MODULO 2
PARTE 5.c

Programmare sul Web 2.0
Open API:
Google Chart e *Google Maps*

Open API - I

Open API:

- **API** = *Application Programming Interface* (Interfaccia di Programmazione di un'Applicazione) = strumento per rendere disponibile ad altri programmatori le funzionalità di un programma
- **Open** = "aperte", disponibili a tutti

Nell'ambito del Web 2.0 le Open API sono disponibili sul **web** (sfruttano le tecnologie – i protocolli, per es. HTTP – del web)

⇒ un programmatore può **includere** nel suo programma (per es. in un suo sito dinamico o applicazione web) **funzionalità offerte da altri programmi = MASHUP**

NB Spesso le Open API sono offerte come **Web Services**
[rif: www.w3schools.com/webservices/ws_intro.asp, www.w3.org/2002/ws/]

Open API - II

Mashup

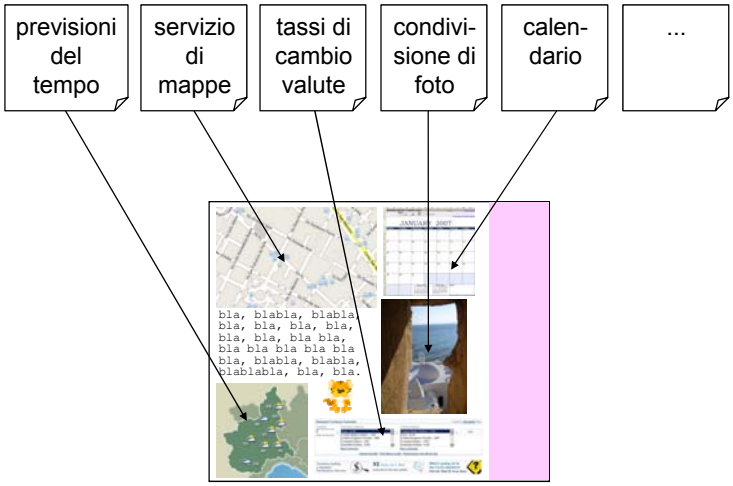
[http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29]:

In Web development, **a mashup is a Web page or application that uses and combines data**, presentation or functionality from two or more sources **to create new services**.

The term implies easy, fast integration, frequently using open APIs and data sources to produce enriched results that were not necessarily the original reason for producing the raw source data

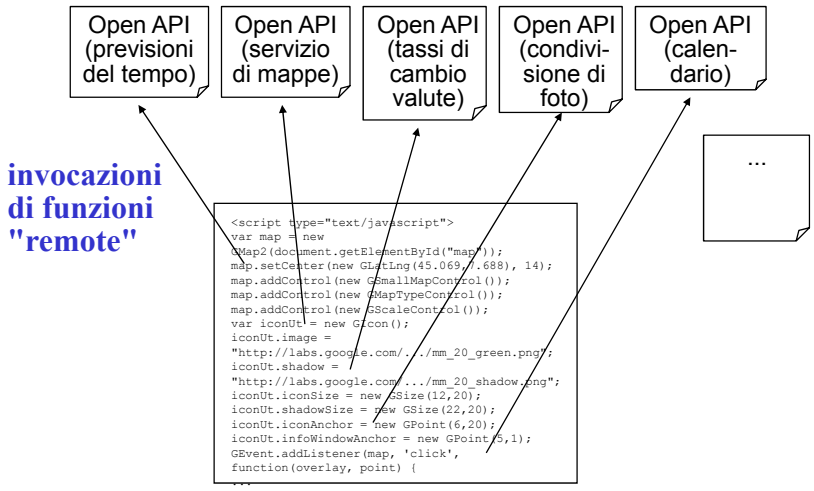
Open API - III

Mashup: fronted



Open API - IV

Mashup: backend ("dietro le quinte")



Open API - V

Per farvi un'idea...

http://www.programmableweb.com/apis/directory
dove (al 17/6/11) erano elencate 3.351 APIs...!
(+ 5.865 esempi di utilizzo = *mashup*)

Qualche esempio (tra i tanti)...

YAHOO! *Yahoo Traffic API*

Traffic Web Services from Yahoo! enable you to get traffic alert information from a given location. Use the Traffic REST API to customize your request with many parameters including indicating locations based on city state, zip code, or a combination of any of these things, latitude-longitude, whether to include a map image, or a search radius in miles. [...]

currencies *Currencies Exchange Rates API*

The Currencies API provides a single JSON structure that is a matrix containing the exchange rates for all known currencies. It is a read only service.

Open API - VI

WeatherBug API

WeatherBug is a full source weather provider featuring exclusive data from its own network of over 8,000 weather observation stations in the USA. The API gives you access to live weather conditions, forecasts and severe weather warnings for all US zip codes.

WhitePages.com API

The WhitePages.com API gives you free access to do people search, reverse phone lookups and reverse address lookups. The powerful database offers access to millions of people records. Build mashups with the REST-based API offering XML and JSON support.

Flickr API

The Flickr API can be used to retrieve photos from the Flickr photo sharing service using a variety of feeds - public photos and videos, favorites, friends, group pools, discussions, and more. The API can also be used to upload photos and video.The Flickr API supports many protocols including REST, SOAP, XML-RPC. Responses can be formatted in XML, XML-RPC, JSON and PHP.

Open API - VII

Entellium API

Online CRM solutions, software and Web-hosted CRM applications for small business. API is available only to partners.

Google Chart API

The Google Chart API lets you dynamically generate charts with a URL string and embed these charts on a web page, or download the image for local or offline use. The Google Chart Tools enable adding live charts to any web page. They provide advantages such as a rich gallery of visualizations provided as image charts and interactive charts and they can read live data from a variety of data sources. Users embed the data and formatting parameters in an HTTP request, and Google returns a PNG image of the chart. Many types of chart are supported, and by making the request into an image tag the chart can be included in a web page.

Google Chart API - I

Google Chart API (http://code.google.com/intl/it-IT/apis/chart/) viene invocato attraverso ***HTTP request***, inserendo nell'URL i dati e i parametri di formattazione → Google restituisce (***HTTP response***) un'immagine in formato PNG

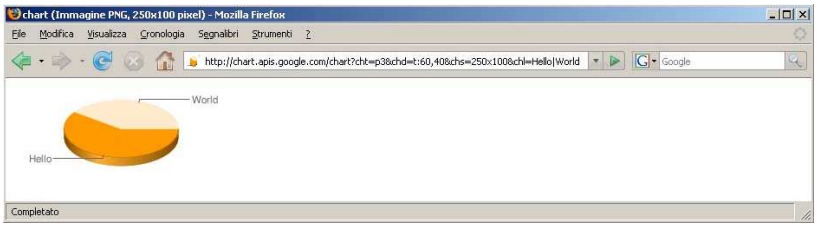


Chart API server programma (che genera il grafico)

http://chart.apis.google.com/**chart**?
chs=250x100&chd=t:60,40&cht=p3&chl=Hello|World

dimensioni (in pixels) dati tipo etichette

parametri (passati al programma)

Google Chart API - II

- Per **includere un grafico in una pagina web** è sufficiente inserire l'URL in un tag **`<img>`**, come valore dell'attributo **`src`**

  Es: demoGoogleChart.html

  NB: quando si include l'URL che invoca i Chart API (come valore dell'attributo **`src`** in un tag **`<img>`**), bisogna usare ***&*** per ottenere il carattere & !!!

  ```
  <img src="http://chart.apis.google.com/chart?
  chs=250x100&amp;chd=t:60,40&amp;cht=p3&amp;chl=Hello|World"
  alt="Sample chart" />
  ```

- **Parametri obbligatori**:
  1. dimensioni
  2. dati
  3. tipo

  Tutti gli altri parametri sono **opzionali** (vedi http://code.google.com/intl/it-IT/apis/chart/basics.html)

Google Chart API - III

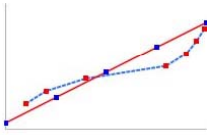
1. **Dimensioni**
   `chs = <larghezza in pixels> x <altezza in pixels>`
2. **Dati**
   ci sono diversi formati; il più semplice è fornire una stringa di numeri (decimali) positivi (compresi tra 0 e 100)
   NB: se ci sono più insiemi di dati, come separatore si usa | ;
   `-1` = *valore mancante* [vedi esempio slide successiva]
   `chd = t:<chart data string>`
   `chd = t:10.2,58.0,-1,95.0|30.0,8.6,63.7,17.0`
3. **Tipo**
   ci sono diversi tipi di grafico: *line charts*, *bar charts*, *pie charts*, ecc... (vedi http://code.google.com/intl/it-IT/apis/chart/types.html)

Google Chart API - IV

- ***Line charts***:
  `cht = <line chart style>`
  In cui `<line chart style>` può essere `lc`, `ls`, `lxy` :

```
cht=lc
chd=t:40,60,60,45,47,75,70,72
```

```
cht=ls
chd=t:27,25,60,31,25,39,25,31,26,28,
      80,28,27,31,27,29,26,35,70,25
```



```
cht=lxy
chd=t:10,20,40,80,90,95,99|
      20,30,40,50,60,70,80|
      -1|
      5,25,45,65,85
```

per **grafici di tipo *lxy*** bisogna fornire **una coppia di insiemi di dati per ogni linea**; il primo insieme di ogni coppia specifica i dati per l'asse x, il secondo per l'asse y

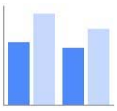
Google Chart API - V

- ***Bar charts***:
  `cht = <bar chart style>`
  In cui **`<line chart style>`** può essere **`bhs`**, **`bhg`**, **`bvs`**, **`bvg`** :

`cht=bhs` (**`h`** = *horizontal*; **`s`** = *stacked*)
`chd=t:50,60,60,65`

`cht=bvs` (**`v`** = *vertical*; **`s`** = *stacked*)
`chd=t:10,50,60,80`

`cht=bhg` (**`h`** = *horizontal*; **`g`** = *grouped*)
`chd=t:50,65,60,70`
`chco=4D89D9,C6D9FD`

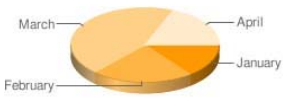


`cht=bvg` (**`v`** = *vertical*; **`g`** = *grouped*)
`chd=t:50,65,60,70`
`chco=4D89D9,C6D9FD`

Google Chart API - VI

- ***Pie charts***:
  `cht = <pie chart style>`
  In cui **`<pie chart style>`** può essere **`p`**, **`p3`**, **`pc`** :

`cht=p`
`...`



`cht=p3`
`...`

`cht=pc` (**`c`** = *concentric*)
NB: bisogna fornire due o più insiemi di dati...

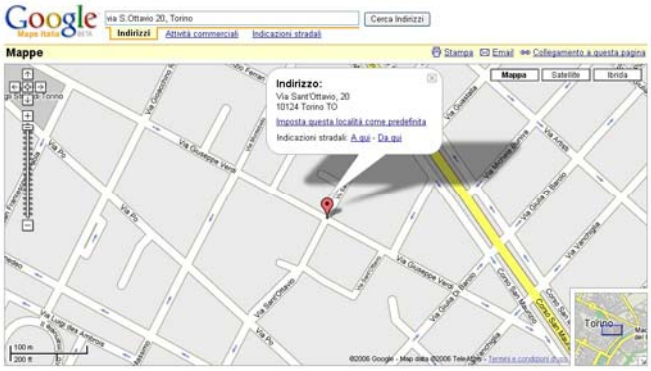
Proposta di esercizio

1. Scrivete una pagina php che **legge i dati dal DB** e disegna un grafico con Google Chart API.

Google Maps API - I

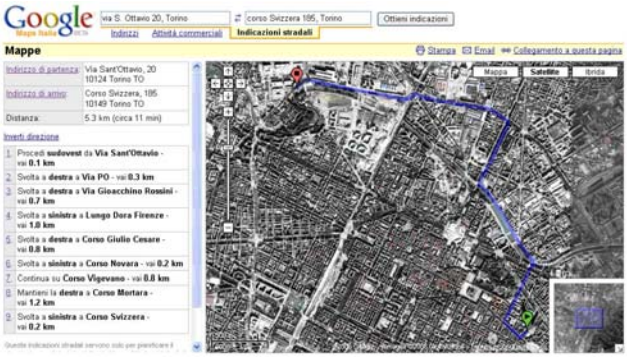
Google Maps API

- Google Maps consente di visualizzare la **mappa** di un luogo prescelto e richiedere **informazioni** di vario tipo, per es. dove si trova un'azienda o una località, insieme ai dati di contatto e alle **indicazioni stradali** per raggiungerli



Google Maps API - II

- L'utente può effettuare sulla mappa varie **operazioni**, per es. il trascinamento (per vedere zone adiacenti senza ricaricare la nuova area), l'ingrandimento o la riduzione
- Può anche scegliere se visualizzare una **semplice mappa** con le sole indicazioni stradali, o una **carta satellitare**



Google Maps API - III

- Le Google Maps sono molto utilizzate anche dagli sviluppatori web grazie alla possibilità di **implementare le mappe anche all'interno del proprio sito** → attraverso **Open API** è possibile invocare le funzionalità delle mappe dalla propria pagina web
- Le API di Google Maps sono state utilizzate in molti **settori**: di trasporti alle previsioni meteorologiche ed ambientali, dall'enogastronomia all'informazione turistica, dallo sport e tempo libero alla ricerca statistica, dal giornalismo alla musica, al cinema, ai videogames...

http://code.google.com/intl/it-IT/apis/maps/

Developer's Guide:
https://developers.google.com/maps/documentation/javascript/tutorial?hl=it-IT

Google Maps API - IV

Le API che utilizzeremo sono scritte in **Javascript**

Per vedere qualche esempio di utilizzo delle API di Google Maps abbiamo bisogno di alcuni concetti importanti:

1. Il formato **JSON**
2. Le **classi principali** di GMaps API (*Map, LatLng, Marker, InfoWindow*)
3. I **listener** (gestori di eventi)
4. La **geocodifica** (*geocoding*)
5. I "**percorsi**" (*directions*)

JSON - I

JSON (JavaScript Object Notation) - *www.json.org*

- è un formato testuale per lo **scambio di dati** in applicazioni client-server, basato su Javascript (anche se è indipendente dal linguaggio di programmazione)
- è basato su due strutture:
  - insiemi di coppie nome-valore (**oggetti** o array associativi)
  - elenchi ordinati di valori (**array**)

  in particolare:
  - un **oggetto** è una serie non ordinata di coppie *nome:valore*, racchiuse tra graffe e separate da virgole
  - un **array** è una serie ordinata di valori racchiusi tra quadre e separati da virgole
  - un **valore** può appartenere ad uno qualunque dei tipi supportati (stringa, numero, boolean, array, oggetto, null)

JSON - II

Utilizzeremo JSON nelle **API di Google Maps**, per esempio: per creare una mappa:

```
var map = new google.maps.Map(
          document.getElementById("map"), myOptions);
```

creiamo un **oggetto JSON** (*myOptions*) che ne definisce alcune proprietà:

```
var point = new google.maps.LatLng(45.06...,7.68...);
var myOptions = {
  zoom: 14,
  center: point,
  mapTypeId: google.maps.MapTypeId.ROADMAP
};
```


JSON - III

Inoltre...

- è diventato una *built-in feature* di Javascript da dic. 2009
- è utilizzato nel protocollo **GoogleData**, nel quale (se utilizzato direttamente = senza librerie), i dati vengono scambiati come *data feed* (liste strutturate), il cui formato può essere:
  - *AtomPub* (*atompub.org*): i dati scambiati sono collezioni di elementi Atom
  - JSON: i dati scambiati sono oggetti JSON

JSON - IV

- è usato in **AJAX** (come alternativa a XML/XSLT), per es:

```
var obj;
var xhr = new XMLHttpRequest();
xhr.open("GET", url, true);
xhr.onreadystatechange = function () {
  if (xhr.readyState == 4) {
    if (xhr.status == 200)
      obj = JSON.parse(xhr.responseText);
      else alert("Errore!");
  }
  xhr = null;
};
xhr.send(null);
```

NB valido solo per browser con supporto nativo a JSON (non per le versioni di Internet Explorer precedenti la 8)

Google Maps API: classi - I

Classi principali di GMaps API:

- **google.maps.Map**: un'istanza di *Map* rappresenta una singola mappa all'interno di una pagina web

```
var map = new google.maps.Map
     (document.getElementById("map"), myOptions);
```

  Il costruttore ha 2 **parametri**:

  – un **elemento HTML** che conterrà la mappa (generalmente un **contenitore** come `<div...></div>`):

```
document.getElementById("map")
```

  – le **proprietà della mappa**, contenute in un **oggetto JSON**:

```
var myOptions = {
 zoom: 14, // livello di zoom
 center: latlng, // oggetto LatLng [vedi prox slide]
 mapTypeId: google.maps.MapTypeId.ROADMAP
 // tipo di mappa (stradale, satellitare, ...)
};
```

Google Maps API: classi - II

- **google.maps.LatLng**: un'istanza di ***LatLng*** rappresenta un **punto "geografico"**, cioè definito in termini di coordinate geografiche (longitudine e latitudine); il costruttore ha 2 **parametri**: latitudine e longitudine

```
var latlng = new google.maps.LatLng(45.06...,7.68...);
```

NOTA: per conoscere le **coordinate geografiche di una città** potete usare un servizio on-line (per es: http://www.backups.nl/geocoding/)

Google Maps API: classi - III

- **google.maps.Marker** : un'istanza di *Marker* rappresenta un marker che può essere aggiunto alla mappa

  Il costruttore ha un **parametro**: le **proprietà del marker**, contenute in un **oggetto JSON**:

```
var marker = new google.maps.Marker({
 position: myLatLng, // oggetto LatLng (punto a cui il
                     // marker è ancorato)- obbligatorio
 map: map, // oggetto Map
 title: "Hello World!" // stringa (tooltip)
});
```

Icona di default:  si può impostare un'icona personalizzata aggiungendo all'oggetto JSON parametro del costruttore:

```
icon: "myIcon.png" // file con l'imm. personalizzata
```

oppure utilizzando il metodo *setIcon(<nomefile>)*

NB: Specificando un oggetto *Map* **il marker viene aggiunto su quella mappa**; alternativamente lo si può fare con il metodo *setMap()*; *setMap(null)* rimuove il marker dalla mappa

Google Maps API: classi - IV

- **google.maps.InfoWindow**:
  un'istanza di *InfoWindow* rappresenta una *info window* che può essere aggiunta sia direttamente alla mappa, sia ad un marker

  Il costruttore ha un **parametro**: le **proprietà della infoWindow**, contenute in un **oggetto JSON**:

```
var infowindow = new google.maps.InfoWindow({
  content: contentString // HTML visualizz. nella finestra
  position: myLatLng // punto di ancoraggio (se non
                     // ancorata a un marker)
});
```

  Per **rendere visibile la infoWindow** si utilizza il metodo *open()*, passandogli la mappa e il marker (opzionale) a cui ancorarla (se il marker non viene passato, la infoWindow si apre sul punto indicato dalla proprietà *position*), per es:

```
infowindow.open(map, marker);
```


Google Maps API: listener

Generalmente una infoWindow si apre al **click dell'utente** per es su un marker...

Alcuni oggetti delle GMaps API sono costruiti in modo tale da rispondere ad **eventi** come il click del mouse; per es. *google.maps.Marker* risponde ad eventi quali: 'click', 'dblclick', 'mouseover', 'mouseout', ...

Per **gestire tali eventi** occorre **registrare un listener**, utilizzando il metodo ***addListener()*** della classe **google.maps.event**; per es:

```
google.maps.event.addListener(marker, 'click', function(){
  infowindow.open(map,marker);
});
```

Il **metodo *addListener()*** ha 3 parametri:

- un **oggetto** (per es un marker)
- un **evento** da "catturare" (per es il 'click')
- una **funzione** (anonima) da invocare quando l'evento viene "catturato" (per es. mostrare la infoWindow)

Geocoding (Google) - I

Geocoding (*https://developers.google.com/maps/documentation/javascript/geocoding?hl=it-IT*)

- La **geocodifica** è il processo di conversione di un **indirizzo** (es: "1600 Amphitheatre Parkway, Mountain View, CA") in **coordinate geografiche** (es: latitude 37.423021, longitude 122.083739), che identificano un punto sulla mappa

Geocoding (Google) - II

- Il **servizio di geocodifica** di Google prevede un'**interazione asincrona** (perché è necessaria una chiamata ad un server esterno) ⇒ per utilizzare le API del servizio è necessario fornire un **metodo** particolare chiamato ***callback***, che verrà eseguito quanto il server avrà elaborato e inviato la risposta:
  - il metodo callback conterrà il codice per elaborare i risultati
  - il principio è lo stesso visto in AJAX (funzione assegnata alla proprietà *onreadystatechange* dell'istanza di *XMLHttpRequest*)

```
xhrObj.onreadystatechange = updatePage;
function updatePage() {
  if (xhrObj.readyState == 4) {
    var risp = xhrObj.responseText;
    document.getElementById("cap").value = risp;
  }
}
```

Geocoding (Google) - III

- Per utilizzare il servizio di geocodifica di Google tramite API si utilizza la **classe** *google.maps.**Geocoder*** ed in particolare il **metodo *geocode()***, che invia una **richiesta** al servizio di geocoding passandogli:
  1. un oggetto di tipo ***GeocodeRequest*** (che contiene l'input)
  2. un **metodo callback** (che verrà eseguito alla recezione della risposta)

Geocoding (Google) - IV

1. L'oggetto ***GeocodeRequest*** è un oggetto JSON (insieme di coppie nome-valore):

```
{
 address*: string,
 latLng*: LatLng,
 bounds: LatLngBounds,
 region: string
}
```

indirizzo da geocodificare

punto (oggetto di tipo *LatLng*) per cui si vuol conoscere il più vicino indirizzo

opzionali (per casi più complessi...)

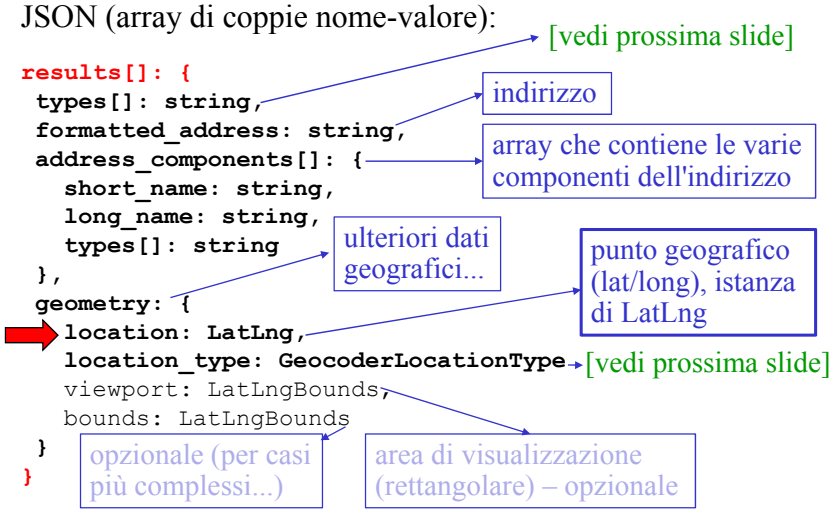
***** Se viene passato un indirizzo, il servizio effettua una geocodifica vera e propria (*geocoding*), restituendo un punto geografico; se viene passato un punto geografico (oggetto di tipo *LatLng*), il servizio effettua una *geocodifica inversa* (*reverse geocode*), restituendo l'indirizzo più vicino

Geocoding (Google) - V

2. Il **metodo callback** ha **2 parametri**:
   a. il **risultato** e
   b. un **codice di stato**

Geocoding (Google) - VI

a. **Risultato** = oggetto di tipo ***GeocoderResults*** = oggetto JSON (array di coppie nome-valore):



Geocoding (Google) - VII

- **`types[]: string`** = array che contiene una lista di stringhe che indicano il tipo del risultato; per es: "locality" e "political" (se il risultato è "Chicago", che è una località, ma anche un'entità politica)

  Tipi supportati: *street_address, route, intersection, political, country, administrative_area_level_1, administrative_area_level_2, administrative_area_level_3, colloquial_are, locality, sublocality, neighborhood, premise, subpremise, postal_code, natural_feature, airport, park*

- **`location_type: GeocoderLocationType`** = il ***GeocoderLocationType*** può essere:
  - ROOFTOP = geocodice preciso
  - RANGE_INTERPOLATED = approssimazione (interpolazione tra due punti precisi)
  - GEOMETRIC_CENTER = centro geometrico (per es nel caso di una regione, rappresentata come un poligono)
  - APPROXIMATE = indica che il risultato è approssimativo

Geocoding (Google) - VIII

b. Codice di stato (*google.maps.GeocoderStatus*) può essere:

- OK = la geocodifica ha avuto successo
- ZERO_RESULTS = la geocodifica ha avuto successo, ma non ha prodotto risultati
- OVER_QUERY_LIMIT = la quota di richieste è stata superata
- REQUEST_DENIED = la richiesta è stata rifiutata dal servizio
- INVALID_REQUEST = la richiesta (l'indirizzo o il punto geografico indicato) non è valida

Geocoding (Google): esempi - I

Esempio

chiediamo al servizio di Google di **geocodificare un indirizzo** e poi mettiamo un **marker** sul corrispondente punto geografico
→ prova_geocoding_1.html

1. definiamo una funzione *initialize* che crea un'istanza di *Geocoder* e una mappa (un'istanza di *Map*) e la invochiamo subito, al caricamento della pagina:

```
<BODY onload="initialize()">
```

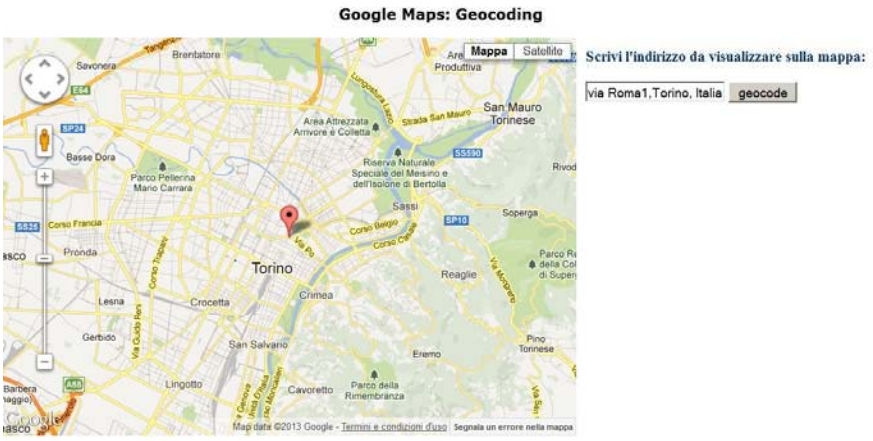

Geocoding (Google): esempi - II

2. definiamo una funzione *codeAddress* che
   - legge l'indirizzo scritto dall'utente
   - invoca il geocoder, passandogli tale indirizzo e chiedendogli di geocodificarlo (cioè di identificare il corrispondente punto geografico)
   - mette un marker sul punto geografico restituito dal geocoder

   e la invochiamo quando l'utente clicca sul pulsante "geocode":

```
<input id="address" type="textbox"
                    value="Torino, Italia">
<input type="button" value="geocode"
                     onclick="codeAddress()">
```

Geocoding (Google): esempi - III



Geocoding (Google): esempi - IV

```
var geocoder;
var map;
function initialize() {
  geocoder = new google.maps.Geocoder();
  var latlng = new google.maps.LatLng(45.06...,7.68...);
  var myOptions = {
    zoom: 12,
    center: latlng,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  }
  map = new google.maps.Map(document.getElementById("map"),
                                                myOptions);
}
```

creiamo un'istanza di *Geocoder*

1) creiamo un **punto** (`latlng`) con latitudine e longitudine di Torino
2) creiamo un oggetto JSON (`myOptions`) che contiene alcune **proprietà della mappa**: il livello di **zoom**, il **punto** su cui centrare la mappa, il **tipo** di mappa
3) creiamo una nuova **mappa** (un'istanza di *Map*) con le proprietà definite (2° par) e la inserimo nel div *map* (1° par):

```
<div id="map"></div>
```

Geocoding (Google): esempi - V

```
function codeAddress() {
  var address = document.getElementById("address").value;
```

leggiamo l'indirizzo scritto dall'utente

```
  geocoder.geocode({'address': address},
   function(results, status) {
     ...
     [vedi prossima slide]
     ...
  });
}
```

chiediamo al **geocoder** di geocodificare l'indirizzo, invocando il metodo *geocode*, che ha 2 parametri:
- un oggetto (JSON) di tipo *GeocodeRequest* (che contiene **l'indirizzo da geocodificare**)
- un **metodo callback** (cioè una funzione che verrà eseguita alla recezione della risposta) che ha 2 parametri: il risultato (`result`) e un codice di stato (`status`)

Geocoding (Google): esempi - VI

Vediamo cosa fa il **metodo callback**:

```
  function(results, status) {
  if (status == google.maps.GeocoderStatus.OK){
     map.setCenter(results[0].geometry.location);
     var marker = new google.maps.Marker({
       map: map,
       position: results[0].geometry.location
     });
   } else {
     alert("Geocode failed: " + status);
   }
}
```

se lo stato (2° par del metodo callback) è OK
- **centriamo la mappa** sul punto restituito dal geocoder nel risultato (`results[0].geometry.location`)
- **creiamo un marker**, passandogli come parametro le sue proprietà (oggetto JSON): la mappa su cui visualizzarlo e la posizione (il punto restituito dal geocoder nel risultato)

altrimenti visualizzo un messaggio di errore (*alert*)

Directions (Google) - I

Directions (*https://developers.google.com/maps/documentation/javascript/directions?hl=it-IT*)

- Il **calcolo del percorso** tra due punti sulla mappa, in base al mezzo di trasporto, è offerto dal servizio *Directions* di Google
- Per utilizzare il servizio di *Directions* tramite API si utilizza la **classe** *google.maps.**DirectionsService*** e per visualizzare i risultati, la **classe** *google.maps.**DirectionsRenderer***
- Il servizio di *Directions* di Google prevede un'**interazione asincrona** (perché è necessaria una chiamata ad un server esterno) ⇒ per utilizzare le API del servizio è necessario fornire un **metodo *callback***, che verrà eseguito quanto il server avrà elaborato e inviato la risposta e che conterrà il codice per elaborare i risultati

Directions (Google) - II

- il metodo da invocare su un'istanza di *DirectionsService* è il **metodo *route()***, che invia una **richiesta** al servizio di geocoding passandogli:
  1. un oggetto di tipo ***DirectionsRequest*** (che contiene l'input)
  2. un **metodo callback** (che verrà eseguito alla recezione della risposta)

Directions (Google) - III

1. L'oggetto ***DirectionsRequest*** è un oggetto JSON (insieme di coppie nome-valore):

```
{
  origin: LatLng | String,
  destination: LatLng | String,
  travelMode: TravelMode,
  transitOptions: TransitOptions,
  unitSystem: UnitSystem,
  waypoints[]: DirectionsWaypoint,
  optimizeWaypoints: Boolean,
  provideRouteAlternatives: Boolean,
  avoidHighways: Boolean,
  avoidTolls: Boolean,
  region: String
}
```

punto di partenza: può essere una stringa o un punto geografico (istanza di LatNng)

punto di arrivo: IDEM

mezzo di trasporto [vedi prox slide]

opzionali (per casi più complessi...)

Directions (Google) - IV

`travelMode: TravelMode` = oggetto che può assumere i seguenti valori:

- google.maps.TravelMode.DRIVING (default)
- google.maps.TravelMode.BICYCLING
- google.maps.TravelMode.TRANSIT (= via public transit routes)
- google.maps.TravelMode.WALKING

2. Il **metodo callback** ha **2 parametri**:
   a. il **risultato** e
   b. un **codice di stato**

Directions (Google) - V

a. **Risultato** = oggetto di tipo ***DirectionsResult*** che contiene il percorso; questo oggetto può essere dato in input ad un'istanza di ***DirectionsRenderer***, il quale lo visualizzerà sulla mappa; in particolare occorre:
   1. creare un'istanza di *DirectionsRenderer*
   2. invocare il metodo *setMap()* (per indicare su quale mappa si vuole visualizzare il percorso)
   3. invocare il metodo *setDirections()*, passandogli il risultato (istanza di *DirectionsResult*)

Directions (Google): esempi - I

Esempio 1

chiediamo al servizio di Google di calcolare il percorso tra Torino e una città scelta dall'utente → prova_directions_1.html

1. creiamo un'istanza di *DirectionsService* (*directionsService*)
2. definiamo una funzione *initialize* che crea un'istanza di *DirectionsRenderer* (*directionsDisplay*) e una mappa (un'istanza di *Map*), a cui agganciamo il *directionsDisplay*, e la invochiamo subito, al caricamento della pagina:

```
<BODY onload="initialize()">
```

Directions (Google): esempi - II

3. definiamo una funzione *calcRoute* che
   - legge la città di arrivo, selezionata dall'utente
   - crea un oggettto (JSON) *request*, che indica la città di partenza, quella di arrivo e il modo di viaggio
   - chiede il percorso al *directionsService*, invocando il metodo *route* e passandogli tale request
   - visualizza il percorso sulla mappa

   e la invochiamo quando l'utente seleziona una città di arrivo:

```
<select id="end" onchange="calcRoute();">
```


Directions (Google): esempi - III

Directions (Google): esempi - IV

```
var directionDisplay;
var directionsService = new google.maps.DirectionsService();
var map;
function initialize() {
  directionsDisplay = new google.maps.DirectionsRenderer();
  var torino = new google.maps.LatLng(45.06...,7.68...);
  var myOptions = {
    zoom: 10,
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    center: torino
  }
  map = new google.maps.Map(document.getElementById("map"),
                                                  myOptions);
  directionsDisplay.setMap(map);
}
```

creiamo un'istanza di *DirectionsService*

creiamo un'istanza di *DirectionsRenderer*

creiamo una mappa…

a cui "agganciamo" il *directionsDisplay*

Directions (Google): esempi - V

```
function calcRoute() {
  var start = "Torino";
  var end = document.getElementById("end").value;
  var request = {
    origin:start,
    destination:end,
    travelMode: google.maps.DirectionsTravelMode.DRIVING
  };
  directionsService.route(request, function(response,
   status) {
    if (status == google.maps.DirectionsStatus.OK) {
      directionsDisplay.setDirections(response);
    }
  });
}
```

leggiamo la città di arrivo selezionata dall'utente

creiamo un oggettto (JSON) *request*, che indica la città di partenza, quella di arrivo e il modo di viaggio

Directions (Google): esempi - VI

chiediamo al ***directionsService*** di calcolare il percorso, invocando il metodo *route*, che ha 2 parametri:
- un oggetto (JSON) di tipo *DirectionsRequest* (che contiene la città di partenza, quella di arrivo e il modo di viaggio)
- un **metodo callback** (cioè una funzione che verrà eseguita alla recezione della risposta) che ha 2 parametri: il risultato (`result`) e un codice di stato (`status`)

Vediamo cosa fa il **metodo callback**:
se lo stato (2° par) è OK
- chiediamo al *directionsDisplay* di visualizzare il percorso sulla mappa, invocando il metodo *setDirections* e passandogli in risultato fornitoci dal *directionsService* (`response`), cioè il percorso:

```
directionsDisplay.setDirections(response);
```


Directions (Google): esempi - VII

Esempio 2

costruiamo un'applicazione web che mostra gli eventi che si svolgono a Torino e visualizza il percorso per raggiungerli

Directions (Google): esempi - VIII

→ prova_geo_dir_2.php + DB "eventi "
NB: attenzione agli indirizzi nel DB!!! Controllateli prima con prova_geocoding_1.html

1. definiamo una **funzione** (*initialize*) che:
   - crea un punto geografico corrispondente a Torino (*torino*)
   - crea una mappa centrata su Torino (*map*) e la mette nel div "map",
   - crea un *Listener* che, al click dell'utente sulla mappa, aggiunge un marker sul punto geografico corrispondente (*puntoUtente*)
   - crea un'istanza di *DirectionsService* (*directionsService*)
   - crea un'istanza di *DirectionsRenderer* (*directionsDisplay*)

   e la invochiamo subito, al caricamento della pagina:

```
<BODY onload="initialize()">
```


Directions (Google): esempi - IX

2. definiamo una **funzione** (*showEvent*) che prende in input i dati relativi ad un evento (nome, indirizzo, immagine, descrizione), trova il punto geografico corrispondente all'indirizzo, lo usa per centrare la mappa (*map*) e per inserire un marker "sensibile" (cioè che al click fa comparire una infoWindow) sulla mappa; la invochiamo quando l'utente clicca sul nome dell'evento:

```
<A HREF="" onClick="showEvent(...); ...">
```

3. definiamo una **funzione** (*calcRoute*) che prende in input due punti geografici e mostra il percorso sulla mappa legata all'oggetto *directionsDisplay* (cioè *map*); la invochiamo quando l'utente clicca sul "percorso" accanto al nome dell'evento:

```
<A HREF="" onClick="calcRoute(...); ...">
```

Directions (Google): esempi - X

4. **leggiamo la lista degli eventi dal DB** (script PHP); per ogni evento, costruiamo:
   - un link che, al click dell'utente, invoca la funzione *showEvent*
   - un link che, al click dell'utente, invoca la funzione *calcRoute*

Directions (Google): esempi - XI

```
function initialize() {
 // creiamo un punto geografico corrispondente a Torino
 var torino = new google.maps.LatLng(45.06...,7.68...);
 // creiamo una mappa centrata su Torino nel div "map"
 var myOptions = {
   zoom: 12,
   center: torino,
   mapTypeId: google.maps.MapTypeId.ROADMAP
 }
 map = new google.maps.Map(document.getElementById("map"),
                                                 myOptions);
// rendiamo la mappa "sensibile" al click dell'utente:
// quando l'utente clicca, mettiamo un marker
 google.maps.event.addListener(map, 'click', function(event){
   var marker = new google.maps.Marker({
     position: event.latLng, //punto cliccato dall'utente
     map: map
   });
   marker.setIcon('img/mm_20_green.png');
   puntoUtente = event.latLng;
 });
```

Directions (Google): esempi - XII

```
 ...
 // creiamo un'istanza di DirecionsService
 // (per chiedere il percorso)
 directionsService = new google.maps.DirectionsService();

 // creiamo un'istanza di DirectionsRenderer
 // e la "leghiamo" alla mappa (per visualizzare il percorso)
 directionsDisplay = new google.maps.DirectionsRenderer();
 directionsDisplay.setMap(map);
}
```


Directions (Google): esempi - XIII

```
// input = dati relativi ad un evento (nome, ind, imm, descr)
function showEvent(name, add, img, des) {
 // creiamo un'istanza di Geocoder
 var geocoder = new google.maps.Geocoder();
 // invochiamo il metodo geocode, passandogli l'indirizzo e
 // definendo il metodo callback
 geocoder.geocode({'address': add},
  function(results, status) {
   if (status == google.maps.GeocoderStatus.OK) {
    // centriamo la mappa sul punto corrisp. all'indirizzo
    map.setCenter(results[0].geometry.location);
    // creiamo un marker su quel punto
    var marker = new google.maps.Marker({
     map: map,
     position: results[0].geometry.location
    });
```

Directions (Google): esempi - XIV

```
    // creiamo il contenuto (HTML) per una infoWindow
    var info = "<p><b>" + name + "</b><br>" + add + "<br>
        <img src='img/" + img + "'><br>" + des + "</p>";
    // creiamo un'istanza di infoWindow
    var infowindow = new google.maps.InfoWindow({
     content: info
    });
    // "leghiamo" la infoWindow al marker (con un listener
    // che, al click sul marker, apre la infoWindow)
    google.maps.event.addListener(marker, 'click',
     function(){
      infowindow.open(map, marker);
     });
   } else {
      alert("Geocode was not successful for the following
            reason: " + status);
   }
  });
}
```


Directions (Google): esempi - XV

```
// input = 2 punti geografici (partenza e arrivo)
function calcRoute(start, end) {
 if (start == null) {
  alert("Devi prima cliccare sulla mappa per indicare il
         punto da cui vuoi partire!");
 }
// creiamo una richiesta (ogg JSON) con partenza, arrivo
// e modo di viaggio
 var request = {
  origin:start,
  destination:end,
  travelMode: google.maps.DirectionsTravelMode.DRIVING
 };
```

Directions (Google): esempi - XVI

```
// invochiamo il metodo route sull'istanza di
// DirectionsService, passandogli la richiesta e
// definendo il metodo callback
 directionsService.route(request,
  function(response, status) {
   if (status == google.maps.DirectionsStatus.OK) {
    // chiediamo all'istanza si DirectionsRenderer di
    // mostrare il percorso restituito da
    // directionsService
    directionsDisplay.setDirections(response);
   }
  });
}
```


Directions (Google): esempi - XVII

```
<?php
 include("dbConn.php");
 $conn = mysql_connect($host, $user, $pwd) or die ...
 mysql_select_db($db) or die ...
 // leggiamo la lista degli eventi dal DB
 $sql1 = "SELECT * FROM eventi_torino";
 $ris1 = mysql_query($sql1) or die ...
 // per ogni evento...
 while ($rec1 = mysql_fetch_array($ris1)) {
   $n = $rec1["nome"];
   $a = $rec1["indirizzo"];
   $i = $rec1["img"];
   $d = $rec1["descr"];
// costruiamo un link che, al click dell'utente, invoca
// la funzione showEvent + un link che, al click
// dell'utente, invoca la funzione calcRoute
?>
```

Directions (Google): esempi - XVIII

```
<P>
 <A HREF="" onClick="showEvent('<?php echo $n; ?>',
                               '<?php echo $a; ?>',
                               '<?php echo $i; ?>',
                               '<?php echo $d; ?>');
                     return false;">
  <?php echo $n; ?>
 </A>

 <A HREF="" onClick="calcRoute(puntoUtente,
                               '<?php echo $a; ?>');
                     return false;">
  percorso
 </A>
</P>
<?php
 }
?>
```