Consiglio Nazionale delle Ricerche
Istituto di Matematica Applicata
e Tecnologie Informatiche
“Enrico Magenes”

REPORT SERIES

L. Roverelli, G. Zereik, B. Bonino, A. Galizia, A. Clematis

Strumenti e procedure per la gestione e la visualizzazione di dati meteo prodotti da sensori eterogenei e distribuiti tramite interfacce web basate su mappe geografiche

18-04

IMATI REPORT Series

Nr. 18-04
April 2018

Editorial Office
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*"
Consiglio Nazionale delle Ricerche
Via Ferrata, 5/a
27100 PAVIA (Italy)
Email: reports@imati.cnr.it
http://www.imati.cnr.it

IMATI Report Series Nr. 18-04
April 11, 2018

Strumenti e procedure per la gestione e la visualizzazione di dati meteo prodotti da sensori eterogenei e distribuiti tramite interfacce web basate su mappe geografiche

Luca Roverelli, Gabriele Zereik, Brigida Bonino, Antonella Galizia, Andrea Clematis

Luca Roverelli
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*"
Consiglio Nazionale delle Ricerche
Via de Marini, 6 (Torre di Francia) 16149 Genova - Italy
E-mail: luca.roverelli@ge.imati.cnr.it

Gabriele Zereik
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*"
Consiglio Nazionale delle Ricerche
Via de Marini, 6 (Torre di Francia) 16149 Genova - Italy
E-mail: gabriele.zereik@ge.imati.cnr.it

Brigida Bonino
Dipartimento di Matematica
Università di Genova
via Dodecaneso 35, 16146 Genova – Italy
E-mail: brigida.bonino@ge.imati.cnr.it

Antonella Galizia
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*"
Consiglio Nazionale delle Ricerche
Via de Marini, 6 (Torre di Francia) 16149 Genova - Italy
E-mail: antonella.galizia@ge.imati.cnr.it

Andrea Clematis
Istituto di Matematica Applicata e Tecnologie Informatiche "*E. Magenes*"
Consiglio Nazionale delle Ricerche
Via de Marini, 6 (Torre di Francia) 16149 Genova - Italy
E-mail: andrea.clematis@ge.imati.cnr.it

Abstract.

Il report ha lo scopo di fornire un'introduzione rapida e direttamente operativa per lo sviluppo di siti e applicazioni web in grado di gestire e visualizzare informazioni geo localizzate provenienti da sensori (soprattutto meteo) distribuiti sul territorio.
Il taglio di questo report è informale e finalizzato a fornire uno strumento di aiuto rapido per lo sviluppo di applicazioni che potranno avere anche un sofisticato contenuto scientifico o tecnologico, ma questo aspetto esula dalle finalità di questo documento.

Keywords: *Sensori eterogenei, sviluppo web, geolocalizzazione, visualizzazione*

Sommario

1. INTRODUZIONE

1.1 Scopo del documento

Il report ha lo scopo di fornire un'introduzione rapida e direttamente operativa per lo sviluppo di siti e applicazioni web in grado di gestire e visualizzare informazioni geo localizzate provenienti da sensori (soprattutto meteo) distribuiti sul territorio.

Il taglio di questo report è informale e finalizzato a fornire uno strumento di aiuto rapido per lo sviluppo di applicazioni che potranno avere anche un sofisticato contenuto scientifico o tecnologico, ma questo aspetto esula dalle finalità di questo documento.

1.2 Struttura del documento

Il presente documento è strutturato come segue:

- Capitolo 1 – è questa introduzione,
- Capitolo 3 – introduce le basi delle principali tecnologie usate per lo sviluppo Web,
- Capitolo 3 – tratta le nozioni di base per la comprensione, l'utilizzo e la definizione di generiche API e di Web Service sia di tipo SOAP che di tipo RESTful,
- Capitolo 4 – tratta gli aspetti di geo-localizzazione e le funzioni disponibili in ambito web,
- Capitolo 5 – analizza alcuni aspetti della raccolta dati da sensori e della loro gestione in ambito web,
- Capitolo 6 – fornisce un primo esempio su come mettere assieme quanto visto nei capitoli precedenti al fine di implementare una applicazione Web che permetta la visualizzazione di dati geolocalizzati.

1.3 Acronimi

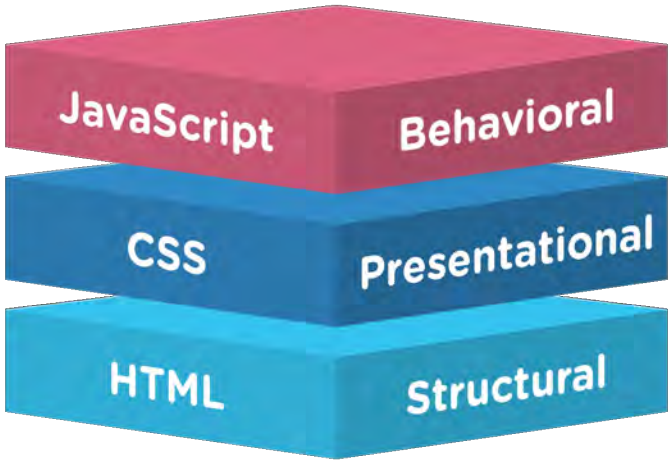
API	Application Programming Interface
BGS	British Geological Survey
GML	Geography Markup Language
KML	Keyhole Markup Language
MMWG	Mass Market Geo Working Group
NOAA	National Oceanic and Atmospehric Administration
O&M	Observations & Measurements Schema
OGC	Open Geospatial Consortium
PWS	Personal Weather Station

SAS	Sensor Alert Service
SOS	Sensor Observations Service
SVG	Scalable Vector Graphics
SWE	Sensor Web Enablement
WebCGM	Web Computer Graphics Metafile
WFS	Web Feature Service
WMO	World Meteorological Organization
WMS	Web Map Service
WNS	Web Notification Services
WRF	Weather Research and Forecasting Model
XML	eXtensible Markup Language

2. INTRODUZIONE ALLO SVILUPPO WEB

I sistemi ambientali (e.g. geologico, idrologico, meteorologico, oceanico) sono strettamente interconnessi e pertanto è fondamentale per coloro che lavorano nei settori delle geo-scienze e ambientali poter scoprire, accedere, rielaborare e condividere dati e risorse relativi a fenomeni geo-spaziali. Sovente, i professionisti di discipline ambientali si scontrano con le difficoltà poste dall'elaborazione di dati eterogenei e dalla loro gestione in ambito web.

In questo capitolo, verranno presentate le tre principali tecnologie per lo sviluppo web. Queste possono essere categorizzate come: HTML, che definisce i contenuti della pagina web; CSS che ne definisce l'aspetto, Javascript che aggiunge funzionalità dinamiche alla pagina.



Verrano nel contempo inquadrati anche framework e librerie di base che semplificano la vita agli sviluppatori web, mettendo a disposizione diversi elementi pre-costituiti per i più svariati scopi; in particolare considereremo Bootstrap e jQuery.

2.1 HTML - definiamo i contenuti

HTML è l'acronimo di *HyperText Markup Language* e non è un linguaggio di programmazione. Si tratta invece di un **linguaggio di markup** (di 'contrassegno' o 'di marcatura'), che permette di indicare come disporre gli elementi all'interno di una pagina web, definendone l'aspetto. Queste indicazioni vengono date attraverso degli appositi marcatori, detti **tag** (*'etichette'*), che hanno la caratteristica di essere inclusi tra parentesi angolari, e sono contenute in un documento html. Nel seguito si propone un documento HTML, cioè quello che una pagina web andrà ad esporre.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Example</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!-- content here -->
        <h1>Page Example</h1>
        <p>Lorem ipsum dolor sit amet.</p>
    </body>
</html>
```

Questo documento ha una struttura ben definita: il tag <html> contiene il tag <head> che contiene i metadati della pagina come il <title>, e il tag <body> che definisce il contenuto della pagina, che sarà poi visualizzato sul browser e quindi mostrato all'utente che naviga sulla pagina. Innumerevoli sono i tag definiti dal linguaggio, ognuno con le proprie caratteristiche.

Per esempio si possono definire diversi tag per titoli e sottotitoli o per paragrafi di testo, o anche per altri contenuti, come le immagini.

- Heading elements:
  - <h1>, <h2>, <h3>, <h4>, <h5>, <h6>
- Paragraph:
  - <p>
- Break line:
  -
- Images:
  - <img>
- Division (section of the document)
  -

Altri tag, come div o span, servono a organizzare il contenuto nella pagina. Per specificare il comportamento dei tag si possono aggiungere ad essi degli attributi. Alcuni generici e validi per ogni tag come class o id, altri, specifici per ogni tag, ne definiscono alcune caratteristiche peculiari, come l'immagine da caricare nel tag img o il tipo di input nel tag <input>, che permette all'utente di inserire un dato.

- Tag attributes:

```
<div class="class1 class2" id="div_id">
<img src="path/to/img">
 <input type="email" placeholder="Write your mail">
```

Nel un esempio di come viene visualizzato il contenuto di un file html su un browser quando

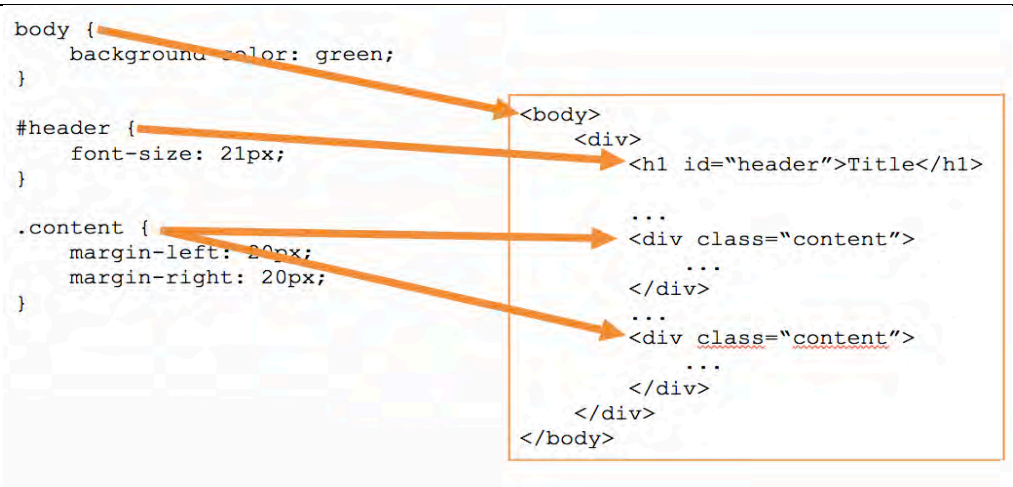
si utilizza un tagli di tipo titolo.


```
<html>
    <head>
        <title>Example 1</title>
    </head>
    <body>
        <h1> Header - Page title </h1>
        <div>
            <p>This is a paragraph
that can contain some useful
information!</p>
        </div>
    </body>
</html>
```

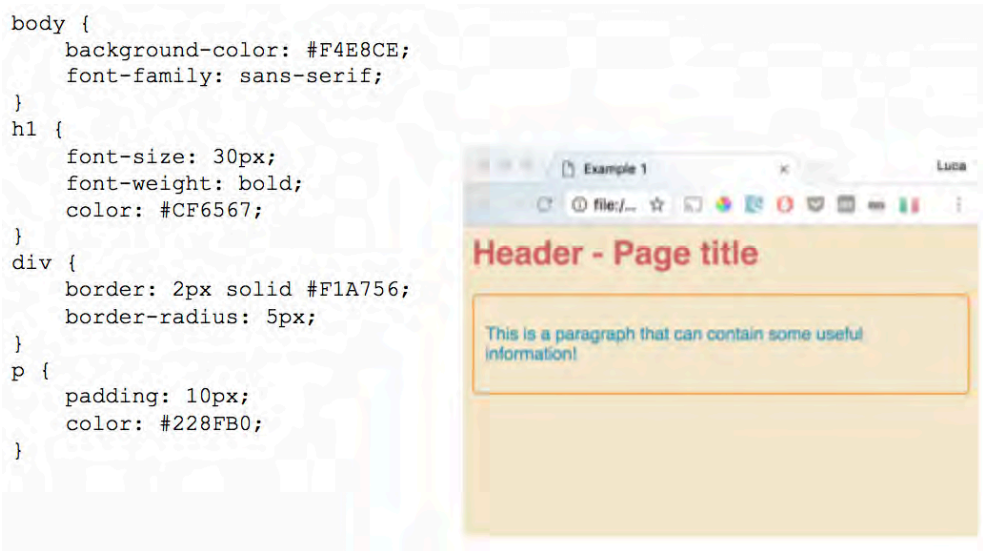
Il tag div (division) è un tag contenitore e indica una sezione di documento separata dal resto. Come si può capire dall'immagine, alcuni elementi html hanno un tag di apertura e uno di chiusura. La struttura che emerge è quella di un **albero**, in cui i rami sono tutti tag contenitori e le foglie sono composte da testi, immagini o altri elementi come le caselle di input. Tutti questi elementi finali non sono contenitori e non necessitano di un tag di chiusura. Per esempio l'elemento p che contiene il nostro testo è considerato, in questo tipo di struttura, figlio dell'elemento div e, più genericamente, discendente dell'elemento body. Al contrario, l'elemento div è il padre del tag p, mentre l'elemento body ne sarà l'antenato.

2.2 CSS - definiamo l'aspetto

L'acronimo **CSS** sta per **Cascading Style Sheets** (fogli di stile a cascata) e designa un linguaggio di stile per i documenti web. I CSS istruiscono un browser o un altro programma utente su come il documento debba essere presentato all'utente, per esempio definendone i font, i colori, le immagini di sfondo, il layout, il posizionamento delle colonne o di altri elementi sulla pagina, etc. Quindi, se l'HTML definisce il contenuto di una pagina web, il CSS ne definisce puramente l'aspetto.



Nel codice CSS riportato sopra possiamo notare la definizione di alcune regole di stile, che hanno una struttura ben definita: prima della parentesi graffa vi è il selettore, che indica l'elemento (o gli elementi) a cui viene applicata tale regola - nell'esempio body, #header, .content. All'interno delle graffe invece, è presente il blocco delle dichiarazioni, che contiene una o più dichiarazioni, costituite da proprietà e valore, da applicare all'elemento: background colour, font-size, etc. Diversi sono i tipi di selettori, per tag, per id (preceduti da #), per classe (preceduti da .) e molti altri (compresi i cosiddetti pseudoselettori). Applicando il foglio di stile all'esempio precedente, possiamo notare come l'aspetto della pagina cambi secondo le regole definite.



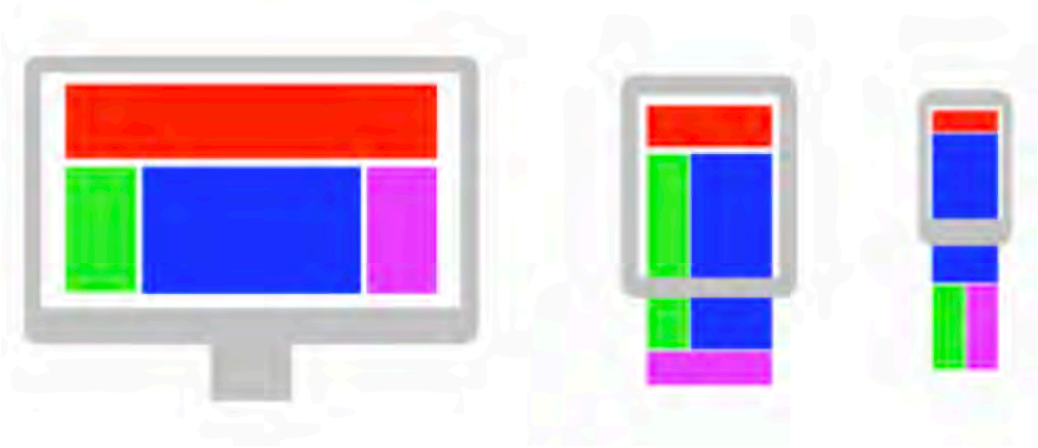
2.2.1 Framework di svilippo Bootstrap

L'evoluzione del web e dei diversi device che vi hanno accesso ha portato alla nascita e allo

sviluppo di diversi strumenti e framework che semplificano la vita agli sviluppatori web, mettendo a disposizione diversi elementi pre-costituiti per i più svariati scopi.

Tra questi framework, il più utilizzato strumento per lo sviluppo front-end (la parte del programma con cui l'utente interagisce direttamente) è sicuramente Bootstrap.

Bootstrap è una raccolta di strumenti per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript. Sono diversi gli elementi che compongono Bootstrap: dai semplici bottoni e input a strumenti più articolati, quali pannelli, modali, o caroselli di immagini, e molti altri. In questi anni in cui l'accesso al web è effettuato da diversi tipi di device con diverse dimensioni e risoluzioni, è importante ottimizzare la pagina web per la visualizzazione su ognuno di questi dispositivi - la così detta responsività. Bootstrap include uno strumento per facilitare questa operazione, il grid system.



Questo strumento permette appunto di creare pagine web Responsive, che modificano la loro struttura a seconda del dispositivo su cui vengono visualizzate. Il grid System di Bootstrap permette di suddividere la pagina web in un numero illimitato di righe, ognuna delle quali può contenere fino a 12 colonne. Righe e colonne hanno un comportamento "fluido", possono ovvero avere differenti dimensioni in base alla dimensione dello schermo su cui sono visualizzate. Ogni, colonna, a sua volta, può contenere diverse righe (tramite la ricorsione), permettendo così la definizione di qualsiasi tipo di layout.

In particolare, questo sistema tende a suddividere gli schermi in 4 tipi:

- Extra small devices (telefoni)
- Small devices (tablet)
- Medium devices (piccoli desktop)

- Large devices (schermi molto grandi)

In pratica, tramite l'applicazione di specifiche classi, il sistema definisce diverse dimensioni per ogni colonna, a seconda dello schermo. Nel seguito una schematizzazione delle possibili opzioni che Bootstrapt propone su diversi device.

Grid options

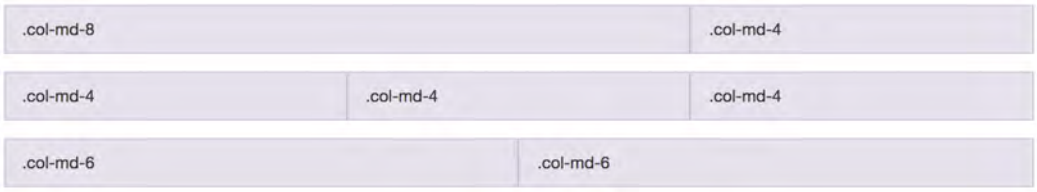
See how aspects of the Bootstrap grid system work across multiple devices with a handy table.

	Extra small devices Phones (<768px)	Small devices Tablets (≥768px)	Medium devices Desktops (≥992px)	Large devices Desktops (≥1200px)
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints		
Container width	None (auto)	750px	970px	1170px
Class prefix	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`
# of columns	12			
Column width	Auto	~62px	~81px	~97px
Gutter width	30px (15px on each side of a column)			
Nestable	Yes			
Offsets	Yes			
Column ordering	Yes			

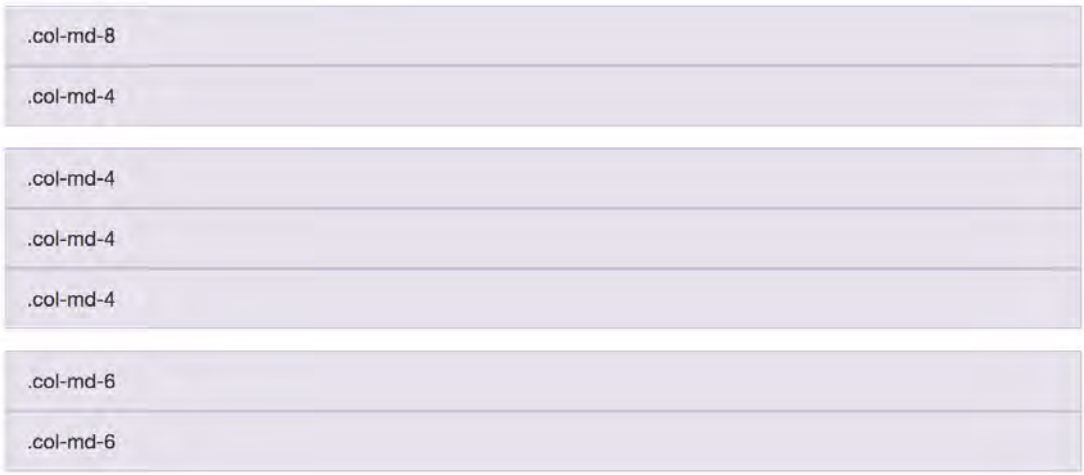
Nell'esempio sottostante, vengono definite tre righe (classe row). Ognuna delle quali contiene diverse colonne.

```
<div class="row">
        <div class=”col-xs-12 col-md-8">.col-md-8</div>
        <div class="col-xs-12  col-md-4">.col-md-4</div>
</div>
<div class="row">
        <div class="col-xs-12  col-md-4">.col-md-4</div>
        <div class="col-xs-12 col-md-4">.col-md-4</div>
        <div class="col-xs-12 col-md-4">.col-md-4</div>
</div>
<div class="row">
        <div class="col-xs-12 col-md-6">.col-md-6</div>
        <div class=" col-xs-12 col-md-6">.col-md-6</div>
</div>
```

La prima riga per esempio contiene due colonne. Queste colonne su schermi Desktop medi e grandi sono larghe rispettivamente 2/3 e 1/3 della riga che le contiene.



Su schermi più piccoli, invece, ognuna delle due colonne avrà larghezza uguale al 100% della larghezza della riga. Questo, ovviamente, comporta il fatto che le due colonne vengano visualizzate, su questi tipi di schermi, una sotto all'altra.



Si è cosi in grado di definire, senza eccessivi sforzi, un'interfaccia ottimizzata su ogni tipo di schermo.

2.3 JavaScript - definiamo la dinamicità

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera, caricamento della pagina ecc...). Tali funzioni di script possono essere opportunamente inserite in file HTML, o in appositi file separati con estensione .js poi richiamati nella logica di business.

Caratteristica principale del linguaggio è che esso viene eseguito direttamente dal browser, ovvero sul pc dell'utente che naviga sulla pagina web, non creando alcun carico computazionale lato server.

```
function factorial(n) {
        if (n === 0 || n === 1) {
                return 1;   // 0! = 1! = 1
        }
        return n * factorial(n - 1);
}

var result = factorial(283);

console.log("The result is: "+result);
```

La sintassi di javascript è relativamente simile a quella del C, del C++ e del Java, ed è un linguaggio debolmente tipizzato, e debolmente orientato agli oggetti. Per questo motivo, in Javascript si possono per esempio definire funzioni, eseguirle e definire anche variabili.

2.3.1 Librerie di base jQuery

Javascript ha avuto molto successo come linguaggio per il web anche grazie alle molte librerie che sono state sviluppate. La più usata tra queste librerie è sicuramente **jQuery**, che nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione degli elementi nelle pagine HTML. Si possono infatti selezionare facilmente, per esempio, uno o più elementi della pagina HTML per poter poi essere manipolati in diverse maniere.

- Easily retrieve elements of the web page using different selectors:
  - by "id" of the element:
    - `$("#main_article")`
  - by the tag:
    - `$("button")`
  - by class:
    - `$(".btn-primary")`
- Similar to CSS, it is possible to chain different selectors:
    - `$("#main_article button")`

Similmente al CSS, si possono definire diversi selettori, (per id, tag, classe, ecc) e si possono inoltre concatenare i selettori per filtrare gli elementi selezionati.

Diversi tipi di manipolazione sono previsti da questa libreria. Vediamone alcuni esempi:

- si possono modificare dinamicamente le diverse proprietà di stile dell'elemento:

```
node.css('color', '#1abc9c');
```

- si può modificarne e/o sostituirne il contenuto HTM

```
node.html('<b>Bold text</b>');
```

- si può prependere o appendere altro contenuto prima o dopo l'elemento

```
node.append('<b>Bold text</b>');
```

- si può nascondere un elemento e ripristinarne la visibilità

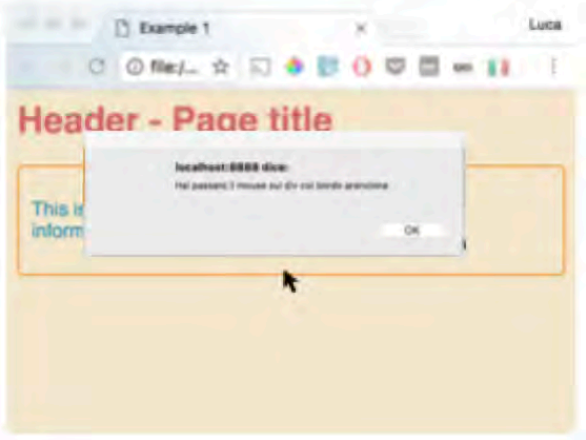
```
node.toggle();
```

jQuery permette inoltre di definire delle funzioni che vengano eseguite al succedersi di un determinato evento: lo snippet nella figura definisce una funzione che viene eseguita quando il bottone con id main-button viene schiacciato dall'utente.

```
$("#main-button").on("click", function() {
        console.log("button pressed")
});
```

Diversi sono i tipi di eventi gestiti da jQuery: alcuni esempi sono l'evento double click (al doppio click del mouse), l'evento change: quando viene per esempio selezionata un'opzione in un elemento di tipo select, o l'evento hover, che avviene quando l'utente passa col mouse sopra all'elemento.

```
$('div').hover(
  function(){
    alert("Hai passato il mouse sul div col bordo arancione");
  }
);
```



Nell'esempio in figura, viene dichiarata una funzione, cosiddetta callback, che venga appunto chiamata quando l'utente passa col mouse sopra all'elemento target (il div con il bordo arancione). Questa funzione non fa altro che aprire un popup standard del browser per visualizzare un messaggio.

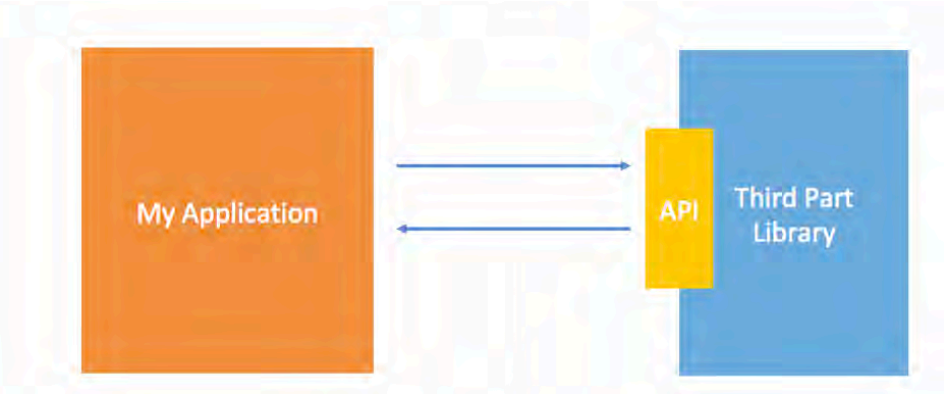
3. WEB SERVICE E API

Questo capitolo presenta le nozioni di base per la comprensione, l'utilizzo e la definizione di generiche API e di Web Service sia di tipo SOAP che di tipo RESTful.

3.1 API - Application Programming Interface

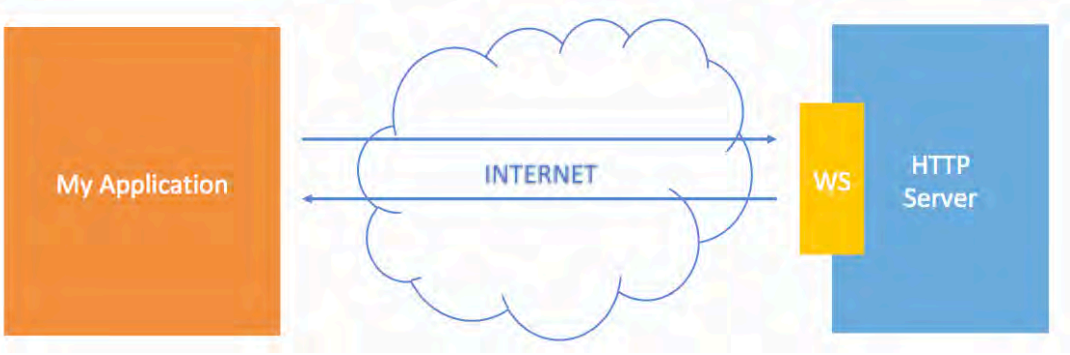
Una **API** è un'interfaccia implementata da un software che permette ad esso di interagire con altri software, nello stesso modo in cui un'interfaccia permette l'interazione tra il software e un utente umano.

Un'API può essere implementata da diversi componenti software (applicazioni, librerie, sistemi operativi…) e definisce il modo in cui altri applicativi possono accedere alle funzionalità di tali componenti. La definizione di tale interazione è costituita dalla definizione di un insieme di routine, protocolli, strutture dati usati per la comunicazione di questa componente con i programmi che vogliono accedere alle sue funzionalità.



3.2 Web Service

Un **Web Service** è un sistema software progettato per supportare un'interazione tra applicazioni in remoto, utilizzando le tecnologie e gli standard Web. Possiamo definirlo quindi come un sistema software in grado di mettersi al servizio di un'applicazione comunicando su una medesima rete tramite il protocollo HTTP (o HTTPS). Un Web service consente dunque, alle applicazioni che vi si collegano di usufruire delle funzioni che mette a disposizione, tramite la definizione di un'API.



Il meccanismo dei **Web Service** consente di far interagire in maniera trasparente applicazioni sviluppate con linguaggi di programmazione diversi, che girano su sistemi operativi eterogenei. Questo grazie all'esistenza di due approcci standard alla creazione di un WS: SOAP (simple object access protocol) e Restful (Representational state transfer).

- Il primo si prefigge di riprodurre in ambito web un approccio a chiamate remote (Remote procedure call). Un **Web Service basato su SOAP**, dunque, espone un insieme di metodi richiamabili da remoto da parte di un client.
- Il secondo è ispirato ai principi architetturali tipici del web e si concentra sulla descrizione di risorse, sul modo di individuarle nel Web e di trasferirle da una macchina ad un'altra. Un **Web Service basato su REST,** quindi, si basa sul concetto di risorsa ed espone un'interfaccia che permette le operazioni base su tali risorse (CRUD). Lo standard di scambio dei messaggi utilizzato da un WS SOAP è XML.

3.2.1 Web Service basati su SOAP

In Figura si presenta un esempio di WS basato su SOAP al fine di spiegarne la struttura.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

https://en.wikipedia.org/wiki/SOAP

Come si evince dalla figura, un messaggio è costituito da un Header e un Body, incapsulati all'interno di un Envelope. L'elemento **Header** è opzionale e contiene informazioni specifiche per l'applicazione. **Body** è un elemento indispensabile che contiene le informazioni scambiate nelle richieste e risposte. L'envelope può contenere inoltre un altro elemento opzionale,

FAULT, che fornisce informazioni riguardo ad eventuali errori manifestati durante la lettura del messaggio.

Ogni WS SOAP include un file (WSDL) che descrive l'interfaccia esposta. Questo file dunque definisce, sempre in formato XML, le modalità di interazione con un determinato Servizio (WS), descrivendone le caratteristiche dei vari punti di accesso (endpoints). Spesso a questo file, si associa un altro file in formato XML, l'XML Schema, che serve per validare il contenuto del WSDL. Un programma client può, quindi, "leggere" il documento WSDL relativo ad un Web Service per determinare quali siano le funzioni messe a disposizione sul server e quindi utilizzare il protocollo SOAP per utilizzare una o più delle funzioni elencate dal WSDL. Il WSDL contiene, come detto, la descrizione dell'API esposta da un ws soap. In particolare, vi si possono trovare informazioni relativamente a:

- Le operazioni messe a disposizione dal servizio,
- Come tali operazioni possono essere utilizzate (protocollo di comunicazione, il formato dei messaggi in input e output ed i dati correlati),
- Dove utilizzare il servizio (ovvero il cosiddetto endpoint, solitamente un indirizzo).

Le operazioni supportate dal Web Service ed i messaggi che è possibile scambiare con lo stesso sono descritti in maniera astratta e quindi non collegati ad uno specifico protocollo di rete o ad uno specifico formato. Nel seguito è riportato un esempio di Web Serive, dove si può evincere che la definizione di un Web Service SOAP è solitamente complicata. Per ridurre tali difficoltà, esistono tool che permettono la generazione automatica del file WSDL, tramite la lettura automatica del codice sorgente. A partire da questo file, è inoltre possibile - tramite altri tool - automatizzare anche la generazione del codice necessario ad un client per l'interazione con tale WS, in qualsiasi linguaggio di programmazione.

```
<?xml version="1.0" encoding ="utf-8"?>
<definitions
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.html.it/php_ws_soap"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://www.html.it/php_ws_soap">
    <types>
        <xs:schema targetNamespace="http://www.html.it/php_ws_soap">
            <xs:element name="name" type="xs:string" />
            <xs:element name="weburl" type="xs:string" />
        </xs:schema>
    </types>
    <message name="getWebUrl">
        <part name="name" type="xs:string" />
    </message>
    <message name="returnWebUrl">
        <part name="weburl" type="xs:string" />
    </message>
    <portType name="WebServiceTest">
        <operation name="getWebUrl">
            <input message="tns:getWebUrl" />
            <output message="tns:returnWebUrl" />
        </operation>
    </portType>
    <binding name="WebServiceSOAPBinding" type="WebServiceTest">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getWebUrl">
            <soap:operation soapAction="http://localhost/ws_server/SearchEngineWS.php/getWebUrl" style="rpc"/>
            <input>
                <soap:body use="encoded" namespace="http://www.html.it/php_ws_soap" encodingStyle="
http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="encoded" namespace="http://www.html.it/php_ws_soap" encodingStyle="
http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
        </operation>
    </binding>
    <service name="GetWebUrl">
        <port name="WebUrl" binding="tns:WebServiceSOAPBinding">
            <soap:address location="http://localhost/ws_server/SearchEngineWS.php"/>
        </port>
    </service>
</definitions>
```

Per quanto riguarda RESTful, più che uno standard esso rappresenta un insieme di linee guida per realizzare un'architettura di sistema. Esso sfrutta lo standard HTTP (ed i suoi verbi/metodi) per eseguire operazioni su una risorsa. I termine risorsa è dunque un concetto chiave in questo contesto: per risorsa si intende un qualsiasi elemento oggetto di elaborazione.

Principio chiave di un WS RESTFUL è che ogni risorsa deve poter essere identificata univocamente (solitamente tramite un indirizzo/URL). Come detto REST sfrutta il protocollo HTTP, e in particolare i suoi metodi per effettuare operazioni base su una risorsa (il cosiddetto CRUD - create read update delete).

La tabella in figura definisce tali operazioni associandole al relativo verbo http utilizzato:

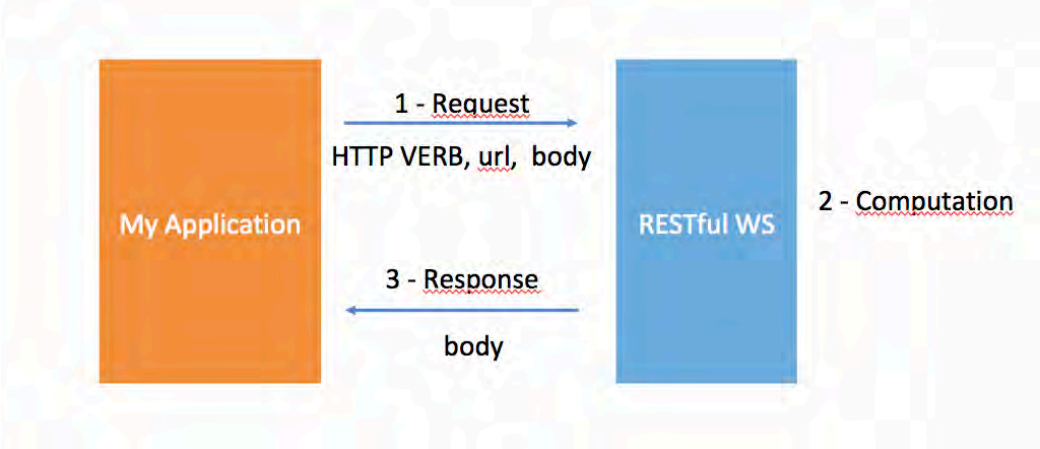
- per creare una nuova risorsa viene utilizzato il verbo POST,
- per ottenere info sulla risorsa si utilizza il GET,
- per modificarla il PUT e per eliminarla si usa il metodo DELETE.

CRUD	HTTP VERB	ACTION
Create	POST	create a new resource
Read	GET	retrieve a resource
Update	PUT	modify a resource
Delete	DELETE	delete a resource

Un WS RESTful può utilizzare due diversi standard per lo scambio di messaggi: XML E JSON. Tuttavia, per la maggiore leggerezza dello standard, è solitamente utilizzato quest'ultimo, in contrapposizione con l'utilizzo di XML nei WS SOAP. **JSON** (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi.

E' possibile descrivere la modalità d'interazione con un WS RESTful in tre passi:

1. Un client fa una richiesta ad una risorsa al WS, specificando il verbo http, l'url della risorsa ed eventualmente un body che contiene informazioni necessarie all'espletamento della richiesta.
2. Il WS riceve la richiesta, la elabora (anche in base alle info contenute nel body) e genera un responso.
3. Il responso viene restituito al client.



3.2.2 Web Service basati su REST

Nel seguito di propongono diversi esempi di utilizzo dei WS RESTful.

In figura è definito un esempio di operazione CREATE di una risorsa di tipo book. Nel body della richiesta vengono specificate le informazioni necessarie per poter creare tale risorsa.Il responso del WS è un oggetto simile a quello della richiesta in cui viene specificato inoltre l'id della risorsa appena creata.

POST http://myapp.com/books

Body of the request:

```
{
    "author": "J.R.R. Tolkien",
    "title": "The Lord of the rings",
    "year": 1954
}
```

Response body:

```
{
    "id": 723,
    "author": "J.R.R. Tolkien",
    "title": "The Lord of the rings",
    "year": 1954
}
```

Per ottenere una risorsa si può utilizzare l'id della stessa, che la identifica univocamente. In questo caso, la richiesta utilizza il metodo HTTP GET, che non prevede l'utilizzo del body, per cui il parametro che definisce l'id della risorsa richiesta fa parte dell'url che la identifica.

GET http://myapp.com/books/723

Response body:

```
{
    "id": 723,
    "author": "J.R.R. Tolkien",
    "title": "The Lord of the rings",
    "year": 1954
}
```

Di seguito troviamo un altro esempio di richiesta che utilizza il metodo GET. In questo caso viene richiesta la lista di tutte le risorse di tipo book presenti nel WEB service: il responso sarà quindi un array di risorse di tipo book, ognuna delle quali contenente le proprie informazioni.

GET http://myapp.com/books

Response body:

```
[
    {
        "id": 723,
        "author": "J.R.R. Tolkien",
        "title": "The Lord of the rings",
        "year": 1954
    },
    {
        "id": 742,
        "author": "George Orwell",
        "title": "1984",
        "year": 1949
    }
]
```

Nell'url di una richiesta si possono inoltre specificare diversi parametri che, se gestiti all'interno del WS, possono essere utilizzati, per esempio, per filtrare le risorse richieste. Nell'esempio è presente la definizione di una richiesta di tutte le risorse di tipo book in cui la proprietà Author contiene la stringa "Orwell".

GET http://myapp.com/books?author=Orwell

Response body:

```
[
    {
        "id": 742,
        "author": "George Orwell",
        "title": "1984",
        "year": 1949
    }
]
```

Il prossimo esempio definisce una richiesta di tipo Update, che sfrutta il metodo http PUT. Tale richiesta ha lo scopo di modificare una risorsa già presente all'interno del WS. Il tipico responso di un WS RESTful in questo tipo di operazione, contiene le informazioni aggiornate della risorsa modificata.

PUT http://myapp.com/books/723

Body of the request:

```
{
    "year": 2017
}
```

Response body:

```
{
    "id": 723,
    "author": "J.R.R. Tolkien",
    "title": "The Lord of the rings",
    "year": 2017
}
```

Infine vediamo un esempio di richiesta di tipo DELETE che sfrutta l'omonimo verbo HTTP. Dato che la risorsa viene identificata univocamente dal proprio id, non serve nessun altro parametro per completare questo tipo di richiesta. Il responso definito nell'esempio contiene i dati della risorsa appena eliminata.

DELETE http://myapp.com/books/723

Response body:

```
{
    "id": 723,
    "author": "J.R.R. Tolkien",
    "title": "The Lord of the rings",
    "year": 2017
}
```

I componenti principali, tipici di un qualsiasi Web Service RESTful, sono 4, e sono **request**, **response**, **handlers** e **routes**.

- La request costituisce i dati inviati dal client e ricevuti dal server. Questi comprendono il body della richiesta, ma anche eventuali headers e cookies.
- Il response è invece la risposta generata dal server, e contiene i dati che devono essere inviati al client che ha eseguito la richiesta. Oltre ai dati, può contenere anche uno status code ed un eventuale errore.
- Gli handlers sono le funzioni che, a partire da una particolare richiesta, eseguono una elaborazione, e generano un response da inviare al client. Qua avviene la vera e propria computazione.
- Infine, le rotte permettono di collegare uno specifico URL e il verbo HTTP usato nella

richiesta ad un particolare handler.

Nell'esempio in figura viene definito in piccolo WS di tipo REST, utilizzando, per semplicità di sintassi, Slim framework, scritto in linguaggio php.

```
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;
require 'vendor/autoload.php';

$app = new \Slim\App;

$app->get('/hello/{name}', function (Request $request, Response $response) {

    $name = $request->getAttribute('name');
    $val = [ "message": "Hello, $name" ];
    $response->getBody()->write($val);
    return $response;

});

$app->run();
```

In particolare, il Web service definisce una rotta di tipo get contenente un parametro di tipo name. Alla rotta viene associato un Handler, ovvero una funzione deputata all'elaborazione della richiesta e alla generazione di un responso, che solitamente sono due argomenti di questo handler.

3.2.3 Valutazione dei due standard per la definizione di WS

Di seguito si fornisce una breve analisi dei pro ed i contro dei due standard presentati per lo sviluppo di Web Service, la tabella successiva sintentizza le caratteristiche considerate.

La tipizzazione dei dati in un WS SOAP fa si che i messaggi scambiati siano pesanti, e questo incide sulle performance. Un web service REST invece non ha una forte tipizzazione dei dati, e quindi ha performance superiori ma è anche più incline a errori.

Il problema della complessità della definizione di un WSDL per SOAP viene in parte superato dalla possibilità di generarlo automaticamente, come d'altronde si può fare anche con la parte client, anche se i tool per la generazione sono spesso complicati da utilizzare. Con REST, il client side è spesso implementato a mano, vista anche la semplicità di definizione. E' inoltre molto semplice, come si è visto anche nell'esempio apportato, definire la parte Server.

SOAP	RESTful
• Typed data	• Not strictly typed
• Heavy and verbose messaging	• Simple and lightweight messaging
• Automatic WSDL, and client side	• Client side to be implemented
• Needs complex tools	• Server implementation very simple
• Complex to be cached	• Simple to be cached

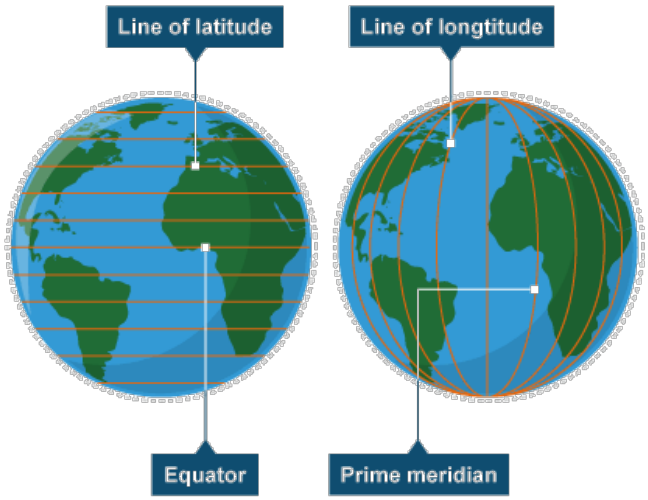
In ultimo, è utile anche menzionare le difficoltà nell'utilizzo della cache per migliorare le prestazioni di un WS soap, visto che in ogni caso è sempre necessario effettuare il parsing della richiesta in toto. Le stesse difficoltà non si trovano in un WS REST, che utilizza JSon come standard di scambio dei messaggi.

4. GEOLOCALIZZAZIONE, MAPPE E IL WEB

Quando si lavora con sensori, è ovviamente fondamentale essere in grado di riconoscerne la posizione geografica. Per questo nel seguito vengono spiegati il concetto di geolocalizzazione e le nozioni di base per l'utilizzo delle funzionalità di geolocalizzazione offerte da HTML5 e delle API JavaScript di Google Maps.

4.1 Cosa significa geolocalizzare un oggetto

La **geolocalizzazione** è l'attribuzione o l'individuazione di un'esatta situazione o posizione spaziale di un oggetto, come una sorgente radar, un telefono o un terminale connesso ad Internet. Essa è costituita da due attributi, la latitudine e la longitudine, espressi in gradi, che determinano rispettivamente la posizione verticale e orizzontale di un oggetto.



La geolocalizzazione può essere stimata attraverso diverse tecniche:

- il GPS, che sfrutta i satelliti,
- localizzazione tramite IP address, che permette di localizzare terminali collegati in rete,
- localizzazione tramite MAC address, che sfrutta lo stesso principio,
- localizzazione tramite Wi-fi, che permette di localizzare oggetti in base al segnale wi fi che intercettano.

Tutti i moderni smartphone hanno diversi sensori per utilizzare le diverse tecniche di geolocalizzazione: da notare come alcune di queste, per esempio il GPS, consumano molta batteria.

4.2 Geolocalizzare con HTML5

I moderni browser hanno a disposizione alcune tecniche di geolocalizzazione, infatti HTML5

ha introdotto delle API per la geolocalizzazione dell'utente direttamente tramite browser. Nel seguito riportiamo un semplice esempio di codice JavaScript per la geolocalizzazione.

```
<script>
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition, showError);
} else {
    console.log("Geolocation is not supported by this browser.");
}

function showPosition(position) {
    console.log("Latitude: " + position.coords.latitude +
    " Longitude: " + position.coords.longitude);
}
...
</script>
```

La funzione getCurrentPosition, asincrona, ha come parametri due callback. La prima viene chiamata in caso di ottenimento della posizione richiesta, mentre la seconda, solo in caso di errore. La callback di errore è chiamata a gestire le varie casistiche in cui non si è riusciti ad ottenere una posizione valida.

```
<script>
...
function showError(error) {
    switch(error.code) {
        case error.PERMISSION_DENIED:
            //User denied the request for Geolocation
            break;
        case error.POSITION_UNAVAILABLE:
            //Location information is unavailable
            break;
        case error.TIMEOUT:
            //The request to get user location timed out
            break;
        case error.UNKNOWN_ERROR:
            //An unknown error occurred
            break;
    }
}
```

I quattro tipi di errore possibili sono:

1. l'utente non ha dato il consenso alla localizzazione,
2. le informazioni sulla geolocalizzazione non sono disponibili,
3. la richiesta non ha avuto responso prima del timeout,
4. un errore sconosciuto è avvenuto.

4.3 Le mappe di Google

La libreria di Google Maps, permette e facilita lo sviluppo di interfacce grafiche basate su mappa. In particolare la libreria permette di:

- visulizzare una mappa interattiva della terra,
- aggiungere e rimuovere marker,
- personalizzare i marker con immagini custom,
- disegnare polilnee e poligoni sulla mappa (ma anche altre figure),
- sovrapporre alla mappa altri layer di immagini (usando il formato KML),
- personalizzare i colori della mappa.

Queste possibilità vengono dettagliate nel seguito fornendo esempi di codice per permetterne lo sviluppo.

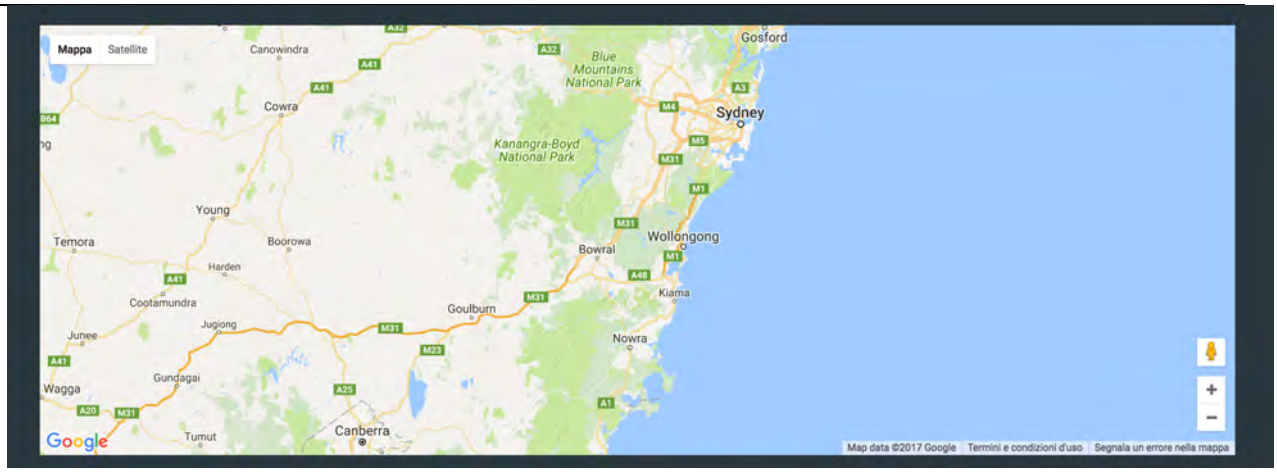
Al fine di utilizzare le mappe Google e le loro funzionalità si devono utilizzare/interrogare le API di Google Maps, per far questo è necessario ottenere un API key gratuita. I passi per farlo sono molto semplici e descritti nel link seguente:

https://developers.google.com/maps/documentation/javascript/

Una volta ottenuta l'API KEY è molto semplice visualizzare una mappa interattiva tramite le funzioni in JavaScript messe a disposizione; nel seguito un frammento di codice

```
...
<div id="map"></div>
<script>
    function initMap() {
        var map = new
google.maps.Map(document.getElementById('map'), {
          center: {lat: -34.397, lng: 150.644},
          zoom: 8
        });
    }
</script>
<script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap"></script>
...
```

Una volta incluso lo script delle API, si crea una funzione che assegni ad un elemento del DOM, la funzione di visualizzare la mappa. Ed ecco il risultato ottenuto:



Per aggiungere un marker si deve associare il marker, in fase di creazione, alla mappa (passandogli l'oggetto della mappa come opzione) e assegnarli una posizione geografica, oltre ad altre opzioni.

```
<script>

  function initMap() {
    var myLatLng = {lat: -25.363, lng: 131.044};

    var map = new
google.maps.Map(document.getElementById('map'), {
      zoom: 4,
      center: myLatLng
    });

    var marker = new google.maps.Marker({
      position: myLatLng,
      map: map,
      title: 'Hello World!'
    });
  }
</script>
```

Il marker verrà visualizzato esattamente nella locazione geografica definita in fase di creazione.

Tra le opzioni per i marker vi è la possibilità di definire un'immagine custom da visualizzare sulla mappa. L'opzione icon del marker può essere impostata ad una stringa contenente l'url dell'immagine.

```
    <script>
      function initMap() {
        var map = new
google.maps.Map(document.getElementById('map'), {
          zoom: 4,
          center: {lat: -33, lng: 151}
        });

        var image = 'https://developers.google.com/maps/
documentation/javascript/examples/full/images/beachflag.png';
        var beachMarker = new google.maps.Marker({
          position: {lat: -33.890, lng: 151.274},
          map: map,
          icon: image
        });
      }
    </script>
```

A questo punto l'immagine del nostro marker sarà quella che abbiamo indicato in fase di creazione.

Le API Google Maps permettono anche di indicare un testo (semplice o html) legato al marker che possa contenere informazioni aggiuntive. Nell'esempio, questo testo viene visualizzato quando l'utente clicca sul marker.

```
      function initMap() {
        var uluru = {lat: -25.363, lng: 131.044};
        var map = new
google.maps.Map(document.getElementById('map'), {
          zoom: 4, center: uluru
        });
        var contentString = 'htmlText';
        var infowindow = new google.maps.InfoWindow({
          content: contentString
        });
        var marker = new google.maps.Marker({
          position: uluru, map: map, title: 'Uluru (Ayers
Rock)'
        });
        marker.addListener('click', function() {
          infowindow.open(map, marker);
        });
      }
```

Come mostra la figura, questa finestra di info viene visualizzata in una posizione tale per cui è molto chiaro a quale marker essa corrisponda.

Altra possibilità offerta è quella di disegnare linee sulla mappa, indicando le varie coordinate che devono essere toccate. Ci sono diverse opzioni anche per questa funzionalità.

Tra queste è interessante l'opzione geodesic: quando questo booleano è settato a true, le linee seguiranno la curvatura della terra.

```
function initMap() {
  var map = new google.maps.Map(document.getElementById('map'),
{
      zoom: 3, center: {lat: 0, lng: -180},
      mapTypeId: 'terrain'
  });

  var flightPlanCoordinates = [
    {lat: 37.772, lng: -122.214}, {lat: 21.291, lng: -157.821},
    {lat: -18.142, lng: 178.431}, {lat: -27.467, lng: 153.027}
  ];
  var flightPath = new google.maps.Polyline({
     path: flightPlanCoordinates,
     geodesic: true, strokeColor: '#FF0000',
     strokeOpacity: 1.0, strokeWeight: 2
  });
  flightPath.setMap(map);
}
```

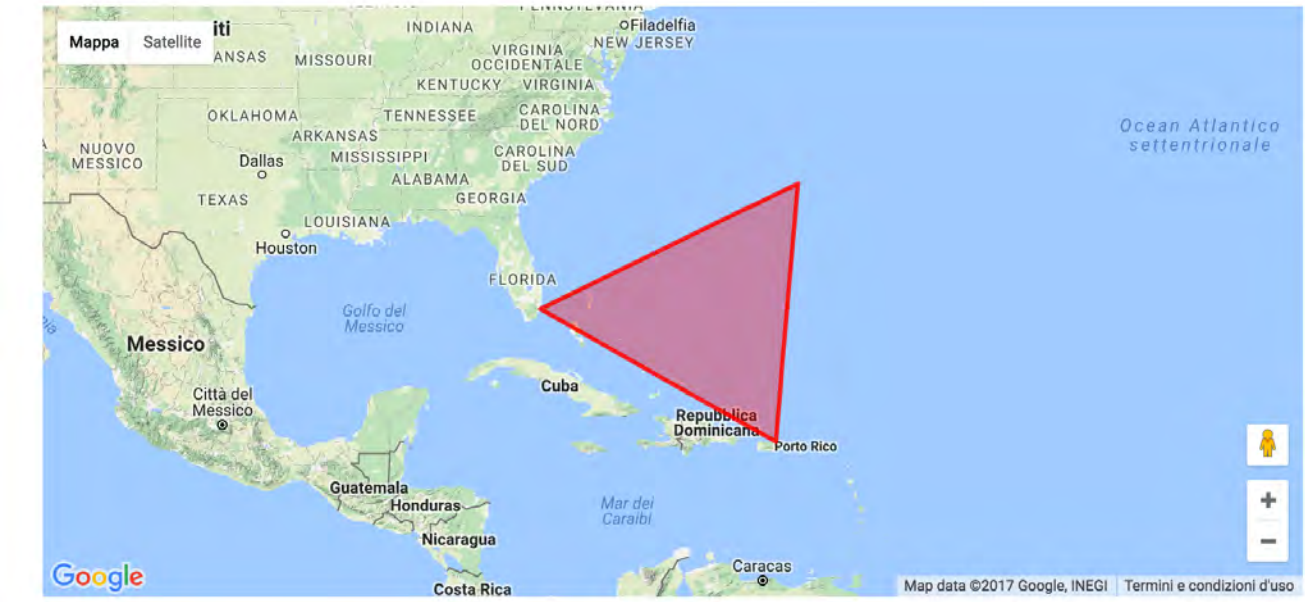
Aggiungendo una linea che tocchi diverse coordinate, il risultato è il seguente.

Si noti la leggera curvatura delle linee dovuta al fatto che è attiva l'opzione geodesic.

Analogamente a quanto visto con le linee, si possono disegnare sopra ad una mappa anche dei poligoni. Per far ciò, si istanzia il corretto oggetto, includendo nelle opzioni un array di coordinate che corrispondono ai vertici del poligono.

```
function initMap() {
    var map = new
google.maps.Map(document.getElementById('map'), {
      zoom: 5,
      center: {lat: 24.886, lng: -70.268},
      mapTypeId: 'terrain'
     });
     var triangleCoords = [
        {lat: 25.774, lng: -80.190},
        {lat: 18.466, lng: -66.118},
        {lat: 32.321, lng: -64.757}
     ];
     var bermudaTriangle = new google.maps.Polygon({
         paths: triangleCoords, strokeColor: '#FF0000',
         strokeOpacity: 0.8, strokeWeight: 3,
         fillColor: '#FF0000', fillOpacity: 0.35
        });
     bermudaTriangle.setMap(map);
}
```

Ed ecco come visualizzare il triangolo delle Bermuda sulla mappa.



Diversi tipi di poligoni e figure sono disponibili e possono essere aggiunti alla mappa.

Nel caso in figura, istanziamo una serie di cerchi, e li aggiungiamo alla mappa impostando centro e raggio dei vari cerchi.

```
var citymap = {
    newyork: {
       center: {lat: 40.714, lng: -74.005},
       population: 8405837
    }, ...
};
function initMap() {
    var map = new google.maps.Map(document.getElementById('map'), {
        zoom: 4, center: {lat: 37.090, lng: -95.712},
    });
    for (var city in citymap) {
        var cityCircle = new google.maps.Circle({
            strokeColor: '#FF0000', strokeOpacity: 0.8,
            strokeWeight: 2, fillColor: '#FF0000',
            fillOpacity: 0.35, map: map,
            center: citymap[city].center,
            radius: Math.sqrt(citymap[city].population) * 100
        });
    }
}
```

Abbiamo poi aggiunto diversi cerchi alla mappa in corrispondenza di alcune città, facendo in modo che il raggio fosse direttamente proporzionale alla popolazione di tali città. In tal modo abbiamo implementato una rappresentazione visiva della popolazione in queste città.

Come accennato nell'introduzione, vi è inoltre la possibilità di sovrapporre immagini alla mappa, impostando la posizione dei suoi 4 bordi.

```
function initMap() {
    var map = new
google.maps.Map(document.getElementById('map'), {
        zoom: 13,
        center: {lat: 40.740, lng: -74.18}
    });
    var imageBounds = {
        north: 40.773941, south: 40.712216,
        east: -74.12544, west: -74.22655
    };
    var historicalOverlay = new
google.maps.GroundOverlay('https://www.lib.utexas.edu/maps/
historical/newark_nj_1922.jpg', imageBounds);

    historicalOverlay.setMap(map);
}
```

Si possono sovrapporre, per esempio, immagini di come era il territorio in passato, per avere un confronto con la conformazione attuale.

Oltre alle immagini, si può sovrapporre un altro tipo di layer, definito in formato KML - Keyhole Markup Language, formato utilizzato per visualizzare dati geografici. KML usa una struttura a tag con elementi nidificati e attributi ed è basato sullo standard XML.

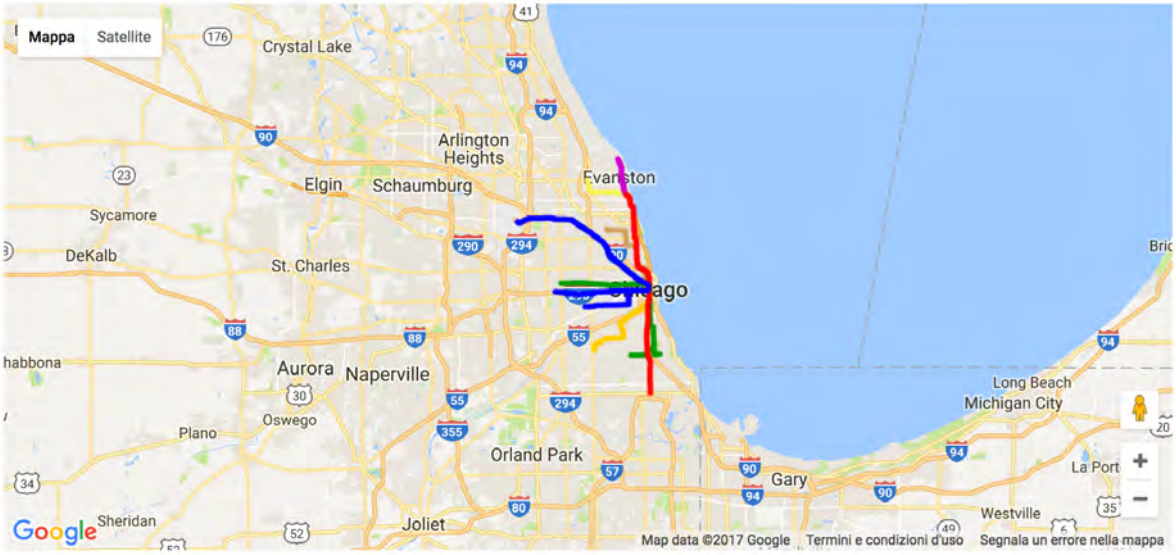
```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Document>
    <name>Chicago Transit Map</name>
    <description>Chicago Transit Authority train lines</description>
    <Style id="yellowLine">
      <LineStyle>
        <color>ff61f2f2</color>
        <width>4</width>
      </LineStyle>
    </Style>
    <Placemark>
      <name>Yellow Line</name>
      <styleUrl>#yellowLine</styleUrl>
      <LineString>
        <altitudeMode>relative</altitudeMode>
        <coordinates>
-87.75250,42.04049,0
-87.74726629257202,42.02620267868042,0
...
        </coordinates>
      </LineString>
    </Placemark>
  </Document>
</kml>
```

Per aggiungere un layer KML alla mappa è sufficiente istanziare un oggetto KmlLayer indicando l'url del file KML corrispondente.

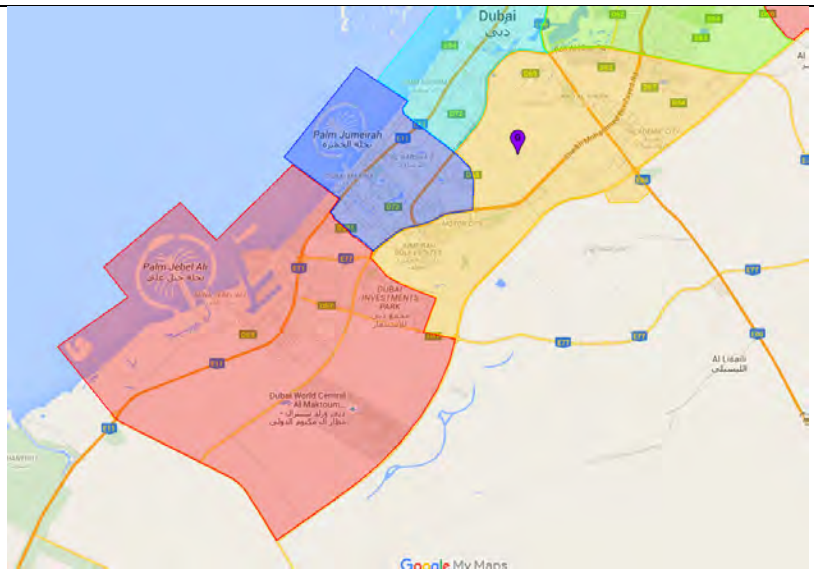
```
function initMap() {
    var map = new
google.maps.Map(document.getElementById('map'), {
        zoom: 11,
        center: {lat: 41.876, lng: -87.624}
      });

      var ctaLayer = new google.maps.KmlLayer({
          url: 'http://googlemaps.github.io/js-v2-samples/
ggeoxml/cta.kml',
          map: map
      });
}
```

In figura viene mostrato il risultato dell'aggiunta di un layer KML molto semplice.



E' tuttavia possibile disegnare quasi ogni tipo di figura, definendo il file KML.



Una volta presa confidenza con le tecnologie di geolocalizzazione e con le API di Google maps, è possibile combinare le due funzionalità. Nell'esempio, dopo aver inizializzato una mappa, viene richiesta la geolocalizzazione tramite HTML5.

```
function initMap() {
    map = new google.maps.Map(document.getElementById('map'), {
        center: {lat: -34.397, lng: 150.644},
        zoom: 6
    });

    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(function(position) {
            var pos = {
                lat: position.coords.latitude,
                lng: position.coords.longitude
            };
            map.setCenter(pos);
        }, function() {
            //handle error
        });
    } else {
        // Browser doesn't support Geolocation
    }
}
```

Se la richiesta ha buon fine, viene impostato il centro della mappa alla posizione corrente. Come risultato, la mappa sarà centrata correttamente sulla posizione corrente ottenuta.


Mappa
Satellite
Francia
Svizzera
Liechtenstein
Graz
Slovenia
Zagabria
Trieste
Ginevra
Limoges
Clermont-Ferrand
Lione
Grenoble
Milano
Verona
Venezia
Padova
Torino
Croazia
Bosnia
Erzegov
Genova
Bologna
Zara
Sar
Nimes
Tolosa
Monaco
Pisa
San Marino
Firenze
Spalato
Montpellier
Marsiglia
Cannes
Italia
Mare Adriatico
Andorra
Girona
Corse
Roma
Barcellona
Google
Map data ©2017 GeoBasis-DE/BKG (©2009), Google, Inst. Geogr. Nacional, ORION-ME
Termini e condizioni d'uso

5. ACQUISIZIONE E ARCHIVIAZIONE DI DATI DA SENSORE

In questo capitolo verranno presentati strumenti e metodologie ottimali per costruire un componente software, in particolare si proporranno dei Web Service, in grado di acquisire, archiviare ed eventualmente elaborare dei dati di diversa natura.

Verranno dunque introdotti e definiti i sistemi di acquisizione e verranno accennate le best practises per costruirne uno. Sarà poi introdotto il concetto di database, con un piccolo approfondimento su alcuni tipi più usati, e saranno accennato alcuni strumenti e pratiche per costruire un sistema che acquisisca e immagazzini dati da sensori di diversa natura.

5.1 Definizione ed architettura di un sistema di acquisizione

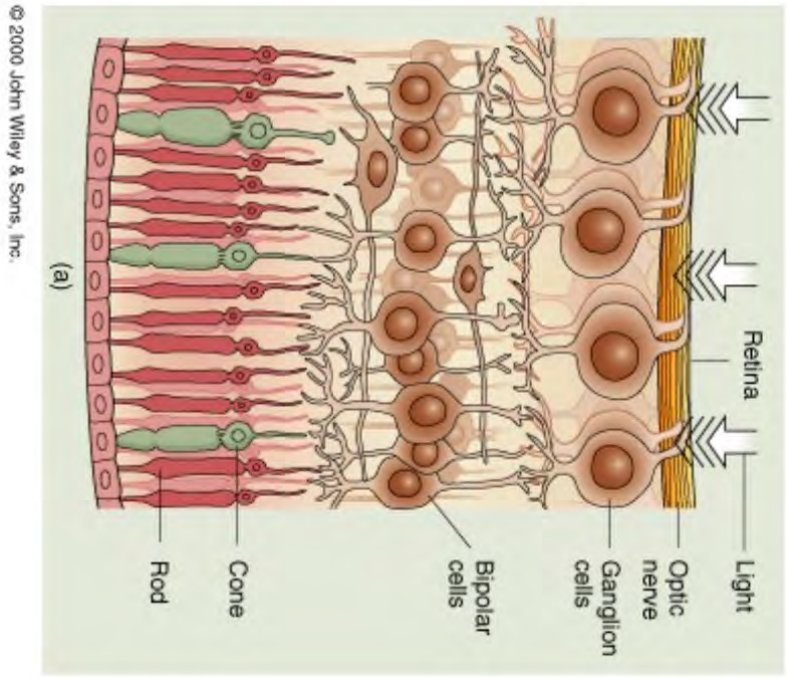
Un sistema di acquisizione è un insieme di strumenti che opportunamente collegati e configurati realizzano una o più catene di misura.

Il caso più elementare di sistema d'acquisizione dati è quello composto da due soli strumenti:

- un sensore che legge la grandezza fisica d'interesse, e la converte in un segnale elettrico;
- un acquisitore che legge il segnale elettrico e provvede alla sua registrazione.

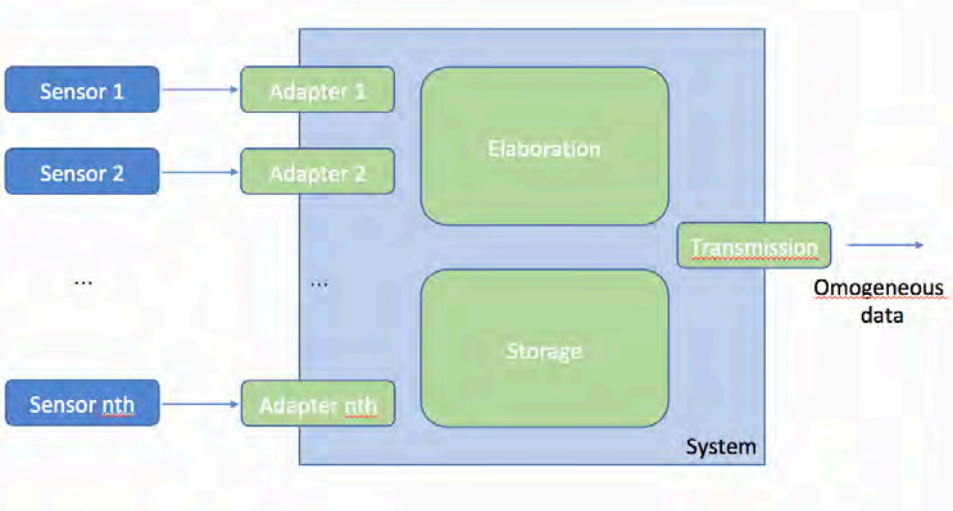
L'intero sistema, dal sensore che viene applicato al misurando, fino al supporto dello strumento registratore, costituisce una catena di misura.

L'occhio, per esempio è un complesso sistema di acquisizione dati che traduce la luce registrata dalla retina (il sensore) in impulsi elettrici trasmessi al cervello e da questo elaborati.



Questi sistemi fanno parte della vita di tutti i giorni più di quanto non ci si possa inizialmente aspettare: da sensori ottici montati su ascensori o cancelli, al microfono che acquisisce audio, agli accelerometri montati sulla maggior parte dei cellulari che ne registrano l'accelerazione nello spazio, e molti altri.

Per costruire un componente software che possa gestire dati acquisiti da diversi tipi di sensori, è importante strutturare architetturalmente questo componente in modo tale da gestire l'eterogeneità dei dati.



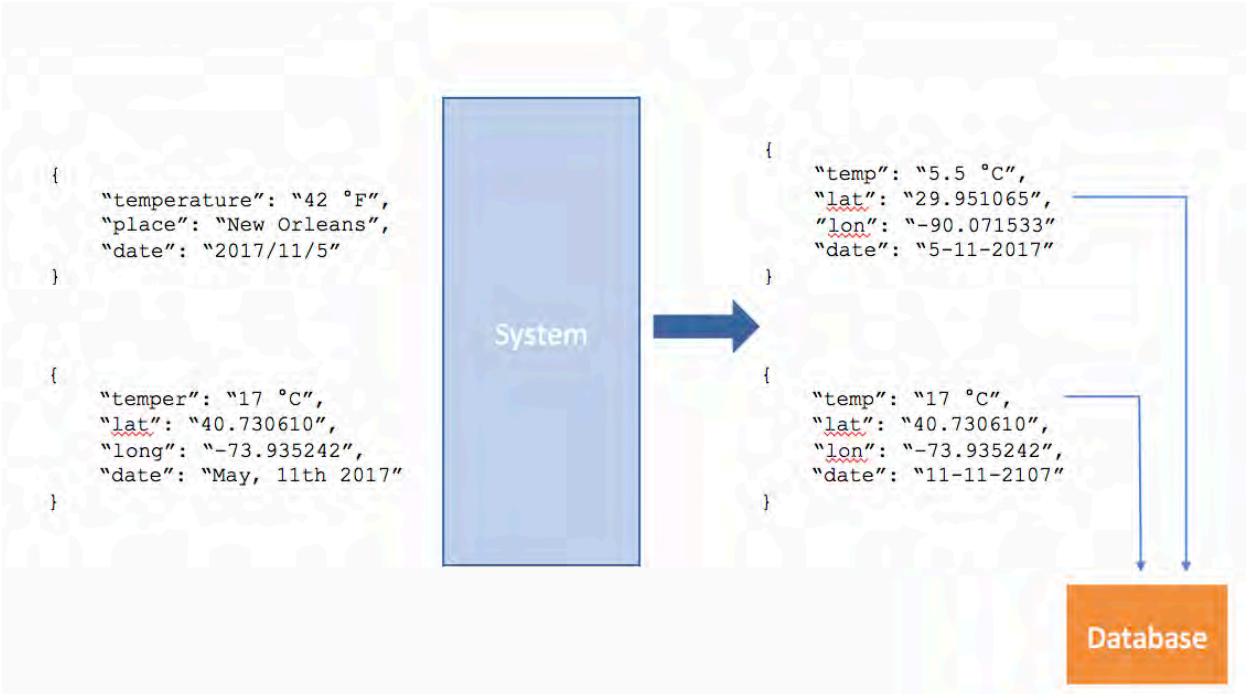
Una tipica architettura per componenti di questo tipo, prevede l'utilizzo di diversi adapter.

Ogni adapter avrà il compito di acquisire e un tipo di dato e di normalizzarlo in modo che all'interno del sistema i dati siano uniformi, abbiano cioè lo stesso formato, nonostante abbiano natura diversa tra loro.

Un classico esempio potrebbe essere un applicativo Web che recuperi dati relativi al meteo da 2 diversi Web services. Ogni Web Service potrebbe potenzialmente esporre i propri dati con un formato diverso dagli altri, e, addirittura, potrebbe usare standard di trasmissione diversi (JSON o XML per esempio).

Per poter archiviare ed eventualmente elaborare questi dati eterogenei, il nostro applicativo Web ha bisogno di adapter diversi per ogni sorgente di informazione interrogata. Gli adapter, come mostrato in figura, avranno il compito di uniformare il formato dei dati in entrata: questo include, per esempio:

- l'eventuale uniformazione dello standard di trasmissione,
- l'uniformazione dei nomi delle proprietà dei dati,
- l'uniformazione delle unità di misura dei dati (per esempio l'unità di misura usata per la temperatura),
- L'uniformazione del formato delle proprietà (es il formato della data),
- La gestione di eventuali informazioni mancanti da una o più fonti,
- E altro..



5.2 Strumenti per l'archiviazione dei dati

Un database è un archivio di dati strutturato in modo da razionalizzare la gestione e l'aggiornamento delle informazioni e da permettere lo svolgimento di ricerche complesse.

Un **Database Management System** è invece un sistema software progettato per consentire la creazione e la manipolazione e l'interrogazione efficiente di un database, per questo detto anche "gestore o motore del database", e ospitato su architettura hardware dedicata oppure su semplice computer.

Esistono diversi tipi di database con caratteristiche diverse, e progettati per gestire al meglio diversi tipi di dati. Tra i più usati ci sono: database relazionali, database document-oriented e Graph database.

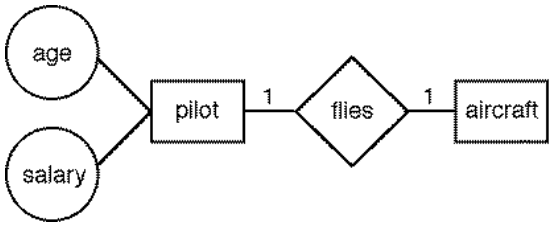
5.2.1 Database relazionali

Il modello **relazionale** è un modello logico di rappresentazione o strutturazione dei dati di un **database** implementato su sistemi di gestione di basi di dati (DBMS), detti perciò sistemi di gestione di basi di dati **relazionali** (RDBMS).

Database relazionali: i database appartenenti a questa categoria si basano sul modello relazionale la cui struttura principale è la relazione, cioè una tabella bidimensionale composta da righe e colonne. Ciascuna riga, che in terminologia relazionale viene chiamata tupla, rappresenta un'entità che noi vogliamo memorizzare nel database. Le caratteristiche di ciascuna entità sono definite invece dalle colonne delle relazioni, che vengono chiamate attributi. Entità con caratteristiche comuni, cioè descritti dallo stesso insieme di attributi, faranno parte della stessa relazione.

Ad esempio, se nel database si dovranno rappresentare delle persone, si potrà definire una relazione chiamata "Persone", i cui attributi descrivono le caratteristiche delle persone. Ciascuna tupla della relazione "Persone" rappresenterà una particolare persona.

il **modello entity-relationship** (anche detto **modello E-R**, in italiano **modello entità-associazione** o desuetamente '***modello entità-relazione***') è un modello per la rappresentazione concettuale dei dati ad un alto livello di astrazione.



Tramite esso si può modellare la struttura di un database, definendo le entità, i loro attributi e le relazioni che intercorrono tra esse.

Questo diagramma, utile nel design di un database, potrà essere utilizzato come punto di partenza per implementare le relazioni (tabelle) della base di dati.

Pilot

name	ge	id
John	35	1
Richard	58	2

Aircraft

type	weight	id
B744	100.000	1
A388	95.000	2

In figura possiamo vedere le due tabelle (Pilot e Aircraft) che sono state implementate partendo dal precedente diagramma.

Numerose sono le operazioni che possono essere effettuate su un database per manipolare i dati al suo interno:

- Insert per inserire una nuova entità,
- Update per aggiornare gli attributi di un'entità già esistente,
- Select per selezionare una o più entità,
- Project: restituisce una relazione con un sottoinsieme degli attributi della relazione a cui viene applicato. Le tuple della relazione risultato vengono composte dalle tuple della relazione originale in modo che continuino ad essere un insieme in senso matematico,
- Join: vengono concatenate le tuple di due relazioni in base al valore di un insieme dei loro attibuti,

- Union: applicando questo operatore a due relazioni compatibili, se ne ottiene una contenente le tuple di entrambe le relazioni. Due relazioni sono compatibili se hanno lo stesso numero di attributi e gli attributi corrispondenti nelle due relazioni hanno lo stesso dominio.

SQL (Structured Query Language) è un linguaggio per interagire con i database relazionali e per manipolare e accedere ai dati in essi contenuti.

Vedremo qualche esempio di richiesta di accesso o manipolazione ai dati. Questo tipo di richiesta è solitamente detta query. La prima query che vediamo ha come effetto la **creazione** di un database con nome database_name:

```
CREATE DATABASE database_name;
```

Una volta creato il database, si può aggiungere ad esso una tabella.

```
CREATE table table_name(
col_name col_type [...]
[ , col_name col_type [...]
...);

CREATE table Book (
ID INTEGER PRIMARY KEY REFERENCES Publication(ID),
title VARCHAR(160) NOT NULL,
publisher INTEGER NOT NULL REFERENCES Publisher(ID),
volume VARCHAR(16),
series VARCHAR(160),
edition VARCHAR(16),
pub_month CHAR(3),
pub_year INTEGER NOT NULL,
note VARCHAR(255)
);
```

Oltre alle diverse opzioni che si possono usare, è obbligatorio indicare almeno un attributo della tabella, indicandone nome e tipo.

Nell'esempio definiamo una tabella book con una serie di attributi. Alcuni attributi, come pub_year (anno di pubblicazione) per esempio, sono dei numeri interi; altri, come title, sono delle stringhe.

Creata la tabella, si può cominciare a popolarla, inserendo dei dati con la query **INSERT**. Nell'esempio popoliamo la tabella Person, aggiungendo una persona di nome Massimo Melucci.

```
INSERT INTO table_name[ ( attribute1, attribute2, ...) ]
VALUES ( value_attr1, value_attr2, ...);
```

```
INSERT INTO Person('id', 'surname', 'name') VALUES ( 3, 'Melucci',
'Massimo' );
```

Per accedere ai dati inseriti nel database, si utilizza la query **SELECT**.

```
SELECT [DISTINCT ] selected_items_list
FROM tables_list
[ WHERE expression]
[ GROUP BY col_list]
[ HAVING expression]
[ ORDER BY col_list]
```

```
SELECT Person.surname  FROM Person;

SELECT Person.surname, Person.given_names FROM Person

SELECT * FROM Book WHERE pub_year = 1983 OR pub_year = 1993 OR
pub_year = 1980
```

Questa query è composta da diverse clausole. Le principali sono:

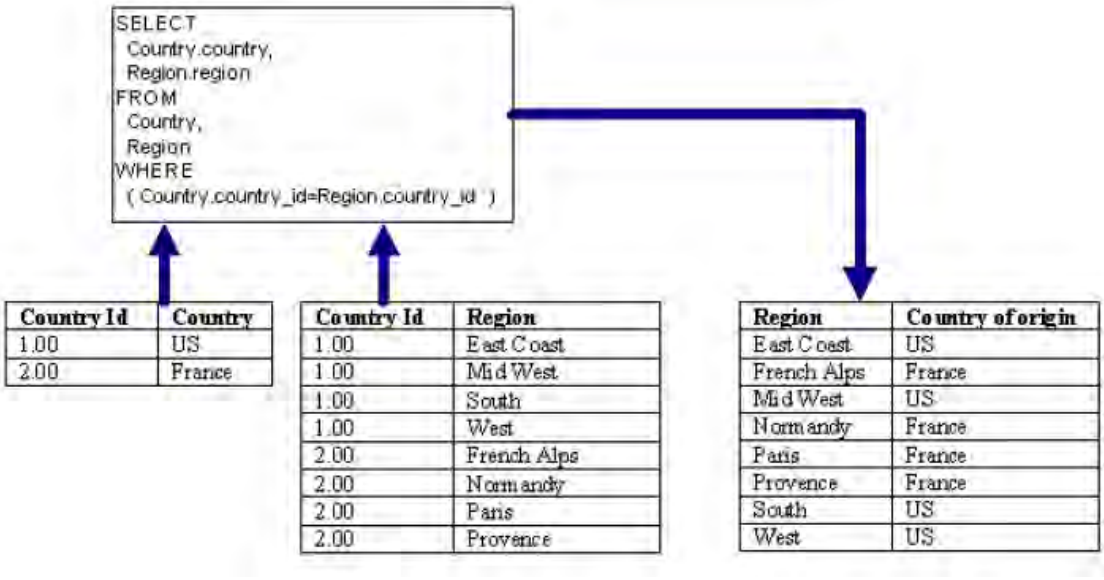
- **Select**: è la parte dove vengono elencati gli attributi che si vogliono selezionare
- **FROM**: dove si indica la tabella (o le tabelle) dalla quale si vogliono estrarre i dati
- **WHERE**: è la clausola dove si filtrano le entità (per esempio se si vuole selezionare tutte le persone maggiorenni, lo si indica in questa clausola).

Il **JOIN** è una clausola del linguaggio SQL che serve a combinare le tuple di due o più relazioni di un database ramite l'operazione di congiunzione dell'algebra relazionale.

Nell'esempio vengono associati i nomi delle regioni ai nomi delle nazioni appartenenti, dati che inizialmente risiedevano in due tabelle diverse.

SQL supporta anche la possibilità di manipolare dati geografici. Geometry è un tipo di attributo usato per salvare coordinate geografiche. SQL offre diverse funzionalità per lavorare con i dati geografici: è possibile infatti creare dati geografici, convertirli in diversi formati, accedere a diverse proprietà del dato, descrivere le relazioni tra diversi dati(per esempio calcolare la

distanza tra due punti), creare nuovi dati geografici.



Country Id	Country
1.00	US
2.00	France

Country Id	Region
1.00	East Coast
1.00	Mid West
1.00	South
1.00	West
2.00	French Alps
2.00	Normandy
2.00	Paris
2.00	Provence

Region	Country of origin
East Coast	US
French Alps	France
Mid West	US
Normandy	France
Paris	France
Provence	France
South	US
West	US

5.2.2 Database Document-Oriented

E' inoltre molto importante considerare anche una seconda categoria di database, ovvero i database orientati ai documenti. In questo caso, il database è pensato per immagazzinare dati semi-strutturati, in modo più "libero" rispetto ai classici database relazionali. E' possibile infatti salvare oggetti eterogenei, diversi tra di loro, all'interno della stessa "collezione". Sarà inoltre possibile eseguire delle query su questi dati.

Un tipico utilizzo di questo tipo di database è quando la tipologia dei dati non è nota al momento del design, o se questa può cambiare in corso d'opera.

Il database orientato ai documenti più popolare è senza dubbio MongoDB, che è tra l'altro un progetto opensource e cross platform.

MongoDB usa un formato simile al JSON per salvare i documenti, e implementa un suo linguaggio di query che permette un'ottima flessibilità. E' possibile, ad esempio, eseguire query selezionando elementi di un array, o se una proprietà esiste oppure no.

5.3 Standard OGC per l'integrazione di dati eterogenei

L'integrazione nei modelli e nell'esperienza dei dati provenienti da sensori eterogenei presenta diversi problemi e lo sviluppo di applicazioni e servizi basati su questi dati richiede di analizzare almeno quattro aspetti principali:

- Eterogeneità dei dati prodotti dai sensori;
- Qualità e affidabilità dei dati prodotti da sensori;
- Modalità di accesso ai dati;
- Definizione di strategie tecnologiche e architetture di riferimento per l'integrazione dei
- dati in modelli, applicazioni e servizi.

In particolare per la gestione di dati da sensori eterogenei si parla degli standard Open Geospatial Consortium (OGC). L'OGC è un consorzio industriale internazionale il cui obiettivo principale è la progettazione di codifiche standard interoperabili e interfacce per servizi geografici e dati spaziali. Gli standards promulgati da OGC supportano soluzioni interoperabili atte a "geo-abilitare" il Web, i servizi geo-localizzati e wireless, così come le applicazioni IT tradizionali (e.g. GIS, workflows di modelli previsionali), agevolando quindi il processo di sviluppo tecnologico, rendendo servizi e informazioni spaziali complessi, accessibili e fruibili a tutti i tipi di applicazioni.

Scopo dell'iniziativa Sensor Web Enablement (SWE) del consorzio, è la costruzione di un quadro unico e rivoluzionario di standard aperti per lo sfruttamento di sensori connessi al Web e sistemi di sensori di tutti i tipi: OGC promuove una serie di standard per l'accesso e la fruizione di dati prodotti da sensori, o di servizi che producono questi dati. Per ragioni di sintesi citiamo tra questi:

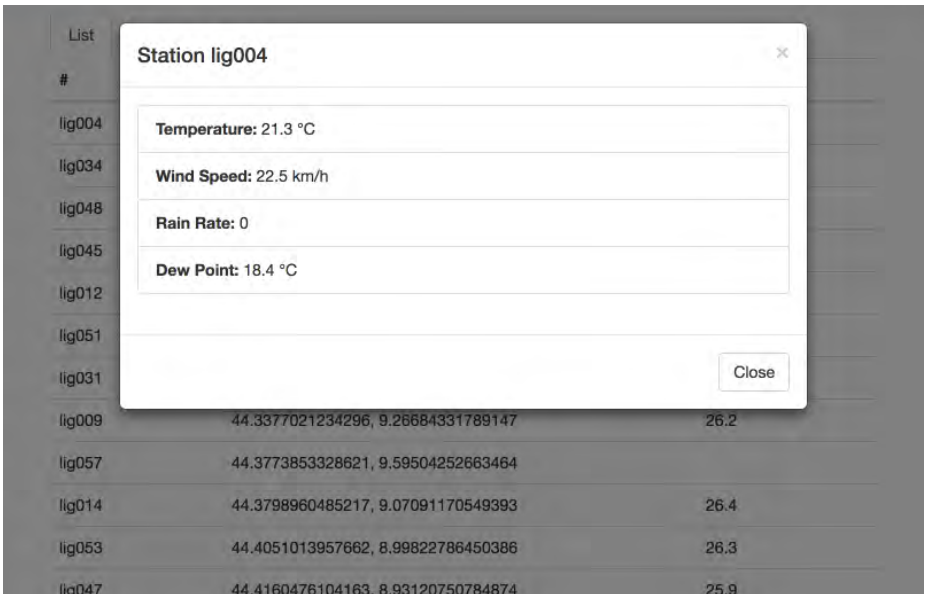
- Observations & Measurements Schema
- Sensor Model Language
- Sensor Alert Service

- Web Notification Services

In particolare, il primo definisce il Modello standard e lo Schema XML per la codifica di osservazioni e misurazioni da un sensore, sia archiviati che in tempo reale.

6. ESEMPIO DI CREAZIONE DI UNA PAGINE WEB PER DATI METEO

In quest'ultimo capitolo, verrà presentata la creazione di una pagina Web, simile a quella mostrata in figura, che permetta la visualizzazione di alcune informazioni delle stazioni meteo in tempo reale utilizzando sia Google Maps che in formato tradizionale in una tabella. Inoltre la pagina permetterà la possibilità di visualizzare i dettagli di ogni stazione meteo cliccando sulla riga o sul marker. Per realizzare questa pagina, verranno quindi utilizzati i concetti e le nozioni definite nel documento.



Il lavoro di creazione della pagina finale può essere diviso in diversi passi.

Per incominciare, verrà realizzato il layout della pagina, utilizzando il framework Bootstrap. Verrà quindi studiato un data model appropriato per rappresentare i dati che vengono recuperati dalle personal weather station. A questo punto, il layout della pagina verrà modificato, aggiungendo alcune funzioni javascript, in modo da visualizzare dei dati che rispecchiano il data model definito nello step precedente. Verrà quindi brevemente introdotto un WS, il cui compito è quello di recuperare ed uniformare i dati relativi alle stazioni meteo. Infine, il WS verrà utilizzato come sorgente di dati dinamici da visualizzare nella pagina web. Come step bonus, inoltre, sarà possibile implementare la visualizzazione di un popup contente maggiori informazioni sulla stazione meteo.

Il primo step è quello di scaricare bootstap direttamente dal sito ufficiale nel link

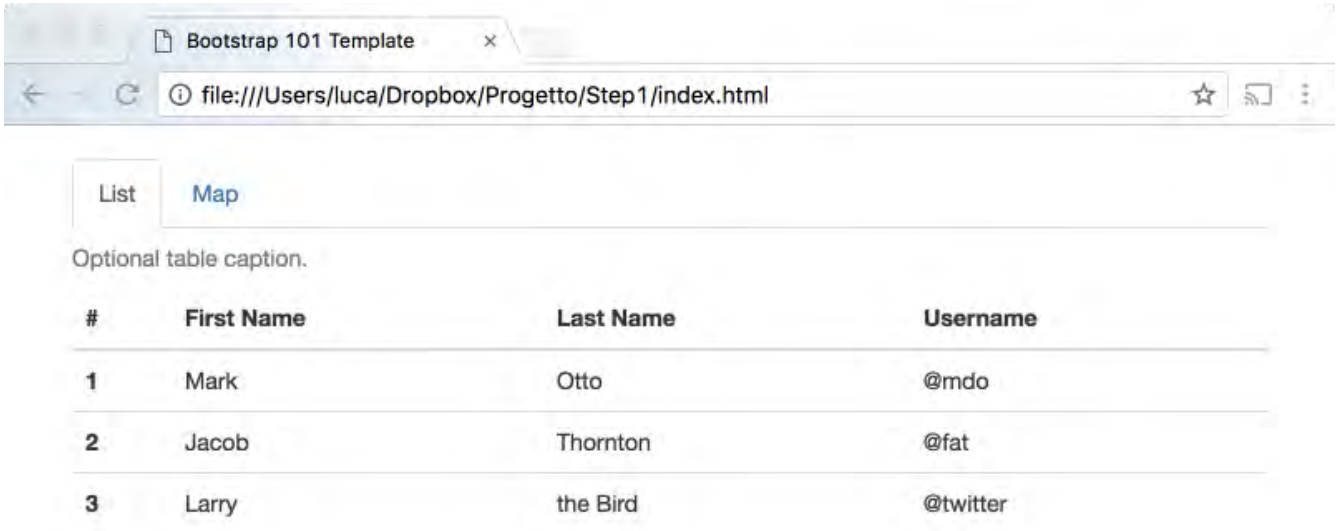
http://getbootstrap.com/getting-started/

e creare quindi una primissima pagina, utile a verificare che sia stato tutto correttamente installato.

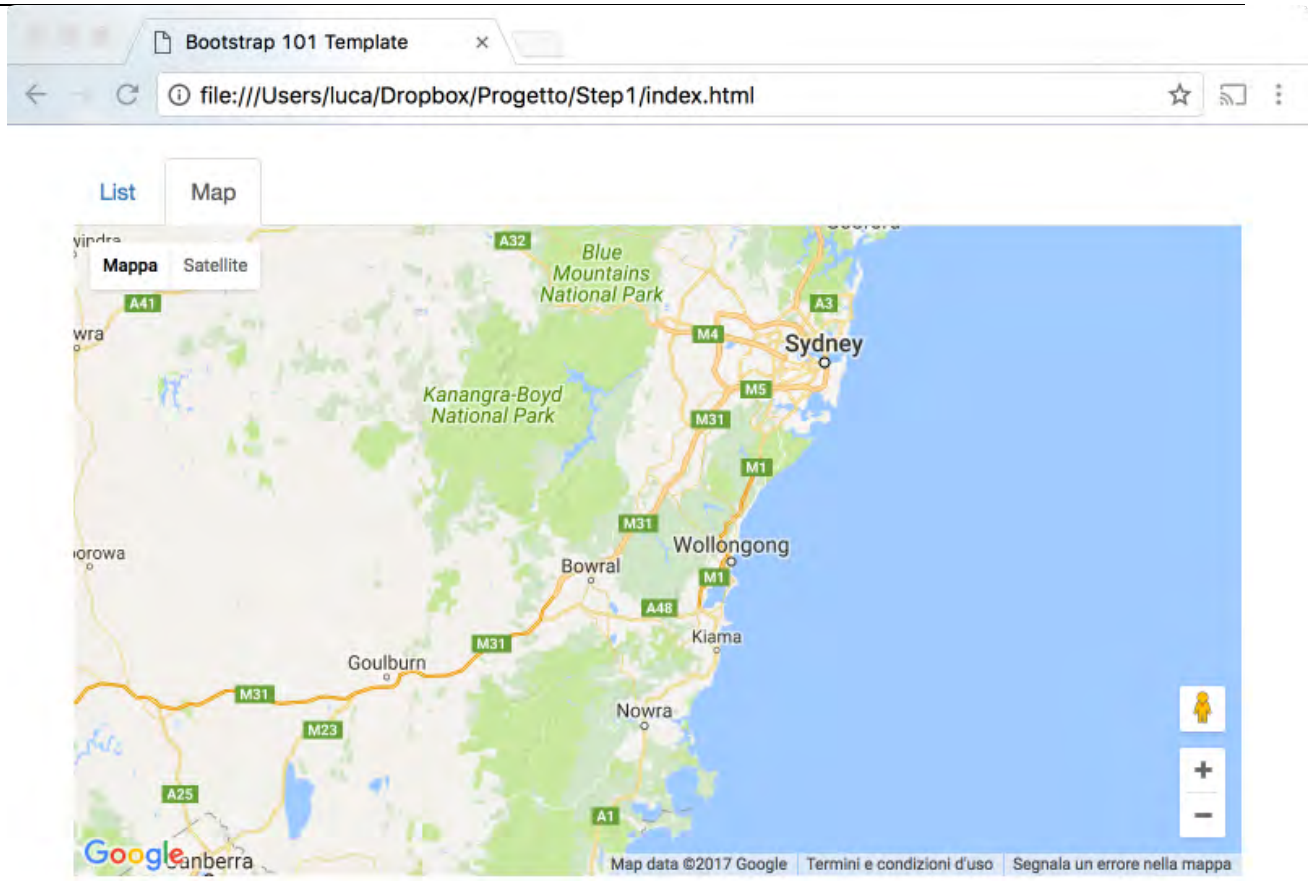
Il passo successivo è quindi quello di utilizzare il componente di Bootstrap “tab” per creare due tab diverse. Usando il codice di esempio della documentazione ufficiale (http://getbootstrap.com/javascript/#tabs) , modificandolo in modo da creare solo due tab, una per la visualizzazione della lista in formato tabellare, e la seconda che conterrà google maps. Il risultato è quello mostrato in figura.

A questo punto, si può passare alla creazione di una tabella per la prima tab (la documentazione relativa alle tabelle si trova al link http://getbootstrap.com/css/#tables).

Utilizzando proprio il codice di esempio, il risultato è il seguente.



Il seconda tab, invece, è un po’ più complessa, e prevede la visualizzazione di google maps. Come integrare la mappa è stato argomento del Capitolo 4.



A questo punto, il layout di base della pagina è stato definito, e possiamo passare allo studio del modello di dati delle weather station. Infatti, il passo successivo è andare a definire un data model per rappresentare i dati delle stazioni che vogliamo visualizzare.

Visto che i dati devono essere visualizzati su una mappa, questi devono essere necessariamente localizzati, e quindi dovranno avere una latitudine e una longitudine, ed ogni stazione avrà un identificativo univoco. Oltre a questi dati di base, ogni stazione dovrà contenere altri dati - cioè quelli di misura. Per semplicità, ci concentriamo sulla temperatura registrata, e quindi sarà presente un campo, “temperatura”, contenente questa informazione.

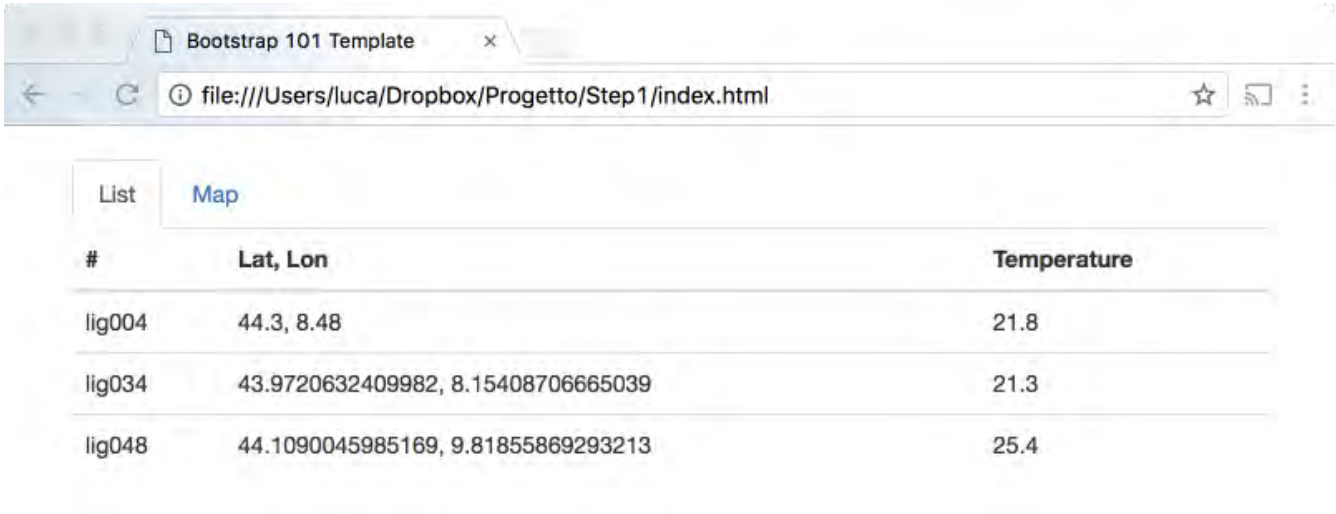
JSON Example:

```
{
    "id": "id9489",
    "latitudine": 12.9,
    "longitudine": 45.0,
    "temperatura": "12"
}
```

Ovviamente, i dati che andremo ad utilizzare saranno array, in quanto vorremo visualizzare

molte stazioni.

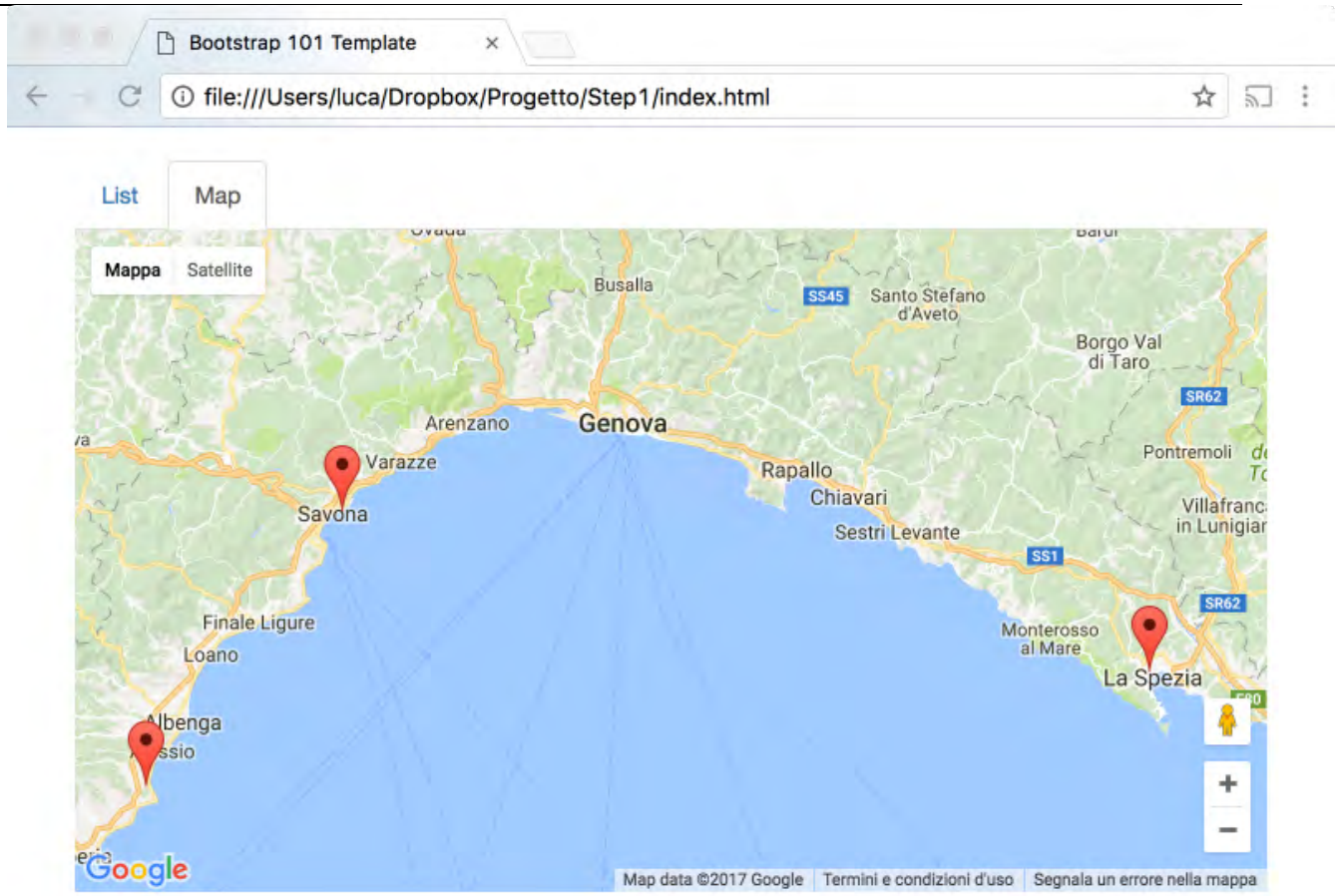
A questo punto, possiamo utilizzare un array di stazioni come dato di esempio, per andare a popolare la tabella e la mappa con dei dati dinamici. Per fare questo, andremo ad utilizzare le funzioni di jQuery viste nei precedenti capitoli. La tabella sarà quindi simile a quella mostrata in figura.



Come aiuto, presentiamo di seguito il codice Javascript necessario per aggiungere una riga alla tabella. Questo dovrà essere inserito in un ciclo, che iteri sull'array delle stazione meteo.

```
$("#table").find("tbody")
      .append($("<tr>")
            .append($("<td>")
                                    .text(data[i].id)
            )
            .append($("<td>")
                                    .text(data[i].latitudine +
", " +                              data[i].longitudine)
            )
                                          .append($("<td>")
                                    .text(data[i].temperatura)
            )
);
```

Nello stesso modo, è anche necessario creare i marker da visualizzare nella mappa, utilizzando le API di Google. Anche in questo caso è già stato trattato come aggiungere marker alla mappa di Google.



Il passo successivo è la definizione di un WS per recuperare i dati delle stazioni e renderli fruibili in modo omogeneo e che rispecchino il data model definito in precedenza.

Esistono due provider, meteo network (http://www.meteonetwork.it/supporto/meteonetwork-api) e weather underground (https://www.wunderground.com/weather/api/), i quali mettono a disposizione due API differenti per recuperare questi dati.

Il Web Service, quindi, deve eseguire periodicamente un task, il quale: recupera i dati da un provider; li normalizza, eseguendo ad esempio conversioni di unità di misura e, per concludere, salva i dati in un database, in modo che siano sempre disponibili e forniscano anche un archivio storico. Il Web Service deve ovviamente fornire una API per ottenere i dati salvati.

Come esempio dimostrativo, abbiamo sviluppato e reso pubblicamente disponibile il web service accessibile tramite il link https://portalts.it/repository/files/wheater_stations/example.json che implementa esattamente quanto detto sopra.

In particolare, questo WS rende fruibili anche altri dati, oltre a quello sulla temperatura, che non sono tuttavia utilizzati in questo esempio applicativo.

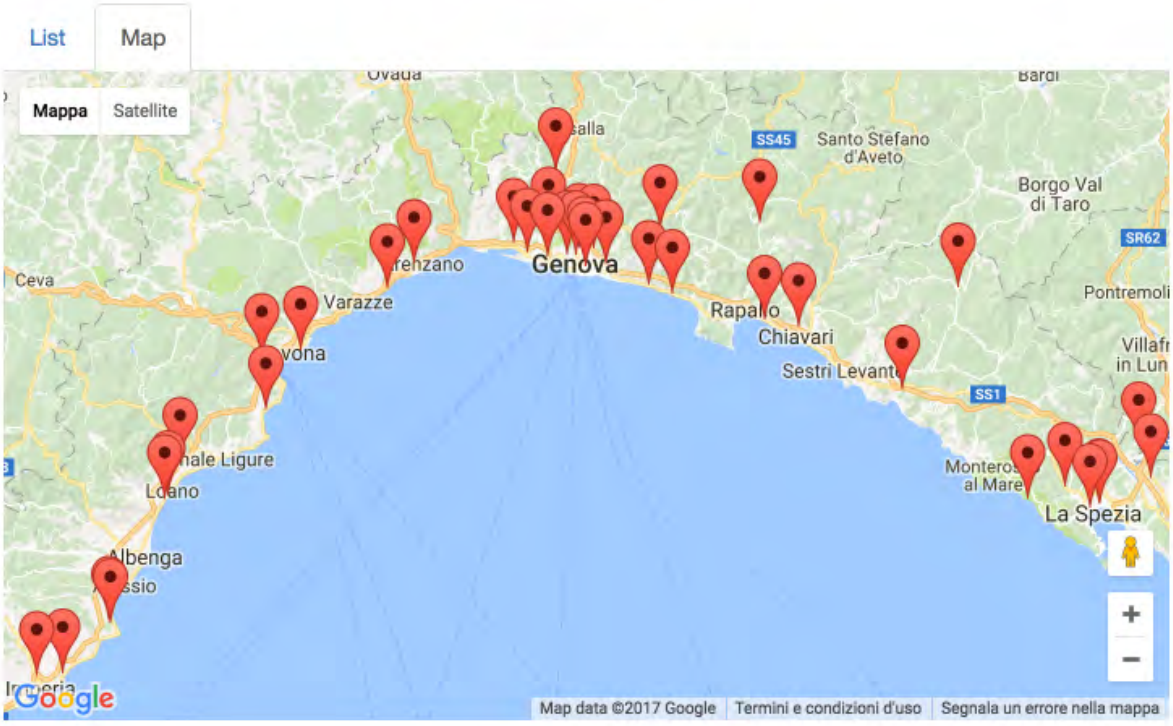
Infine, interroghiamo dinamicamente il web service per ottenere le informazioni delle stazioni, e visualizzarle all'interno della tabella e su Google Map.

Per arrivare a questa funzionalità, si possono usare le funzioni ajax di jQuery, che permettono di eseguire delle richieste web in modo dinamico.

List Map

#	Lat, Lon	Temperature
lig004	44.3, 8.48	21.9
lig034	43.9720632409982, 8.15408706665039	20.9
lig048	44.1090045985169, 9.81855869293213	25.8
lig045	44.1142735479457, 9.83495235443115	24.7
lig012	44.12, 8.25	22.8
lig051	44.145446252325, 9.92164134979248	26.2
lig031	44.1658289815979, 8.27493667602539	
lig009	44.3377021234296, 9.26684331789147	24.4
lig014	44.3798960485217, 9.07091170549393	24.8
lig053	44.4051013957662, 8.99822786450386	21.9
lig016	44.4094088501931, 8.9606299996376	22.5

Quella mostrata in figura è invece la visualizzazione delle weather station su mappa.



Come step aggiuntivo, consigliamo anche la creazione di un popup quando l'utente clicca su di una riga della tabella o di un marker su Google Map.

Per implementare il popup, è possibile utilizzare i modali di bootstrap (http://getbootstrap.com/javascript/#modals), che sono attivabili anche tramite una funzione Javascript. Ovviamente, sarà necessario iniettare le informazioni corrette della singola weather station da visualizzare nel popup, utilizzando le funzioni jQuery html o altre simili.

Per rendere più carina l'interfaccia, è inoltre possibile appoggiarsi ai dati ritornati dal WS di

esempio in modo da visualizzare anche altre informazioni oltre a quelle già prese in considerazione in precedenza.

Grazie a questo esempio dimostrativo, abbiamo quindi utilizzato tutti gli argomenti visti nei capitoli precedenti e li abbiamo uniti, per poter creare una vera e propria applicazione Web che permetta la visualizzazione di dati geolocalizzati.

Ovviamente la stessa strategia potrebbe essere utilizzata per visualizzare qualsiasi tipo di dato, ad esempio si potrebbero visualizzare i dati degli ultimi terremoti, o segnalare eventuali incendi o altri eventi pericolosi.

7. BIBLIOGRAFIA

Rif.	Data	Titolo
1	2015	Muller C.L., Chapman L., Johnson S., Kidd C., Illingworth S., Foody G., Overeem A. and Leigh R.R.: Crowdsourcing for climate and atmospheric sciences: current status and future potential, International Journal of Climatology, 2015, wileyonlinelibrary.com
2	2014	Tsai P-H., Lin Y-J., Ou Y-Z., Chu Edward T-H., Liu Jane W.S.: A framework for fusion of human sensor and physical sensor data, IEEE Transactions on Systems, Man ans Cybernetics: Systems, vol.44, No.9, September, 2014
3	2011	Boulos M.N.K., Resch B., Crowley D.N., Breslin J.G., Sohn G., Burtner R., Pike W.A., Jezierski E., Chuang K.-Y.S.: Crowdsourcing, citizen sensing and sensor web technologies for public and environmental health surveillance and crisis management: trends, OGC standards and application examples, International Journal of Health Geographics 10: 67, 2011
4	2012	Bedrina T., Parodi A., Quarati A., Clematis A.: ICT approaches to integrating institutional and non-institutional data services for better understanding of hydro-meteorological phenomena, Natural Hazards Earth Systems Science, 12, 1961-1968, 2012
5	2012	Bröring, A., Stasch, C., Echterhoff, J.: OGC® Sensor Observation Service Interface Standard Version 2.0. OGC document 12-006, 2012. Accessible from: http://www.opengis.net/doc/IS/SOS/2.0
6	2013	Fiourucci, P., Molini, L., Parodi, A.: Spatio-temporal relative humidity patterns and extreme wildfires in the Mediterranean. International Journal of Wildland Fire, 2013
7	2015	NOAA Citizen Weather Observer Program (CWOP). Accessible from http://www.wxqa.com
8	2008	MADIS Meteorological Surface Quality Control, 2008. Accessible from http://madis.noaa.gov/madis_qc.html
9	2004	Zahumensky, I.: Guidelines on Quality Control Procedures for Data from Automatic Weather Stations, World Meteorological Organization, Commission for basic systems open programme area group on integrated observing systems, Expert team on requirements for data from automatic weather stations, Third session, CBS/OPAG-IOS/ET AWS-3/Doc. 4(1), 25.V.2004, 2004
10	2012	Taylor, P.:OGC® WaterML 2.0, Version 2.0, OGC Document 10-126, 2012. Accessible from: http://www.opengis.net/doc/waterml/2.0
11	2015	T. Bedrina, A. Clematis, A. Quarati, Crowdsourcing Contribution in Weather Observation, Rapporto Tecnico IMATI – CNR 2015
12	2007	Chow, S.-W.: PHP Web 2.0 Mashup Projects. In: Packt Publishing, Packt Publishing Ltd., Birmingham, B27 6PA, UK, 304, ISBN-13: 978-1-84719-088-8, 2007
13	2009	Ogrinz, M.: Mashup Patterns: Designs and Examples for the Modern Enterprise. ISBN-13:978-0-321-57947-8, 2009
14	2009	Fichter, D., What is Mashup? University of Saskatchewan

Rif.	Data	Titolo
		Library, 2009
15	2009	Peenikal, S., Mashups and enterprise. White Paper. Mphasis an HP company, 2009
16	2011	WILLIAMS, M., CORNFORD, D., BASTIN, L., JONES, R., PARKER, C.: AUTOMATIC PROCESSING, QUALITY ASSURANCE AND SERVING OF REAL-TIME WEATHER DATA. COMPUTERS AND GEOSCIENCES, 37, 353-362, 2011
17	2009	WANG, P., BISHOP, I.D., STOCK, C.: REAL-TIME DATA VISUALIZATION IN COLLABORATIVE VIRTUAL ENVIRONMENTS FOR EMERGENCY RESPONSE. IN: PROCEEDINGS OF THE SURVEYING & SPATIAL SCIENCES INSTITUTE BIENNIAL INTERNATIONAL CONFERENCE, ADELAIDE, SURVEYING & SPATIAL SCIENCES INSTITUTE, 435-441, ISBN: 978-0-9581366-8-6, 2009
18	2009	YONG, L., HILL, D., MARINI, L., KOOPER, R., RODRIGUEZ, A., MYERS, J: WEB 2.0 GEOSPATIAL VISUAL ANALYTICS FOR IMPROVED URBAN FLOODING SITUATIONAL AWARENESS AND ASSESSMENT. ACM GIS '09 , NOVEMBER 4-6, 2009
19	2014	T. Bedrina, A. Parodi, A. Clematis, A. Quarati, Mashing-up weather networks data to support Hydro-Meteorological research. In Advanced Research and Trends in New Technologies, Software, Human-Computer Interaction, and Communicability, Chapter: 23, Publisher: IGI Global, Editors: Francisco Vicente Cipolla-Ficarra (ALAIPO – AINCI, Spain and Italy, pp.245-254, 2014.
20	2009	Pastorello, Jr. G.Z., A.Senra, R.D., B.Medeiros, C., A standards-based framework to foster geospatial data and process interoperability. Journal of the Brazilian Computer Society, 15(1):13-25, 2009.
21	2010	Tomasz Kobialka, Rajkumar Buyya, Peng Deng, Lars Kulik, Marimuthu Palaniswami, Sensor Web: Integration of Sensor Networks with Web and Cyber Infrastructure, Handbook of Research on Developments and Trends in Wireless Sensor Networks: From Principle to Practice, 447-473pp, H. Jin and W. Jiang (eds), ISBN: 978-161-520-701-5, IGI Global, Hershey, PA, USA, Feb. 2010.
22	2013	Delano Medeiros Beder, Jó Ueyama, João Porto de Albuquerque, Marcos Lordello Chaim, FlexFT: A Generic Framework for Developing Fault-Tolerant Applications in the Sensor Web, International Journal of Distributed Sensor Networks, Vol. 2013, Article ID 385892, 12 pages, Hindawi Publishing Corporation http://dx.doi.org/10.1155/2013/385892
23	2010	Ingelrest, F., Barrenetxea, G., Schaefer, G., Vetterli, M., Couach, O., and Parlange, M. 2010. Sensor- Scope: Application-specific sensor network for environmental monitoring. ACM Trans. Sensor Netw. 6, 2, Article 17 (February 2010), 32 pages.DOI = 10.1145/1689239.1689247 http://doi.acm.org/10.1145/1689239.1689247
24	2009	XU LI, WEI SHU, MINGLU LI, HONG-YU HUANG, PEI-EN LUO, MIN-YOU WU, PERFORMANCE EVALUATION OF VEHICLE-BASED MOBILE SENSOR NETWORKS FOR TRAFFIC MONITORING, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, VOL. 58, NO. 4, PP. 1647-1653, 2009
25	2010	Binbin Zhou, Jiannong Cao, Xiaoqin Zeng, Hejun Wu, Adaptive Traffic Light Control in Wireless Sensor Network-based Intelligent Transportation System, …, 2010
26	2012	M. Gentili, P.B. Mirchandani, Locating sensors on traffic networks: Models, challenges and research opportunities, Transportation Research Part C 24 (2012) 227–255
27	2011	Haifeng Li, Xinsha Fuc, An Intelligence Traffic Accidence Monitor

Rif.	Data	Titolo
		System Using Sensor Web Enablement, Procedia Engineering 15 (2011) 2098 – 2102, Elsevier
28	2009	Jaehun Joo , Jaegeol Yim & Choong-Ki Lee (2009) Protecting cultural heritage tourism sites with the ubiquitous sensor network, Journal of Sustainable Tourism, 17:3, 397-406, DOI: 10.1080/09669580802582498
29	2010	Kenteris, Michael, Damianos Gavalas, and Aristides Mpitziopoulos. "A mobile tourism recommender system." Computers and Communications (ISCC), 2010 IEEE Symposium on. IEEE, 2010.
30	2013	Groen, Maarten, Wouter Meys, and Mettina Veenstra. "Creating smart information services for tourists by means of dynamic open data." Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication. ACM, 2013

Recent titles from the IMATI-REPORT Series:

2018

18-01: *Arbitrary-order time-accurate semi-Lagrangian spectral approximations of the Vlasov-Poisson system*, L. Fatone, D. Funaro, G. Manzini.

18-02: *A study on 3D shape interaction in virtual reality*, E. Cordeiro, F. Giannini, M. Monti, A. Ferreira.

18-03: *Standard per la gestione e procedure di validazione di dati meteo da sensori eterogenei e distribuiti,* A. Clematis, B. Bonino, A. Galizia.

18-04: *Strumenti e procedure per la gestione e la visualizzazione di dati meteo prodotti da sensori eterogenei e distribuiti tramite interfacce web basate su mappe geografiche,* L. Roverelli, G. Zereik, B. Bonino, A. Gallizia, A. Clematis.

Istituto di Matematica Applicata e Tecnologie Informatiche ``*Enrico Magenes",* CNR
Via Ferrata 5/a, 27100, Pavia, Italy
Genova Section: Via dei Marini, 6, 16149 Genova, Italy • **Milano Section:** Via E. Bassini, 15, 20133 Milano, Italy
http://www.imati.cnr.it/