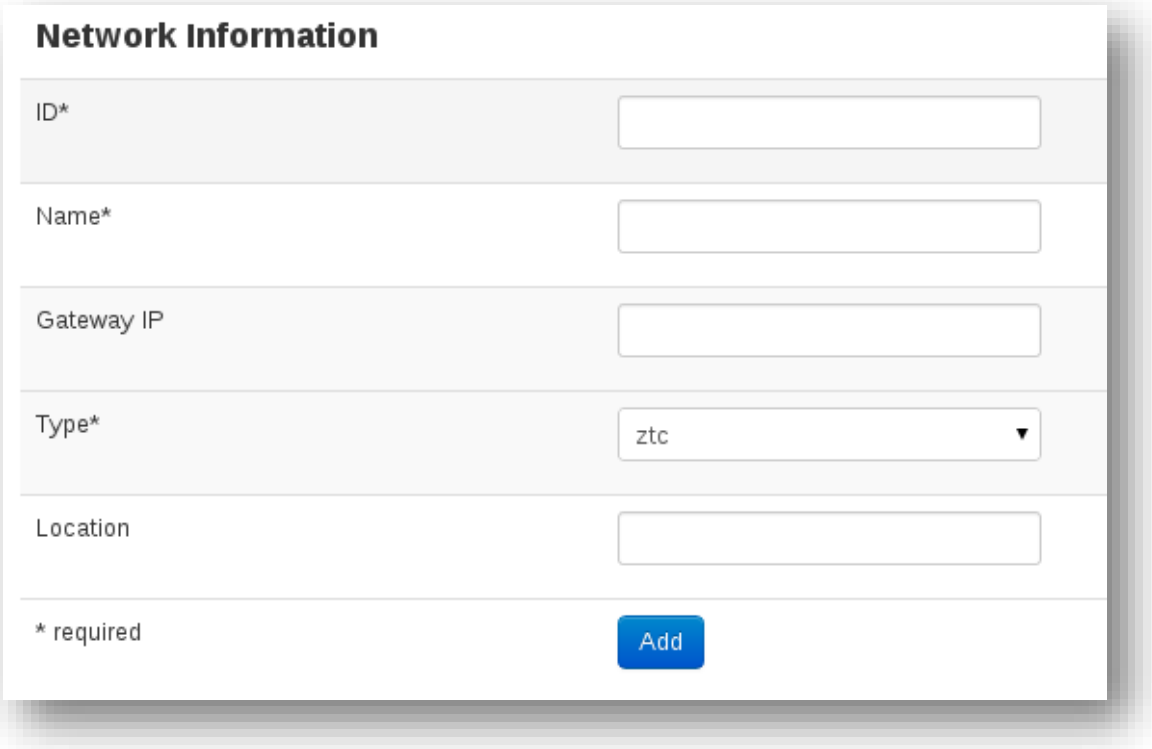


Figura 22 - Aggiunta di una rete al sistema

Alla pressione del tasto “Add” viene generata una chiave AES, che viene salvata insieme ai dati della rete nel database di Tornado.

Dalla lista delle reti, l’amministratore può selezionarne una in particolare, aprendo in questo modo la pagina di amministrazione di rete.

Pagina di Amministrazione di rete (Net Configuration Page)

In questa pagina sono presenti

- Una sezione di monitoraggio (Figura 23) in cui sono elencati i dispositivi associati alla rete di cui l'utente ha visibilità. Da qui è possibile aggiungere nuovi dispositivi, o modificare/cancellare un dispositivi esistente;

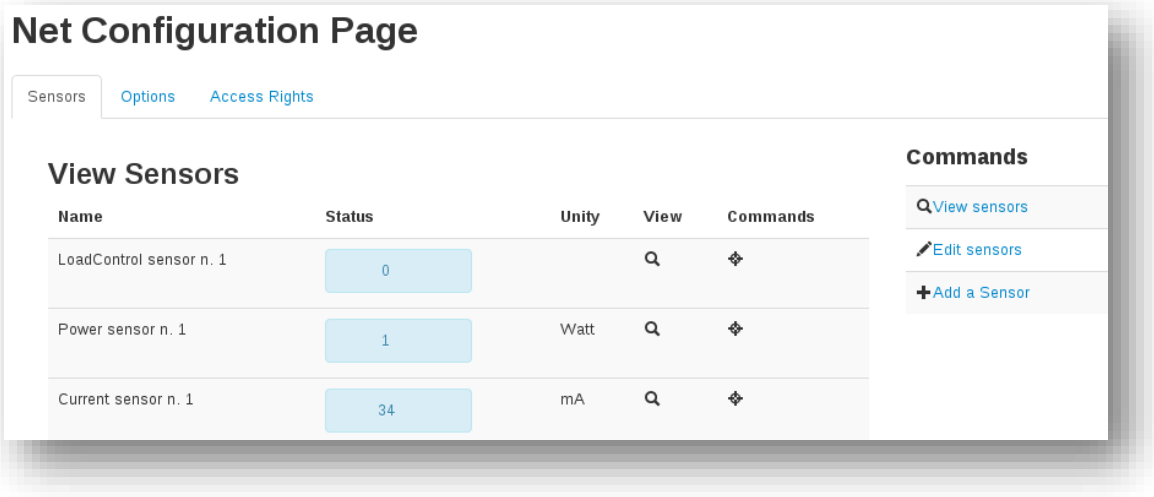


Figura 23 - Sezione di monitoraggio della pagina di amministrazione di rete

- Una sezione per l'assegnazione/rimozione/modifica dei diritti (Figura 24) alla rete (visibile solo all'amministratore);

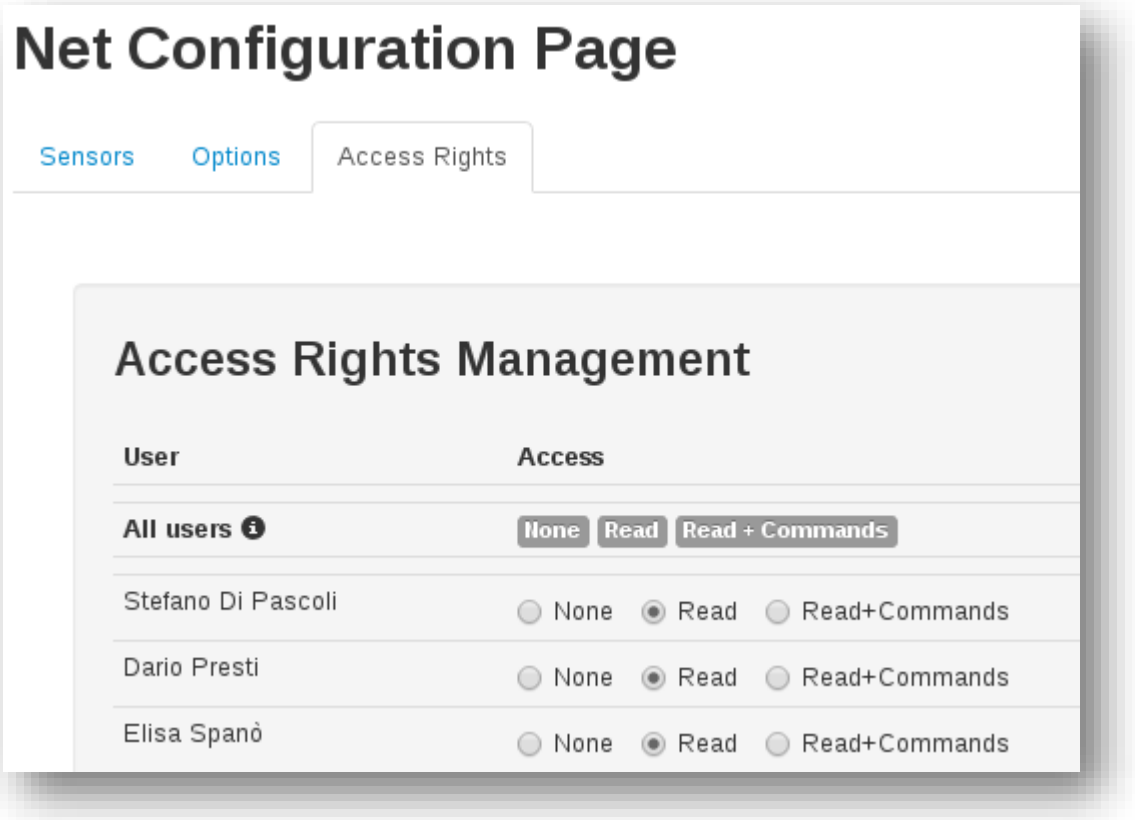


Figura 24 - Sezione per l'assegnazione dei diritti di accesso alla rete

- Una sezione per l'invio di comandi alla rete (visibile solo all'amministratore e agli utenti con permesso di lettura e invio comandi). Da qui è possibile inviare attualmente il solo comando di riavvio (restart) alla rete (Figura 25). È possibile, comunque, aggiungere altri comandi sulla falsa riga di quello già implementato.

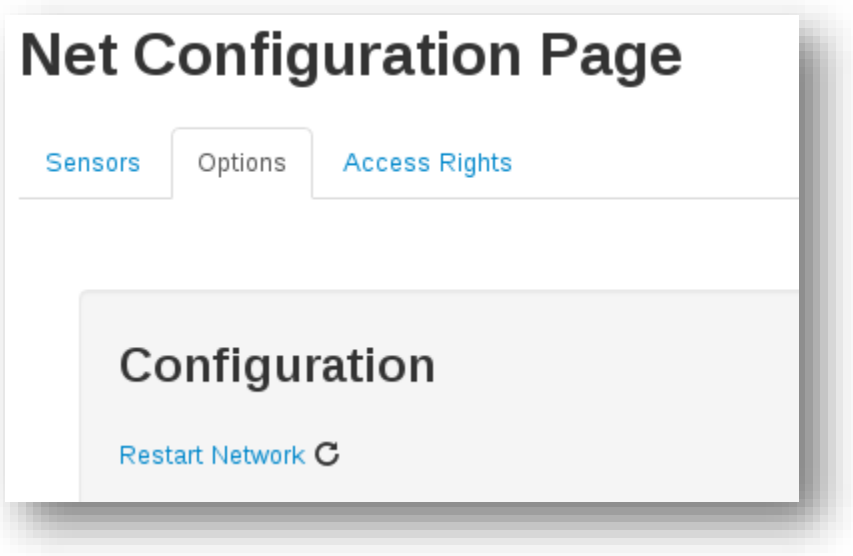


Figura 25 - Sezione per l'invio di comandi ad una rete

Pagina di configurazione di un dispositivo (Devices configuration page)

Dalla lista dei dispositivi presenti nella pagina della rete, l'amministratore può selezionarne uno e accedere alla sua pagina di configurazione.

In questa pagina sono presenti varie sezioni:

- Una sezione in cui vengono visualizzati i dati provenienti dal dispositivo (es. id, potenza del segnale, nome, descrizione, posizione, frequenza, ecc.). Questa sezione è visibile solo all'amministratore e agli utenti che hanno almeno il diritto di lettura sul dispositivo (Figura 26);

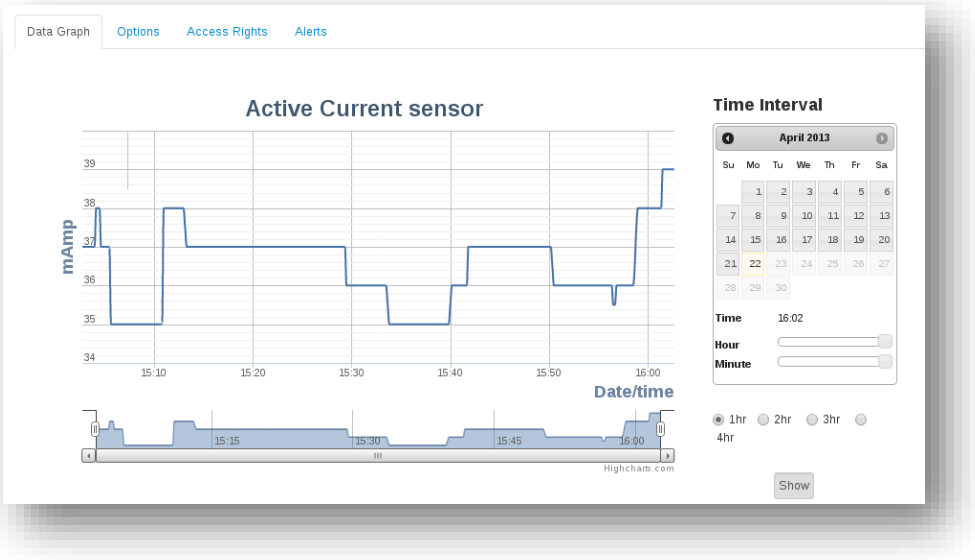


Figura 26 - Sezione per la visualizzazione dei dati provenienti dai dispositivi

- Una sezione per l'invio di comandi al dispositivo (visibile solo all'amministratore e agli utenti con permesso di lettura e invio comandi - Figura 27);

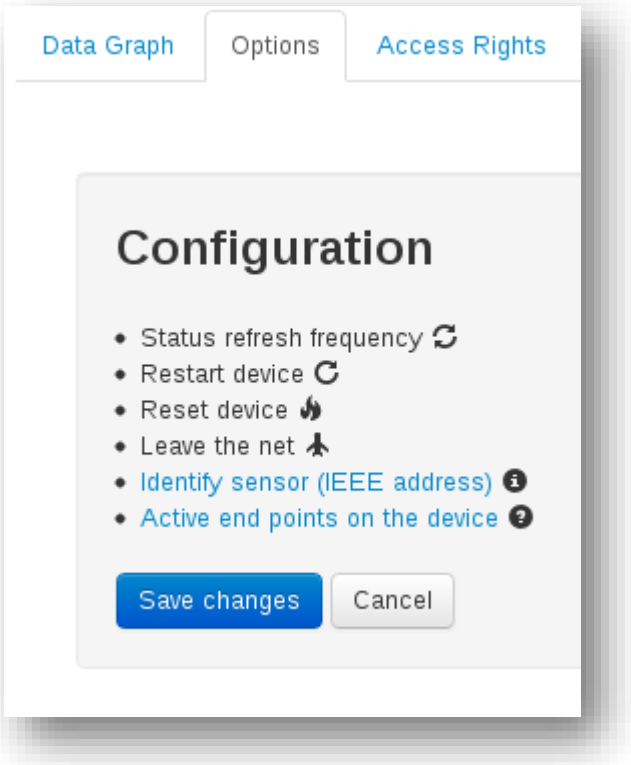


Figura 27 - Sezione per l'invio di comandi ai dispositivi

Nella implementazione attuale dalla sezione di invio di comandi al dispositivo è possibile:

  - Richiedere al dispositivo di identificarsi (IEEE address)
  - Visualizzare quanti sensori (active endpoints) sono presenti sul nodo

- Una sezione di assegnazione/rimozione/modifica dei diritti diaccesso (visibile solo all'amministratore - Figura 28).

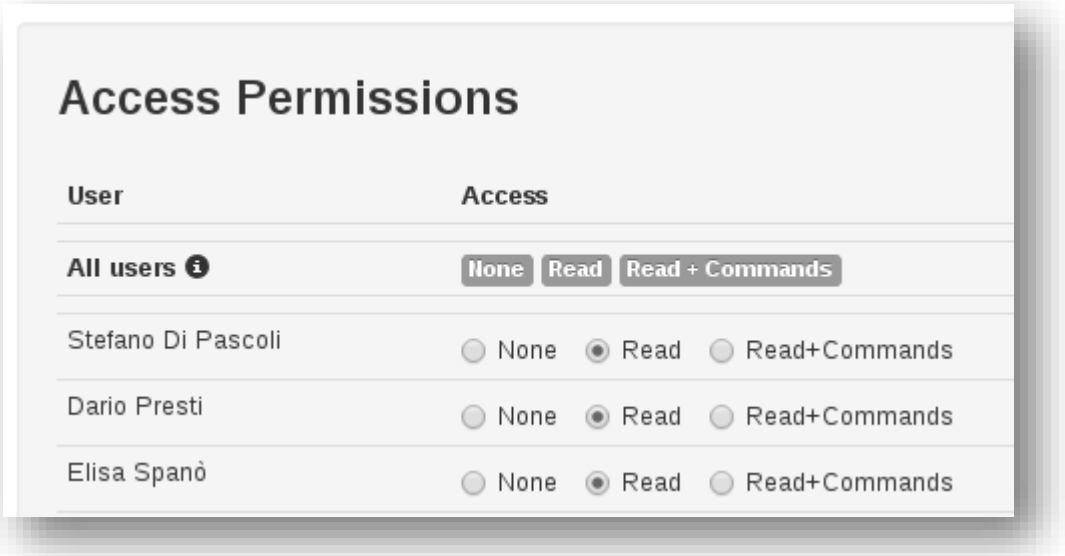


Figura 28 - Sezione per l'assegnazione dei diritti di accesso ai dispositivi

4.1.3. Assegnazione di diritti

All'interno della pagina di amministrazione della singola rete (Net configuration page) e di un singolo dispositivo (Devices configuration page) è possibile assegnare i diritti di visualizzazione ed, eventualmente, di invio comandi.

L'assegnazione dei diritti avviene in tre passaggi:

1. Individuare tra gli utenti presenti nella lista (sono visibili tutti gli utenti registrati a cui è possibile assegnare permessi) quello al quale si vuole modificare il permesso di accesso;

2. Selezionare il permesso da assegnare (è possibile assegnare simultaneamente lo stesso permesso a tutti gli utenti della lista);

3. Il sistema modifica l'account utente in modo che abbia i diritti scelti.

4.2. Implementazione dell'interfaccia di visualizzazione

Come già evidenziato precedentemente, per quanto riguarda la parte meramente grafica dell'interfaccia utente, è stato utilizzato Twitter Bootstrap.

4.2.1. Utilizzo di Twitter Bootstrap

L'utilizzo di Twitter Bootstrap permette di realizzare interfacce grafiche più gradevoli dal punto di vista dell'impatto visivo. Altri vantaggi evidenti che hanno portato alla scelta dell'utilizzo di tale framework sono stati la velocità di sviluppo consentita da Bootstrap, la presenza di un layout responsive automaticamente compatibile per mobile e tablet ed, infine, la disponibilità di componenti Javascript integrati, che riducono a zero la possibilità di incorrere in conflitti fra librerie e aumentano perciò l'affidabilità del progetto.

Di seguito vengono raccontate le diverse parti del progetto dell'interfaccia grafica utilizzanto gli elementi di BootStrap, tra i quali layout a griglia e plug-in.

Layout a griglia

Come già detto, il concetto di base di Bootstrap è quello del grid system, ovvero lavorare a "griglie" di 12 (default di Bootstrap) elementi.

Il layout responsive (Tabella 1) e quello a griglie è stato utilizzato molto per disporre gli elementi nelle pagine del progetto.

Layout Fisso	Layout responsive	Descrizione
container	container-fluid	Classe da attribuire al contenitore di tutto il progetto
row	row-fluid	Classe che identifica la riga
spanN	spanN	Classe che identifica una colonna

Tabella 1 - Tabella degli attributi del layout di Bootstrap

Nel file *base.html* (che descrive la parte comune a tutte le pagine web del progetto), vengono dichiarati elementi *class=”row-fluid”* per sfruttare il design responsive e elementi di tipo *class=”span9”* o “span3”, che sfruttano il concetto di griglia di Bootstrap.

Nella pagina *base.html* sono stati definiti i due blocchi base di ogni pagina html del sistema MetroPower.

Il primo elemento è la barra di navigazione superiore, così definita:

```
<!-- navigation bar
              ================================================= -->
<!-- Placed at the top of the page -->

<div class="navbar navbar-fixed-top">
<div class="navbar-inner">
<div class="container-fluid">
<a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-
collapse"><span class="icon-bar"></span><span class="icon-
bar"></span><span class="icon-bar"></span></a>
<a class="brand" href="/">MetroPower</a>
<div class="nav-collapse">
<ul class="nav">
<li>
<a href="/">Home</a>
</li>
<li>
<a href="/staticpage/about">About</a>
</li>
<li>
<a href="/staticpage/people">People</a>
</li>
{% if current_user %}
<li>
<a href="/networks?action=view">
                              Networks
</a>
</li>
{% end %}
</ul>
<p class="pull-right">
```

```
{% if current_user %}
<ul class="nav pull-right">
<li class="divider-vertical"></li>
<li class="dropdown">
<a href="#" class="dropdown-toggle" data-
toggle="dropdown"><iclass="icon-user icon-
white"></i>{{current_user.name}}<b class="caret"></b></a>
<ul class="dropdown-menu">
<li><a href="/auth/logout?next={{ url_escape(request.uri) }}">Sign
out</a></li>
</ul>
</li>
</ul>
<br/>
{% else %}
<button class="btn btn-inverse">
{{ _('<a href="%(url)s">Sign in</a>') % {"url": "/auth/login?next="
+ url_escape(request.uri)} }}
</button>
{% end %}
</p>
</div>
</div>
</div>
</div>
```

La barra (Figura 29) è composta nella parte sinistra dal nome del progetto MetroPower e dalle sezioni di navigazione Home, About, People e Network (quest'ultima presente solo per gli utenti registrati). Nella parte destra invece è presente il pulsante per il login/logout dell'utente.

Figura 29 - Barra di navigazione del sistema MetroPower

L’altro elemento presente in ogni pagina web del sistema MetroPower è la colonna di navigazione che si trova a sinistra e così definita:

```
<div class="row-fluid">
<div class="span3">
<div class="well sidebar-nav">
{% block navigation %}
<ul class="nav nav-list">
<li class="nav-header">
                            Navigation Menu
</li>
<li>
<a href="/"><iclass="icon-home"></i>Home</a>
</li>
{% block menu %}{% end %}
{% if 'nets' in globals() %}
<li class="active">
<a href="/networks?action=view"><i class="icon-signal"></i>
Networks</a>
</li>
{% end %}
{% if 'net' in globals() %}
<li class="active">
<a href="/networks?action=view"><i class="icon-signal"></i>
Networks</a>
</li>
{% if net.has_key('name') %}
{% if not 'sens' in globals() %}
<li>
{{ net.name }}
</li>
{% else %}
<li>
<a href="/networks/{{ net.id }}/sensors?action=view">{{ net.name
}}</a>
</li>
{% end %}
{% end %}
{% end %}
{% if 'sens' in globals() and sens.has_key('name') %}
<li>
{{ sens.name }}
</li>
{% end %}
</ul>
{% end %}
</div>
</div>
```

Il risultato è visibile nella Figura 30:

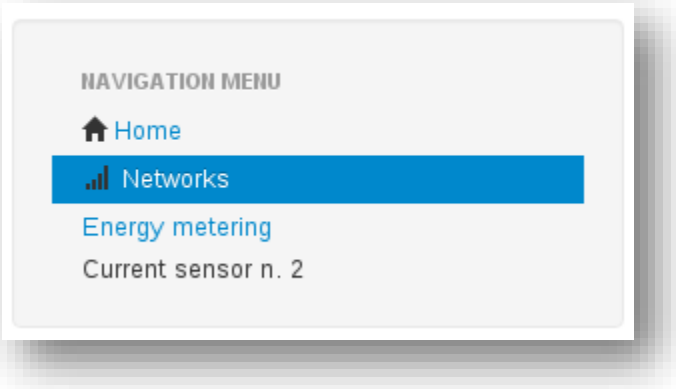


Figura 30 - Colonna di navigazione del sistema MetroPower

Nella pagina *base.html* è presente, infine, la dichiarazione del contenuto di ogni pagina, inserito in un elemento di grandezza 9 su 12, secondo il layout a griglia di Bootstrap:

```
<div id="content" class="span9">
{% block content %}
{% end %}
</div>
```

Plug-in

I plug-in Javascript con cui Bootstrap viene accompagnato, sono quelli più utilizzati sul web. Bootstrap supporta slide di immagini e contenuti testuali, popup, menu a tendina, scroll di testo, tab di contenuti, aree di messaggistica, popover, alert, bottoni, ecc.

Tutto questo senza cambiare framework, per evitare molti problemi di incompatibilità.

I plug-in possono essere inclusi individualmente o tutti insieme. Essi sono contenuti nel file Javascript *bootstrap.min.js*.

Tutti i plug-in accettano un'opzione, una stringa che indirizza un particolare metodo. Inoltre, Bootstrap fornisce eventi personalizzati per le azioni della maggior parte dei plug-in.

I plug-in utilizzati nel progetto MetroPower sono:

- **Modals**: semplici, ma flessibili, finestre di dialogo con funzionalità minime richieste e impostazioni predefinite intelligenti. Sono state

utilizzate per richiedere all'utente la conferma dell'eliminazione di una rete (Figura 31) o di un dispositivo. Ad esempio:

```
<td>
  <div class="modal hide" id="myModal_{{ net.id }}">
    <div class="modal-header">
      <a class="close" data-dismiss="modal">×</a>
      <h3>Are you sure you want delete " {{ net.name }} " ?</h3>
    </div>
    <div class="modal-footer">
      <a href="#" class="btn" data-dismiss="modal">Close</a>
      <a href="/networks/delete?id={{ net.id }}" class="btn btn-primary">Save changes</a>
    </div>
  </div>

  <a data-toggle="modal" href="#myModal_{{ net.id }}"><i class="icon-trash"></i></a>
  </div>
</td>
```

In questo caso il plug-in viene attivato senza ricorrere al Javascript, ponendo l'attributo *data-toggle="modal"* su un elemento icona.

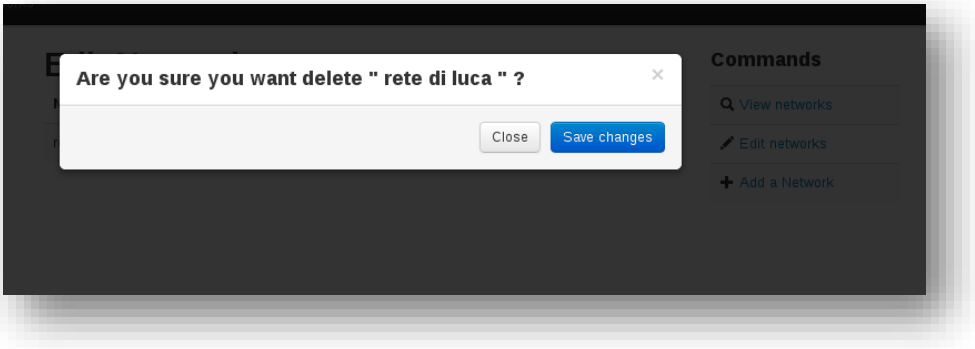


Figura 31 - Modal Plug-in di Twitter Bootstrap

- **Dropdowns**: aggiunge un menu a discesa. È stato utilizzato sul bottone di login/logout dell'utente presente sulla barra superiore delle pagine web del progetto (Figura 32). Ad esempio:

```
<li class="dropdown">
  <a href="#" class="dropdown-toggle" data-toggle="dropdown"><i class="icon-user
icon-white"></i> {{current_user.name}} <b class="caret"></b></a>
  <ul class="dropdown-menu">
    <li><a href="/auth/logout?next={{ url_escape(request.uri) }}">Sign out</a></li>
  </ul>
</li>
```

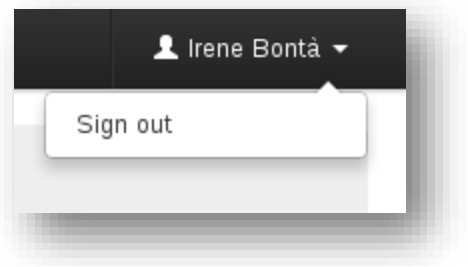


Figura 32 - Dropdown Plugin di Twitter Bootstrap

- **Togglable tabs**: aggiunge una funzionalità di scheda veloce e dinamica per la transizione attraverso etichette. Il plug-in è stato utilizzato per creare le schede nelle pagine di gestione di reti e dispositivi, al fine di snellire la navigazione (Figura 33). Ad esempio:

```
<div class="tabbable"><!-- Only required for left/right tabs -->
  <ul class="nav nav-tabs">
    <li class="active"><a href="#tab1" data-toggle="tab">Sensors</a></li>
    {% if writeable %}
```

```
    <li><a href="#tab2" data-toggle="tab">Options</a></li>
    <li><a href="#tab3" data-toggle="tab">Access Rights</a></li>
    {% end %}
  </ul>
```

Per far sì, inoltre, che fosse possibile indirizzare una particolare scheda specificandola attraverso il proprio hash tag nella URL della pagina, è stata aggiunta questa funzione Javascript:

```
$(function () {
   var activeTab = $('[href=' + location.hash + ']');
activeTab && activeTab.tab('show');
 });
```

Questo è stato necessario in quanto la versione corrente di Bootstrap non supporta tale funzionalità.

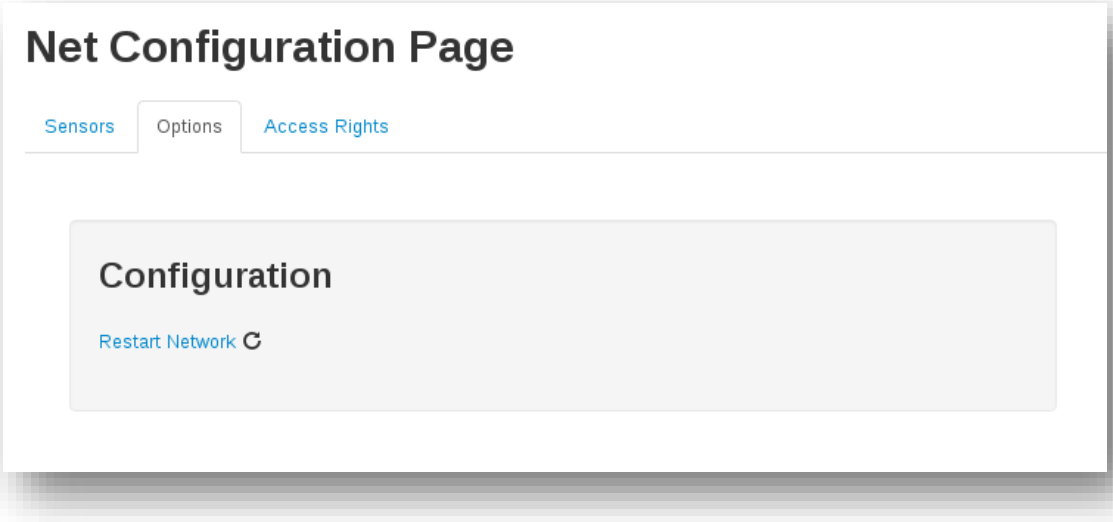


Figura 33 - Togglable Tabs Plugin di Twitter Bootstrap

- **Tooltips**: ispirati dal plug-in jQuery.tipsy scritto da Jason Frame, sono una versione aggiornata che non si basa su immagini e non utilizza il CSS3 per le animazioni. Per ragioni di prestazioni, le API per i tooltip sono opzionali, perciò occorre inizializzarli da soli. Sono stati utilizzati per visualizzare aiuti inline nelle pagine web (Figura 34). Ad esempio:

```
<div>All users<i class="icon-info-sign" rel="tooltip" id="someid" data-original-title="Set the same permission to all users on the same sensor"></i></div>

<script>
$(document).ready(function(){
   $("[rel=tooltip]").tooltip({ placement: 'top'});
});
</script>
```

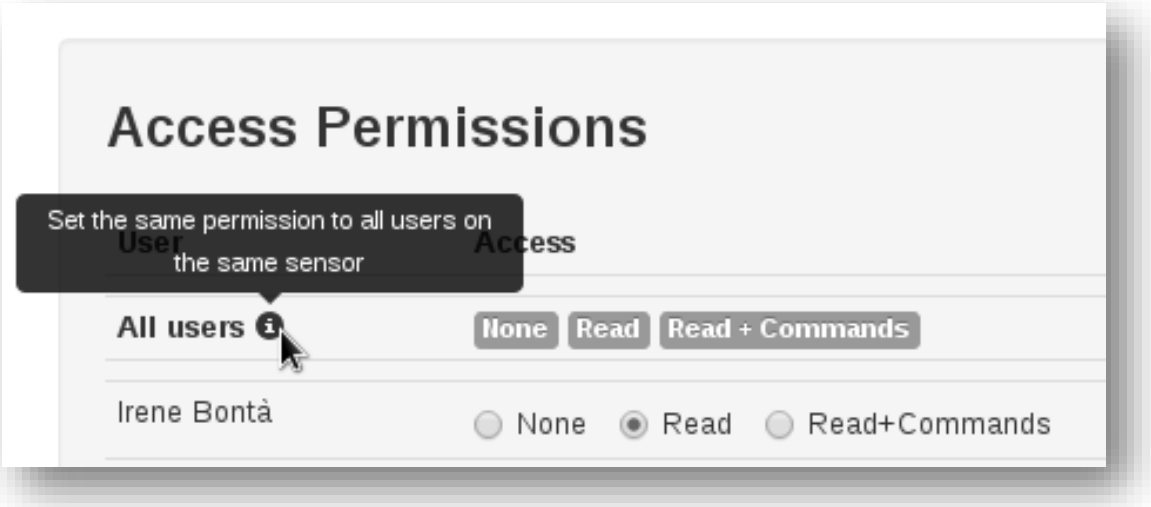


Figura 34 - Tooltip Plugin di Twitter Bootstrap

4.2.2. Utilizzo di Highcharts/Highstock

Per quanto riguarda la visualizzazione dei dati provenienti dai dispositivi e la costruzione dei relativi grafici, sono state utilizzate le API di Highcharts/Highstock [11]. Come già detto, il maggior vantaggio derivante dall'utilizzo di queste librerie, è quello dell'essere interamente scritte in Javascript, per cui non occorre installare particolari plug-in sui browser per visualizzare i grafici.

È stato fatto l'uso, in particolare, della libreria Highstock (Figura 35) nella pagina web *sensordata.html*. La scelta è ricaduta su tale libreria perché permette di aggiungere elementi quali scrollbar e finestre di navigazione, per consultare più velocemente i dati dei dispositivi.

Tramite una richiesta AJAX al server (effettuata dalla funzione Javascript *como_get*), la funzione Javascript che si occupa di generare il grafico (funzione *visualize*) ottiene una serie di dati in formato JSON.

```
como_get("{{ net.id }}","{{ sens.id
}}",utc_start,utc_end,"absolute",visualize);


// Wrapper function to the AJAX request that get Sensor Data
function como_get(net,sens,start,end,mode,callback){
```

```
$.ajax({url:"/networks/"+net+"/sensors/"+sens+"/data?start="+start+"
&end="+end+"&mode="+mode,
type:"GET",
async:true,
dataType:"json",
success:callback
});
}
```

Highstock è in grado di elaborare automaticamente i dati JSON provenienti dal server e questo è un grande vantaggio, in quanto non c'è bisogno di elaborare manualmente i dati.

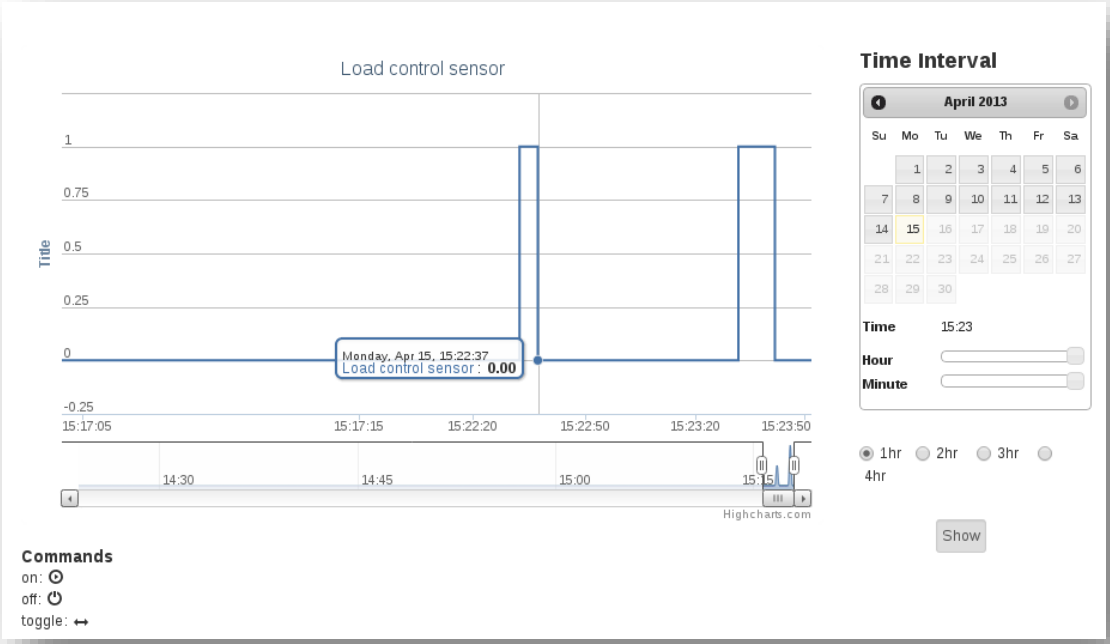


Figura 35 - Visualizzazione dei dati con Highstock

4.3. Implementazione dell'interfaccia di configurazione

Il front-end del sistema MetroPower è realizzato con il server web Tornado e un database MySQL.

4.3.1. Struttura del Database MySQL

Il database MySQL contiene 6 tabelle (vedi Figura 36):

- NETWORKS
- DEVICES
- USERS
- NETS_PERMISSIONS
- DEVICES_PERMISSIONS
- CONFCOMMANDS

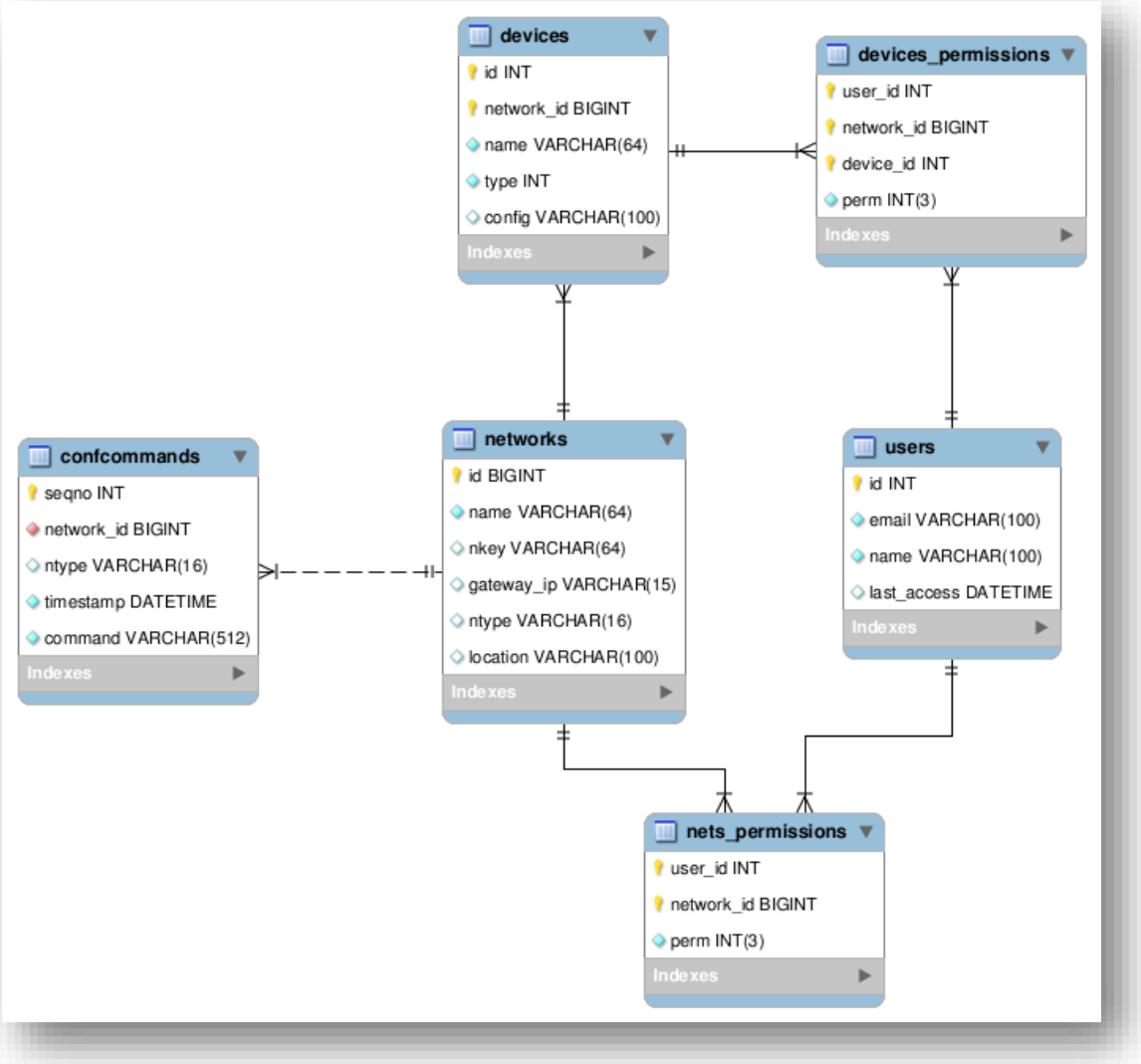


Figura 36 - Schema del database MySQL

Tabella NETWORKS

Contiene l'elenco delle reti registrate. Ciascun item è identificato tramite i seguenti campi:

- Id della rete;
- Nome;
- Chiave AES;
- Ip del gateway;
- Tipo della rete;
- Posizione fisica della rete.

Tabella DEVICES

La tabella devices raccoglie tutti i dispositivi registrati indipendentemente dalla rete cui sono connessi ma tiene traccia della rete di appartenenza.

Ciascun item della tabella è identificato tramite i seguenti campi:

- Id del dispositivo;
- Id della rete di appartenenza;

- Nome;
- Tipo;
- Opzioni di configurazione.

Tabella USERS

La tabella users contiene le informazioni degli utenti registrati al sistema.

Ha una riga per ciascun utente, di cui vengono memorizzati:

- Id;
- Email;
- Nome;
- Timestamp dell'ultimo accesso al sistema.

Tabella NETS_PERMISSIONS

La tabella nets_permissions contiene le informazioni riguardanti i permessi degli utenti sulle reti. Contiene tuple composte da:

- Id dell'utente;
- Id della rete;
- Permesso (nessun permesso, lettura, lettura + invio comandi, amministratore).

Tabella DEVICES_PERMISSIONS

La tabella devices_permissions contiene le informazioni riguardanti i permessi degli utenti sui sensori. Contiene tuple composte da:

- Id dell'utente;
- Id della rete;
- Id del dispositivo;
- Permesso (nessun permesso, lettura, lettura + invio comandi).

Tabella CONFCOMMANDS

La tabella confcommands contiene i comandi che gli utenti inviano alle reti, ai nodi o ai sensori. Le tuple sono identificate dai seguenti campi:

- Numero sequenziale;
- Id della rete a cui viene inviato il comando;
- Tipo della rete;
- Timestamp di invio del comando;
- Comando.

Visibilità

L'amministratore di una rete può impostare i diritti di accesso di altri utenti sulle proprie reti e sui propri dispositivi. Le reti ed i sensori vengono inizialmente creati con visibilità privata al solo amministratore. Successivamente l'amministratore può rendere ciascuna rete e/o dispositivo:

- Pubblico: visibile a tutti gli utenti registrati;
- Pubblico con possibilità di invio comandi;
- Ristretto: visibile solo ad un certo numero di utenti espressamente specificati dall'amministratore.

4.3.2. Struttura del web server

La progettazione del webserver basato su Tornado, richiede la scrittura di un insieme di “handler” di risposta a richieste HTTP.

La maggior parte del lavoro nel progettare l’applicazione di Tornado è stata quella di definire classi che estendono la classe di Tornado *RequestHandler*.

Struttura del web server

In cima al programma, sono state importate varie librerie di Tornado. Le librerie importate aggiungono al sistema MetroPower gli strumenti necessari per gestire l’autenticazione, il dialogo con il database, la connessione al server e l’arrivo di richieste HTTP asincrone dal client, ecc. Di seguito è possibile vedere quali moduli di Tornado sono stati inclusi nell’applicazione MetroPower.

```
# import tornado packages
import tornado.auth
import tornado.database
import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web
import tornado.gen
from tornado.httpclient import AsyncHTTPClient

# import standard packages
import unicodedata
import os
import datetime# for timestamps
import random# for network key generation
import time# for time.time()
```

La libreria *tornado.options* è stata importata per far leggere a Tornado le opzioni specificate per la nostra applicazione. Tramite tale libreria, per il sistema MetroPower, sono stati specificati l'indirizzo del web server e la porta sulla quale l'applicazione ascolta le richieste HTTP, il nome e la password del database MySQL e l'indirizzo del server CoMo e la relativa porta di ascolto per le query provenienti dal lato client.

Ogni opzione, specificata in uno statement *define* in questo modo, è disponibile come attributo dell'oggetto globale *options*.

```
# Define Global Configuration
# DB connection
define("port",default=8000,help="run on the given
port",type=int)
define("mysql_host",default="127.0.0.1:3306",help="WSN
database host")
define("mysql_database",default="wsn",help="WSN database
name")
define("mysql_user",default="wsnadmin",help="WSN database
user")
define("mysql_password",default="baffone",help="WSN database
password")
# Other
define("cookie_alerts",default="alerts",help="Cookie used for
client-side session alerts")
# Remote CoMo instance
define("como_address",default="131.114.52.207",help="IP
address of the machine that runs the CoMo instance")
define("como_port",default="44444",help="TCP port number for
the CoMo query interface")
```

Quando gestisce una richiesta, Tornado istanzia la relativa classe e chiama il metodo corrispondente al metodo HTTP della richiesta. La classe di Tornado *RequestHandler* ha un certo numero di metodi pre-costruiti, incluso il metodo *get_argument*, che nel sistema MetroPower è stato utilizzato per recuperare un argomento dalla stringa della query. Un altro metodo della classe *RequestHandler* utilizzato negli handler del nostro sistema è il metodo *write*, che prende una stringa come parametro e la scrive nella risposta.

Le righe che permettono all'applicazione di Tornado di lanciare l'esecuzione del sistema MetroPower sono le seguenti:

```
def main():
tornado.options.parse_command_line()
http_server=tornado.httpserver.HTTPServer(Application())
http_server.listen(options.port)
tornado.ioloop.IOLoop.instance().start()

if __name__=="__main__":
main()
```

Per prima cosa, si utilizza la libreria *options* di Tornado per fare il parsing della riga di comando e recuperare le varie opzioni del sistema MetroPower.

```
tornado.options.parse_command_line()
```

Poi viene creata un'istanza della classe *Application* di Tornado:

```
class Application(tornado.web.Application):
def __init(self):
# define Nodes for the Application graph
handlers=[
(r"/*",HomeHandler),
#(r"/auth/login", AuthLoginHandler),
(r"/auth/login",FakeAuthLoginHandler),
(r"/auth/logout",AuthLogoutHandler),
(r"/networks/*",NetworksHandler),
(r"/staticpage/([a-z]+)/*",StaticPageHandler),
(r"/networks/edit",NetworkEditHandler),
(r"/networks/delete",NetworkDeleteHandler),
(r"/networks/([0-9]+)/status/*",NetworkStatusHandler),
(r"/networks/([0-9]+)/config/*",NetworkConfHandler),
(r"/networks/([0-9]+)/sensors/*", NetworkSensorsHandler),
(r"/networks/([0-9]+)/sensors/([0-9]+)/*",SensorHandler),
(r"/networks/([0-9]+)/send/*",NetworkSendHandler),
(r"/networks/([0-9]+)/sensors/([0-
9]+)/send/*",SensorSendHandler),
(r"/networks/([0-9]+)/sensors/edit/*",SensorEditHandler),
(r"/networks/([0-9]+)/sensors/delete/*",SensorDeleteHandler),
(r"/networks/([0-9]+)/sensors/([0-
9]+)/data/*",SensorDataHandler),
(r"/networks/([0-9]+)/unreg",NetworkUnregSensorsHandler),
(r"/networks/types",NetworksTypesHandler),
(r"/networks/([0-9]+)/rights",NetworkRightsHandler),
(r"/networks/([0-9]+)/sensors/([0-
9]+)/rights",SensorRightsHandler),
(r"/networks/([0-9]+)/all*",NetworkAllRightsHandler),
(r"/networks/([0-9]+)/sensors/([0-
9]+)/all*",SensorAllRightsHandler)
]
```

In tale classe, come si vede dal codice, sono stati definiti i vari path che le richieste HTTP provenienti dal lato utente dovranno corrispondere al fine di lanciare il corretto handler per l'elaborazione delle varie pagine web dell'applicazione. Le classi usate per gestire tali richieste sono state

elencate nell'argomento *handlers* della classe *Application* e passate in questo modo al metodo *__init__*.

Il parametro *handlers* è una lista di tuple, ognuna contenente come primo elemento un'espressione regolare da confrontare e come secondo elemento una classe *RequestHandler*.

Tornado utilizza le espressioni regolari nelle tuple per abbinare il path della richiesta HTTP. Quando una parte dell'espressione regolare è racchiusa tra parentesi, i contenuti corrispondenti di quel gruppo vengono passati all'oggetto *RequestHandler* come parametri del metodo corrispondente della richiesta HTTP.

Per esempio, nel sistema MetroPower, quando una richiesta HTTP corrisponde alla tupla

```
(r"/networks/([0-9]+)/sensors/([0-9]+)/*",SensorHandler),
```

L'id della rete e l'id del sensore vengono passati come parametri del metodo get() al corrispondente *SensorHandler*:

```
class SensorHandler(BaseHandler):
# Requires authentication
@tornado.web.authenticated
def get(self,nid,sid):
```

Una volta creata l'istanza della classe *Application*, tale oggetto viene passato all'oggetto *HTTPServer* di Tornado, che quindi si mette in ascolto sulla porta specificata nelle opzioni dell'applicazione (recuperata attraverso l'oggetto *options*).

```
http_server=tornado.httpserver.HTTPServer(Application())
http_server.listen(options.port)
```

Infine, viene creata un'istanza di *IOLoop* di Tornado, dopodiché il programma è pronto ad accettare richieste HTTP.

```
tornado.ioloop.IOLoop.instance().start()
```

Come già precedentemente visto, Tornado è in grado di gestire diversi codici di stato HTTP. Tra i codici di stato sono presenti 4 codici di errore (404, 400, 405, 500).

Quando si verifica uno di questi errori, Tornado di default invia un breve frammento di HTML al client con il codice di stato e informazioni riguardo all'errore.

Nell'applicazione Tornado del sistema MetroPower è stato ridefinito il metodo *write_error*, in modo da rimpiazzare la risposta predefinita con una risposta personalizzata al client. La funzione è stata scritta in questo modo:

```
def write_error(self, status_code, **kwargs):
error_trace=None
if self.settings.get("debug") and "exc_info" in kwargs:
import traceback
error_trace=""
for line in traceback.format_exception(*kwargs["exc_info"]):
error_trace+=line
self.render("error.html", status_code=status_code,
error_trace=error_trace)
     self._finished=True
```

La funzione ritorna in questo modo la pagina di errore personalizzata *error.html*, nella quale sono state inserite le informazioni su come si è generato l'errore (*error_trace*), utili soprattutto in fase di debug.

Template

Tornado contiene un linguaggio a template veloce e flessibile nel modulo *tornado.template*.

Nel sistema MetroPower è stato ampiamente sfruttato l'utilizzo di template. Innanzitutto, come si vede nel codice qua sotto

```
template_path=os.path.join(os.path.dirname(__file__),"templates"),
```

è stato passato il parametro *template_path* al metodo *__init__* dell'oggetto *Application*. Tale parametro descrive a Tornado dove cercare i file template. Nel caso del sistema MetroPower i file relativi si trovano nella directory "templates". Una volta dichiarato a Tornado dove trovare i template, negli handler è stato utilizzato il metodo *render* della classe *RequestHandler* per dire a Tornado di leggere in un file template, interpolare il codice trovato in esso e inviare il risultato al browser.

Per esempio, il *NetworkStatusHandler*, ovvero la classe che gestisce la pagina di stato della rete, utilizza tale metodo per inviare al browser le informazioni della rete:

```
# Render the networks page
self.render("networkstatus.html",net=net)
```

Per quanto riguarda, invece, l'utilizzo di contenuti statici, la dichiarazione del path è simile a quella usata per i template. Nel sistema MetroPower è stata definita questa impostazione di configurazione di Tornado:

```
static_path=os.path.join(os.path.dirname(__file__),"static"),
```

In questo modo si è detto a Tornado di cercare i file statici nella directory "static".

Il modulo template di Tornado fornisce una funzione chiamata *static_url* per generare URL a file trovati in tale directory. Tale funzione è stata utilizzata nel sistema MetroPower soprattutto per includere librerie Javascript o fogli di stile alle pagine HTML. Per esempio, in questo modo

```
<script type='text/javascript' src="{{ static_url("js/highstock.js") }}"></script>
```

è stata inclusa la libreria di Highstock nella pagina di visualizzazione del grafico dei dati dei sensori. L'URL tradotta sarà del tipo

```
/static/js/highstock.js?v=95ae1
```

Si è scelto di usare il metodo fornito da Tornado invece che semplicemente riportare il path nei template per vari motivi.

Innanzitutto, la funzione *static_url* crea una stringa hash basata sul contenuto del file e la scrive in append alla fine dell'URL. Nell'esempio appena visto, tale stringa corrisponde all'argomento *v*=95ae1 dell'URL generata.

La stringa assicura che il browser carichi sempre l'ultima versione di un file invece di affidarsi a una precedente versione presente nella cache. Questo è utile sia nello sviluppo e nella distribuzione dell'applicazione MetroPower, in modo che gli utenti non debbano pulire la propria cache del browser per vedere cambiamenti nei contenuti statici.

Altro vantaggio ritenuto molto importante per un progetto in divenire come il sistema MetroPower è quello di poter potenzialmente in futuro modificare la struttura delle URL dell'applicazione, senza dover però modificare il codice nei template.

Database

Tornado contiene un modulo precostruito per la gestione di un database MySQL, il modulo *tornado.database.* La funzionalità principale che tale modulo fornisce è quella di fare da "wrapper" alla connessione MySQLdb DB-API, così da consentire un semplice accesso al database, semplicemente indirizzando le colonne delle tabelle per nome.

Nel costruttore dell'applicazione MetroPower sono state inserite le informazioni necessarie a creare una connessione al nostro database MySQL. Come si vede dal seguente codice

```
# Open DB connection. Shared among all handlers.
self.db=tornado.database.Connection(
host=options.mysql_host,database=options.mysql_database,
user=options.mysql_user, password=options.mysql_password)
```

è stato creato l'oggetto *Connection*, che consente così di accedere al database al quale il server si connette, conoscendo appropriatamente server, nome del database, username e password per accedere ad esso. Tali informazioni sono state specificate, come visto in precedenza, tra le opzioni di configurazione dell'applicazione MetroPower.

Inoltre, l'oggetto *db* della classe *tornado.database.Connection* è stato utilizzato in questo modo per interrogare le tabelle del database MySQL.

I metodi usati nell'applicazione MetroPower sono:

- *query(query, *parameters)* – restituisce una lista per la query data. Per esempio:

```
# Retrieve the networks on which the user has at least read
privileges
nwks=self.db.query("SELECT * FROM networks as n JOIN
nets_permissions as p \on n.id = p.network_id \
WHERE p.user_id=%s AND p.perm>0", int(usr_id))
```

  in questa parte di codice, presa dal *NetworksHandler*, vengono recuperate le reti visibili dall'utente. Il risultato della query sarà una lista di reti, per questo è stato utilizzato questo metodo.

- *get(query, *parameters)* – restituisce la prima riga restituita dalla query data. Per esempio:

```
usr=self.db.get("SELECT network_id, user_id FROM
nets_permissions \WHERE user_id=%s AND network_id=%s", usrid,
netid)
```

in questa parte di codice, presa dal *NetworkRightsHandler*, viene recuperato il permesso dell'utente sulla rete selezionata. In questo caso, il risultato della query sarà un unico valore intero, per cui è stato utilizzato questo metodo, piuttosto che il metodo precedente *db.query*.

- *execute(query, *parameters)* – esegue la query data. Per esempio:

```
self.db.execute("DELETE FROM devices_permissions \
WHERE user_id=%s AND device_id=%s", usrid, sensid)
```

in questa parte di codice, presa dal *SensorRightsHandler,* viene eseguita una query di DELETE, che elimina dalla tabella dei permessi sui dispositivi un determinato utente. In questa parte di codice,

```
self.db.execute("UPDATE devices_permissions SET perm=%s \
WHERE user_id=%s AND device_id=%s", perm, usrid, sensid)
```

invece, viene eseguita una query di UPDATE, che aggiorna il permesso di un determinato utente su un determinato dispositivo.

Richieste asincrone

Nel sistema MetroPower numerose sono le richieste asincrone da gestire. A tal scopo è stato innanzitutto importato, come già visto, il modulo *tornado.httpclient*, in particolare la classe *AsyncHTTPClient*. Questo ha consentito di riscrivere molte volte il comportamento predefinito di Tornado, ovvero quello di chiudere automaticamente le richieste HTTP al termine dei request handler.

Per esempio, nel codice sottostante è stato utilizzato il decoratore *@tornado.web.asynchronous* per far sì che la connessione con il client rimanga aperta nonostante la funzione che gestisce la richiesta ritorni.

```
@tornado.web.asynchronous
@tornado.gen.engine

...

como_url="".join(['http://',options.como_address,':',options.c
omo_port, '/ztc_config?netid=0&opcode_group=0&opcode=0&start=-
20s&end=-1s'])

http_client=AsyncHTTPClient()

response=yield tornado.gen.Task(http_client.fetch,como_url)

ret={}
if response.error:
ret['error']='Error while retrieving the response'
self.write(tornado.escape.json_encode(ret))
self.finish()
else:
    ...
```

Poiché però in questo caso Tornado non chiuderà mai la connessione, è stato esplicitamente detto a Tornado di chiudere la richiesta chiamando il metodo *self.finish* dell'oggetto *RequestHandler*.

L'altro decoratore presente sul metodo, visibile nell'esempio appena riportato, è *@tornado.gen.engine.* Questo decoratore è un'interfaccia basata su generatori che rende più semplice lavorare in un ambiente asincrono. La programmazione mediante tale modulo è tecnicamente asincrona, ma consente la scrittura di un singolo generatore invece che di una serie di funzioni separate.

Sicurezza

Utilizzando le funzioni di Tornado *set_secure_cookie()* e *get_secure_cookie()*si inviano e si recuperano i cookie del browser che l'utente utilizza per interfacciarsi al sistema. Per usare queste funzioni, è stato specificato il parametro *cookie_secret* nel costruttore dell'applicazione in questo modo:

```
cookie_secret="11oETzKXQAGaYdkL5gEmGeJJFuYh7EQnp2XdTP1o/Vo=",
```

Per quanto riguarda invece il *Cross-Site Request Forgery*, nel sistema MetroPower è stata abilitata la protezione XSRF includendo il parametro *xsrf_cookies* nel costruttore dell'applicazione in questo modo:

```
xsrf_cookies=True,# Use Cross-Site attack secured cookies?
```

Così Tornado rifiuta le richieste POST, PUT e DELETE che non contengono il corretto valore *_xsrf* come parametro della richiesta.

Il valore di XSRF è stato ovviamente incluso nei form HTML del sistema MetroPower, così da autorizzare legittime richieste. Per avere un'idea più

chiara, si veda il codice del form scritto per modificare le informazioni di una rete già esistente:

```
<form action="{{ request.path }}" method="post" class="networkedit">
<table class="table table-striped">
<thead><tr>
<th><h3>Network Information</h3></th>

</tr></thead>
<hr/>
<tfoot><tr>
<td>{{ required }} required</td>
<td>

<button class="btn btn-primary" type="submit">{{ _("Save") if net
else _("Add") }}</button>
{{ xsrf_form_html() }}
</td>
</tr></tfoot>
<tr>
<td>ID{{ required }}</td>
<td><input name="id" {{"readonly='readonly'"ifnetelse''}}
type="text" value="{{ net.id if net else '' }}"/></td>
</tr>
<tr>
<td>Name{{ required }}</td>
<td><input name="name" type="text" value="{{ escape(net.name) if net
else '' }}"></td>
</tr>
<tr>
<td>Gateway IP</td>
<td><input name="gateway_ip" type="text" value="{{
escape(net.gateway_ip) if net else '' }}"></td>
</tr>
<tr>
<tr>
<td>Type{{ required }}</td>
<td><select name="ntype" id="ntype" type="select"></select></td>
</tr>
<tr>
<td>Location</td>
<td><input name="location" type="text" value="{{
escape(net.location) if net else '' }}"></td>
</tr>
{% if net %}
<tr>
<td>Key</td>
<td><input name="nkey" readonly="readonly" type="text"
value={{escape(net.nkey)}}></td>
<tr>
```

```
{% end %}
</table>
<input name="action" type="hidden" value={{'edit'ifnetelse'add'}}>
</form>
```

Nella parte evidenziata del form si vede la chiamata alla funzione *xsrf_form_html* nel template.

UI Modules

Tornado supporta gli User Interface Modules (UI modules) per rendere facile il supporto a widget standard e riusabili all'interno dell'applicazione. Gli UI modules sono come speciali chiamate di funzione per presentare componenti della pagina che vengono pacchettizzati con i propri CSS e Javascript.

Nel sistema MetroPower è stato fatto uso di questo strumento per presentare all'utente, nella pagina di riepilogo dei sensori di una rete, lo stato attuale di ogni dispositivo.

Per impostare Tornado all'utilizzo degli UI modules è stato innanzitutto definito il modulo all'interno del costruttore dell'applicazione:

```
ui_modules={"SensorGraphSparkline":SensorGraphSparklineModule},
```

Infine è stato inserito l'elemento relativo al modulo all'interno della pagina *sensors.html*:

```
<div class="sens_status">
{{ modules.SensorGraphSparkline(net.id, sens.id) }}
</div>
```

Il modulo *SensorGraphSparkline* è una pagina html, che si trova nella directory "modules", alla quale vengono passati come argomenti l'id della rete e l'id del sensore dall'handler *SensorGraphSparklineModule* così fatto:

```
class SensorGraphSparklineModule(tornado.web.UIModule):
def render(self,netid,sensid):
return
self.render_string("modules/sensor_graph_sparkline.html",netid=netid
,sensid=sensid)
```

In questo modo lo stesso elemento viene utilizzato nella stessa pagina per visualizzare lo stato corrente del sensore per tutti i dispositivi appartenenti alla rete.

All'interno del modulo viene impostata la grandezza del widget:

```
<div id="chart_div_{{ sensid }}"style="height:40px;width:120px;">
<img src="{{ static_url('img/loading.gif') }}"></div>
```

E poi, mediante una funzione Javascript, viene recuperato l'ultimo valore dello stato del sensore letto.

```
<script type="text/javascript">
//<![CDATA[
function refresh_{{sensid}}(){
como_get("{{ netid }}","{{ sensid }}","-301s","-
1s","relative",visualize_{{sensid}});
}
function visualize_{{sensid}}(json_response){
var err=json_response['error'];
if(err){
$("#chart_div_{{ sensid }}").html("<div
class='alert'>"+err+"</div>");
return;
}
var data=json_response['data'];
if(data.length==0){
$("#chart_div_{{ sensid }}").html("<div class='alert alert-error'><i
class='icon-warning-sign'></i>  No Data!</div>");
return;
}
// Only show the last reading from the sensor
var last=data.length;
$("#chart_div_{{ sensid }}").html("<div class='alert alert-
info'>"+data[last-1][1]+"</div>");
// Schedule an update in 5 seconds
setTimeout(refresh_{{sensid}},5000);
}
//]]>
</script>
```

Lo stato del sensore viene riletto ogni 5 secondi, in modo da non dover riaggiornare la pagina manualmente.

I widget che mostrano lo stato corrente di tutti i dispositivi vengono mostrati come si vede in Figura 37.

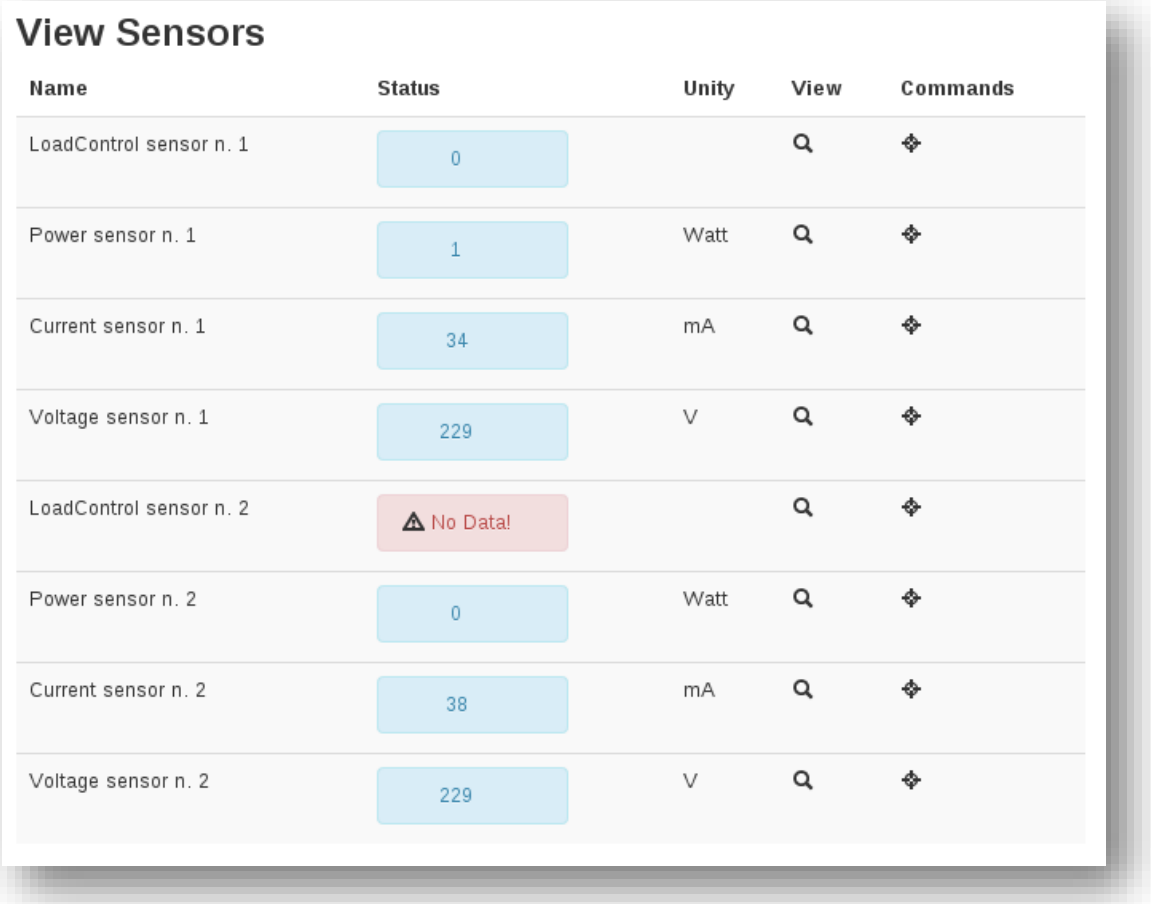


Figura 37 - UI modules nel sistema MetroPower

Autenticazione con servizi esterni

Come già visto, Tornado offre una serie di mix-in di Python che aiutano a sviluppare l'autenticazione con servizi esterni. Il modulo di Tornado *tornado.auth* fornisce classi per OpenID, Twitter, FriendFeed, Google OpenID, Facebook, ecc. Nel progettare il sistema MetroPower si è scelto di utilizzare l'autenticazione attraverso un account Google OpenID.

La classe che è stata creata per gestire l'autenticazione mediante tale metodo è così fatta:

```
class AuthLoginHandler(BaseHandler,tornado.auth.GoogleMixin):
@tornado.web.asynchronous
def get(self):
if self.get_argument("openid.mode",None):
self.get_authenticated_user(self.async_callback(self._on_auth))
return
self.authenticate_redirect()

# Authentication-OK callback.
# Save user info on the first connection.
# Only save a last-login timestamp otherwise.
def _on_auth(self,user):
if not user:
raise tornado.web.HTTPError(500,"Google auth failed")

str_time=datetime.datetime.now().isoformat()

usr=self.db.get("SELECT * FROM users WHERE email=%s",user["email"])
if not usr:
# Create user entry in the WSN-database
self.lock_tables("write",['users'])
usr_id=self.db.execute("INSERT INTO users (email, name, last_access)
\
                                                    VALUES
(%s,%s,%s)",
user["email"],user["name"],str_time)
self.unlock_tables()
else:
```

```
self.lock_tables("write",['users'])
usr_id=usr["id"]
self.db.execute("UPDATE users SET last_access=%s WHERE id=%s",
str_time,usr_id)
self.unlock_tables()

self.set_secure_cookie("user",str(usr_id))
      self.info("Hello <b>"+user["name"]+"</b>!")
self.redirect(self.get_argument("next","/"))

# Do not log Login info
def _log(self):
pass
```

Come si vede dal codice, sono stati utilizzati i metodi *authenticate_redirect* e *get_authenticated_user* per recuperare i dati dell'account Google dell'utente, consentirgli di loggarsi sul servizio esterno e ritornare all'applicazione MetroPower le informazioni necessarie da salvare nel nostro database MySQL.

Una volta che l'utente del sistema si è autenticato sul servizio di Google, viene chiamata la funzione callback *_on_auth* che aggiorna nella tabella USERS del database MySQL il timestamp dell'ultimo accesso dell'utente, se questo era già presente, oppure aggiunge il nuovo utente registrato alla tabella, con le informazioni di identità dell'utente provenienti dal servizio esterno di Google, quali email e nome.

4.3.3. Flusso di invio dei comandi dal database MySQL al gateway

Il server per la comunicazione remota con i gateway delle varie reti, recupera i comandi dal database MySQL di Tornado. Ciascun processo figlio presente sul server ha accesso al database.

La tabella dei comandi *confcommands*, come già descritto, include i seguenti campi:

- Network_id (integer, ID della rete);
- Ntype (stringa, tipo della rete);
- Timestamp;
- Command (string al massimo 512 caratteri).

Il corpo del pacchetto (command) è inserito in una stringa di tipo JSON[2].

Il suo valore è la stringa (ASCII) che contiene il pacchetto:

{ 'data': 'value' }

[2] JSON, acronimo di JavaScript Object Notation, è un formato adatto per lo scambio dei dati in applicazioni client-server.

Il formato dei dati nella tabella CONFCOMMANDS è visibile in Figura 38. Il campo ntype della tabella nel database indica la "famiglia" del pacchetto. Allo stato attuale, il gateway riconosce due famiglie, "ztc" e "uni":

- *ztc* : indica che il pacchetto è codificato in esadecimale-ascii e deve essere inviato così com'è al gateway;
- *uni*: si usa per pacchetti "universali" che devono essere inviati al gateway e interpretati da questo. Servono per inviare comandi di configurazione/riprogrammazione del gateway.

seqno	network_id	ntype	timestamp	command
402	2459546910990453036	ztc	2013-04-23 16:00:21	{"data": "70501002d13d0000000000008060008000001d9...
403	2459546910990453036	ztc	2013-04-23 16:00:26	{"data": "70501002d13d0000000000008060008000000d8...
404	2459546910990453036	ztc	2013-04-23 16:00:52	{"data": "a20106d13dd13d0100a4"}
405	2459546910990453036	ztc	2013-04-23 16:01:02	{"data": "a20504d13dd13da3"}

Figura 38 -Esempio di dati presenti nella tabella CONFCOMMANDS

Ciascun processo figlio legge dal database MySQL il comando destinato alla rete a lui associata (identificabile dall'indirizzo di rete a 64 bit).

Il processo figlio elimina dal database MySQL, ogni 5 minuti, le righe più vecchie di 15 minuti. Se il processo figlio termina, per una qualsiasi ragione, prima di uscire elimina dal database MySQL tutte le righe destinate a quell'indirizzo di rete.

Invio di un comando a un dispositivo

Per capire meglio il flusso di invio dei comandi dal database MySQL al gateway, viene descritto qui nel dettaglio come è stata implementata la parte di front-end per il sistema MetroPower.

Esaminiamo il comando on/off/toggle, che è possibile inviare a sensori di tipo LoadControl.

Quando l'utente invia il comando ON al sensore, il comando inviato viene passato come argomento della richiesta all'handler *SensorSendHandler*. Viene prima di tutto controllato che l'utente corrente abbia i privilegi necessari per inviare il comando al dispositivo.

```
class SensorSendHandler(BaseHandler):
# Requires authentication
@tornado.web.authenticated
@tornado.web.asynchronous
@tornado.gen.engine
def get(self, nid, sid):
command=self.get_argument('command').upper();

usr=self.get_current_user()
usr_id=usr['id']

self.lock_tables("read",['nets_permissions as n'])
perm=self.db.get("SELECT n.perm FROM nets_permissions as n \
                                WHERE n.network_id=%s AND
n.user_id=%s", nid, int(usr_id))
self.unlock_tables()

# Check whether the user has access to the network
perms=self.check_network_access(nid)
# Check wether the sensor exists in this network
self.check_sensor_in_network(sid,nid)
```

Dopodiché l'handler controlla che il comando inviato sia tra quelli disponibili. Se non lo è, viene inviato un codice di errore 404 al client.

```
if command not in['ON','OFF','TOGGLE','IDENTIFY','HOWMANY']:
raise tornado.web.HTTPError(404,"Unknown command: "+str(command))
```

Altrimenti l'handler costruisce il relativo pacchetto da inviare al dispositivo.

```
#Command OnOffCmd_SetState
if command in['ON','OFF','TOGGLE']:
sens_id=hextransform(int(sid)>>16,4)
sens_id_little=invert2bytes(sens_id,0)

if command=='ON':
cmd_data="01"
elif command=='OFF':
cmd_data="00"
elif command=='TOGGLE':
cmd_data="02"

op_group_hex=0x70
op_code_hex=0x50

cmd_meta="02%s000000000000080600080000%s"
mycommand=cmd_meta%(sens_id_little,cmd_data)
len_mycommand=len(mycommand)/2

mynet_type="ztc"
cmdjson=packet2json(op_group_hex,op_code_hex,mycommand)
```

I pacchetti devono essere scritti nel database nel linguaggio che il nodo è in grado di interpretare. La formattazione dei pacchetti "ztc" destinati ai sensori ZigBee Freescale è descritta nel manuale "BeeStack™ BlackBox ZigBee™ Test Client (ZTC)" e per il comando onoff è la seguente (Figura 39):

B.6.27 OnOffCmd_SetState

Description

For the 3 commands On = 01, off = 00 and Toggle = 02.

Parameters

Table B-346. OnOffCmd_SetState Parameters

Parameter	Size (bytes)	Comments
OpGroup	1	0x70
OpCode	1	0x50
Length	1	Length in bytes of the following parameters
DestAddressMode	1	Indirect = 00, Group = 01, Direct 16it = 02, Direct 64bit = 03
DestAddress	8	Destination Address
DestEndPoint	1	Destination EndPoint, ignored for indirect or group address mode
ClusterId	2	Cluster Id
SrcEndPoint	1	Source EndPoint
TxOptions	1	Transmit options
Radius	1	Radius of transmission
Command	1	

Figura 39 - Parametri del comando OnOffToggle Command

Una volta impostato correttamente il pacchetto, viene costruito il comando da scrivere nella tabella *confcommands* del database MySQL, dove vengono aggiunte informazioni sull'id della rete, sul tipo di rete e sul timestamp di invio del comando, in questo modo:

```
ts=datetime.datetime.now().isoformat()
self.lock_tables("write",['confcommands'])
self.db.execute("INSERT INTO confcommands (network_id, ntype,
timestamp, command) \
 VALUES (%s,%s,%s,%s)", nid, mynet_type, ts, cmdjson)
self.unlock_tables();
```

4.3.4. Flusso dei messaggi di stato dalla rete al DB

È possibile inviare ai nodi della rete comandi e messaggi in un formato che essi sono in grado di comprendere. Le risposte a questi comandi passano da CoMo ma vengono interpretate da tornado.

CoMo ha un database diverso per ogni modulo. Per quanto riguarda le letture di dati provenienti dai sensori, la struttura dei record definita è la seguente:

```
struct record {
   timestamp_t ts;
   uint64_t net_id;
   uint32_t sens_id;
   uint8_t  sens_type;
   int32_t value;
}
```

Il sistema è composto da WSN di vario tipo (ZTC, Wi-Fi, Bluetooth, ecc.). Ogni rete di sensori è collegata ad un gateway, che ne gestisce lo scambio di dati con il server di comunicazione remota. Per ciascuna nuova connessione da un gateway, viene generato dal server un processo per la gestione della comunicazione e l'inoltro dei messaggi, dei comandi e delle configurazioni.

I dati provenienti dai sensori vengono scritti in un file (/tmp/ztc) in questo formato:

```
222210e261fec92c 517694eb
*ZTCR;517694eb;000ee2d7;4e;9D01240200000002E4DCFF0000170704260000000
000018100000000000000000000000000000#

222210e261fec92c 517694eb *ZTCR;517694eb;000ee2d8;08;A0970100#

222210e261fec92c 517694eb
*ZTCR;517694eb;000ee2d9;28;9D001102D13D00000000000000000000000083#

222210e261fec92c 517694eb
*ZTCR;517694eb;000ee2da;4e;9D01240200000002E4DCFF0000170804E5000000000
000018100000000000000000000000000000#

222210e261fec92c 517694ec
*ZTCR;517694eb;000ee2e0;28;9D001102E4DC00000000000000000000000084#
```

Da questo file, uno sniffer di CoMo, come sarà meglio descritto nel prossimo capitolo, filtra i messaggi e li inserisce nel database secondo il formato timestamp-UUID-sensid-comando.

Le query a questo database vengono fatte tramite URL dal web server Tornado, che fa parte del front-end. Tornado si appoggia su un database MySQL di configurazione. Al database di Tornado vengono fatte le query provenienti dall'interfaccia di amministratore/utente.

Ricezione della risposta di un sensore ad un comando

Nel sistema MetroPower attualmente è possibile inviare due comandi che prevedono una risposta dai sensori:

- il comando IDENTIFY, che richiede al dispositivo di identificarsi sulla rete restituendo il proprio IEEE address a 64 bit,
- il comando HOWMANY, che richiede al nodo di conoscere quanti sono gli “active endpoints” presenti su di esso.

Per capire meglio il flusso dei messaggi di stato provenienti dalle reti al front-end, viene descritto qui nel dettaglio come è stata implementata la parte relativa al front-end per il sistema MetroPower.
Esaminiamo innanzitutto la ricezione della risposta al comando IDENTIFY.

La risposta al comando IDENTIFY viene descritta nel manuale BeeStack™ BlackBox ZigBee™ Test Client (ZTC)Reference Manual come in Figura 40:

Table B-170. ZDP-IEEE_addr.response Parameters

Parameter	Size (bytes)	Comments
OpGroup	1	0xA0
OpCode	1	0x81
Length	1	Length in bytes of the following parameters
Status	1	Result of write cmd Possible values: 0x00: Success 0x80: Inv_RequestType (Success) 0x81: Device_Not_found (Device Not found)
IEEEAddrRemoteDev	8	IEEE Address of Remote Device
NWKAddrRemoteDev	2	NWK Address of Remote Device
NumOfAssociatedDevice	1	NWK Address of Remote Device
StartIndex	1	NWK Address of Remote Device
ListOfShortAddress	2 x NumOfAssociatedDevice	NWK Address of Remote Device

Figura 40 - Parametri della risposta al comando IDENTIFY

Sapendo, dunque, in che modo è strutturata la risposta al comando inviato, nel *SensorSendHandler* sono stati impostati l'opgroup e l'opcode sui valori indicati dal manuale. È stata fatta poi una richiesta HTTP

asincrona al modulo *ztc_config* di CoMo per recuperare le risposte inviate dai sensori negli ultimi 20 secondi.

```
elif command=='IDENTIFY':
opcodegroupr=int("A0",16)
opcoder=int("81",16)


como_url="".join(['http://',options.como_address,':',options.como_po
rt,'/ztc_config?netid=0&opcode_group=0&opcode=0&start=-20s&end=-
1s'])

http_client=AsyncHTTPClient()


yield tornado.gen.Task(tornado.ioloop.IOLoop.instance().add_timeout,
time.time()+5)
response = yield tornado.gen.Task(http_client.fetch,como_url)
```

A questo punto l'handler controlla che non si siano verificati errori. Se l'esito è negativo, recupera la risposta relativa al comando IDENTIFY inviato e stampa a video all'utente l'indirizzo IEEE del dispositivo.

La richiesta HTTP che viene fatta al modulo di CoMo ritorna una stringa di questo tipo:

1366729234245954691099045303616012913 0 170 170 170 170 170 170 170 170 209 61 0 0

Il codice di Tornado scritto per recuperare la risposta, va dunque a "splittare" la stringa nelle varie parti di cui è composta:

1. timestamp (evidenziato in rosso)
2. id della rete (evidenziato in giallo)
3. opgroup (evidenziato in verde)
4. opcode (evidenziato in azzurro)

Le parti successive rappresentano il pacchetto di risposta e contengono le informazioni da ritornare al client, ovvero:

1. lunghezza
2. indirizzo IEEE su 8 byte
3. altre informazioni

Il codice che elabora la stringa è il seguente:

```
ret={}
if response.error:
    ret['error']='Error while retrieving the response'
    self.write(tornado.escape.json_encode(ret))
    self.finish()
else:
    for line in response.body.split("\n"):
        if line!="":
            opg=int(line.split(" ")[2])
            opc=int(line.split(" ")[3])
            status=int(line.split(" ")[5])
        if command=='IDENTIFY':
            if opg==opcodegroupr and opc==opcoder:
                if status==0:
                    IEEEAddrRemoteDev="".join(line.split(" ")[6:13])
                    ret['success']="The IEEE address of the remote
device is %s "%(IEEEAddrRemoteDev)
                    self.write(tornado.escape.json_encode(ret))
```

```
            self.finish()
        elif status==128:
            ret['success']="Invalid request type"
            self.write(tornado.escape.json_encode(ret))
            self.finish()
        elif status==129:
            ret['error']="Device not found"
            self.write(tornado.escape.json_encode(ret))
            self.finish()
```

Analogo è il funzionamento della ricezione della risposta del dispositivo al comando HOWMANY.

La risposta del dispositivo a tale comando è del tipo descritto in Figura 41:

Table B-128. ZDP-Active_EP_rsp.response Parameters

Parameter	Size (bytes)	Comments
OpGroup	1	0xA0
OpCode	1	0x85
Length	1	Length in bytes of the following parameters
Status	1	Result of write cmd Possible values: 0x00: Success 0x80: Inv_RequestType (Success) 0x89: No_Descriptor (Success) 0x81: Device_Not_found (Device Not found)
NwkAddress	2	Nwk Short Address of Remote Device
ActiveEPCount	1	Count of Active Endpoints of Remote Device
ActiveEPList	ActiveEPCount	List of Active Endpoints

Figura 41 - Parametri della risposta di un dispositivo al comando HOWMANY

4.4. Modulo query CoMo

4.4.1. Modulo ztc_config

Il modulo ztc_config di CoMo è il modulo scritto per ottenere lo stato della configurazione e ricevere le informazioni specifiche di una data rete.

Tale modulo cattura pacchetti 'ztc' del tipo:

```
/* common data format shared by Capture, Transform, Storage
and Query callbacks*/
#define DATAFMT \
    timestamp_t ts; \
    uint64_t net_id; \
    uint8_t opcode_group;\
    uint8_t opcode;\
    uint8_t payloadlength; \
    uint8_t payload[256]; \
```

I record vengono salvati nel database di CoMo nel seguente formato:

```
typedef struct ZTCMessage_tag{
ztcOpcodeGroup_t opcode_group;
ztcMsgType_t opcode;//opcode
uint8_t length;//len
uint8_t data[256];
} ZTCMessage_t;
```

La risposta proveniente dai sensori è così strutturata:

seq | net_id | opcode_group | opcode | payloadlength | val

Per esempio:

1366729234 | 2459546910990453036 | 160 | 129 | 13 | 01701701701701701701701702096100

4.4.2. Interpretazione dei dati da un nuovo tipo di dispositivo: modifiche ai moduli di CoMo e Tornado

Modifiche apportate a CoMo

Ogni modulo, in CoMo, ha una sua directory contenente almeno questi file:

- *data.h*
- *capture.c*
- *query.c*

Il file *ztc.h* contiene le definizioni di costanti e strutture dati relative al protocollo ZTC.

Per quanto riguarda il file *data.h*, si è voluto dichiarare il payload del pacchetto come un array di interi a 8 bit di dimensione 256

Lo stesso è stato fatto nel file *ztc_cmd.h* per la struttura che contiene il messaggio ZTC.

Nel file *capture.c* è stata quindi aumentata la capacità del buffer, dal quale vengono raccolti i dati:

```
uint8_t buffer[256];
ZTCMessage_t *zp;
uint8_t data_len;
```

In questo modo si legge la stringa del pacchetto ZTC, viene rimosso il carattere “#” finale, si trasforma la stringa in un pacchetto ZTC e si creano le tuple da salvare nel database.

```
// Read the ZTC packet string
sscanf(pkt->payload+57,"%s",zbpktstr);
// Remove the '#' char at the end of string
zbpktstr[strlen(zbpktstr)-1]='\0';
// Convert ZTC packet string to raw binary
hexstr2raw(zbpktstr,buffer);
// Cast the buffer to a ztc packet
// (all fields are 1-byte long, no need to adjust endianness)
zp=(ZTCMessage_t*)buffer;

data_len=zp->length;

// Create Tuple to be saved on disk
t=mdl_tuple_new();
t->ts=HTONLL(COMO(ts));
t->net_id=HTONLL(netid);
t->opcode_group=zp->opcode_group;
t->opcode=zp->opcode;
t->payloadlength=zp->length;
int i;
for (i=0;i<data_len;i++)
t->payload[i]=zp->data[i];
```

In questo modo è possibile stampare il payload byte a byte e, di conseguenza, andare a leggere al momento della query corrispondente,

solamente i byte che interessano ai fini della risposta, in base alla lunghezza del payload.

Per esempio:

```
IEEEAddrRemoteDev="".join(line.split(" ")[6:13])
```

Nel file *query.c*, infatti, i dati provenienti dai sensori vengono stampati in questo modo:

```
val=rec->payloadlength;

mdl_print("%u %llu %u %u %u ", sec, NTOHLL(rec->net_id),
rec->opcode_group,
rec->opcode,
rec->payloadlength);

int i;
for (i=0;i<val;i++)
mdl_print("%u ",rec->payload[i]);
mdl_print("\n");
```

Dal codice, si vede chiaramente come si sia scelto di stampare byte a byte il payload in modo da poter riconoscere facilmente i byte di cui esso è composto.

Il risultato è visibile in Figura 42:

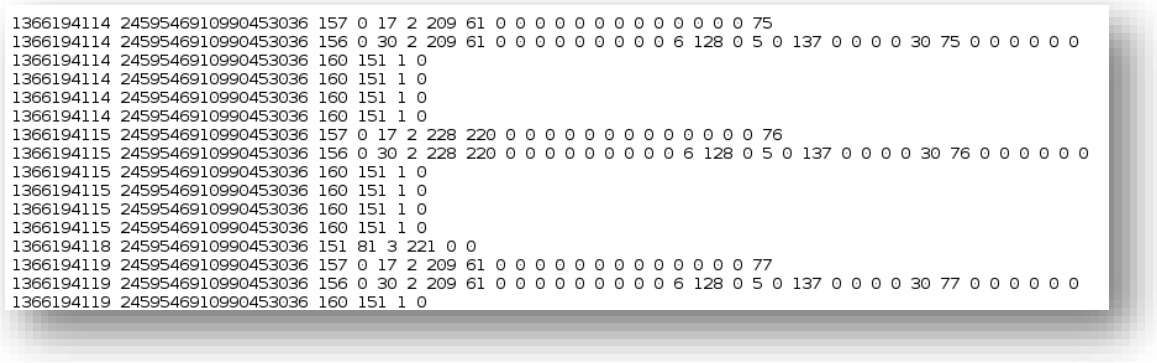


Figura 42 - Output del modulo ztc_config di CoMo

Modifiche apportate a Tornado

Per far sì che l'interfaccia interpreti correttamente i dati che arrivano da un nuovo tipo di dispositivo, occorre aggiungere il tipo di tra quello supportati.

Nel file *wsn.py*, la struttura *sensors_config* definisce tutti i tipi di sensori visualizzabili: all'interno di *sensor_config* è possibile assegnare un nome ed una serie di comandi che possono essere ricevuti dai sensori.

```
# Define the configuration for all the different type of sensors in
all the possible networks
sensors_config={
'ztc':{
```

```
0:{
'type':'Generic',
'commands':[]
},
1:{
'type':'Temperature',
'commands':[]
},
2:{
'type':'Contact',
'commands':[]
},
3:{
'type':'LoadControl',
'commands':['on','off','toggle']
},
4:{
'type':'Power',
'commands':[]
},
      5:{
'type':'Current',
'commands':[]
},
      6:{
'type':'Voltage',
'commands':[]
},
7:{
'type':'ECG',
'commands':[]
},
},
'wifi':{},
'bluetooth':{},
}
```

In tale struttura sono stati definiti 8 tipi di sensori ‘ztc’: Generico, Temperatura, Contatto, LoadControl, Potenza, Corrente, Voltaggio, ECG. Per il momento è possibile inviare i comandi specifici *on*, *off* e *toggle* ai sensori LoadControl (sensori di contatto).

Per interrogare il modulo ztc_config e ottenere le risposte dai sensori, invece, è necessario solamente fare una richiesta http al modulo, nella quale vengono specificati l'indirizzo ip del server, la porta su cui è in ascolto, il modulo, l'id della rete, l'opcode del gruppo, l'opcode e l'intervallo di tempo sul quale vogliamo che vengano estratti i dati.

Per esempio:

http://131.114.52.207:44444/ztc_config?netid=0&opcode_group=0&opcode=0&start=-5m&end=-1s

Se opcode_group e opcode sono uguali a '0', vengono restituiti tutti i record.

5. Modo d'uso

5.1. Invio dei comandi e visualizzazione dei dati dai sensori

Ogni utente registrato sul sistema MetroPower può inviare comandi alle proprie reti ed ai propri dispositivi.

5.1.1. Invio di comandi alle reti

Una volta effettuato l'accesso al sistema MetroPower, accedere alla pagina di riepilogo di una rete di cui si è amministratori e portarsi nella seconda scheda "Options" (Figura 43).

Da questa scheda è possibile inviare comandi alla propria rete. Attualmente il sistema MetroPower supporta il solo comando di restart della rete.

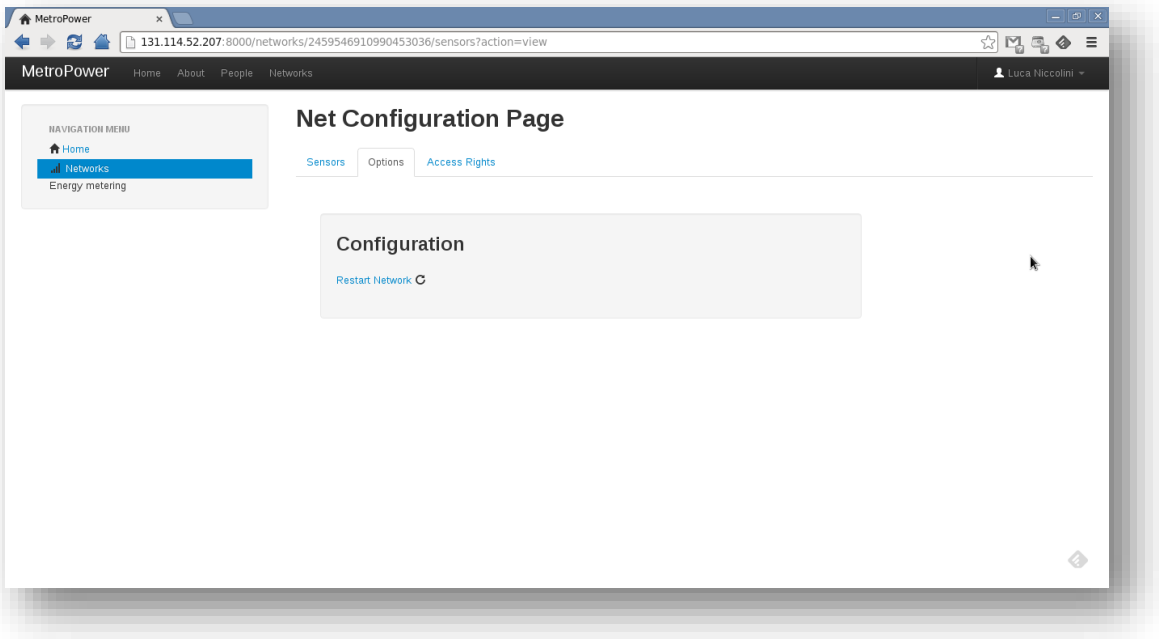


Figura 43 - Pagina di invio comandi alla rete

Cliccare su Restart Network per inviare il comando di riavvio alla rete. Il sistema notificherà l'avvenuto invio del comando (Figura 44).

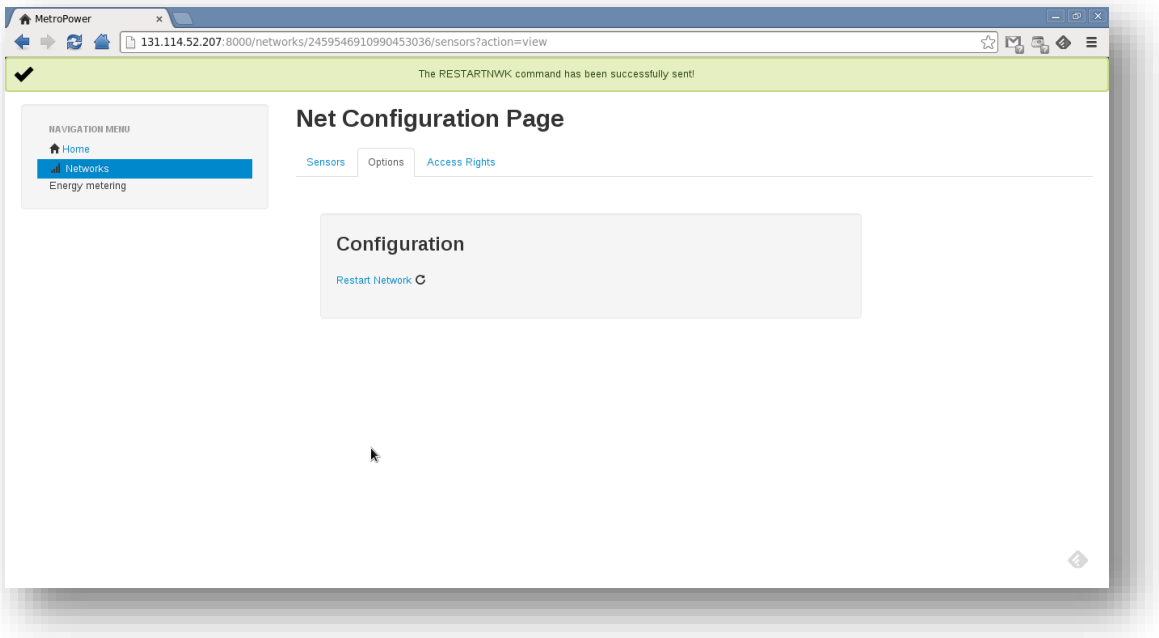


Figura 44 - Notifica di avvenuto invio del comando di riavvio della rete

5.1.2. Invio di comandi ai dispositivi

Nella pagina di riepilogo della rete, vengono visualizzati tutti i sensori appartenenti alla rete stessa (Figura 45).

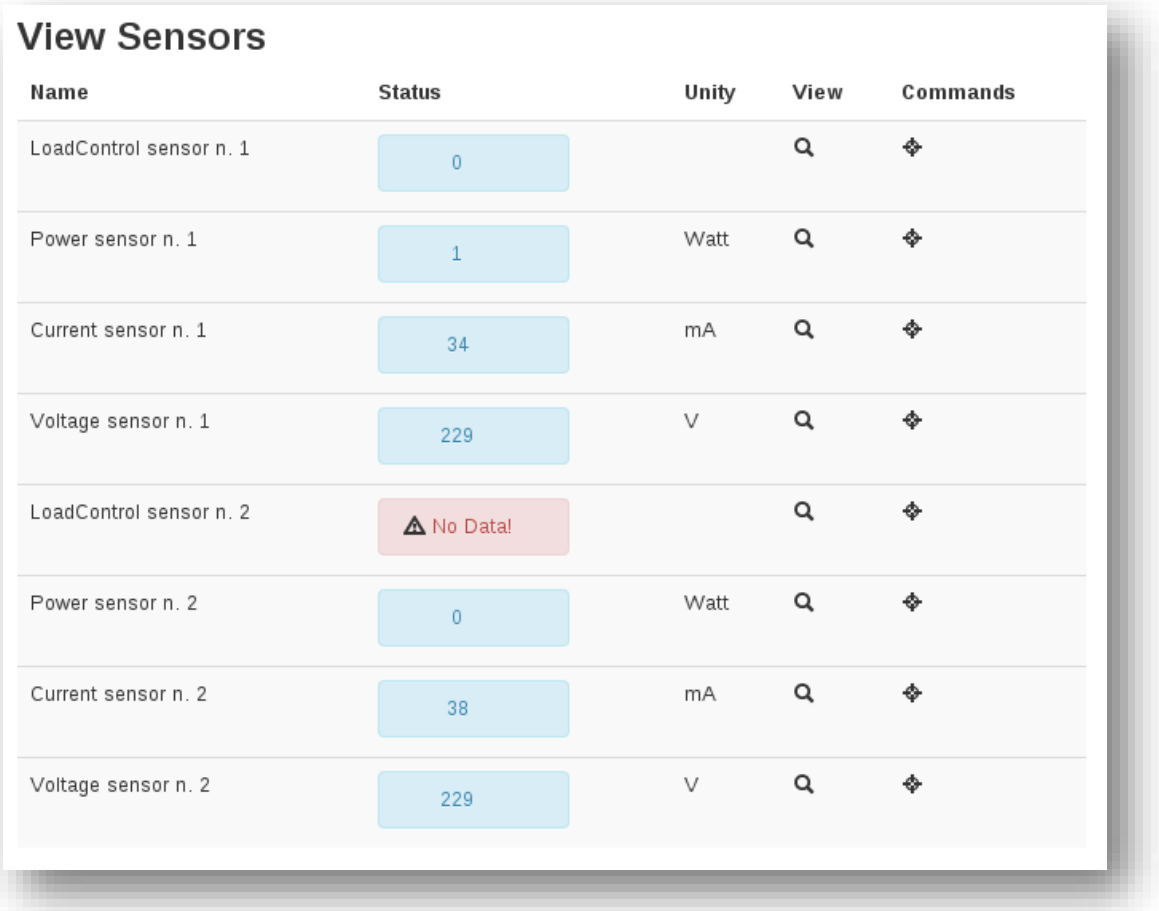


Figura 45 - Pagina di riepilogo della rete

Per quanto riguarda i sensori LoadControl, è possibile inviare direttamente dalla pagina di riepilogo il comando on/off/toggle (Figura 46).

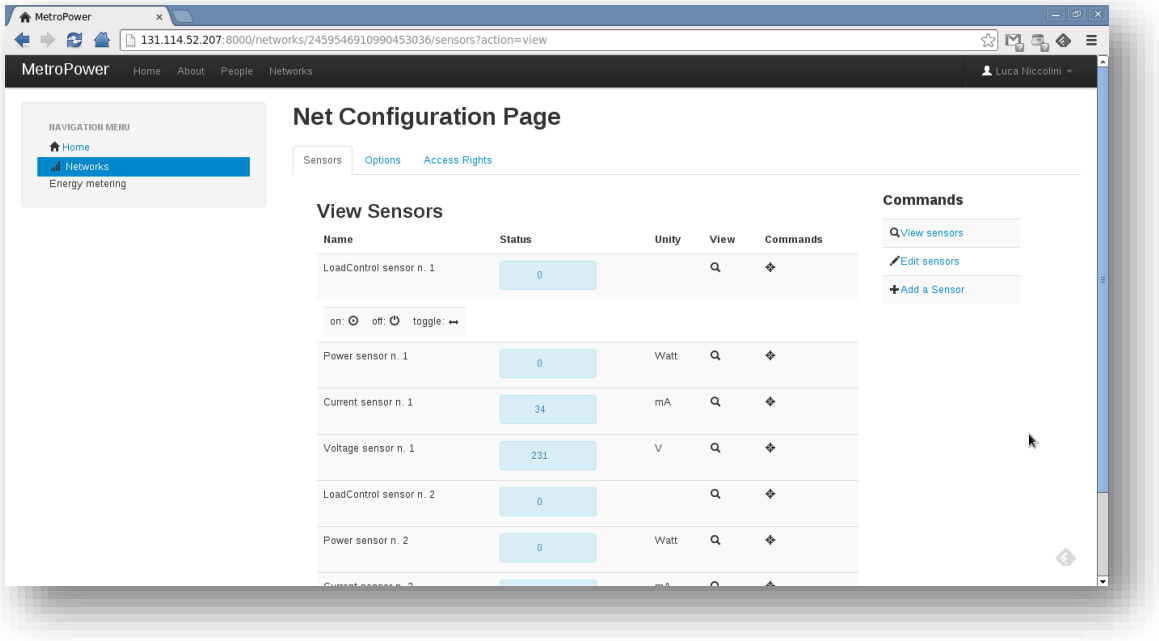


Figura 46 - Comando on/off/toggle per sensori LoadControl

Tale comando, come descritto nel manuale di riferimento BeeStack™ BlackBox ZigBee™Test Client (ZTC), non prevede alcuna risposta da parte del sensore.

Come si vede in Figura 46, il sensore attualmente è spento, poiché il suo valore registrato è pari a “0”. Quando l’utente clicca sul comando ON, viene attivato il seguente collegamento (file /front-end/templates/sensors.html).

```
{% for cmd in sens_config[sens.type]['commands'] %}
<td><div class="sens_{{cmd}}">{{cmd}}:<a
href='javascript:void(send_command_to_sensor("{{net.id}}",
"{{sens.id}}", "{{cmd}}"));'>{% if cmd == 'on' %} <i class="icon-
play-circle"></i>{% end %}
{% if cmd == 'off' %} <i class="icon-off"></i>{% end %}
{% if cmd == 'toggle' %}<i class="icon-resize horizontal"></i>{% end
%}
```

Viene richiamata la funzione Javascript *send_command_to_sensor*, presente nel file /front-end/static/js/wsn.js, passando come parametri l’id della rete, l’id del dispositivo ed il comando inviato.

```
function send_command_to_sensor(net,sens,command){
$.ajax({url:"/networks/"+net+"/sensors/"+sens+"/send?command="+comma
nd,
type:"GET",
async:true,
dataType:"json",
success:function(json_response){
var err=json_response['error'];
      if (err){
      show_alert('error',err);
      return;
      }
      var success=json_response['success'];
      if (success){
      show_alert_2('success',success);
      return;
      }
      show_alert('alert',"This should not happen!");
}
});
}
```

Tale funzione effettua una richiesta HTTP di tipo GET a Tornado ed invia così il comando al dispositivo.

Sul client viene notificato così il corretto invio del comando ON al dispositivo e, come si vede in Figura 47, lo stato del sensore passa a “1”.

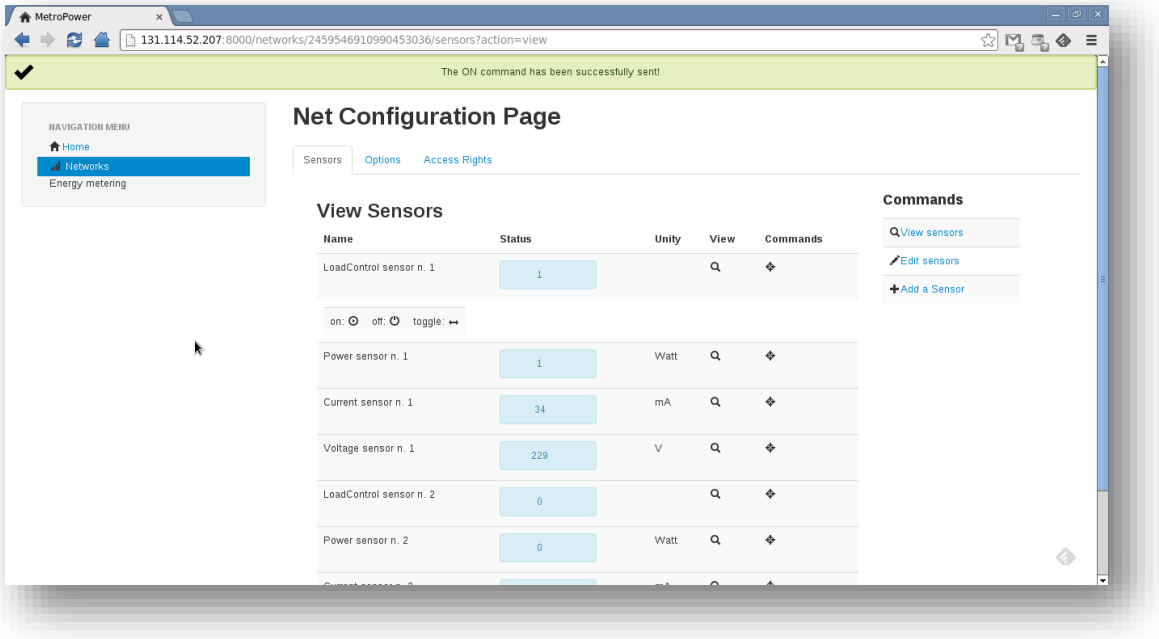


Figura 47 - Notifica di avvenuto invio del comando ON al dispositivo

5.2. Ricezione risposte

Analogamente a quanto visto per il comando on/off/toggle, è possibile inviare attualmente altri due comandi ai dispositivi della propria rete. Mentre il comando on/off/toggle non prevede risposta da parte del dispositivo al quale viene inviato, gli altri due comandi prevedono una risposta dal sensore.

È possibile inviare, infatti, il comando IDENTIFY, che richiede al dispositivo di identificarsi sulla rete restituendo il proprio IEEE address a 64 bit.

È, inoltre, possibile inviare il comando HOWMANY, che richiede al nodo di conoscere quanti sono gli “active endpoints” presenti su di esso.

Vediamo nel dettaglio la ricezione della risposta a questi due comandi.

5.2.1. Ricezione della risposta al comando IDENTIFY

Per inviare il comando IDENTIFY ad un dispositivo, occorre portarsi nella pagina di amministrazione dello stesso ed accedere alla seconda scheda "Options" (Figura 48).

Da qui occorre cliccare sul link *Identify sensor (IEEE address)* per inviare il comando, analogamente a quanto visto per il comando on/off/toggle.

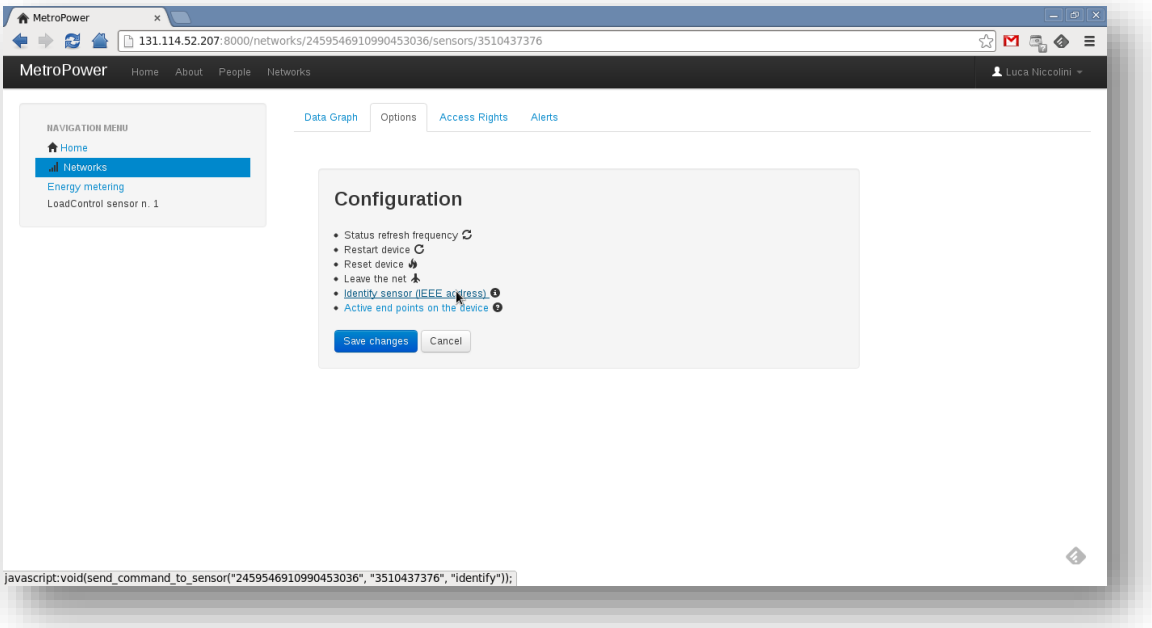


Figura 48 - Opzioni di invio comandi ai dispositivi

Il flusso di esecuzione dell'invio del comando è identico a quello visto per il comando on/off/toggle. In questo caso però l'esecuzione prosegue con il recupero della risposta del dispositivo.

Se non ci sono stati errori nel processo di invio del comando e di risposta da parte del sensore, viene stampato a video sul client l'indirizzo IEEE del dispositivo (Figura 49).

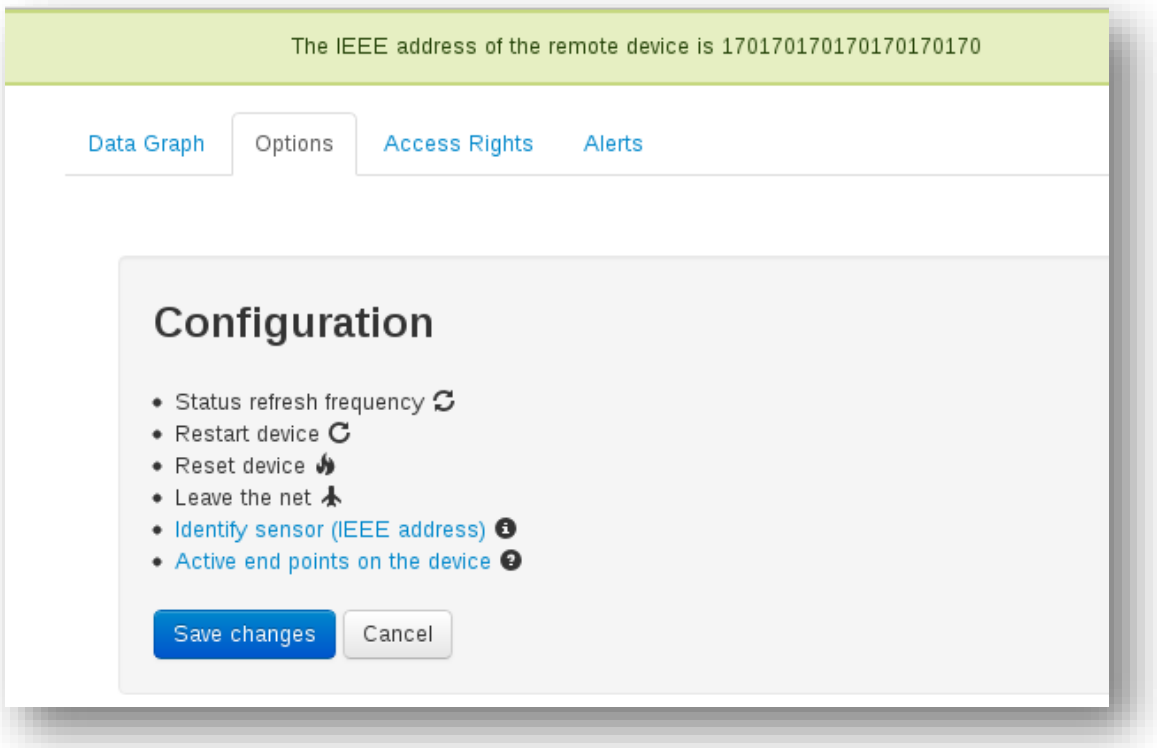


Figura 49 - Risposta di un dispositivo al comando IDENTIFY

5.2.2. Ricezione della risposta al comando HOWMANY

Analogo è il funzionamento della ricezione della risposta del dispositivo al comando HOWMANY.

Se il processo di invio del comando e di risposta da parte del sensore è andato a buon fine, viene stampato a video sul client il numero di active endpoints che sono presenti sul dispositivo remoto (Figura 50).

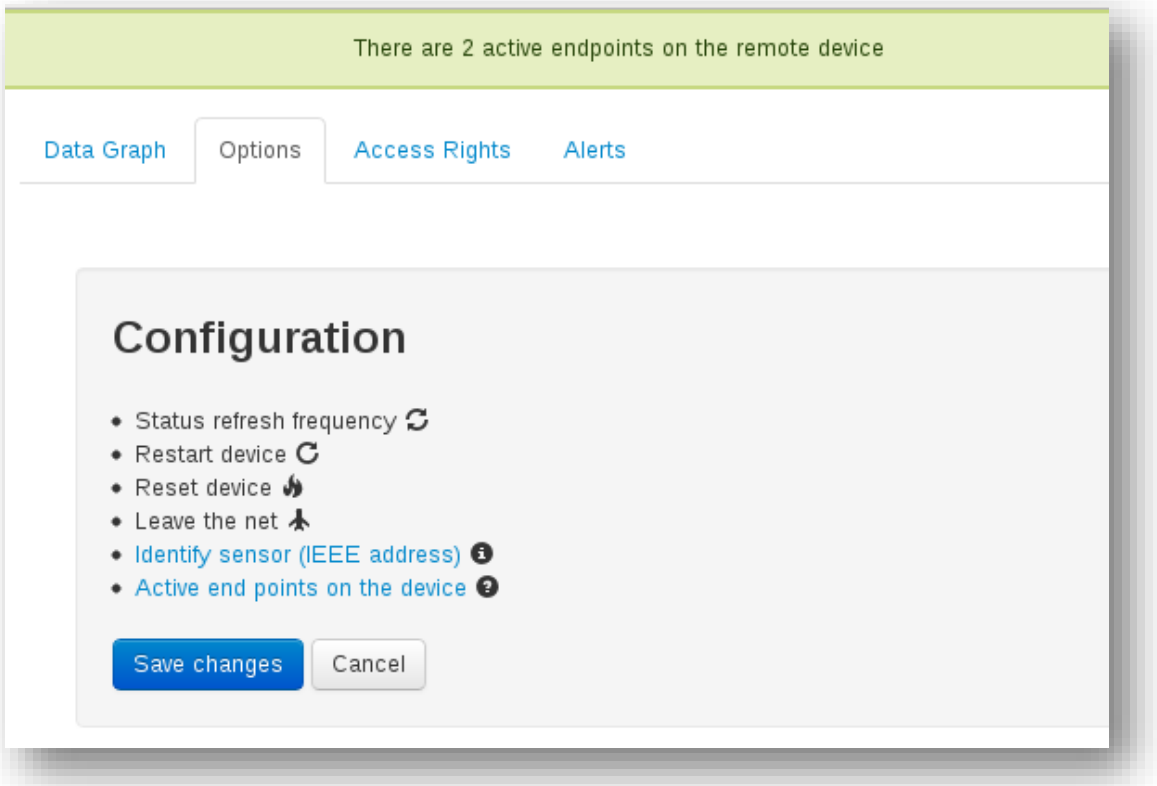


Figura 50 - Risposta di un dispositivo al comando HOWMANY

5.3. Visualizzazione dati sensori

Per visualizzare il grafico dello stato di un sensore nel tempo, occorre accedere alla pagina specifica del dispositivo.

Dalla pagina di riepilogo della rete, cliccare sull'icona "View" accanto al nome del dispositivo di cui si vogliono vedere nel dettaglio i dati (Figura 45).

Viene chiamato al rendering della pagina l'handler di Tornado *SensorHandler*.

```
class SensorHandler(BaseHandler):
# Requires authentication
@tornado.web.authenticated
def get(self,nid,sid):

# Retrieve the current user
usr=self.get_current_user()
usr_id=usr['id']

self.lock_tables("read",['nets_permissions as n'])
perm=self.db.get("SELECT n.perm FROM nets_permissions as n \
                                WHERE n.network_id=%s AND
n.user_id=%s", nid, int(usr_id))
self.unlock_tables()

# Check whether the user has access to the network
perms=self.check_network_access(nid,perm['perm'])

# Check wether the sensor exists in this network
self.check_sensor_in_network(sid,nid)
# Get Network info
net=self.get_network(nid)
# Get Sensor info
sens=self.get_sensor(sid,nid)

# check if the user is the net admin
writeable=self.get_network_access(net.id,PERM_WRITE)

# Retrieve the current permission of the user on the device
self.lock_tables("read",['nets_permissions as n'])
usrpermnet=self.db.query("SELECT perm FROM nets_permissions as n
```

```
WHERE user_id = %s AND network_id=%s",int(usr_id),net.id)
self.unlock_tables()

self.lock_tables("read",['devices_permissions as d'])
usrpermdev=self.db.query("SELECT perm FROM devices_permissions as d
\
                                    WHERE user_id = %s AND
network_id=%s AND device_id=%s", int(usr_id), net.id, sens.id)
self.unlock_tables()

if len(usrpermdev)>0:
usrperm=usrpermdev
else:
usrperm=usrpermnet

# Retrieve the current rights
self.lock_tables("read",['users as u','nets_permissions as
n','devices_permissions as d'])
user=self.db.query("SELECT DISTINCT t.id, t.name, d.device_id,
d.perm FROM \
                                (SELECT u.id, u.name, n.network_id
FROM users as u LEFT OUTER JOIN nets_permissions as n \
                                ON u.id = n.user_id WHERE u.id!=%s
AND n.network_id=%s AND n.perm!=0) as t LEFT OUTER JOIN
devices_permissions as d \
                                ON d.network_id = t.network_id AND
d.user_id = t.id", int(usr_id), net.id)


self.unlock_tables()

sens_config=sensors_config[net.ntype]

# Render the networks page
# Send also the server time
self.render("sensordata.html",net=net,sens=sens,writeable=writeable,
users=user,usrperm=usrperm,usrpermdev=usrpermdev,sens_config=sens_co
nfig,srvtime=int(time.time()))
```

L'handler recupera l'utente corrente che vuole accedere alla pagina e controlla che egli abbia accesso ad essa. Controlla che il sensore esista sulla rete e recupera il permesso dell'utente sullo stesso dispositivo.

Infine invia le informazioni recuperate al client, compreso il *server time*, necessario per la creazione del grafico.

Una volta demandata la costruzione della pagina al client, viene chiamata la funzione *get_data*:

```
function get_data(){
var end = new Date($('#timeivl').val())-0;
var start=end-$("input:radio[@name=ivllength]:checked").val()*1000;
utc_start=""+newDate(start).toUTCString();
utc_end=""+newDate(end).toUTCString();
// Insert the 'loading' image
if (chart==null){
$("#chart_div").html('<img src="{{ static_url('img/loading.gif')
}}">');
}
// Gets sensor data (in JSON format) and call 'visualize' when done
como_get("{{ net.id }}","{{ sens.id
}}",utc_start,utc_end,"absolute",visualize);
}
```

Tale funzione recupera i dati passati in formato JSON da Tornado chiama a sua volta la funzione Javascript *visualize*. Questa funzione controlla che non si siano verificati errori:

```
function visualize(json_response){
var err=json_response['error'];
if (err){
$("#chart_div").html("<div class='alert'>"+err+"</div>");
return;
}

var data=json_response['data'];
data=data.map(function(d){return[d[0]*1000,d[1]]});

// Set the global options for HighCharts
Highcharts.setOptions({
global:{
useUTC:false}});
```

Se l'esito è negativo, imposta i dati ricevuti in maniera corretta per la visualizzazione e setta le opzioni corrette della libreria Highcharts. L'opzione globale *useUTC: false* serve per utilizzare nella visualizzazione del grafico il tempo locale, invece che il tempo coordinato universale (UTC).

Dopodiché, se non ci sono dati da visualizzare, viene mostrato un messaggio di errore. Altrimenti, la funzione controlla che l'utente, nel caso egli non sia amministratore della rete, abbia almeno il permesso di lettura sul dispositivo.

```
{%if writeable=="False"%}
    {%if not usrpermdev%}
    $("#chart_div").html("<div
class='alert'><strong>Warning!</strong> You don't have the
permission to view this page.</div>")
    {%elif usrpermdev==0%}
    $("#chart_div").html("<div
class='alert'><strong>Warning!</strong> You don't have the
permission to view this page.</div>")
{%end%}
```

In caso di esito negativo, viene visualizzato un messaggio di errore.

Altrimenti, se il dispositivo è un sensore di contatto, viene visualizzata una tabella TIMESTAMP/VALORE (Figura 51):

```
{%elif net.ntype=="ztc" and sens.type==2%}
// Create the table for the load control sensor
$("#chart_div").hide();
$("#table_div").html('<tr><td style="font-weight:bold;">TIMESTAMP <i
class="icon-time"></i></td><td style="font-weight:bold;">VALUE <i
class="icon-off"></i></td></tr>');

$.each(data,function(index,value){
$('#table_div >
tbody:last').append('<tr><td>'+value[0]+'</td><td>'+value[1]+'<td></
tr>');
});
```

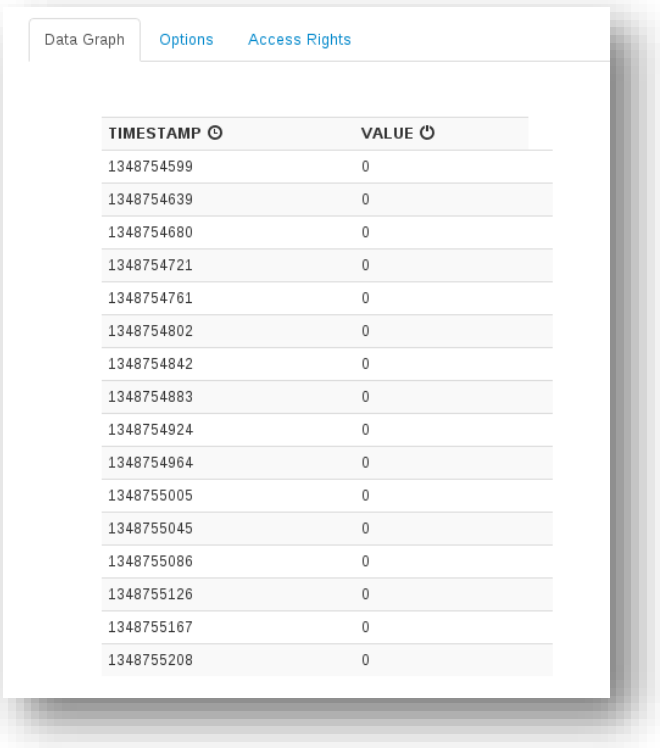


Figura 51 - Visualizzazione dati di un sensore di contatto

Per gli altri tipi di sensori, invece, viene sfruttata la libreria Highstock. Di seguito viene riportato il codice Javascript che consente di impostare il grafico di un Active Voltage Sensor.

```
{%elif net.ntype=="ztc" and sens.type==6%}
// Create the chart for the power sensor
window.chart=new Highcharts.StockChart({
chart:{
renderTo:'chart_div',
},

rangeSelector:{
enabled:false
},

title:      {
text:'Active Voltage sensor',
style:{
fontWeight:'bold',
fontSize:'24px'
}

},
yAxis:{
title:{
text:'Volt',
style:{
fontWeight:'bold',
fontSize:'18px'
}
},

minorGridLineColor:'#F0F0F0',
minorTickInterval:'auto'
},
xAxis:{
title:{
text:'Date/time',
align:'high',
style:{
fontWeight:'bold',
fontSize:'18px'
}
},
gridLineWidth:1,
gridZIndex:1,
```

```
},
series:[{
name:'Volt',
data:data,
type:'spline',
tooltip:{
valueDecimals:2
}

}]
})
```

Il risultato è il visibile in Figura 52.

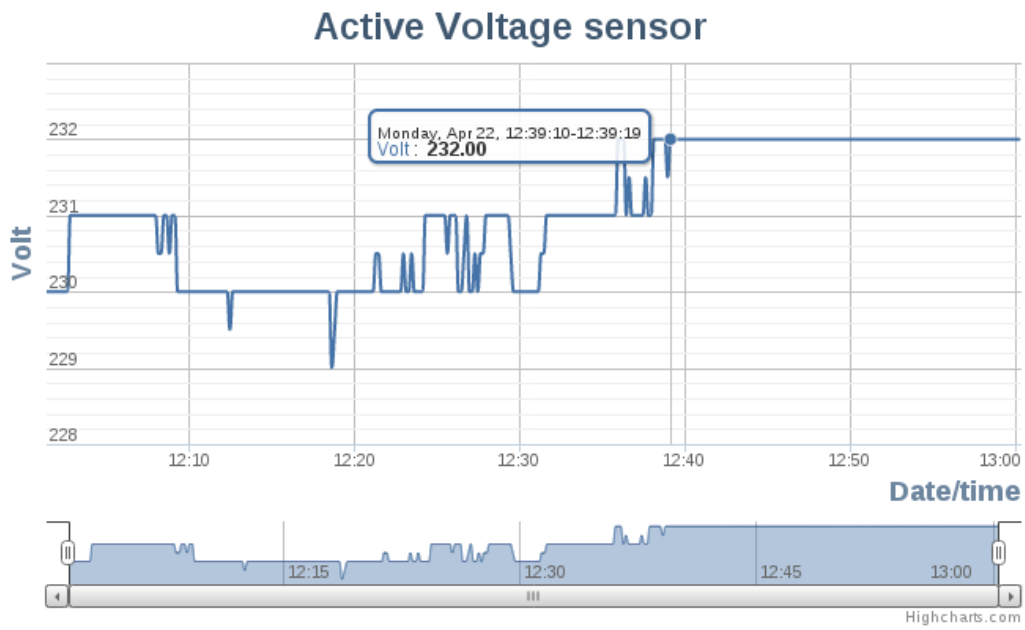


Figura 52 - Grafico dei dati di un Active Voltage Sensor

Per navigare i dati, è possibile utilizzare il navigatore che si trova sotto al grafico stesso o, in alternativa, impostare manualmente un intervallo di data e tempo utilizzando il timepicker (Figura 53) presente di lato a destra.

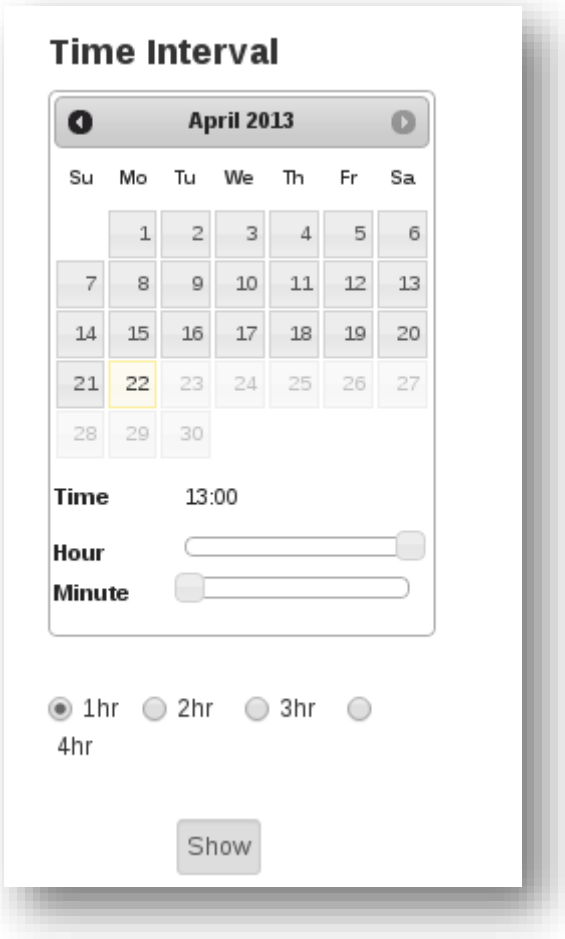


Figura 53 - TimePicker

5.4. Login and session control

Per analizzare questi modi d'uso viene utilizzato l'handler *FakeAuthLoginHandler*.

Questo è un handler che gestisce l'accesso degli utenti manualmente, comunicando direttamente con il database e bypassando l'autenticazione con il servizio esterno di Google.

```
class FakeAuthLoginHandler(BaseHandler):

def get(self):

user={'email':u'mario.rossi@gmail.com','first_name':u'Mario','last_n
ame':u'Rossi','locale':u'it','name':u'Mario Rossi'}

#user = {'email': u'carlo.bianchi@gmail.com', 'first_name':
u'Carlo','last_name': u'Bianchi','locale': u'it','name': Carlo
Bianchi'}

#user = {'email': u'giuseppe.verdi@gmail.com', 'first_name':
u'Giuseppe','last_name': u'Verdi','locale': u'it','name': Giuseppe
Verdi'}

str_time=datetime.datetime.now().isoformat()

self.lock_tables("write",['users'])
usr=self.db.get("SELECT * FROM users WHERE email=%s",user["email"])

if not usr:
# Create user entry in the WSN-database
usr_id=self.db.execute("INSERT INTO users (email, name, last_access)
VALUES (%s,%s,%s)",
user["email"],user["name"],str_time)
```

```
else:

usr_id=usr["id"]
self.db.execute("UPDATE users SET last_access=%s WHERE id=%s",
str_time,usr_id)
self.unlock_tables()

self.set_secure_cookie("user",str(usr_id))
      self.info("Hello <b>"+user["name"]+"</b>!")
self.redirect(self.get_argument("next","/"))

def _on_auth(self,user):
if not user:
raise tornado.web.HTTPError(500,"Fake auth failed")
```

Sono stati creati, a tal scopo, all'interno dell'handler tre utenti fake, per velocizzare la fase di test del sistema MetroPower.

5.4.1. Test controllo degli accessi al sistema

È stato effettuato un test per verificare se l'accesso alle pagine, alle reti ed ai sensori viene correttamente negato se un utente non è connesso o non ha i privilegi necessari. Inizialmente nessun utente è loggato al sistema (Figura 54).

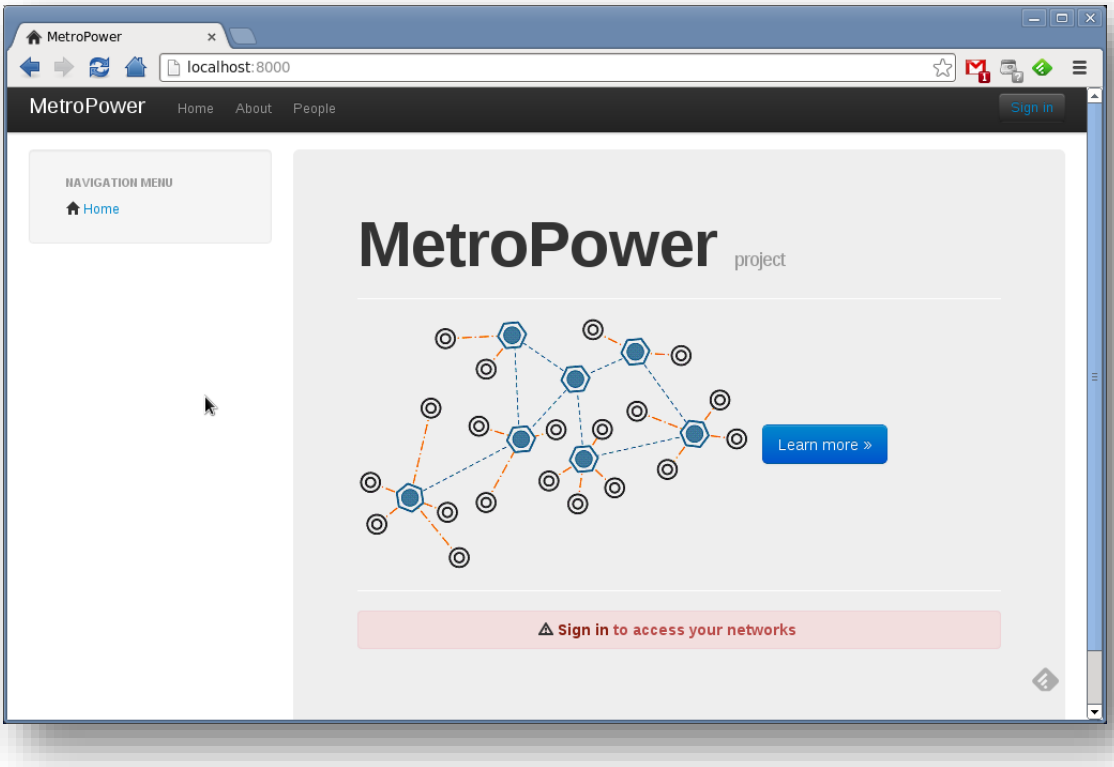


Figura 54 - Pagina di login del sistema MetroPower

Si vede già dalla prima schermata che, se un utente non è loggato al sistema MetroPower, non è in alcun modo visibile la pagina di accesso alle reti. Infatti, nella barra di navigazione superiore, non compare la scheda relativa alle pagine di visualizzazione delle reti.

L'utente "Mario Rossi" accede al sistema (Figura 55).

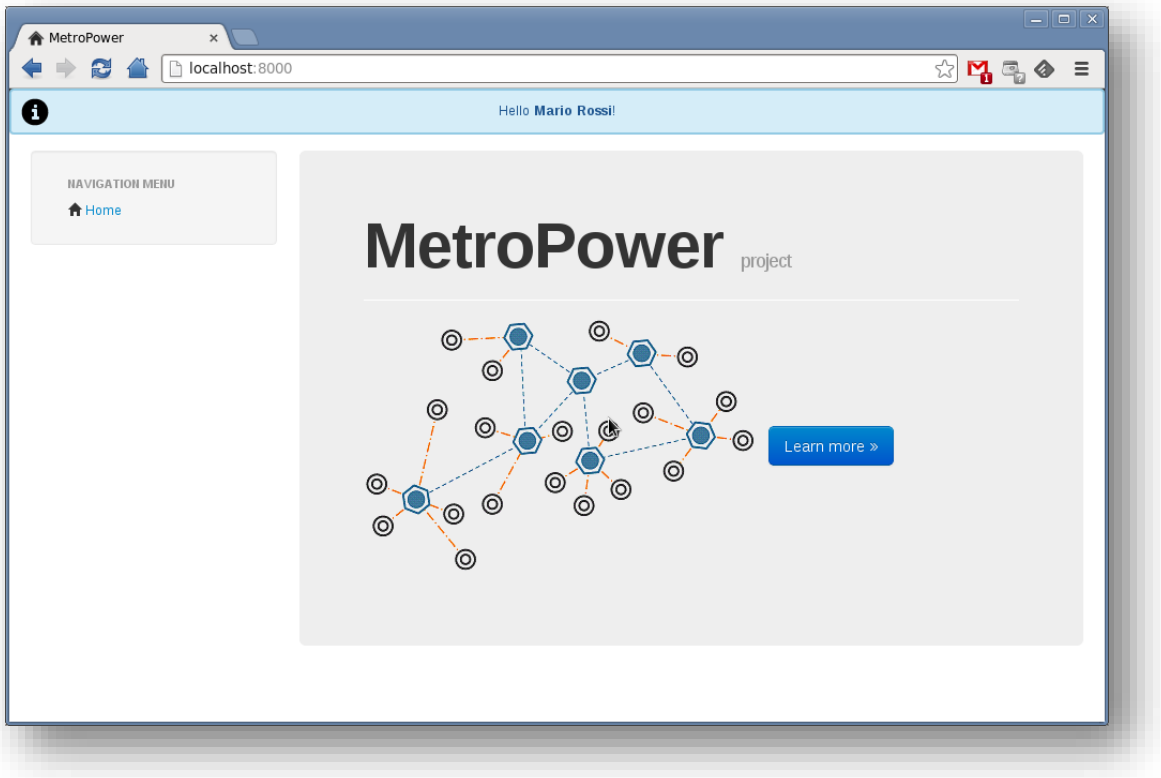


Figura 55 - Login di un utente al sistema MetroPower

Nella pagina di amministrazione l'utente Mario Rossi ha una rete "test case 1" (Figura 56).

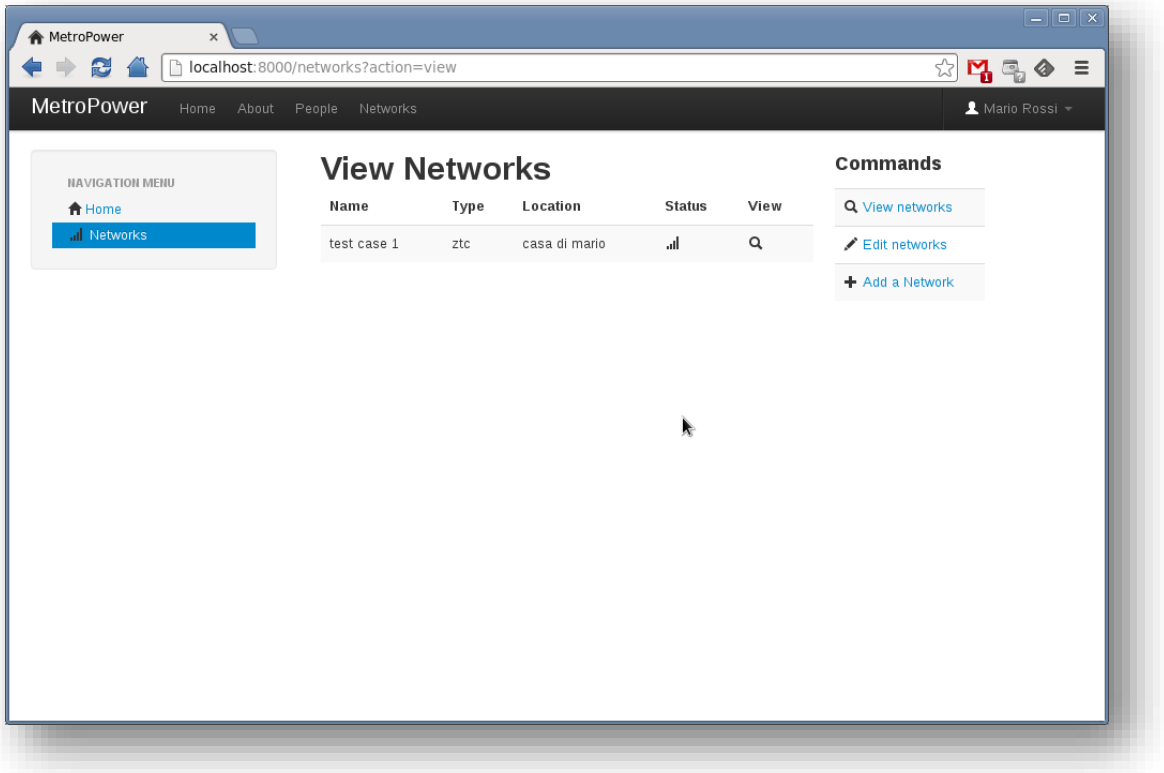


Figura 56 - Pagina di amministrazione di Mari Rossi

Qualsiasi altro utente che si logga al sistema MetroPower non vede la rete di Mario Rossi nella propria pagina di visualizzazione (Figura 57).

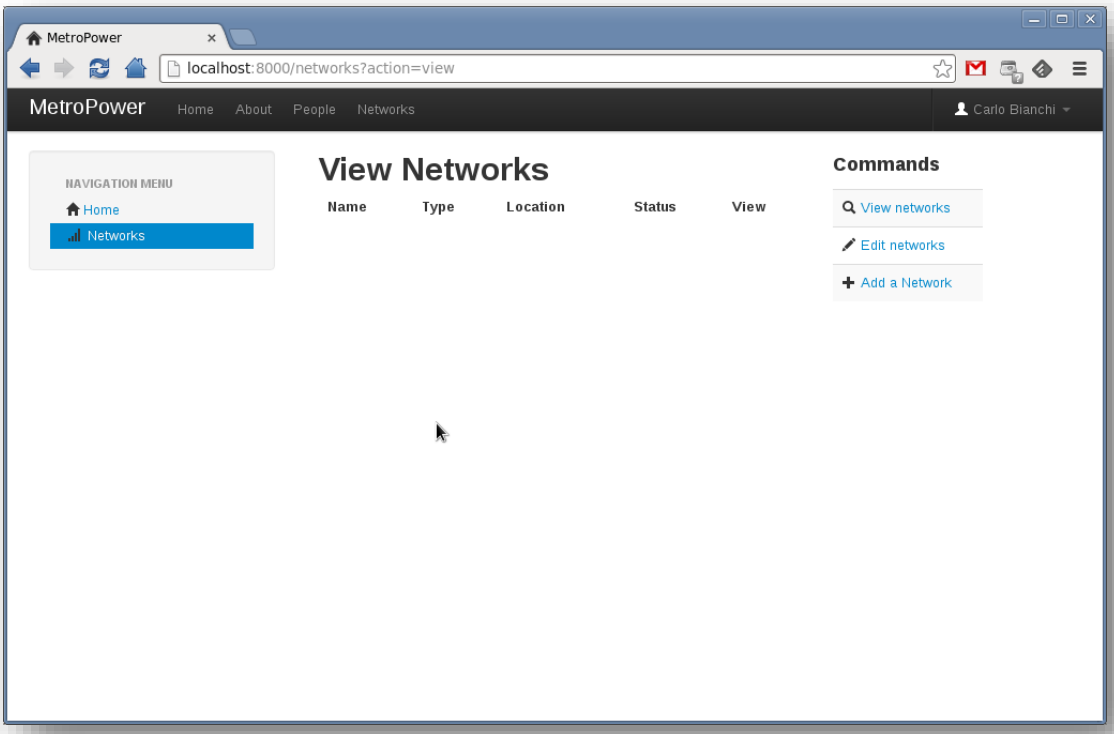


Figura 57 - Pagina di amministrazione di Carlo Bianchi

L’utente Mario Rossi decide di dare il permesso di visualizzare la propria rete al solo utente Carlo Bianchi.

Dalla pagina di amministrazione della rete, nella sezione per l’assegnazione dei permessi agli utenti, è presente la lista degli utenti ai quali Mario Rossi può assegnare diritti sulla propria rete (Figura 58).

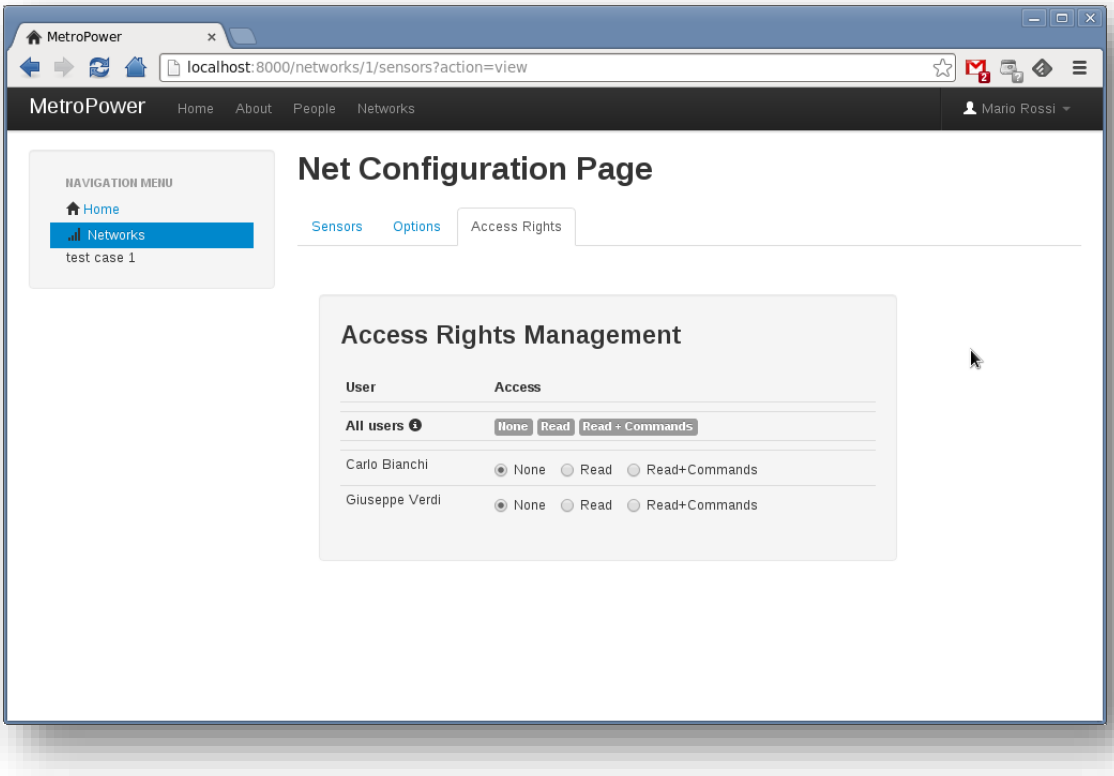


Figura 58 - Pagina di assegnazione dei permessi di Mario Rossi

L’utente Mario Rossi assegna all’utente Carlo Bianchi il diritto di visualizzare la propria rete “test case 1” (Figura 59).

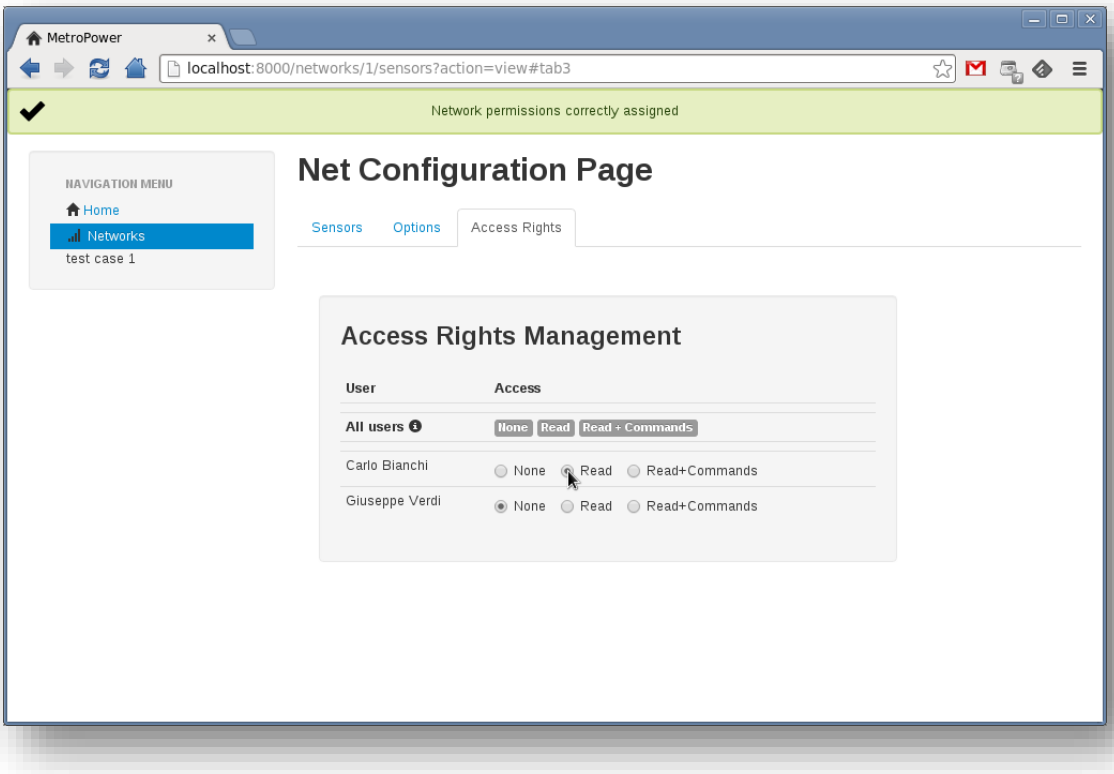


Figura 59 - Assegnazione del diritto di lettura all'utente Carlo Bianchi

Quando l'utente Carlo Bianchi si logga al sistema MetroPower, vedrà nella propria pagina di amministrazione delle reti, la rete "test case 1" dell'utente Mario Rossi (Figura 60).

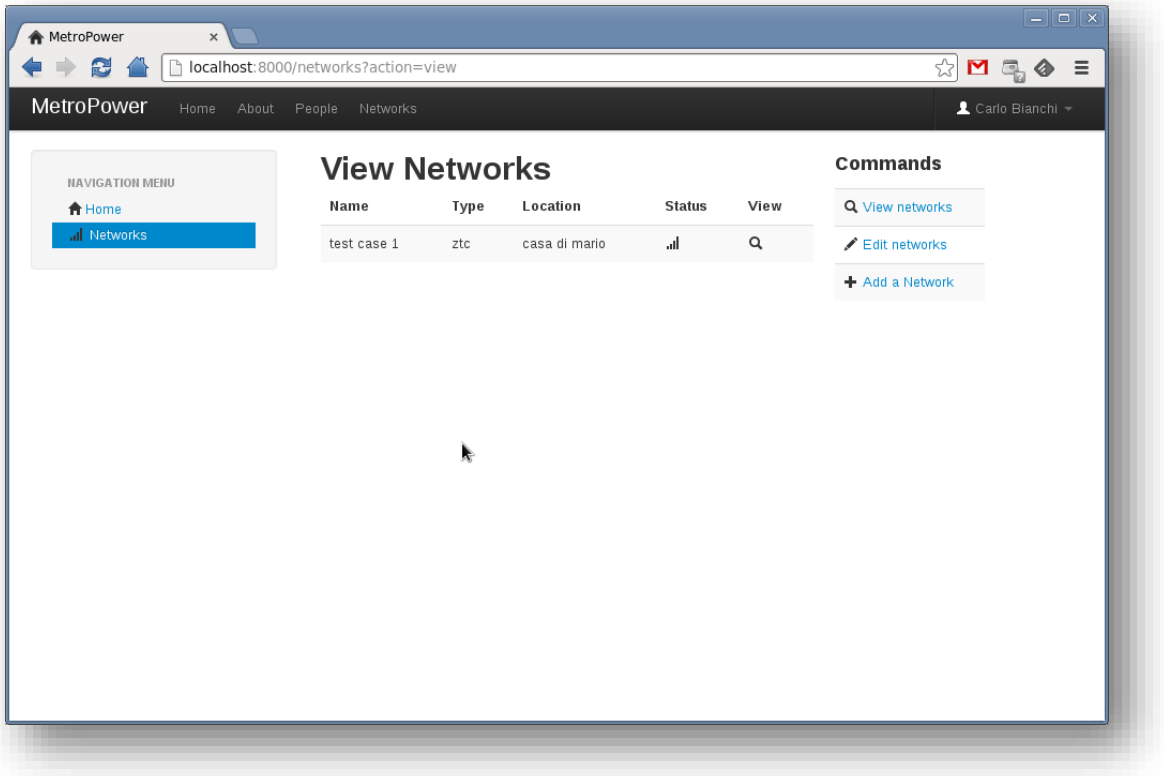


Figura 60 - Pagina di amministrazione delle reti di Carlo Bianchi dopo l'assegnazione del diritto di lettura da parte dell'utente Mario Rossi

Si vede come l'utente Carlo Bianchi non abbia acquisito in alcun modo diritti di amministrazione sulla rete "test case 1". Se Carlo Bianchi vuole

modificare in qualche modo la rete dell'utente Mario Rossi cliccando su "Edit" di lato a destra, questa azione gli viene negata (Figura 61):

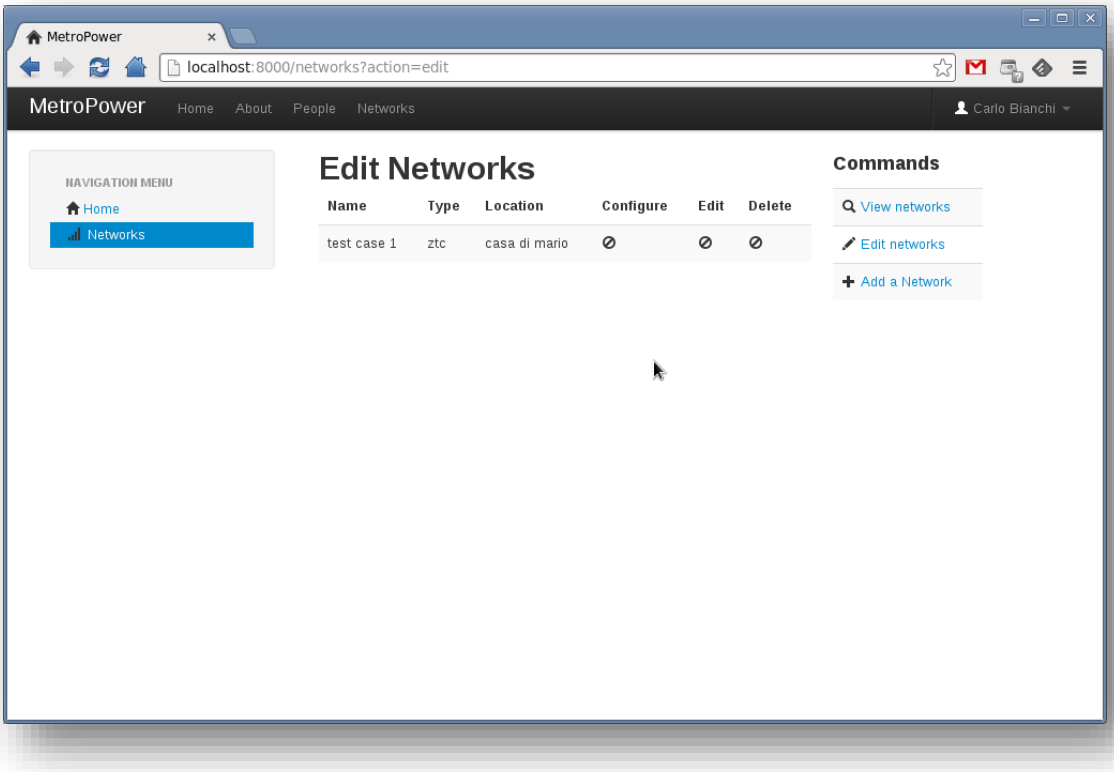


Figura 61 - Modifiche proibite sulle reti a chi non è amministratore

5.4.2. Accesso multiutente

È stato verificato l'accesso al sistema MetroPower da parte di più utenti contemporaneamente, senza verificare alcuna perdita di prestazioni e senza alcun conflitto nell'utilizzo delle risorse del sistema.

Questo grazie a Tornado e a CoMo, due server costruiti appositamente per gestire numerose richieste, senza presentare perdite di informazioni o degrado sostanziale delle prestazioni del sistema.

5.5. Database-driven testing

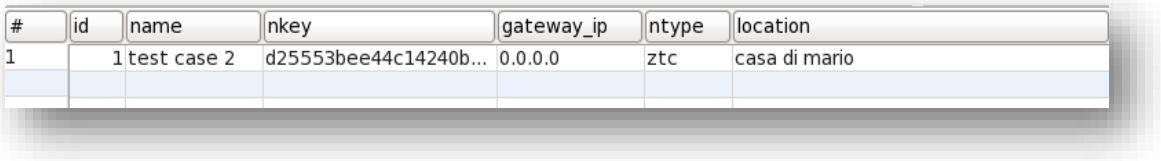
5.5.1. Verificare il corretto funzionamento con DB vuoto

Sono stati eliminati tutti i dati dal database MySQL. Facciamo login con il primo utente, Mario Rossi, e vediamo (Figura 62) che si crea il relativo record nella tabella USERS del database.

#	id	email	name	last_access
1	8	mario.rossi@gmail.com	Mario Rossi	2013-04-24 16:20:08.0

Figura 62 - Salvataggio di un nuovo utente nella tabella CONFCOMMANDS

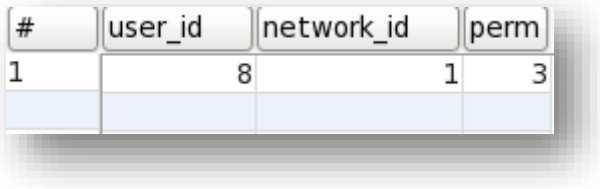
Le tabelle NETWORKS e DEVICES sono vuote. L'utente Mario Rossi crea una nuova rete "test case 2" e il relativo record viene creato all'interno della tabella NETWORKS (Figura 63).



#	id	name	nkey	gateway_ip	ntype	location
1	1	test case 2	d25553bee44c14240b...	0.0.0.0	ztc	casa di mario

Figura 63 - Salvataggio di una nuova rete nella tabella NETWORKS

Inoltre viene assegnato il privilegio di amministratore (perm=3) all'utente Mario Rossi. La tabella NETS_PERMISSIONS riporta un nuovo record corrispondente (Figura 64).



#	user_id	network_id	perm
1	8	1	3

Figura 64 - Assegnazione del privilegio di amministrazione su una rete ad un utente

L'utente con id pari a "8", ovvero Mario Rossi, ha il privilegio di amministrazione sulla rete con id "1", ovvero la rete "test case 2".

A differenza della creazione di una rete, in cui l'utente amministratore compare con permesso pari a "3" nella relativa tabella, quando un utente aggiunge dispositivi alle proprie reti, non viene assegnato nessun permesso di amministrazione nella tabella DEVICES_PERMISSIONS. Questo perché basta il solo permesso di amministratore sulla rete per assegnare lo stesso privilegio su tutti i dispositivi della stessa.

L'utente Mario Rossi aggiunge un dispositivo "sensor 2.1" alla rete "test case 2" (Figura 65 - Figura 66).

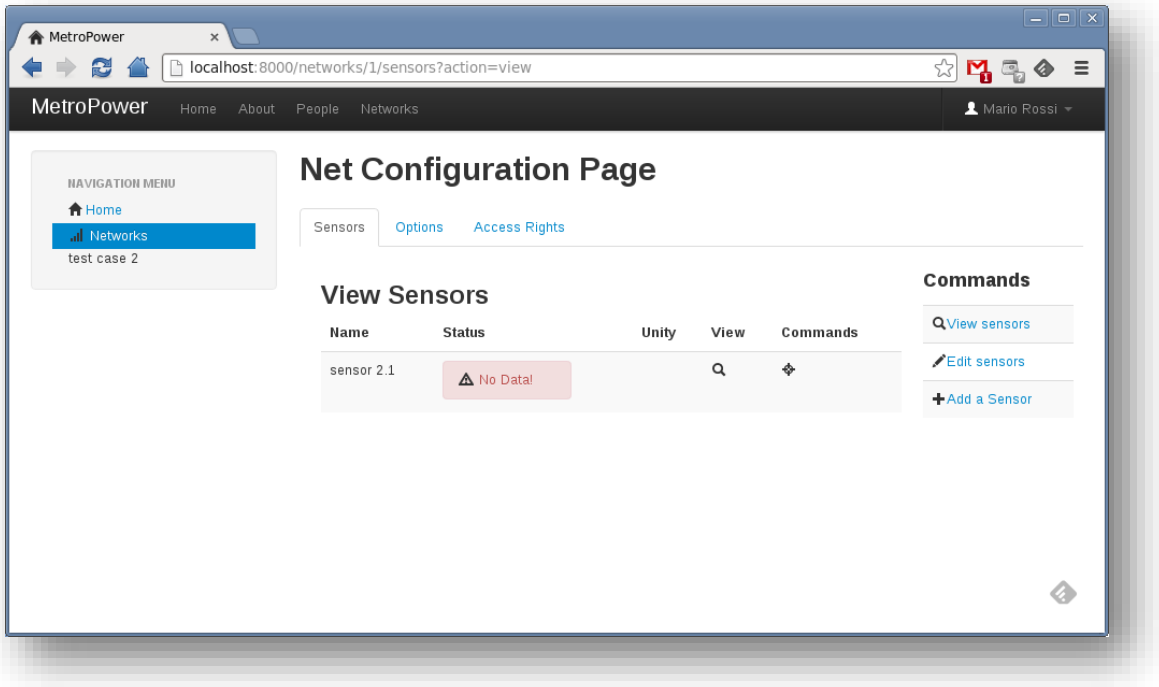
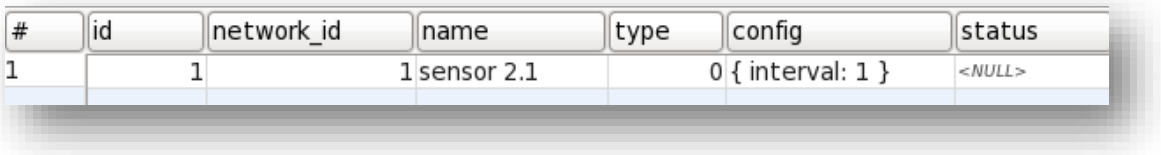


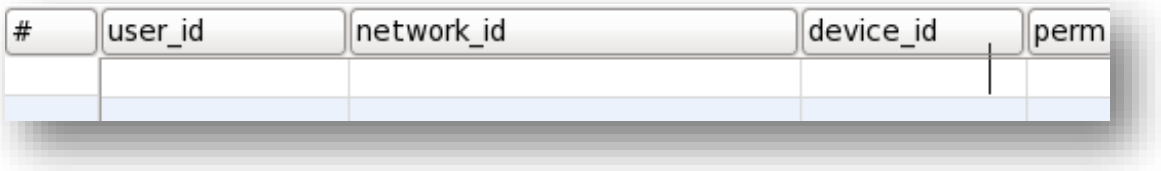
Figura 65 - Aggiunta di un sensore da parte di Mario Rossi

Come si vede in Figura 67, però, non compare alcun permesso nella tabella DEVICES_PERMISSIONS nel database.



#	id	network_id	name	type	config	status
1	1	1	sensor 2.1	0	{ interval: 1 }	<NULL>

Figura 66 - Creazione di un sensore da parte di un utente



#	user_id	network_id	device_id	perm

Figura 67 - Tabella DEVICES_PERMISSIONS vuota

Per verificare che il database funzioni, l'utente Mario Rossi vuole assegnare il permesso di lettura sulla propria rete all'utente Carlo Bianchi, mentre vuole assegnare il permesso di lettura più invio comandi all'utente Giuseppe Verdi (Figura 68).

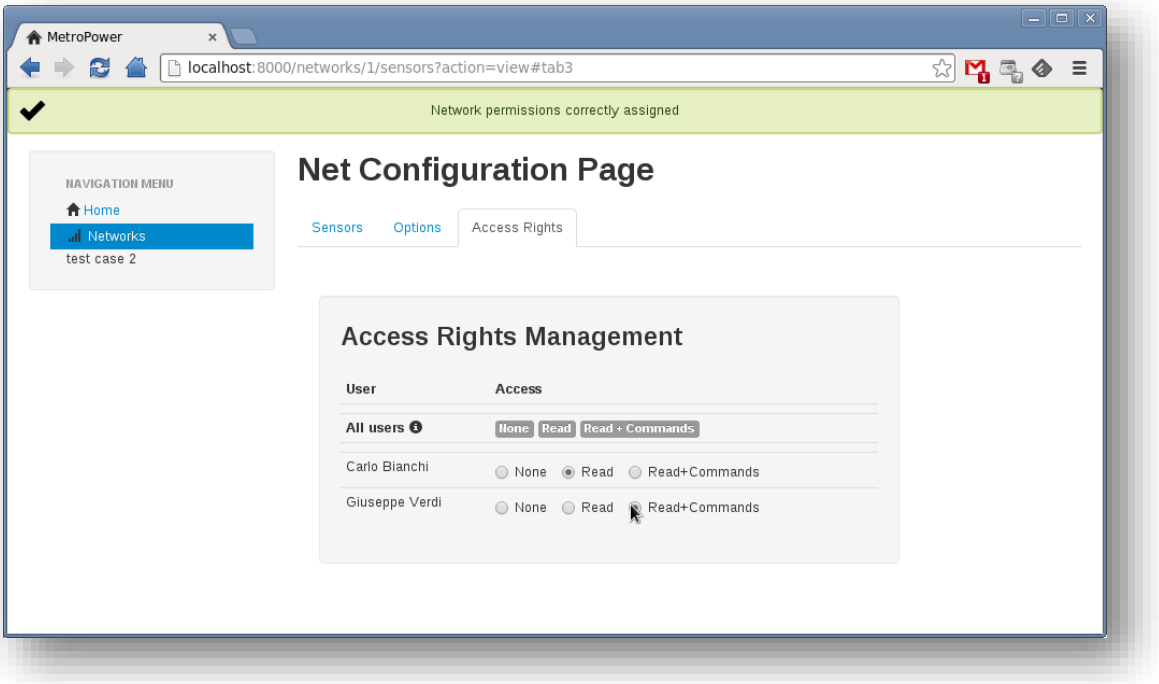


Figura 68 - Assegnazione dei permessi da parte di Mario Rossi

Una volta assegnati i permessi, nella tabella NETS_PERMISSIONS compaiono i relativi record (Figura 69).

#	user_id	network_id	perm
1	8	1	3
2	9	1	1
3	10	1	4

Figura 69 - Permessi assegnati correttamente nel database

Il test di sistema a database vuoto è concluso ed il risultato è positivo.

5.5.2. Verificare il corretto accesso in lettura al DB MySQL

Per verificare il corretto accesso in lettura, continuiamo l'esempio al paragrafo precedente.

Una volta assegnati i permessi di lettura su una determinata rete, gli utenti che hanno ottenuto tale privilegio compaiono nella lista di utenti a cui è possibile assegnare permessi sui dispositivi.

In Figura 70 si vede infatti la pagina di assegnazione dei permessi del sensore “sensor 2.1”.

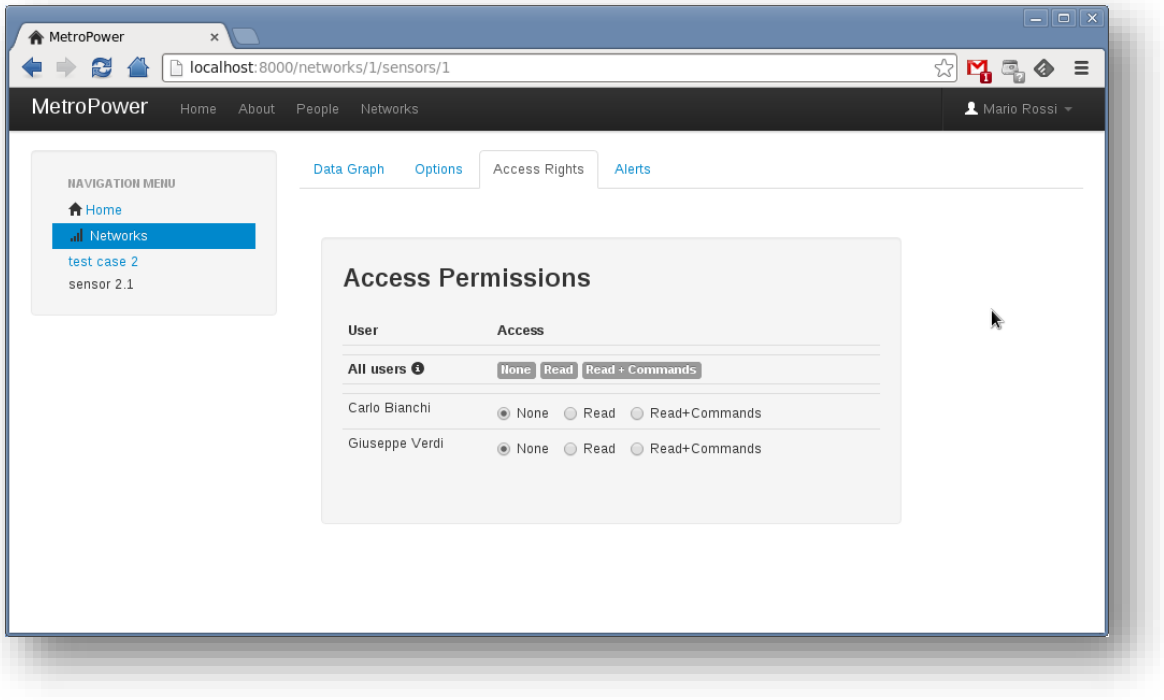


Figura 70 - Pagina di assegnazione dei permessi sui sensori

L’accesso in lettura al database MySQL funziona correttamente.

5.5.3. Verificare la comunicazione con CoMo

Per verificare la comunicazione con CoMo, innanzitutto occorre controllare che il server sia stato correttamente avviato. Per verificare ciò, si fa uno "screen" di CoMo, in modo da vedere se il server invia e riceve messaggi dai sensori. Una volta avviato il server, occorre lanciare il comando

```
screen -d -r como
```

Il risultato è il seguente (Figura 71):

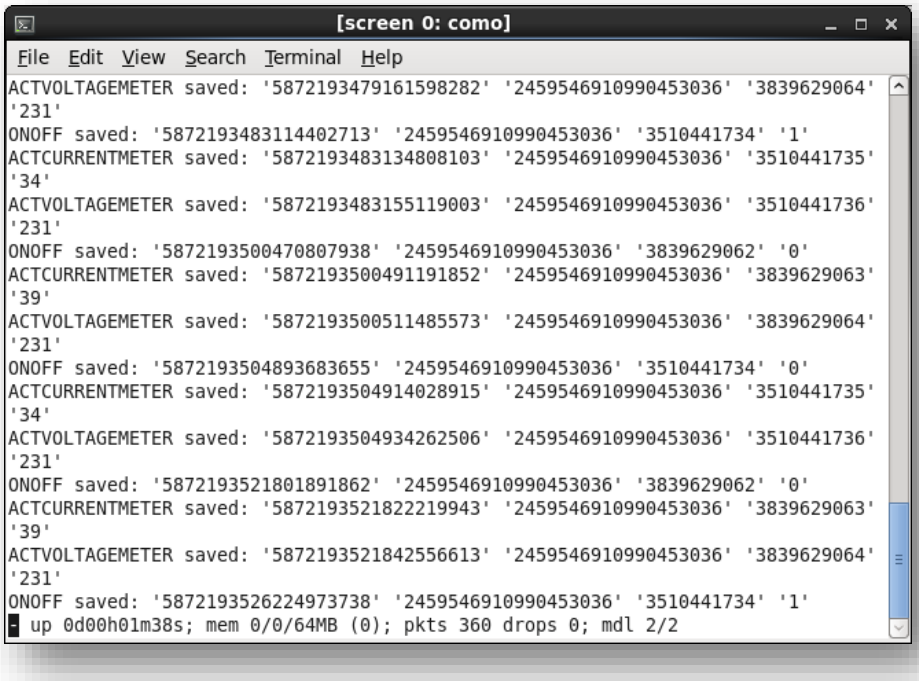


Figura 71 - Screen di CoMo

Dopodiché si va a verificare che il front-end, ed in particolare l'interfaccia del sistema MetroPower, visualizzi correttamente i dati provenienti dagli stessi sensori (Figura 72):

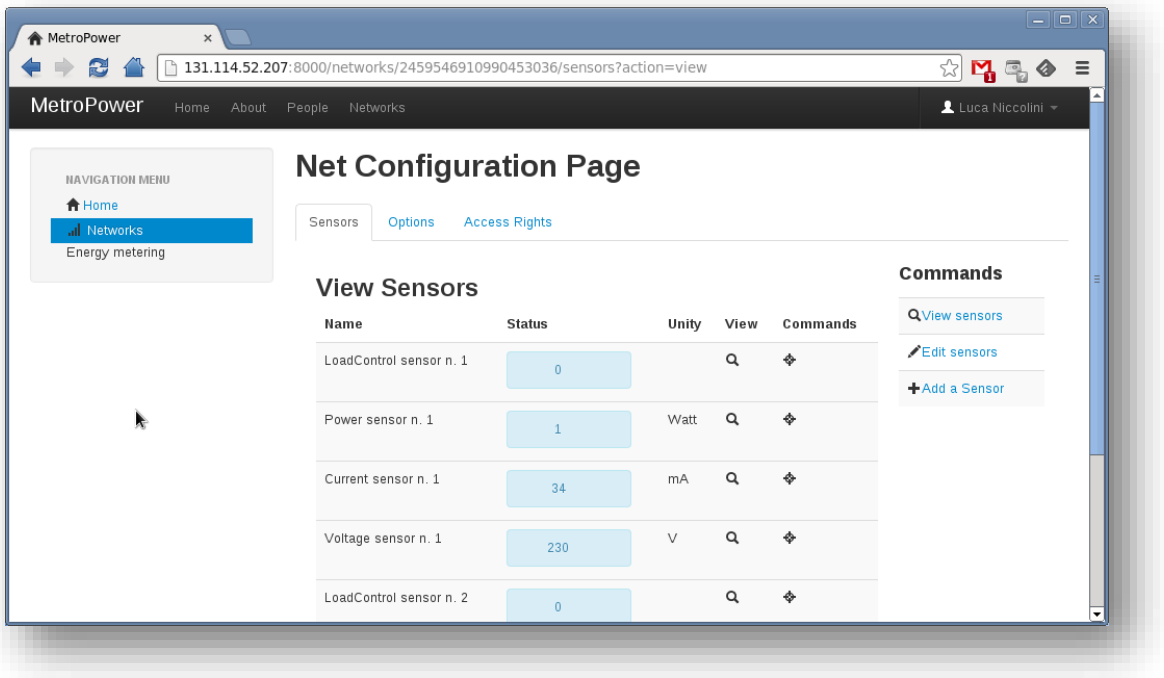


Figura 72 - Corretta comunicazione con CoMo

La comunicazione con CoMo avviene correttamente.

5.5.4. Verificare la corretta scrittura dei comandi nel DB MySQL

Tramite il client visuale “phpmyadmin”, si va a verificare la corretta scrittura dei comandi nel database MySQL.

Si accede al database con le credenziali di accesso.

Si inviano ad un sensore di contatto in sequenza il comando ON e successivamente dopo pochi secondi il comando OFF.

Nella console di Tornado compaiono i due comandi inviati al database (Figura 73).

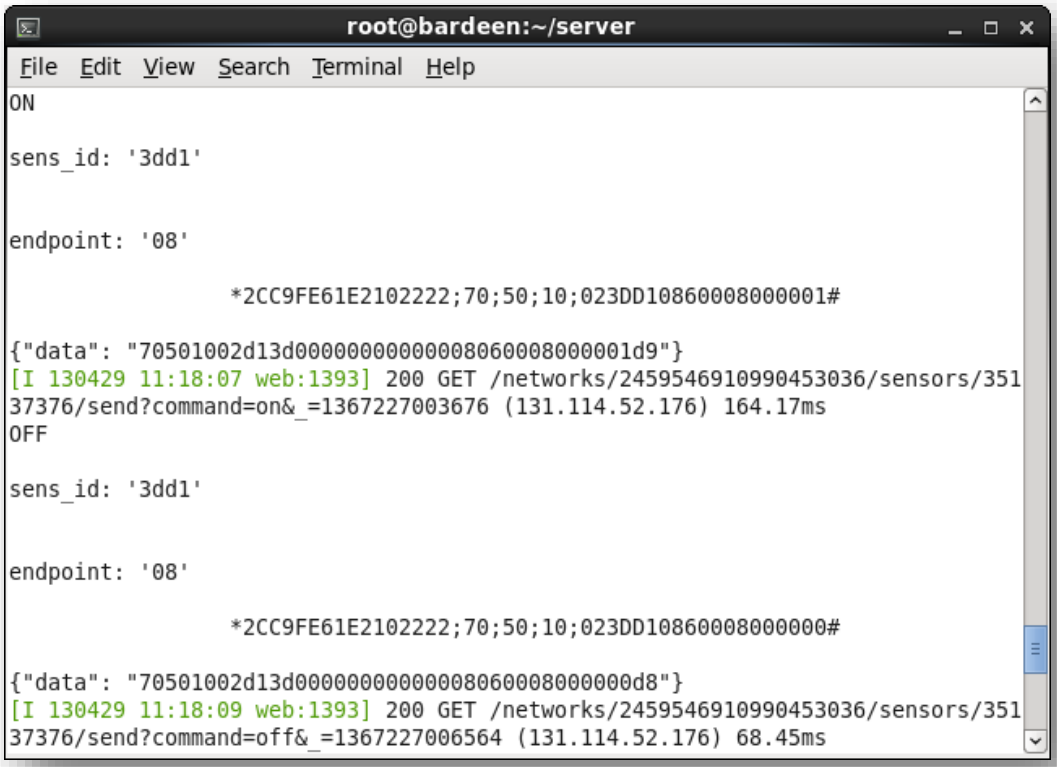


Figura 73 - Screen di Tornado

Nella tabella CONFCOMMANDS del database MySQL compaiono così i due comandi (Figura 74):

seqno	network_id	ntype	timestamp	command
407	2459546910990453036	ztc	2013-04-29 11:16:30	{"data": "70501002d13d0000000000008060008000001d9...
408	2459546910990453036	ztc	2013-04-29 11:16:45	{"data": "70501002d13d0000000000008060008000000d8...

Figura 74 - Comandi correttamente scritti nel database MySQL

La scrittura dei comandi nel database avviene quindi correttamente.

6. Bibliografia

[1] M. Otto, "Building Twitter Bootstrap," 17 Gennaio 2012. [Online]. Available: http://alistapart.com/article/building-twitter-bootstrap.

[2] Italian A List Apart, «La creazione di Twitter Bootstrap,» 3 Febbraio 2012. [Online]. Available: http://www.italianalistapart.com/articoli/57-numero-43-3-febbraio-2012/229-la-creazione-di-twitter-bootstrap.

[3] M. Dory, A. Parrish e B. Berg, Introduction to Tornado, O'Reilly Media, 2012, p. 138.

[4] B. Taylor, "Tornado," 10 Settembre 2009. [Online]. Available: http://backchannel.org/blog/tornado.

[5] «Tornado Documentation,» [Online]. Available: http://www.tornadoweb.org/en/stable/documentation.html.

[6] Highsoft Solutions AS, «Home: Highcharts JS,» [Online]. Available: http://www.highcharts.com/.

[7] The CoMo Project, «Overview: The CoMo Project,» [Online]. Available: http://como.sourceforge.net/index.php.

[8] «The CoMo Project: An Overview,» in *International Workshop on Digital Communications*, Taormina, 2005.

[9] G. Iannaccone, C. Diot, D. McAuley, A. Moore, I. Pratt and L. Rizzo, "The CoMo White Paper," 2004.

[10] G. Iannaccone, «CoMo: An open infrastructure for network monitoring -- Research Agenda,» 2005.

[11] J. Kuan, Learning Highcharts, Packt Publishing, 2012, p. 362.