ALMA MATER STUDIORUM

UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Titolo:

Web of Things:Restful API con Node.js

Relatore

Prof. Omicini Andrea

Presentata da

Martello Luca

Sessione III

Anno Accademico 2017/2018

Indice

Capitolo I

Web of Things, Tecnologie, Ambito.

1.1 Definire l'Internet of things

Negli ultimi anni l'Internet of things (IoT) è diventato uno degli sviluppi più promettenti della tecnologia applicata agli oggetti di uso comune. Basti pensare a quanti dispositivi e elettrodomestici (la vostra televisione se dovessi azzardare un'ipotesi) oggi forniscono possibilità di essere raggiungibili attraverso la rete e fornire servizi aggiuntivi alle mere funzionalità base dell'oggetto in questione.

Potremmo definire l'Iot come un insieme di oggetti fisici che possono essere raggiunti, monitorati, controllati e manipolati da dispositivi elettronici che comunicano attraverso varie interfacce di rete e che possono essere infine connessi a una rete più ampia, Internet appunto.

Tali oggetti fisici possono essere considerati "intelligenti" grazie alla progressiva introduzione di dispositivi integrati (computer e microcontrollori) sempre più potenti, economici e piccoli.Come oggetto intelligente possiamo catalogare ogni oggetto fisico che sia digitalmente potenziato con uno più dei seguenti dispositivi

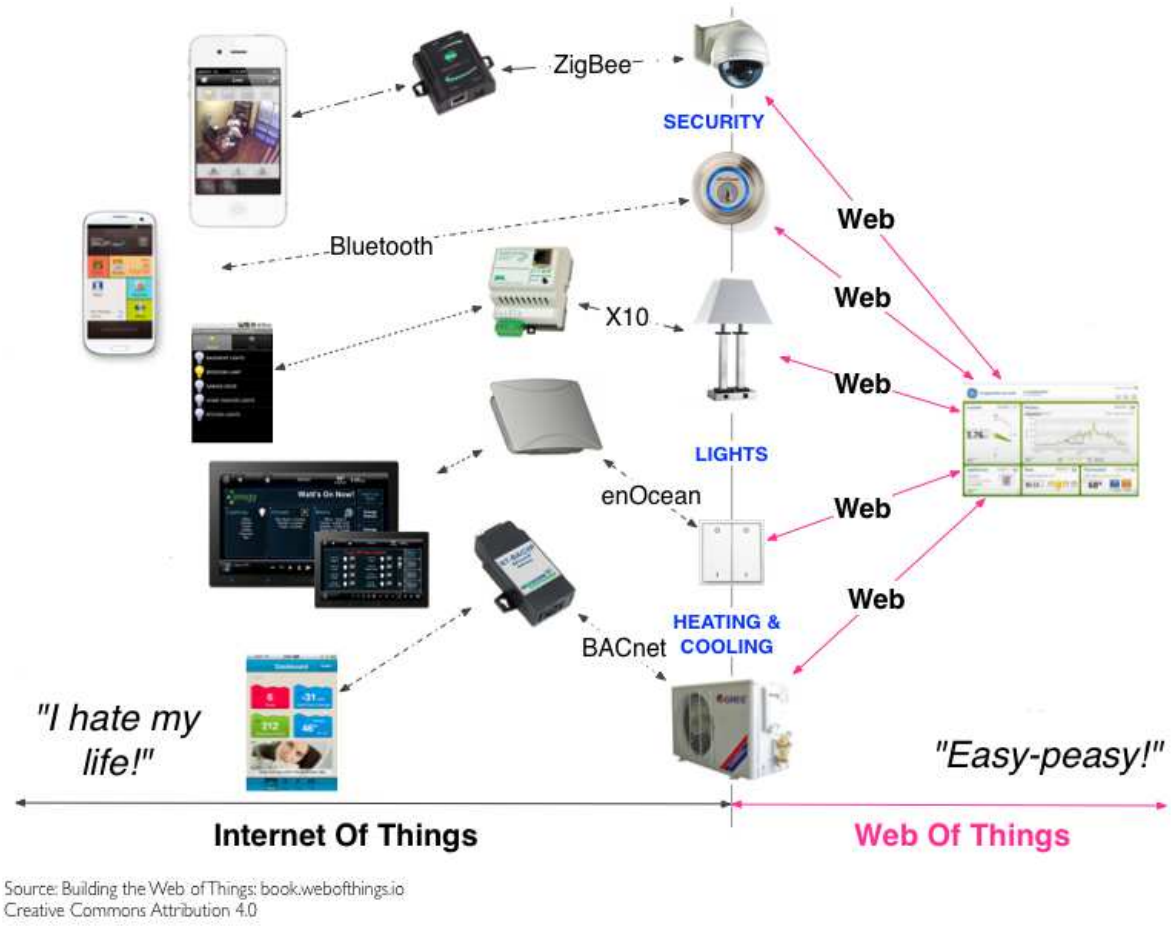
Fino a poco tempo fa i progetti IoT si concentravano principalmente sul costruire dei sistemi chiusi, di piccola scala e generalmente isolati dal mondo esterno in cui i singoli dispositivi dovevano dialogare solo tra loro e non erano progettati per essere acceduti o riprogrammati facilmente.

Questo sopracitato accoppiamento forte tra funzione dell'oggetto e interfaccia applicativa per un dato scenario di utilizzo significa che un eventuale cambiamento nelle funzioni del singolo oggetto o anche una semplice aggiunta di funzionalità risulti costosa sia in termini di tempo sia difficoltosa in quanto richiede competenze specifiche, inoltre limita fortemente le capacità di evoluzione dell'IoT , la grande capacità di evoluzione invece è stata il punto forte del successo del web.

Il Web of Things è una specializzazione dell'IoT che utilizza tutte le pratiche di progettazione che hanno reso il web uno strumento pervasivo e di successo e le trasporta ai dispositivi integrati in modo da rendere lo sviluppo di applicazioni integrate accessibile al maggior numero di sviluppatori possibile.

1.2 Il Web of Things: Scenario tipo

Come illustrerò più avanti le limitazioni dell'Internet of things diventano evidenti appena uno sviluppatore cerca di integrare dispositivi di diversi produttori in una singola applicazione o sistema.
Per fare un esempio astratto e mostrare come il Web of Things può gestire queste limitazioni immaginiamo di voler controllare, connettendo digitalmente, tutti gli elettrodomestici di tutte le stanze di un albergo, in modo che possiamo monitorare, controllare e migliorare la gestione dei consumi energetici attraverso un'applicazione centrale unica.
Creare questo sistema hotel integrato richiederà probabilmente una serratura elettronica costruita e venduta dalla compagnia *Alpha,* telecamere di sicurezza dalla compagnia *Beta* e un'applicazione di controllo per gestire tutto questo fatto dalla compagnia *Gamma.*
Fare in modo che questi dispositive parlino e lavorino tra loro richiederebbe una buona dose di integrazione software e risolvere svariati problemi d incompatibilità.
Potremmo voler contattare una compagnia specializzata e spendere altri fondi in un corposo progetto che richiederebbe mesi per completarsi.
Un progetto tanto complesso avrebbe poca resistenza e affidabilità sarebbe pieno di bug e hack e quindi sarebbe un incubo da mantenere e estendere.
In questo scenario non ce dubbio che i costi sarebbero altissimi e probabilmente supererebbero i benefici prima di riuscire a ottenere il sistema che ci interessa.
Se l'ipotetico proprietario di albergo fosse pratico nel fai da te potrebbe certamente decidere di costruirsi l'intero sistema da solo.
Dovrebbe comprare tutte le attrezzature dalla stessa casa di produzione se non vuole incorrere in problemi di incompatibilità.
Sfortunatamente è improbabile che trovi un singolo produttore che ha tutti i sensori e le attrezzature richieste.
Persino se trovasse questo sistema perfetto, ci sono buone probabilità che l'applicazione di controllo fornita col sistema non sia quello che vuole: facile da usare e configurare.
Probabilmente dovrebbe scrivere un'intera nuova applicazione centro di controllo da solo e se la volesse anche scalabile, affidabile e sicura, ci impiegherebbe il doppio se non il triplo del tempo della versione base.
Potremmo aver fatto un esempio iperbolico ma purtroppo si avvicina a quello che è la realtà dell'IoT moderno.
Non sarebbe fantastico se ogni dispositivo potesse essere facilmente integrato e i suoi dati gestiti da ogni applicazione indipendentemente dai protocolli di rete o standard utilizzati?
Questo è esattamente quello che permette di fare il Web of Thing.



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

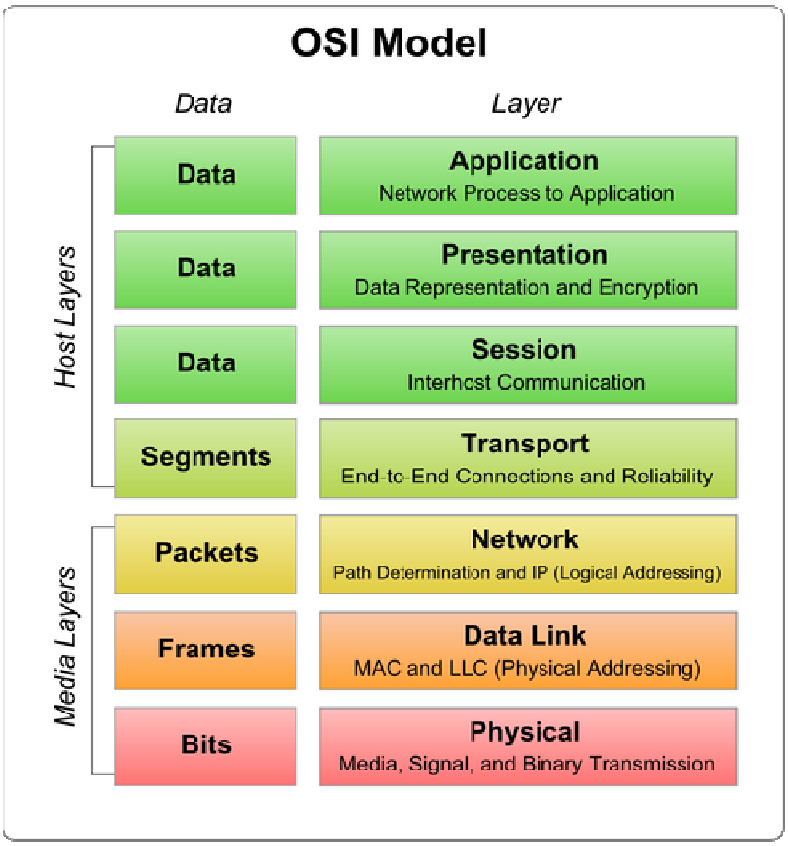
1.3 Comparazione tra IoT e WoT

Poiché sempre più oggetti di uso comune stanno diventando “digitalmente aumentati” il passo successivo più ovvio è utilizzare il World Wide Web e la sua infrastruttura per creare applicazioni per l’Internet of things cercando di eliminare la tendenza che ogni dispositivo debba avere una sua applicazione dedicata.
Sarebbe interessante integrare in ognuno di questi dispositivi la stessa tecnologia che ha permesso a siti come Facebook e Google di servire milioni di utenti contemporaneamente permettendo di scalare il servizio verso numeri più alti o eventualmente bassi secondo le necessità senza compromettere le performance o la sicurezza del servizio.
L’idea di riutilizzare strumenti e tecniche già presenti e consolidate nello sviluppo web e applicarle allo sviluppo di scenari dell’Internet of Things è l’obiettivo finale del Web of Things .**(figura)**
Mentre l’Internet of Things si è occupato di principalmente di risolvere problem di network, il Web of Things si appoggia esclusivamente sugli strumenti e i protocolli dell’Application Layer (livello 7 dell’Open System Interconnection secondo il modello OSI).
Mappare ogni dispositivo con un ottica orientata al Web rende il Web of Things non dipendente dai protocolli adottati dai layer di trasporto (quarto del modello OSI) e fisico (primo del modello OSI)degli stessi dispositivi.

La cosa importante è che attraverso ponti hardware (gateway) è possibile collegare al web un qualsiasi protocollo o standard IoT .
Astrarre la complessità e la diversità dei vari protocolli dei livelli più bassi dietro il semplice modello del web offre molteplici vantaggi.
Proprio come il web è diventato la piattaforma di integrazione globale per applicazioni distribuite su Internet, il Web of Things semplifica l'integrazione di ogni sorta di dispositivo e le conseguenti applicazioni che interagiscono con esso.
In altre parole, nascondendo la complessità e la differenza tra i diversi protocolli di trasporto usati nell'IoT il web of things permette agli sviluppatori di concentrarsi sulla logica delle loro applicazioni senza doversi preoccupare di come un certo protocollo o un certo dispositivo funzioni.
Tornando allo scenario descritto nel paragrafo precedente, se tutti i dispositivi (indipendentemente dal produttore) avessero offerto una API web standard, l'integrazione dei dati attraverso i vari oggetti e applicazioni sarebbe stata naturale poiché tutti i dispositivi avrebbero parlato compreso lo stesso linguaggio applicativo.
In questo caso il proprietario (il responsabile del sistema) avrebbe solo dovuto preoccuparsi di creare l'applicazione centro di controllo per la camera che probabilmente sarà un applicazione web singola che combina diversi dati e servizi da diverse fonti.
Non avrebbe dovuto preoccuparsi di imparare le specifiche di ogni protocollo utilizzato dai diversi dispositivi che vuole usare.
Tutto ciò non solo richiederebbe significativamente meno tempo per essere costruito ma ridurrebbe anche gli sforzi richiesti per mantenere il sistema ogni volta che un dispositivo o un servizio viene aggiunto, rimosso o aggiornato.
Usare HTTP e altri strumenti standard web per interagire con i dispositivi integrati sembra la scelta più semplice per ottenere simili risultati.
Qualche anno fa l'idea sembrava irrealizzabile poichè i web server integrati nei dispositivi IoT avevano risorse hardware molto più limitate dei client che vi accedevano(come browser o smartphone).
Ma le cose sono cambiate:recentemente i web server integrati con features avanzate possono essere implementati con solo 8 KB di memoria.
Grazie all'efficienza ottenuta attraverso ottimizzazioni cross-layer TCP/http possono operare anche su piccoli dispositivi integrati o anche su smartcards.
Inoltre grazie agli enormi sforzi della community JavaScript, è diventato sostanzialmente più semplice spostare buona parte del carico di lavoro dai dispositivi integrati alle applicazioni client e persino a risorse cloud.
Nel Web of Things, I dispositive e I loro servizi sono totalmente integrati con il Web perchè utilizzano gli stessi standard e tecniche dei siti web tradizionali.
Questo significa che è possibile scrivere applicazioni che interagiscono con i dispositivi integrati nella stessa maniera in cui voi interagireste con un qualsiasi altro servizio web che utilizza web API, in particolare le architetture RESTful.

1.4 I Vantaggi del Web of Things

I limiti dell'Internet of things (*IoT*) diventano evidenti appena si vuole tentare di connettere diversi dispositivi di diversi produttori in un'unica applicazione.
La maggior parte dei sistemi IoT non hanno posto molta attenzione al problema di una rete di dispositivi eterogenei che tentano di comunicare tra loro.
Questo perché l'IoT si è concentrato principalmente sui livelli più bassi del modello ISO/OSI (fig.1) e su come i diversi dispositivi potevano comunicare tra



1.Modello OSI

loro non su come facilitare lo sviluppo di nuove applicazioni, ovvero come i dati forniti da tali dispositivi potevano essere raccolti, acceduti e visualizzati.
In particolare non sono stati compiuti molti sforzi per permettere interoperabilità tra diversi dispositivi.
La principale ragione non è tanto tecnica quanto prettamente commerciale.
Una moltitudine di protocolli di comunicazione per l'*IoT* è stata proposta negli ultimi anni da organismi di standardizzazione, consorzi industriali e venditori.
Ma la realtà è che nessuno di questi standard ha raggiunto abbastanza popolarità per diventare " l'unico " protocollo universale dell'*IoT*.
Ancora oggi se si vuole una "Smart House" ed evitare problemi di incompatibilità è conveniente comprare dispositivi dello stesso produttore.
A causa di ciò l'unica opzione possibile per controllare tutto il sistema sarà attraverso le applicazioni che sono fornite dal produttore dell'hardware.

Se quelle stesse applicazioni sono state principalmente sviluppate per iPhone e non sono disponibili per Android sarete costretti a possederne uno.
Se l'applicazione è stata progettata male, è terribilmente lenta, o possiede solo la metà delle caratteristiche che vi occorrono, sarete comunque costretti a utilizzarla.
In parole povere la maggior parte delle soluzioni sul mercato oggi hanno poco in comune con l'Internet(un'unica rete globale, dove tutti i partecipanti sono connessi).
La maggior parte dell'Internet of things oggi sarebbe più propriamente identificato come una "intranet of things" poiché è comparabile come una serie di reti con proprie funzionalità isolate tra loro e che non sono state progettate per comunicare tra loro.
Sebbene un crescente numero di dispositivi connessi offra API per controllarne e accederne ai dati una specifica applicazione deve essere ancora sviluppata per ognuna di quelle API.
Il motivo è che non solo ogni differente dispositivo ha differenti funzionalità ma anche perché ogni API è stata scritta utilizzando un protocollo differente e utilizza un modello dati differente il tutto senza neppure un linguaggio di programmazione standard.
La semplicità e l'apertura del web e dei suoi standard(URL, HTTP, HTML, JavaScript, etc.) sono molto probabilmente la causa della sua diffusione e successo e che lo hanno reso anche world wide.
L'integrazione di una gran varietà di contenuti è stata grandemente semplificata progettando pagine web, browser, server e diversi servizi in modo che parlassero e utilizzassero lo stesso linguaggio applicativo.
Lo stesso unificatore purtroppo non ha ancora trovato la sua strada tra i dispositivi dell'Internet of things.
Nelle prossime pagine descriverò le limitazioni e i problemi che si possono incontrare con gli attuali approcci all'IoT i quali non danno la giusta priorità ad avere un Application Layer Protocol il più semplice possibile per i propri dispositivi.
Per ognuna delle limitazioni illustrerò i benefici di usare un approccio orientato al Web of Things.

- ***Facilità di programmazione***

INTERNET OF THINGS
Il primo problema con le soluzioni e i prodotti in vendita oggi è che molti di quei protocolli sono complessi e difficili da usare.
Questa skill barrier, che anche Internet ebbe negli anni 70, rende l'IoT fuori portata a persone non specializzate nel settore.
Imparare a connettere diversi dispositivi che utilizzano diverse interfacce e protocolli e un'impresa particolarmente ardua che scoraggerebbe chiunque volesse ottenere un prototipo in tempi brevi e con costi ridotti.
Provate a dare un'occhiata alle specifiche del protocollo ZigBee o DPWS(Device Profile for Web Services).

WEB OF THINGS

I protocolli web possono essere facilmente usati per scrivere e leggere dati dai dispositivi e sono inoltre molto meno complessi e piu veloci da imparare (quindi anche piu semplici da mantenere e implementare) dei protocolli IoT.
In aggiunta se tutti i dispositivi potessero offrire una API Web, gli sviluppatori potrebbero usare lo stesso modello di programmazione per interagire con ognuno di essi.
Una volta ottenute le competenze base richieste per costruire semplici applicazioni web, è possibile comunicare rapidamente con nuovi dispositivi con uno sforzo minimo.

- ***Open Standard e estendibilità***

INTERNET OF THINGS
Un altro problema è che molti di questi protocolli evolvono continuamente appena nuovi use-case sono resi possibili da avanzamenti tecnologici.
Siccome questi standard sono finanziati e governati da una singola o comunque un numero limitato di grandi corporazioni non sono così neutrali e malleabili come i progetti open-source guidati da community di sviluppatori.
Inoltre queste compagnie potrebbero decidere di introdurre cambiamenti drastici come più gli fa comodo rendendo i dispositivi esistenti e le loro relative applicazioni incapaci di comunicare tra loro.
Ancora peggio ho riscontrato che la documentazione di alcuni di questi standard non è aperta al pubblico e non possono essere semplicemente utilizzati e implementati senza pagare una quota annuale.
Questo automaticamente limita i loro utilizzo a solo grandi organizzazioni industriali.
I protocolli chiusi e proprietari sono inoltre legati a precisi venditori e fornitori autorizzati.
Assicurarsi che passare a un fornitore differente sia una attività dispendiosa in tempo e denaro è una nota strategia di business delle grandi compagnie software.
Nel contesto IoT le barriere sono ancora più alte poiché cambiare protocollo implica spesso cambiare l'hardware (come utilizzare un chip radio differente).Similmente cambiare protocollo applicativo richiede update al firmware, che sono molto complessi da applicare nel mondo reale.

WEB OF THINGS
Il motive per cui i web standard hanno raggiunto una tale popolarità è che sono completamente aperti e gratuiti, quindi non ce rischio cambino all'improvviso.
Assicurano che i dati vengano trasferiti attraverso diversi sistemi in modo rapido e semplice quindi HTTP e REST sono una scelta ovvia quando si vuole far accedere i dati a un largo pubblico.

- ***Veloce e facile da implementare, integrare e mantenere.***

INTERNET OF THINGS
Poichè l'intero sistema dovrà usare un singolo protocollo, è necessario uno sforzo significativo per integrare dei convertitori ad-hoc ogni volta che un nuovo dispositivo o programma che viene aggiunto.La manutenzione di un sistema con un codice tanto multistrato quanto frammentato richiederebbe ulteriori spese in risorse e tempo.

WEB OF THINGS
Non c'è rischio che il web improvvisamente smetta di funzionare o richieda un aggiornamento e inoltre le possibilità del web non hanno smesso di essere aumentate e migliorate nel tempo.

In contrasto, ci sono sempre nuovi dispositivi e protocolli nel mondo iot e ogni volta che un dei molti protocolli cambia, tutti i pezzi del puzzle che utilizzano quel dispositivo necessitano di aggiornamento.

- ***Disaccopiamento tra le parti***

INTERNET OF THINGS
The implication of the previous sections is most importantly a tight coupling between the devices and applications in the network. Il sistema funziona bene finchè tutte le parti si comportano come previsto e vengono utilizzate come previsto.Sfortunatamente ciò non lascia spazio alla gestione di situazioni anomale o usecase non previsti.

WEB OF THINGS
HTTP è debolmente accoppiato di design poiché i contratti (le specifiche delle API) tra gli attori del web è semplice e ben definito, il che lascia poco spazio alle ambiguità.
Questo permette ogni partecipante di evolvere e cambiare indipendentemente da ogni (almeno finchè il contratto non cambia).Questo è il perché è possibile visitare pagine web che non vengono aggiornate dagli anni 90.
La possibilità per dispositivi IoT di comunicare ai nuovi dispositivi mano a mano che vengono integrati senza richiedere aggiornamenti firmware è essenziale per un web of thing globale.

1.5 JavaScript

Gli esperimenti presenti in questa tesi sono stati eseguiti attraverso JavaScript.JavaScript è un linguaggio di programmazione dinamico nel quale script eseguiti dal web browser lato client possono processare dati in modo asincrono e alterare l'aspetto della pagina web visualizzata.

Grazie al fatto di essere supportato dalla maggioranza dei browser utilizzati normalmente, alla sua relativa semplicità d'uso e flessibilità, JavaScript è diventato la soluzione più diffusa nella realizzazione di applicazioni dinamiche lato client.
Dal lato server, spesso la parte applicativa è implementata usando diversi linguaggi quali PHP, Python, Ruby o Java.Tuttavia anche per il lato server JavaScript sta diventando una soluzione diffusa.Infatti, JavaScript sta essendo adottato sempre più spesso per realizzare applicazioni lato server altamente scalabili in particolare in ambienti runtime come Node.js.
Negli ultimi anni grazie a Node.js(o semplicemente Node) JavaScript è riuscito a infiltrarsi anche tra i dispositivi del mondo reale abbandonando la sua vocazione tipicamente web.
In un settore tipicamente dominato da dispositivi che utilizzano programmi scritti attraverso linguaggi di basso livello come C, JavaScript e Node sono riusciti a emergere come un'alternativa pratica e di facile utilizzo.
Diverse piattaforme software basate su Linux e integrate in dispositivi moderni supportano JavaScript e Node per citarne i nomi più conosciuti Raspberry Pi, Edison dell'Intel e Beagle Board.
Ovviamente un'applicazione integrata che richieda un comportamento assolutamente prevedibile viene meglio realizzata in un linguaggio di basso livello come il C.

JavaScript è stato fortemente criticato dai sui detrattori a causa della mancanza di typing statico e dalla presenza di una miriade di pattern di programmazione e stili che spesso conducono a un codice più difficile da mantenere specialmente in progetti di grandi dimensioni che impiegano un gran numero di persone.
Nonostante questo la sua pervasività , la portabilità e il suo modello ad eventi asincroni rendono JavaScript un ottimo candidato per progetti più contenuti in dimensione e la prototipazione di dispositivi.

1.6 Node.js

Node.js fu originariamente scritto da Ryan Dahl nel 2009 all'incirca tredici anni dopo l'introduzione del primo ambiente JavaScript lato server, LiveWire Pro Web di Netscape.
Node.js è un ambiente run-time, open-source e multi piattaforma per eseguire codice JavaScript lato server.
Con la possibilità di eseguire il codice JavaScript lato server e produrre pagine web dinamiche prima che le stesse siano inviate al web browser dell'utente Node è diventato uno dei pilastri del paradigma "*JavaScript everywhere*", permettendo lo sviluppo di applicazioni web intorno a un singolo linguaggio di programmazione piuttosto che affidarsi a script lato server realizzati in un linguaggio di programmazione differente (per esempio PHP o Python).
Normalmente quando si utilizza PHP con Apache oppure Java con web server come Tomcat si crea un applicazione web e la si avvia su un server esistente mentre con Node il server è l'applicazione stessa.
Un'altra grande differenza tra Node e PHP è che la maggior parte delle funzioni in PHP sono bloccanti fino al loro completamento (i comandi vengono eseguiti solo dopo che precedenti comandi sono stati eseguiti), mentre in Node le varie funzioni sono ideate per essere non bloccanti (i vari comandi vengono eseguiti in parallelo e utilizzano funzioni di callback per segnalare il completamento o il fallimento).
Node possiede un'architettura ad eventi capace di realizzare operazioni input/output (I/O) asincrone, questa scelta di design punta a ottimizzare la velocità di esecuzione e la scalabilità in applicazioni web con molte operazioni I/O e anche per applicazioni web in tempo reale.
Quindi Node porta la programmazione ad eventi ai web server permettendo agli sviluppatori di gestire le applicazioni concorrenti senza ricorrere al threading.
La gestione della programmazione concorrente lato server è complessa in molti linguaggi di programmazione e può condurre a cattive performance(basti pensare alle funzioni PHP bloccanti).Node permette di creare web server e diversi strumenti di rete attraverso JavaScript e soprattutto grazie a una collezione di "moduli" che gestiscono le varie funzionalità base di un server.Proprio come Java ha le sue "*repositories*", Ubuntu ha apt-get e Ruby ha Gem anche Node ha il suo package o module manager chiamato npm.Oltre a essere un package manager per Node, npm funge da package manager anche per JavaScript fornendo un'apposita utility da linea di comando una volta installato.
I moduli utilizzati per la realizzazione dell'API REST sono i seguenti:
-http per poter effettuare richieste HTTP.
-serialport, consente di importare i dati provenienti dalla porta serial desiderata
-msgpack5, si tratta di un efficiente formato per la serializzazione binaria. Come JSON , permette lo scambio di dati tra linguaggi diversi, ma in maniera più veloce e leggera.

-node-json2html, questo modulo implementa il meccanismo di conversione dal formato JSON ad HTML.
A differenza di altre piattaforme, come ad esempio Microsoft Visual Studio, Node è molto leggero, quindi adatto a girare anche su dispositivi poco potenti e con poca memoria (ovviamente se comparati ai personal computer).

1.7 Hypertext Transfer Protocol

L'hypertext transfer protocol (HTTP) è un protocollo a livello di applicativo usato principalmente nella trasmissione di informazioni sul web.
Il funzionamento di HTTP si basa su un meccanismo richiesta- risposta: il client manda una richiesta al server, il quale resistuirà una risposta. Nell'uso comune di questo protocollo, il client corrisponde al browser ed il server alla macchina su cui risiede il sito web desiderato.
A differenza di altri protocolli a livello di applicazione, le connessioni HTTP vengono chiuse quando la richiesta è stata soddisfatta: ciò rende HTTP ideale per il World Wide Web, in cui le pagine spesso contengono link che rimandano ad altre richieste allo stesso server o ad altri, dal momento che viene limitato il numero di connessioni attive a quello effettivamente necessario. Questo ovviamente migliora l'efficienza sia sul client sia sul server.
Una richiesta HTTP è composta da: verbo, URI e versione del protocollo.
L'URI (uniform resrource identifier) indica l'oggetto della richiesta, ovvero la risorsa a cui si vuole accedere. Vediamo invece quali sono i verbi (o metodi) HTTP:

- **GET**. Con tale metodo, il client richiede al server una rappresentazione della risorsa specificata dall'URI. Dal momento che la risposta contiene appunto una rappresentazione della risorsa, GET non va ad incidere in alcun modo sulla risorsa stessa. Inoltre N richieste identiche consecutive, con N>0, daranno sempre lo stesso risultato: Questo verbo è quindi sicuro e idempotente.
- **HEAD**. Il body della richiesta è identico a quello del GET, la risposta invece è diversa: essa non contiene il body. L'HEAD è quindi preferibile al GET quando si vogliono recuperare meta-informazioni scritte negli headers della risposta, evitando così di dover trasferire l'intero contenuto.
- **POST**. Attraverso l'uso del POST, il client richiede al server di creare una nuova risorsa identificata dall'URI. Appare subito chiaro che il POST non è un verbo sicuro né idempotente.
- **PUT**. Il metodo PUT consente di richiedere al server di modificare una risorsa esistente.Se tale risorsa esiste, allora verranno attuate le modifiche desiderate, in caso contrario, la risorsa verrà creata ex novo (in questo modo il PUT diventa un POST).Tale metodo può essere utilizzato, per esempio, per modificare lo stato di attuatori, i parametri soglia di regole e così via. Tale metodo è idempotente, ma non sicuro.
- **DELETE**. Con questo verbo si elimina la risorsa specificata dalla URI. Ovviamente il DELETE non è sicuro, ma solo idempotente.
- **PATCH**. Il patch permette di applicare alla risorsa richiesta delle modifiche parziali. Tale metodo non è né sicuro né idempotente.Vi sono altri tre verbi (OPTIONS, CONNECT, TRACE) che però non verranno trattati in questa tesi.

La prima riga della risposta HTTP viene detta *riga di stato* ed include un *codice di stato* numerico e una frase che lo descrive (es. "Not found").
I codici di stato sono divisi in 5 gruppi principali:

- **1XX Informazione**
- **2XX Successo**
- **3XX Reindirizzamento**
- **4XX Errore del client**
- **5XX Errore del server**

1.8 REST

REST (Representational State Transfer) è uno stile architetturale utilizzato per lo sviluppo di applicazioni distribuite e costituisce il fondamento del Web moderno.
Lo scopo di REST è quello di creare servizi a basso accoppiamento (coupling), in modo tale da poter essere riutilizzati facilmente, per esempio attraverso l'uso di URI's e HTTPs. Se l'architettura di un generico sistema distruibuito segue i principi REST, esso si dice RESTful. Ciò massimizza la scalabilità e l'interoperabilità del sistema, che sono componenti fondamentali del Web.

Vediamo quali sono i vincoli di REST:

1)**CLIENT-SERVER**. Le interazioni tra componenti sono basate su uno schema *richiesta risposta* ovvero un client manda una richiesta ad un server e riceve da esso una risposta.
Ciò permette di minimizzare l'accoppiamento tra componenti, dal momento che il client non necessita di informazioni sulle modalità di funzionamento del server, ma solo sul come inviare ad esso le richieste per accedere ai dati di interesse.
Analogamente, il server non ha bisogno di conoscere lo stato del client o come esso userà i dati.

2)**INTERFACCE UNIFORMI**. Il basso accoppiamento tra componenti può essere realizzato solamente attraverso l'uso di un'interfaccia uniforme che venga rispettata da tutti
i componenti di tale sistema.
Questo punto è essenziale per il funzionamento del WoT, in quanto nuovi e diversi dispositivi vengono aggiunti e rimossi dal sistema in continuazione: è perciò importante limitare tutte le interazioni possibili ad un sottoinsieme limitato di operazioni generiche ben definite dall'interfaccia uniforme.

3)**STATELESS**. . La comunicazione client–server è ulteriormente vincolata in modo che nessun contesto client venga memorizzato sul server. Ogni richiesta da ogni client contiene tutte le informazioni necessarie per richiedere il servizio, e lo stato della sessione è contenuto sul client. Nonostante ciò, lo stato della sessione può essere trasferito al server attraverso un altro servizio posto a persistere, ad esempio la memorizzazione su database.

4)**CACHABLE**. La possibilità di caching costituisce un elemento chiave del Web moderno: si tratta di far si che i clients e i loro intermediari possano salvare alcuni dati localmente, in modo tale che, se il client necessita nuovamente di essi, non occorra fare il fetch sul server per ogni futura richiesta. Ciò implica un minor numero di interazioni client-server, che migliora ovviamente le performance del sistema, riducendo la latenza e migliorando la scalabilità del server.

5)**SISTEMA A STRATI**. Le interfacce uniformi semplificano la fase di progettazione di un sistema a strati, ovvero un sistema in cui diverse componenti intermedie nascondono ciò che sta dietro esse: questo vincolo è strettamente legato a quello del chaching, in quanto i sistemi a strati permettono di fare caching in varie zone del mondo, consentendo ai client di accedere ad alcuni dati più rapidamente.

1.9 Api:Application Programming Interface

Le Application Programming Interfaces (APIs) sono degli insiemi di funzionalità, strumenti e protocolli che vengono messi a disposizione di utenti e programmatori.
Le API forniscono un livello di astrazione che evita al programmatore di sapere come funzionano le stesse ad un livello più basso, favorendo quindi il "riciclo" del codice e di funzioni preesistenti: un esempio classico di API sono le librerie software.
Questi strumenti vengono spesso messi a disposizione dei programmatori da parte di colossi del web, quali Facebook, Google, Amazon e Microsoft, per facilitare lo sviluppo e la realizzazione di applicazioni di vario genere.
Di fatto si può pensare ad un parallelismo che esiste tra le GUIs (graphical user interfaces) e le API. Prendiamo come esempio quello di un'applicazione per le e-mail: il GUI fornisce all'utente dei comandi intuitivi, come dei pulsanti, per andare a leggere le email o spostarle da una cartella all'altra; analogamente un'API fornisce allo sviluppatore delle funzioni per implementare la sua applicazione, come ad esempio quella per copiare i file da una locazione all'altra, senza che egli si debba occupare di come il file system gestisce le operazioni.
Le API sono difatti delle interfacce aperte poste tra lo sviluppatore e i programmi (o parte di essi) che sarebbero altrimenti inaccessibili, ampliandone le funzionalità iniziali.
Per questa tesi la nostra attenzione andrà a concentrarsi sulle API Web: il nostro scopo è quello di fornire un'interfaccia Web che esponga degli endpoints per un sistema basato su messaggi richiesta-risposta.

1.10 Express

Rilasciato come software open-source sotto la licenza MIT, Express è un framework per applicazioni web di Node.js flessibile e leggero che fornisce una serie di funzioni avanzate. Una delle sue peculiarità è quella di fornire allo sviluppatore metodi di utilità HTTP e middleware, semplificando e rendendo più efficiente ed agevole la creazione di APIs.
Il che significa non dover ripetere lo stesso codice in continuazione che in un ottica di mantenere la semplicità nell'implementazione mi sembra fondamentale.
Node.js possiede un meccanismo I/O di basso livello con un modulo HTTP dedicato. Se si vuole utilizzare il modulo http integrato in node molta parte del lavoro di comunicazione come tradurre i payload, gestire i cookie, salvare sessioni in memoria, selezionare il giusto route per ogni azione basandosi sugli strumenti di base di node andrebbe re-implementato. Mentre con Express.js questi procedimenti sono tutti automatizzati. Ultimo ma non meno importante Express.js rimane uno dei framework piu popolari per server node il che rende molto più semplice trovare supporto attraverso la già vasta comunità presente online.

1.11 JSON

JSON è l'acronimo di JavaScript Object Notation. E' un sottoinsieme di JavaScript, rappresenta una semplice soluzione per salvare informazioni in modo organizzato. Fornisce la possibilità di avere dati in memoria che sono facilmente leggibili e possono essere acceduti attraverso JavaScript in modo diretto. JSON ha un formato leggero composto da solo testo il che semplifica il trasferimento di dati tra un server e un'applicazione web. Il suo formato è utilizzato per serializzare e trasmettere strutture dati attraverso Internet. La sua semplicità e flessibilità gli permettono di essere usato attraverso differenti applicazioni, linguaggi applicativi e framework. Sebbene sia fortemente associato a JavaScript, JSON ha un formato che è indipendente dal linguaggio utilizzato e che può essere usato anche con altri linguaggi di programmazione Python, PERL, Java, Ruby e PHP. E' possibile trovare funzioni metodi e procedure già integrati che permettono di utilizzare JSON come strumento di rappresentazione delle risorse nei linguaggi sopra citati. Utilizzando una funzione JavaScript già integrate si può facilmente trasformare una stringa JSON in un oggetto nativo JavaScript che può essere usato come ogni altro oggetto JavaScript in un applicazione. Altri linguaggi di programmazione forniscono già loro funzioni per convertire dati JSON in formati più congeniali a loro. In definitiva per favorire la massima compatibilità e rimanere fedeli all'inclusività di Internet per quando riguarda lo scambio di dati JSON è considerata la scelta più diffusa.

1.12 Ambito applicativo

Questa tesi si propone di sviluppare un'API REST per il monitoraggio e controllo delle luci, della temperatura e umidità di una camera, i cui valori vengono acquisiti attraverso l'uso di sensori collegati ad una scheda Raspberry-p.
Lo scopo è quello di progettare e realizzare una serie di funzioni che permettano il monitoraggio di vari sensori e renderli allo stesso tempo monitorabili attraverso il web secondo i principi RESTFUL e utilizzando l'HTTP come protocollo di trasporto.
Ovviamente la nostra attenzione andrà a concentrarsi più sull'API REST che non sul dispositivo, in quanto i sensori considerati sono fittizi.
Siccome il dispositivo opera da server per il proprio hardware grazie a Node.js, con una connessione Internet sarà possibile interpellare i singoli sensori da un terminale remoto e grazie alle rappresentazioni offerte visualizzarle in html con un semplice browser.

Capitolo 2

Progettazione

Durante la fase di progettazione, ci sono 4 step da tenere in considerazione per ottenere dei buoni risultati:

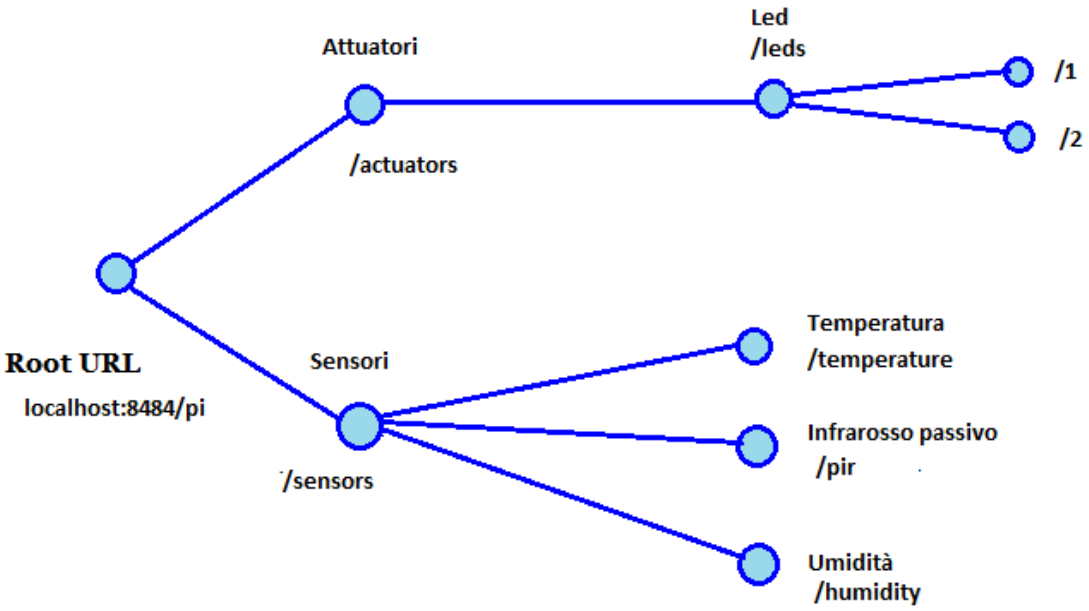
- Definire una strategia di integrazione
- Progettare le risorse
- Decidere quali rappresentazioni fornire
- Progettare un'interfaccia

2.1 Strategia di integrazione

E'opportuno scegliere uno schema adeguato per integrare le cose col web e con Internet.
Per la tesi abbiamo deciso di proporre uno **schema a integrazione diretta** con la api direttamente disponibile sul dispositivo in quanto lo stesso è in grado di supportare un sistema operativo e un applicazione server (Node.js).

2.2Progettazione delle risorse: occorre identificare le funzionalità e i servizi offerti da un oggetto, per poi organizzarli gerarchicamente.

Nel nostro caso, il nodo principale (Root URL) ha come figli i nodi degli Attuatori e quello dei sensori: questi ultimi hanno come nodi foglia dei sensori veri e propri (temperatura, umidità, PIR, led).



2.3 Il modello delle risorse

Le risorse sono funzionalità o servizi offerti da un oggetto. In questa API, le risorse sono costituite da sensori e attuatori: i sensori fanno riferimento ad oggetti fisici che, una volta collegate alla scheda Raspberry, si occupano di raccogliere i valori di temperatura e umidità della stanza.
Per questo le sotto-risorse della route 'sensors' sono i tre sensori presi in considerazione per questo progetto: temperatura, umidità e presenza umana.
Gli attuatori della route actuators invece sono i led: queste risorse supportano anche il verbo PUT per poterne controllare l'accensione e lo spegnimento da li la decisione di chiamarli attuatori.
Di seguito sono mostrati i modelli JSON dei sensori e attuatori che sono inoltre accompagnati dalle informazioni riguardanti il nostro server.(*/resources/resources.json*)


```
{
"pi": {
  "name": "WoT Pi",
  "description": "A simple WoT-connected Raspberry PI",
  "port": 8484,
  "sensors": {
    "temperature": {
      "name": "Temperature Sensor",
      "description": "An ambient temperature sensor.",
      "unit": "celsius",
      "value": 0,
      "gpio": 12
    },
   "humidity": {
     "name": "Humidity Sensor",
     "description": "An ambient humidity sensor.",
     "unit": "%",
     "value": 0,
     "gpio": 12
   },
   "pir": {
     "name": "Passive Infrared",
     "description": "A passive infrared sensor.When 'true' someone is
present.",
     "value": true,
     "gpio": 17
   }
},
 "actuators": {
   "leds": {
     "1": {
       "name": "LED 1",
       "value": false,
       "gpio": 4
     },
     "2": {
       "name": "LED 2",
       "value": false,
       "gpio": 9
      }
    }
   }
  }
}
```

Fatto ciò ho reso disponibile attraverso al file *model.js* il nostro schema JSON con un *exports* JavaScript.(*resources/model.js*)

```
var resources = require('./resources.json');
module.exports = resources;
```

2.4 Le route Express

Ora dobbiamo collegare queste risorse ai diversi URL a cui il web server risponderà.
Per routing si intende determinare come un'applicazione risponde a una richiesta client a un endpoint particolare, il quale è un URI (o percorso) e un metodo di richiesta HTTP specifico (GET, POST e così via).Ciascuna route può disporre di una o più funzioni dell'handler, le quali vengono eseguite quando si trova una corrispondenza per la route.

La definizione della route ha la seguente struttura:

```
app.METHOD(PATH, HANDLER)
```

Dove:

- app è un'istanza di express.
- METHOD è un metodo di richiesta HTTP.
- PATH è un percorso sul server.
- HANDLER è la funzione eseguita quando si trova una corrispondenza per la route.

Creiamo delle route Express una per i sensori e una per gli attuatori, mostro di seguito il file dei sensori:

```
var express = require('express'),
router = express.Router(),
resources = require('./../resources/model');

router.route('/').get(function (req, res, next) {
res.send(resources.pi.sensors);
});

router.route('/pir').get(function (req, res, next) {
res.send(resources.pi.sensors.pir);
});

router.route('/temperature').get(function (req, res, next) {
res.send(resources.pi.sensors.temperature);
});

router.route('/humidity').get(function (req, res, next) {
res.send(resources.pi.sensors.humidity);
});
```

```
module.exports = router;
```

2.5 L’applicazione Express

Ora che abbiamo le routes bisogna caricarle all’interno di un server HTTP.
In pratica quello che viene mostrato nel codice seguente è un server HTTP incapsulato al framework Express.

```
var express = require('express'),
  actuatorsRoutes = require('./../routes/actuators'),
  sensorRoutes = require('./../routes/sensors'),
  resources = require('./../resources/model'),
  converter = require('./../middleware/converter'),
  cors = require('cors'),
  bodyParser = require('body-parser');

var app = express();

app.use(bodyParser.json());

app.use(cors());

app.use('/pi/actuators',  actuatorsRoutes);
app.use('/pi/sensors',  sensorRoutes);
app.use('/things',  thingsRoutes);

app.get('/pi',  function (req,  res) {
  res.send('This is the WoT-Pi!')
});

// per scegliere la rappresentazione corretta
app.use(converter());
module.exports = app;
```

2.6 Progettazione delle rappresentazioni

Un'altra questione importante è quella riguardante la scelta del formato della rappresentazione che l'API fornirà per ogni risorsa.
In questo progetto si è optato per una rappresentazione JSON con possibilità di conversione in HTML, viene inoltre inserito un decodificatore per **msgpack5** per dare un ulteriore selezione ancora più efficiente di JSON.
MessagePack è un formato per la serializzazione binaria. Permette di scambiare dati tra diversi formati come JSON ma in maniera più rapida e con minore consumo di byte. I numeri

interi di piccole dimensioni vengono ridotti a singoli byte e le stringhe necessitano di un singolo extra byte prima di venire serializzate.
Conviene adoperare l'accorgimento di fornire di default una rappresentazione JSON della risorsa richiesta nel caso in cui il formato richiesto non corrisponda a nessuno dei tre previsti.
Questa scelta è mossa da motivazioni ben precise: innanzitutto, sarebbe superfluo soffermarsi eccessivamente sul formato della rappresentazione per poterne fornire molteplici, in quanto questo argomento esula dall'obiettivo principale della tesi. Inoltre, sono stati scelti questi due formati poiché HTML (HyperText Markup Language) è il linguaggio più diffuso per la formattazione e l'impaginazione di documenti nel World Wide Web, mentre JSON (JavaScript Object Notation) è il formato più adatto per lo scambio di dati tra applicazioni client-server, oltre ad essere sia facilmente comprensibile e scrivibile dagli esseri umani sia dalle macchine che possono analizzare e generare dati JSON.
Ci sono diversi modi di supportare altre rappresentazioni in Express, si è scelto un approccio modulare basato sul middleware pattern. Molte librerie Node, incluso Express sostengono l'idea di concatenare funzioni che hanno accesso agli oggetti request e response (req, res).

Di seguito viene mostrato il codice contenuto in */middleware/converter.js* che si occupa di convertire la rappresentazione da JSON a HTML, e da JSON a msgpack5.

```
// richiede i 2 moduli e inizializza il codificatore a msgpack
var msgpack = require('msgpack5')(),
  json2html = require('node-json2html');
  encode = msgpack.encode,

module.exports = function () {
//In Express,  il middleware è una funzione che richiede un'altra funzione

  return function (req,  res,  next) {
    console.info('Representation converter middleware called!');
 // Controlla se il middleware ha lasciato un risultato in req.result
  if (req.result) {
      switch (req.accepts(['json',  'html',  'application/x-msgpack'])) {
//Sceglie la rappresentazione appropriata a seconda dell'Accept header

        case 'html':
          console.info('HTML representation selected!');
          var transform = {'tag': 'div',  'html': '${name} : ${value}'};
          res.send(json2html.transform(req.result,  transform));
//Se è richiesto HTML usa json2html per trasformare JSON in semplice HTML

          return;
        case 'application/x-msgpack':
          console.info('MessagePack representation selected!');
          res.type('application/x-msgpack');
          res.send(encode(req.result));
//Codifica il risultato JSON in MessagePack e ritorna il risultato al client
```

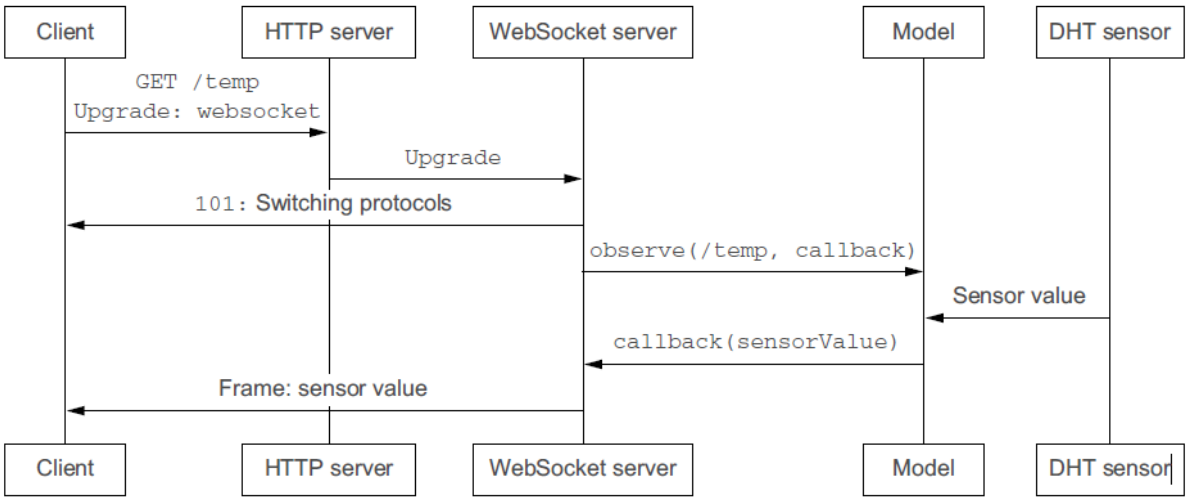
```
        return;
      default:
//Per altri formati il default è JSON

        console.info('Defaulting to JSON representation!');
        res.send(req.result);
        return;
    }
  }
  else {
    next();
//Se non ci sono dati in req.result,  chiama il middleware successivo

  }
 }
};
```

2.7 Progettazione dell'interfaccia

Si decidono i comandi possibili per ogni servizio, ovvero come è possibile manipolare le risorse attraverso l'API REST.
Nel nostro caso, i sensori saranno accessibili attraverso il metodo GET, che restituirà nel formato richiesto le informazioni riguardanti la risorsa.
Per i led è inoltre disponibile il PUT, ovviamente per accenderli e spegnerli.
Per questa API RESTFUL non sono state previste tutte le azioni possibili, ovvero i metodi descritti nel capitolo precedente, ma solamente due: GET e PUT.
Il GET, come già detto, permette accedere ad una rappresentazione della risorsa richiesta, senza andare ad incidere o a modificare in alcun modo quest'ultima.
Per il nostro scopo è infatti fondamentale solo poter accedere ai dati acquisiti dai sensori oppure con il PUT forzare l'accensione e lo spegnimento dei led.

2.8 Interfaccia Pub/sub con WebSockets

L'ultima parte nel design dell'interfaccia è stato permettere anche il supporto a una connessione TCP attraverso i WebSockets.
L'obiettivo è offrire la possibilità di sottoscriversi ai cambiamenti di stato di un sensore attraverso Websockets con un semplice upgrade del protocollo senza dover così continuare a richiedere continuamente la pagina web generando comandi GET in attesa di un nuovo valore del sensore in questione.
Immaginate dover richiedere in continuazione lo stato di un sensore infrarosso che potrebbe cambiare di stato solo per pochi secondi al giorno certamente continuare a inviargli richieste non è il metodo più efficace.
Ci sono diverse implementazioni dei Web Sockets per Node.js, per il nostro server ho scelto WS un'implementazione minimalista ma ad alte performance.
L'implementazione è illustrata di seguito.
.



Si crea un WebSocket server e si inserisce un listener al server HTTP di Express in ascolto di richieste di upgrade del protocollo a WebSocket.
Infine si prendono le richieste di upgrade e usi l'URL della richiesta per osservare i cambiamenti nel modello delle risorse.
Appena si verifica un cambiamento si notifica al client attraverso la connessione WebSocket.
Come client ho realizzato una semplice pagina HTML con il relativo script per la richiesta di upgrade di connessione a WebSocket.
Di seguito il codice nella sezione "script" del file *\serverweb\WSclient\websocketsClient.html*

```
function subscribeToWs(url,  msg) {
        var socket = new WebSocket(url);
```

```
    socket.onmessage = function (event) {
        console.log(event.data);
    };
    socket.onerror = function (error) {
        console.log('An error occurred while trying to connect to a Websocket!');
        console.log(error);
    };
    socket.onopen = function (event) {
        if (msg) {
            socket.send(msg);
        }
    };
}
subscribeToWs('ws://localhost:8484/pi/sensors/temperature');
```

Capitolo 3
Implementazione

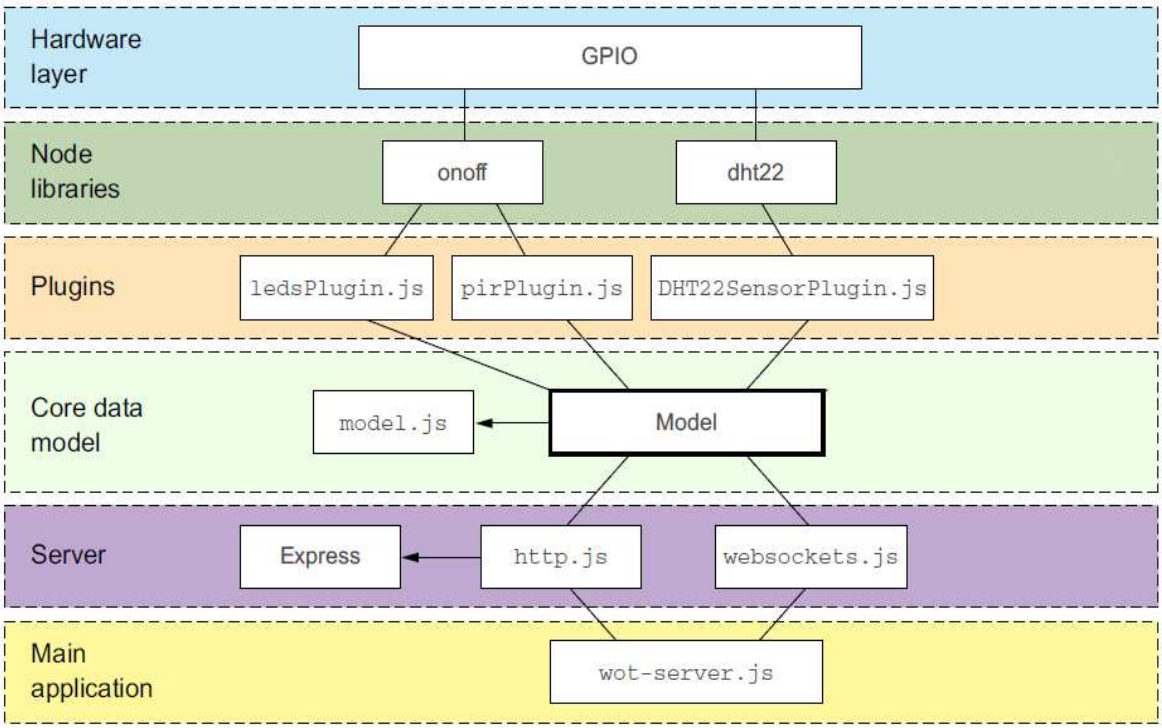
3.1 API Restful integrata sul dispositivo

Il più semplice metodo per integrare un API per il Web of Things è farlo direttamente sul dispositivo stesso.
Questo però richiede al dispositivo di essere accessibile attraverso i protocolli Internet (TCP/IP) ed essere in grado di fare da host a un server HTTP direttamente.
I server web non sono tanto esosi in fatto di spazio di memoria occupato e possono essere facilmente contenuti anche in alcuni dei dispositivi più economici presenti in commercio.

Alcuni dei server http più piccoli possono girare con meno di 50 bytes di RAM, incluso lo stack TCP/IP il che rende possibile la comunicazione attraverso HTTP anche per piccoli dispositivi a 8-bit.

Sebbene sia possibile implementare protocolli web per la maggior parte dei dispositivi integrati, il pattern a integrazione diretta risulta la scelta più efficace quando il dispositivo non è alimentato a batterie e quando l'accesso diretto dai clients, come ad esempio attraverso applicazioni mobile per il web o un semplice browser, è un requisito richiesto. Un buon esempio può essere la domotica, in cui la corrente è generalmente disponibile (come nel nostro caso)e le interazioni a bassa latenza sono importanti, immaginiamo di accendere e spegnere le luci della camera.
Abbiamo già descritto la fase di progettazione delle risorse, seguendo uno schema gerarchico ad albero.
Occorre innanzitutto creare un Modello delle Risorse, trasponendo la struttura ad albero creata in precedenza in un file JSON (il quale è già stato illustrato nella sezione di progettazione)che viene utilizzato nel codice per esporre la struttura URL corretta.
Il modello andrà quindi importato nel codice per poter essere utilizzato.
Il passo successivo consiste nella creazione delle routes con Express, attraverso le quali si vincolano i sensori agli URLs ai quali questo Web server risponderà: queste sono state già illustrate nel capitolo 2 sulla progettazione.
La parte finale quindi sarà semplicemente scrivere il file principale (*wot-server.js*) per utilizzare tutti i vari moduli di cui è composta la nostra applicazione.

Di seguito è mostrata un'immagine dello schema generale del progetto e delle varie inter-relazioni.



3.2 Hardware Plug-in

Per rendere possibile una futura integrazione sul server di hardware reale si è deciso di introdurre all'interno dei plug-in, attraverso le librerie di Node i driver per un sensore umidità/temperatura dht22,un sensore passivo infrarosso e i led.
Tutti i driver sono disponibili attraverso il package manager di Node(npm) e sono richiesti attraverso la funzione –**connectHardware().**
Si creano tanti plug-in quanti sono i sensori e gli attuatori (nel nostro caso quattro) e ognuno di questi aggiorna il modello ogni volta che sui sensori vengono letti nuovi dati.

I plugin vengono dotati delle seguenti funzioni:

-start(). Questa funzione consente l'avvio del plugin in modo tale da poter essere accessibile da altri file ed essere esportato. Se si è optato per simulare i dati acquisiti dai sensori, esso chiamerà la funzione *simulate(),* altrimenti *connectHardware().*

```
exports.start = function (params) {
  localParams = params;
  if (params.simulate) {
    simulate();
  } else {
```

```
    connectHardware();
  }
};
```

-**stop()**. In modo duale alla precedente, con questa funzione si blocca l'accesso di altri file al plugin.

```
exports.stop = function () {
if (localParams.simulate) {
clearInterval(interval);
} else {
sensor.unexport();
}
console.info('%s plugin stopped!',  pluginName);

};
```

-connectHardware(). Questa funzione permette di connettere i driver dell'hardware vero e proprio e di configurarlo. I dati vengono raccolti dai sensori della scheda raspberry su cui è stato caricato il server: attraverso il file JavaScript "**wot-server.js**", Node.Js acquisisce questi dati tramite i gpio del raspberry.Infine il modello (resources.json) viene aggiornato con i nuovi valori.
Viene qui riportato l'esempio per il plugin legato al sensore umidità/temperatura DHT22.

```
function connectHardware() {
 var sensorDriver = require('node-dht-sensor');
  var sensor = {
    initialize: function () {
    //*Inizializza il driver del sensore DHT22 sulla GPIO 12(come
specificato nel modello json)*//
      return sensorDriver.initialize(22, model.temperature.gpio);
    },
    read: function () {
      var readout = sensorDriver.read(); //Recupera il valore dai sensori
      model.temperature.value = parseFloat(readout.temperature.toFixed(2));
      model.humidity.value = parseFloat(readout.humidity.toFixed(2));
      showValue();

      setTimeout(function () {
        sensor.read();
//*Poichè il driver non prevede interrupt,  chiediamo valori nuovi al
sensore con una funzione timeout e impostiamo sensor.read() come
callback*//
      },  localParams.frequency);
    }
```

```
  };
  if (sensor.initialize()) {
    console.info('Hardware %s sensor started!',  pluginName);
    sensor.read();
  } else {
    console.warn('Failed to initialize sensor!');
  }
};
```

-**simulate()**. Questa funzione è molto importante in fase di scrittura del codice, in quanto permette di simulare i valori dei dati provenienti dai sensori e concentrarsi sulla logica dell'API senza doversi preoccupare del dispositivo hardware che potrebbe essere collegato in seguito.

```
function simulate() {
  interval = setInterval(function () {
    model.temperature.value = utils.randomInt(0,  40);
    model.humidity.value = utils.randomInt(0,  100);
    showValue();
  },  localParams.frequency);
  console.info('Simulated %s sensor started!',  pluginName);
};
```

-**showValue()** che permette di visualizzare i dati sulla console che si sta utilizzando.

```
function showValue() {
  console.info('Temperature: %s C,  humidity %s \%',
    model.temperature.value,  model.humidity.value);
};
```

3.3 Entry point

La parte finale della nostra applicazione è creare un file eseguibile da Node.js che si occupi di recuperare tutti i vari moduli e dipendenze e mettersi in ascolto sulla porta che abbiamo specificato nel file JSON per testare che tutto funzioni.
Bisogna caricare le routes sul server HTTP Express (cosa che è già stata trattata in fase di progettazione) e creare un entry point per avviarlo.
L'ultimo step di questa fase consiste nel richiedere i plugins e tutti i moduli nel *wot-server.js* tramite l'istruzione *'require'* e avviarli.

```
var httpServer = require('./servers/http'),
  wsServer = require('./servers/websockets'),
  resources = require('./resources/model');

// Richiedi tutti i plugin per i sensori
var ledsPlugin = require('./plugins/internal/ledsPlugin'),
  pirPlugin = require('./plugins/internal/pirPlugin'),
  dhtPlugin = require('./plugins/internal/DHT22SensorPlugin');

// *Internal Plugins per I sensori connessi alle PI GPIOs
Sensori settati su simulazione "true" per i test *//
pirPlugin.start({'simulate': true,  'frequency': 2000});
ledsPlugin.start({'simulate': true,  'frequency': 10000});
dhtPlugin.start({'simulate': true,  'frequency': 10000});



// server che mette in listen HTTP Server e Websocket server
var server = httpServer.listen(resources.pi.port,  function () {
  console.log('HTTP server started...');
wsServer.listen(server);

  console.info('Your WoT Pi is up and running on port %s',  resources.pi.port);
});
```

3.4 Conclusioni

Connettere ogni singolo dispositivo del nostro mondo fisico a Internet e renderlo disponibile e accessibile tramite il web significa renderlo vulnerabile ad attacchi hacker, virus, o a compagnie poco rispettabili che potrebbero utilizzare suddetti dispositivi per recuperare informazioni in modo non consensuale.
La sicurezza e la privacy dei dati che si potrebbero esporre involontariamente sulla rete è un problema di grande attualità su cui non ci siamo soffermati.
Un singolo pezzo di informazione che potrebbe essere innocuo combinato con altri potrebbe più non esserlo.
Determinati dispositivi potrebbero controllare risorse critiche per la sicurezza o il buon funzionamento di interi sistemi.
Più di una volta è sorto il dilemma di quale sia la giusta quantità di esposizione di un sistema alla rete: i nostri computer possono restare isolati in reti locali ma questo ne riduce le potenzialità in fatto di applicazioni anche se ne aumenta la facilità di controllo.
Le seguenti problematiche di sicurezza sono comunque valide anche nell'ambiente "chiuso" delle varie soluzioni proprietarie dei protocolli IoT.
Detto ciò abbiamo visto come sia possibile connettere dispositivi integrati senza utilizzare soluzioni e protocolli proprietarie di terze parti ma sfruttando i layer di rappresentazione forniti da Internet e i suoi protocolli (HTTP e WebSocket).
Le tecnologie utilizzate (JavaScript, Node.js, JSON) hanno favorito una implementazione semplice e alla portata della maggior parte dei web developer senza richiedere skill specifiche dell'IoT.
Il vantaggio di utilizzare standard utilizzati giornalmente da milioni di persone significa che sono costantemente testati e il fatto che siano "open" garantisce la presenza di una comunità a supporto di essi in costante sviluppo e crescita.
Il controllo della domotica di una stanza è solo una della molteplici possibilità quando si parla della gamma di Things: in commercio sono presenti una miriade di sensori compatibili con Raspberry-p ma nonostante le differenze che ci possono tra le loro funzioni e hardware possono essere tutti ugualmente esposti attraverso Internet utilizzando un architettura REST.

Capitolo 4
Bibliografia e sitografia

.

Building the Web of Things – D.D.Guinard e V.M.Trifa, editore Manning

Node.js in Action - Mike Cantelon, Marc Harter, T.J. Holowaychuk, Nathan Rajlich editore Manning

Learn REST: a RESTful tutorial - http://www.restapitutorial.com/

Node.js https://nodejs.org/en/

Express http://expressjs.com/it/

Oreilly-JavaScript The Definitive Guide 6th Edition Apr 2011

Oreilly-Web Development with Node Express Ethan Brown