

# Display locations from a database on a map using Google Maps JavaScript API and PHP

📅 17. December, 2011. / 📁 Development / 💬 2 Comments / 👍 Like / 🗨️ / ⭐ Favourite

In this relatively short blog post we'll use PHP/MySQL and [Google Maps Javascript API](#) to display locations from a MySQL database on a map.

We'll make a simple "City guide to Zagreb", meaning we'll select interesting locations from the database, display them on a map, show some info about them and show directions from user's current position to a location of his/her choice.



## What you'll need

- PHP/MySQL Development Environment
- Some understanding of JavaScript
- Some free time

As you can see it's really easy to get started. So, let's do it!

## The Database

First, let's create a table for our data:

```
CREATE TABLE `locations` (  
  `id` INT(10) NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(150) NOT NULL,  
  `address` VARCHAR(255) NOT NULL,  
  `lat` FLOAT(10,6) NOT NULL,  
  `lon` FLOAT(10,6) NOT NULL,  
  `description` TEXT NOT NULL,  
  PRIMARY KEY (`id`)  
)  
COLLATE='utf8_general_ci'  
ENGINE=InnoDB
```

Then, use phpMyAdmin or your favorite MySQL client to enter some data about locations. Or, if you don't feel like it, here's my data:

```
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (1, 'Archaeological Museum', 'Nikola &S
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (2, 'Modern Gallery', 'Andrije Hebranga
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (3, 'Technical Museum', 'Savska cesta 1
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (4, 'St. Mark\'s Church', 'Saint Mark\'
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (5, 'Zagreb Cathedral', 'Kaptol 31, 100
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (6, 'The Grounded Sun', 'Bogovićeva, 10
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (7, 'Croatian National Theatre', 'Marsh
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (8, 'Museum of Contemporary Art', 'Dubr
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (9, 'Maksimir Park', 'Maksimir Park, 10
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (10, 'Mimara Museum', 'Trg Franklina De
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (11, 'Strossmayer Gallery of Old Master
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (12, 'Vatroslav Lisinski Concert Hall',
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (13, 'Lotr&scaron;čak Tower', 'Dverce,
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (14, 'Jarun Lake', 'Jarun Lake, 10000,
INSERT INTO `locations` (`id`, `name`, `address`, `lat`, `lon`, `description`) VALUES (15, 'Botanical Garden', 'Trg Marka Mar
```

NOTE that some locations have empty coordinates, and have only their addresses entered. This is deliberate so we can write code to handle those situations as well. If we encounter a location without coordinates entered into the database, we'll try to Geocode it's address into coordinates.

## Create a Simple map

First, let's create a simple map, just to verify how things should work in a map.

Create a new file named **index.html** which will display our results and do other cool stuff with maps. For now, we'll only create a basic map:

```
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8" />
<title>Google Maps Example</title>
<style type="text/css">
body { font: normal 14px Verdana; }
h1 { font-size: 24px; }
h2 { font-size: 18px; }
#sidebar { float: right; width: 30%; }
#main { padding-right: 15px; }
.infoWindow { width: 220px; }
</style>
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript">
//

var map;

// Ban Jelačić Square - Center of Zagreb, Croatia
var center = new google.maps.LatLng(45.812897, 15.97706);

function init() {

var mapOptions = {
zoom: 13,
center: center,
mapTypeId: google.maps.MapTypeId.ROADMAP
}

map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);

var marker = new google.maps.Marker({
map: map,
position: center,
});
}
//]]&gt;
&lt;/script&gt;
&lt;/head&gt;
&lt;body onload="init();"&gt;

&lt;h1&gt;Places to check out in Zagreb&lt;/h1&gt;

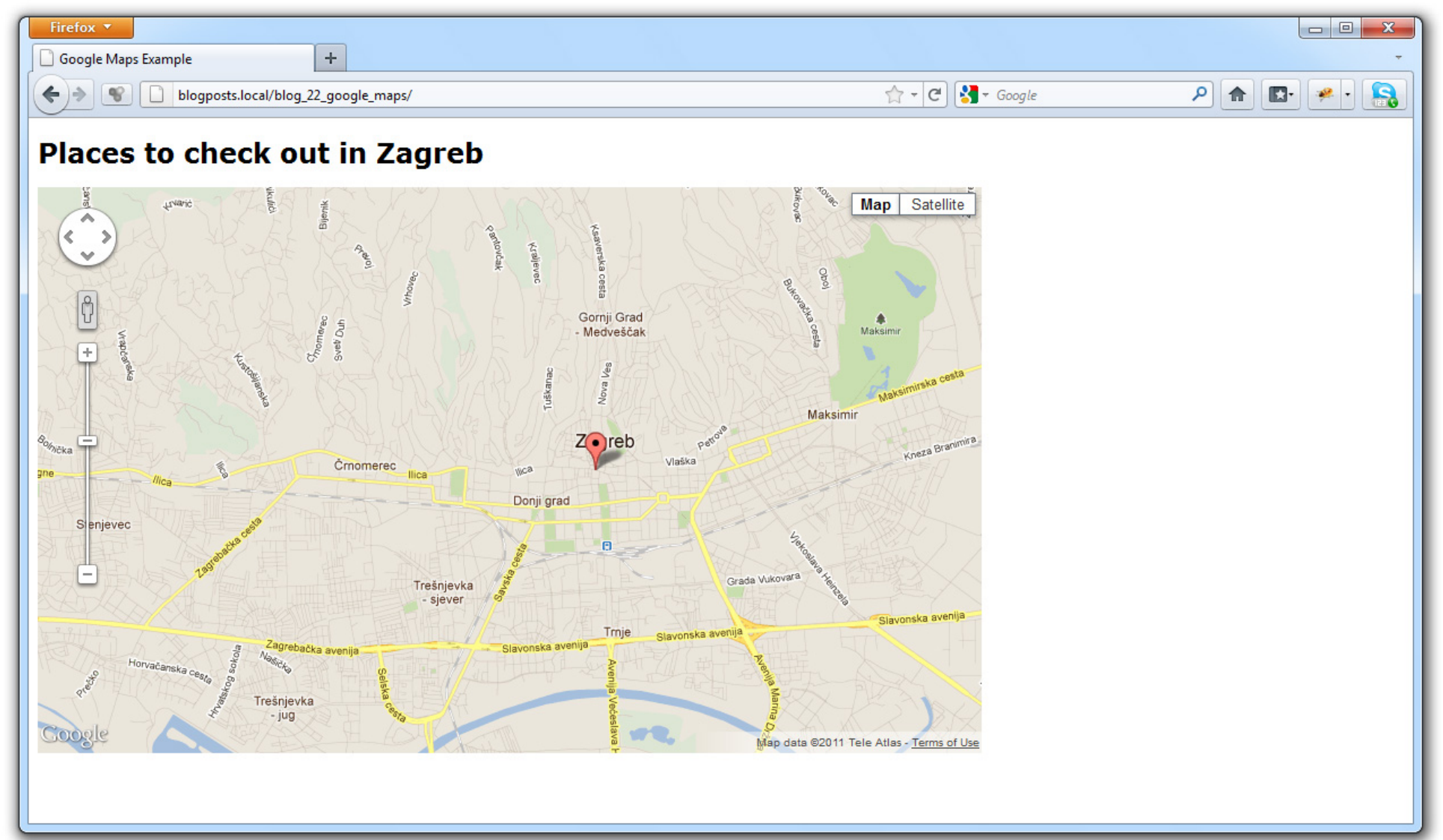
&lt;section id="sidebar"&gt;
&lt;div id="directions_panel"&gt;&lt;/div&gt;
&lt;/section&gt;

&lt;section id="main"&gt;
&lt;div id="map_canvas" style="width: 70%; height: 500px;"&gt;&lt;/div&gt;
&lt;/section&gt;

&lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="921 943 952 966" data-label="Image"><img alt="A small, empty square box, likely a placeholder for a logo or icon."/></div><div data-bbox="32 978 968 988" data-label="Page-Footer"><div>https://isitokto.com/post/display-locations-from-a-database-on-a-map-using-google-maps-javascript-api-and-php</div><div>2/13</div></div>
```

In body we created a div to hold the map. Because the map takes dimensions from the container div it needs to have dimensions. We use **onload** function to run the init function when our page loads. In the init function we create a new map that uses options we specified. Then we create a map marker and display it.

Save the file and run it in the browser to see what we got.



OK, we've scratched the surface, now we need some data to display on the map.

## Getting data from the database

Let's fetch data from the database to display on our map. We'll use Ajax request to get location data. From PHP script we'll echo data in JSON format and then process it via JavaScript. I know, it may sound confusing for beginners, but it's really not. Just follow along, things will be more clear.

First, let's create a **config.php** file. Inside we'll configure some variables needed for database access:

```
<?php
$server      = 'localhost';
$username    = 'root';
$password    = 'YOUR_PASSWORD';
$database    = 'YOUR_DATABASE';

$dsn         = "mysql:host=$server;dbname=$database";
```

Create our file that will return the actual data. I named it **get\_locations.php**. This file should contain the following code:



```
<?php

require 'config.php';

try {

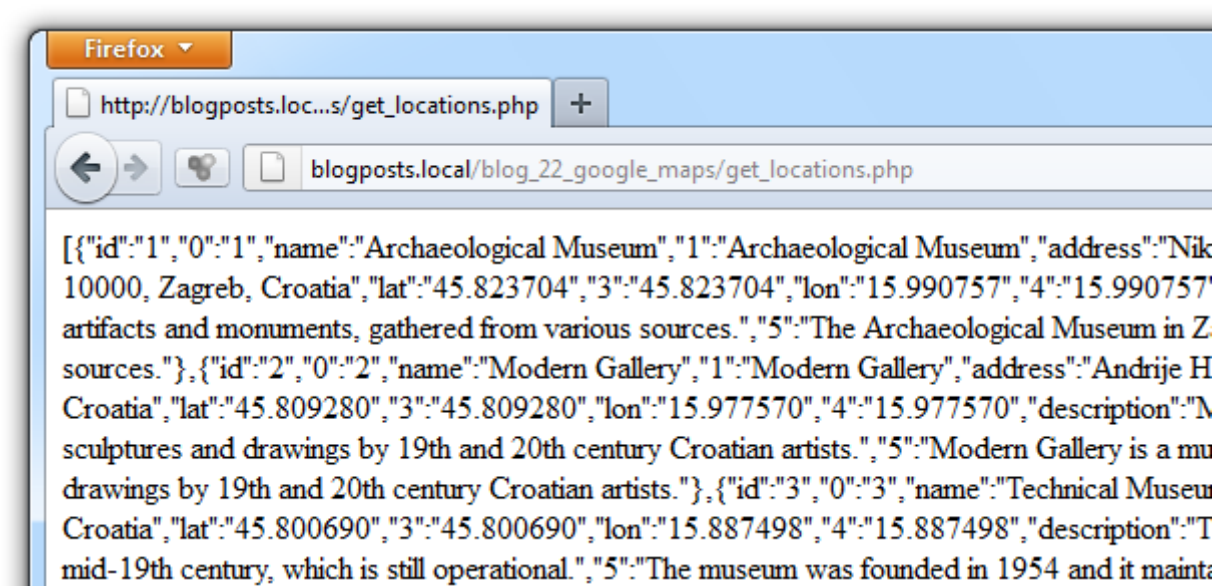
$db = new PDO($dsn, $username, $password);
$db->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

$sth = $db->query("SELECT * FROM locations");
$locations = $sth->fetchAll();

echo json_encode( $locations );

} catch (Exception $e) {
echo $e->getMessage();
}
```

You can run the `get_locations.php` file to verify that it correctly returns our locations as a JSON string:



We're done with the PHP side of things. Cool, right?

## Displaying locations on the map

First thing we'll need to do is a way to make an Ajax request and a way to process the results that request returns to us. As we already know, our PHP file will return JSON to us.

We won't use an external library, so inside a JavaScript block of our `index.html` file create a new function for making Ajax requests.

```
function makeRequest(url, callback) {
var request;
if (window.XMLHttpRequest) {
request = new XMLHttpRequest(); // IE7+, Firefox, Chrome, Opera, Safari
} else {
request = new ActiveXObject("Microsoft.XMLHTTP"); // IE6, IE5
}
request.onreadystatechange = function() {
if (request.readyState == 4 && request.status == 200) {
callback(request);
}
}
request.open("GET", url, true);
request.send();
}
```

This is a pretty standard function, everyone's using some variation of it, so there's no need to get into details. All you need to know is that we'll provide two parameters for this function: `url` where to make an Ajax request and a callback function which will be called with a data sent back as a parameter.

So, let's use our newly added function inside our `init` function. Let's add the call after we create a map. Also, let's add a `Geocoder` and `Infowindow` objects because we'll need them here.



```
var map;

// Ban Jelačić Square - City Center
var center = new google.maps.LatLng(45.812897, 15.97706);

var geocoder = new google.maps.Geocoder();
var infowindow = new google.maps.InfoWindow();

function init() {

var mapOptions = {
  zoom: 13,
  center: center,
  mapTypeId: google.maps.MapTypeId.ROADMAP
}

map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);

makeRequest('get_locations.php', function(data) {

var data = JSON.parse(data.responseText);

for (var i = 0; i < data.length; i++) {
  displayLocation(data[i]);
}
});
}
```

So, we added a **makeRequest** call. The callback function first uses **JSON.parse** to convert JSON string into array of objects. Each object is basically one row from a database and it represents a location, so inside a loop we call a **displayLocation** function with object as a parameter.

In the **displayLocation** function we'll display that location on the map:

```
function displayLocation(location) {

var content = '<div class="infowindow"><strong>' + location.name + '</strong>'
+ '<br/>' + location.address
+ '<br/>' + location.description + '</div>';

if (parseInt(location.lat) == 0) {
  geocoder.geocode( { 'address': location.address }, function(results, status) {
    if (status == google.maps.GeocoderStatus.OK) {

var marker = new google.maps.Marker({
  map: map,
  position: results[0].geometry.location,
  title: location.name
});

google.maps.event.addListener(marker, 'click', function() {
  infowindow.setContent(content);
  infowindow.open(map, marker);
});
});
} else {
var position = new google.maps.LatLng(parseFloat(location.lat), parseFloat(location.lon));
var marker = new google.maps.Marker({
  map: map,
  position: position,
  title: location.name
});

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

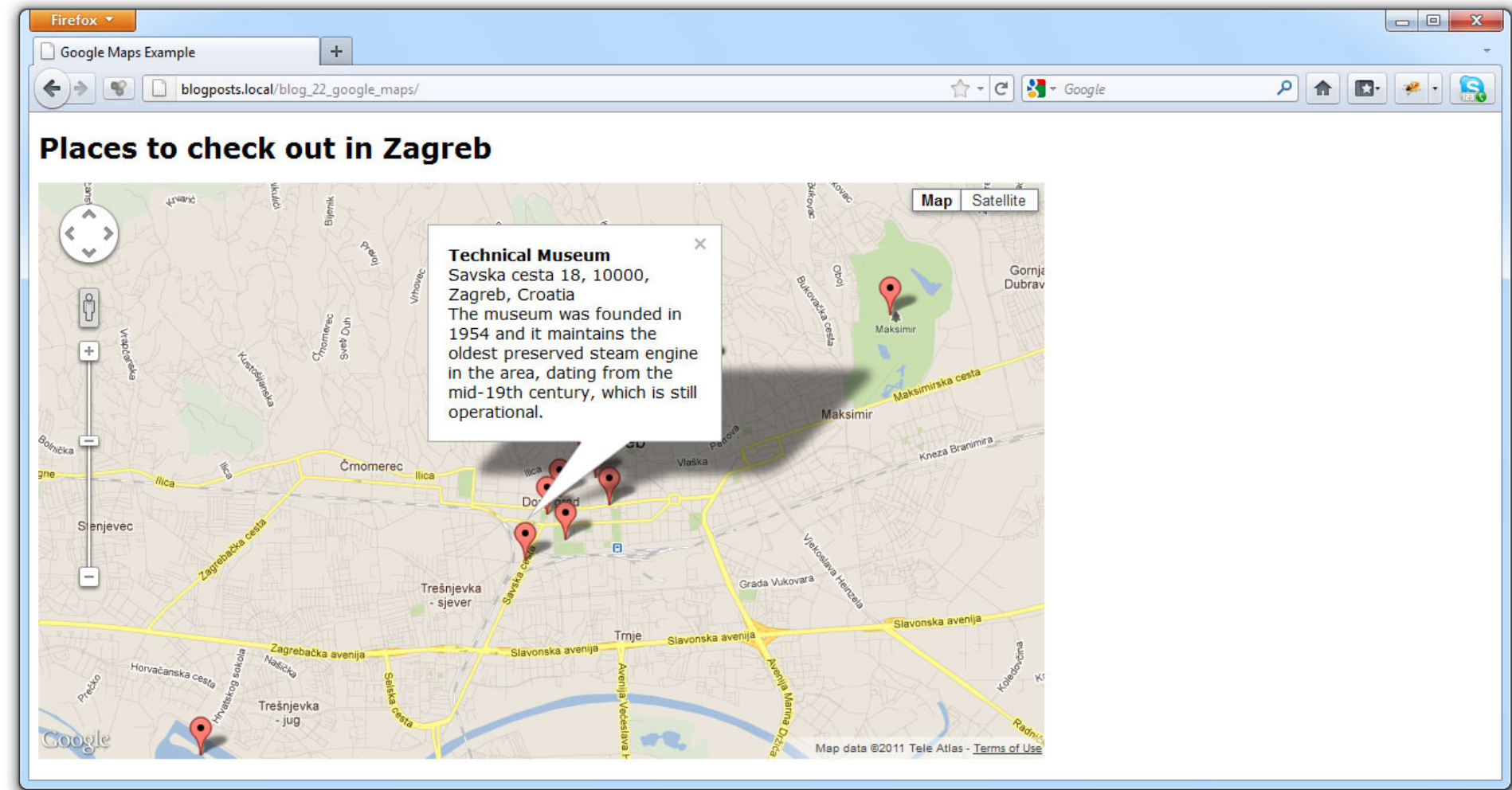
As you can see the function is basically simple. First we create a HTML content that'll go inside a marker, so we have it ready. Then we check if we have latitude and longitude for that location and if we don't we'll use Geocoding to get the latitude and longitude from a location's address. If the geocoding process succeeds we use that data to create a marker.

NOTE: We can't use geocoding service for a lot of results, because of the **limitations**. That is precisely the reason we need to have coordinates stored into the database. For more information please take a look at **Geocoding Strategies** on Google maps api documentation.

If we have latitude and longitude we use that to create a marker. When we create a marker we'll add a event listener to the marker -> When we click on the marker open a InfoWindow containing the information about the location.

Save the file and open index.html via web server to see what we got.





We've successfully added all the locations to the map. Now, let's add the directions to each of the locations.

## Displaying Directions to locations

Inside the body create a form so that user can enter his/her position and choose a location. Also, add a panel in which to display directions.

```
<body onload="init();">

<h1>Places to check out in Zagreb</h1>

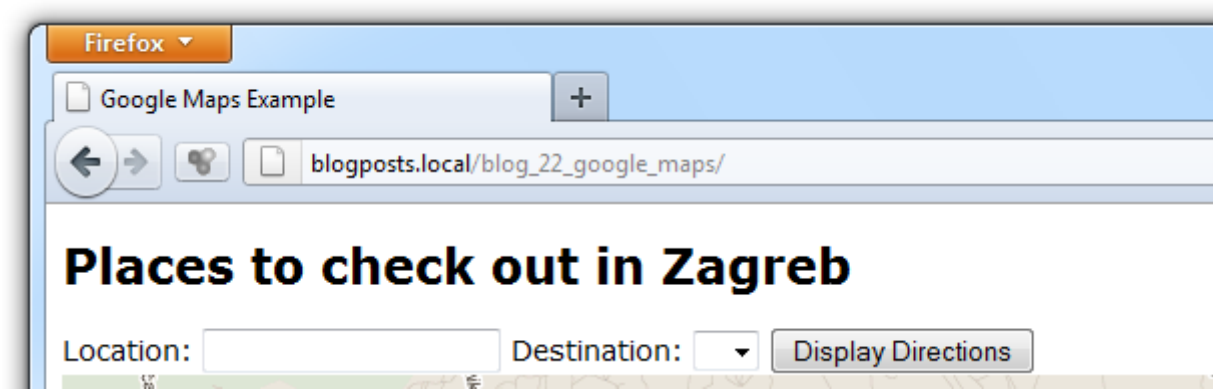
<form id="services">
Location: <input type="text" id="start" />
Destination: <select id="destination" onchange="calculateRoute();"></select>
<input type="button" value="Display Directions" onclick="calculateRoute();" />
</form>

<section id="sidebar">
<div id="directions_panel"></div>
</section>

<section id="main">
<div id="map_canvas" style="width: 70%; height: 500px;"></div>
</section>

</body>
```

If you refresh the page now, you'll see that the form is added, but the destinations select box is empty. So, we need to fill that as well. In addition, we'll try to automatically detect user's location.



First, we'll need to create to new objects before the init function - DirectionsService and DirectionsRenderer.

```
var directionsService = new google.maps.DirectionsService();
var directionsDisplay = new google.maps.DirectionsRenderer();
```

We need to configure the **directionsDisplay** object. We can do it inside our init function. Also, let's add autodetection of user's location to our init function.

```
var map;

// Ban Jelačić Square - City Center
var center = new google.maps.LatLng(45.812897, 15.97706);

var geocoder = new google.maps.Geocoder();
var infowindow = new google.maps.InfoWindow();

var directionsService = new google.maps.DirectionsService();
var directionsDisplay = new google.maps.DirectionsRenderer();

function init() {

var mapOptions = {
  zoom: 13,
  center: center,
  mapTypeId: google.maps.MapTypeId.ROADMAP
}

map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);

directionsDisplay.setMap(map);
directionsDisplay.setPanel(document.getElementById('directions_panel'));

// Detect user location
if(navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(function(position) {

var userLocation = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);

geocoder.geocode( { 'latLng': userLocation }, function(results, status) {
  if (status == google.maps.GeocoderStatus.OK) {
    document.getElementById('start').value = results[0].formatted_address
  }
});

}, function() {
  alert('Geolocation is supported, but it failed');
});
}

makeRequest('get_locations.php', function(data) {

var data = JSON.parse(data.responseText);
var selectBox = document.getElementById('destination');

for (var i = 0; i < data.length; i++) {
  displayLocation(data[i]);
  addOption(selectBox, data[i]['name'], data[i]['address']);
}
});

function addOption(selectBox, text, value) {
var option = document.createElement("OPTION");
option.text = text;
option.value = value;
selectBox.options.add(option);
}
```

We use the W3C Geolocation. If the geolocation succeeds we use Geocoding to get the user's address. If this succeeds we enter that location to the location text box.

We also added the **addOption** function which adds options to a select box. So, inside our for loop we call the addOption function with selectbox, name and address parameters. First parameter indicates which selectbox to fill.

Next, add a **calculateRoute** function inside our JavaScript:



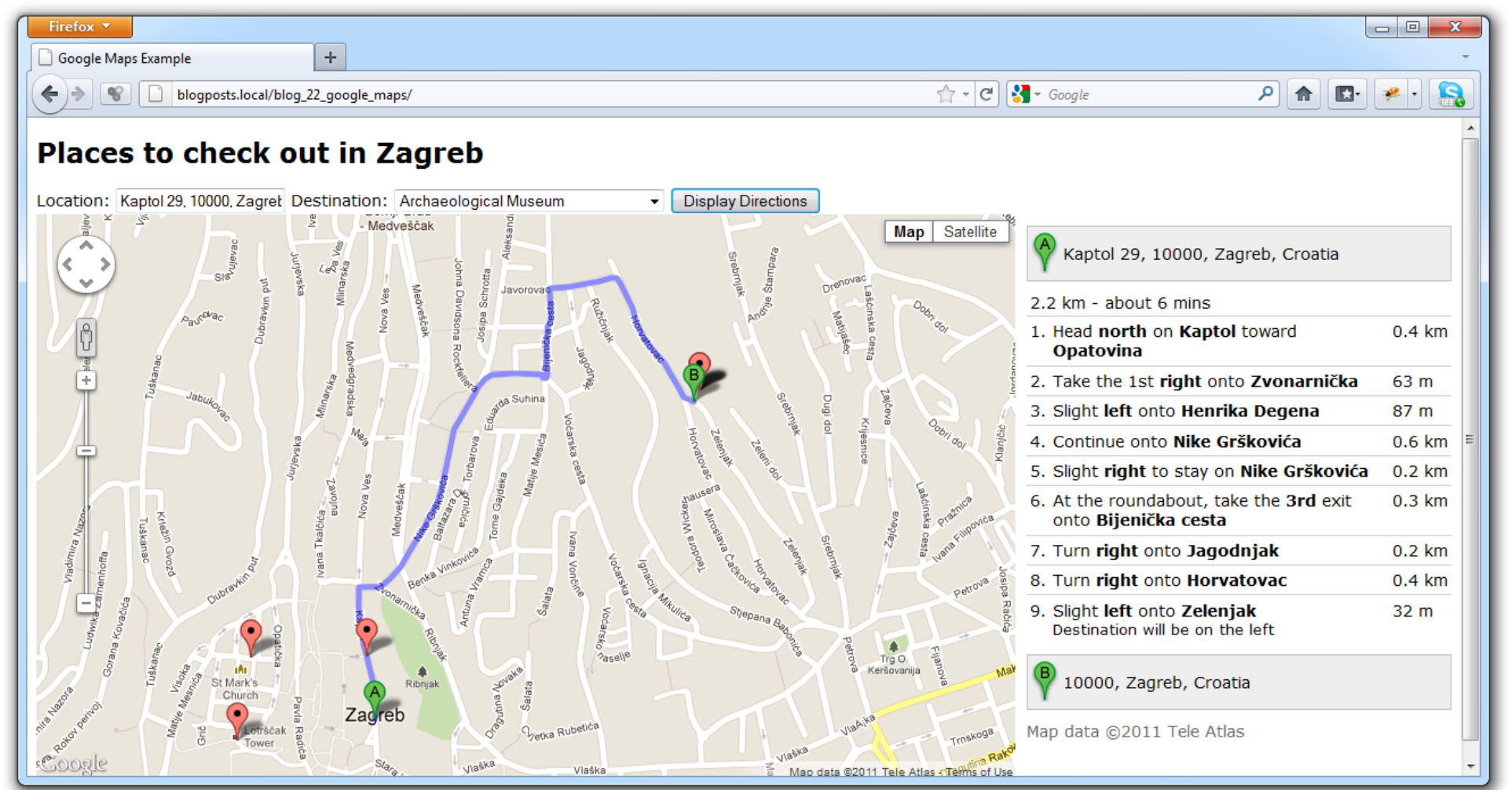


```
function calculateRoute() {  
  
    var start = document.getElementById('start').value;  
    var destination = document.getElementById('destination').value;  
  
    if (start == '') {  
        start = center;  
    }  
  
    var request = {  
        origin: start,  
        destination: destination,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

This function is pretty simple. It uses text from input field as a origin address and selected location as a destination. If the field is empty, we'll use city center as a starting point.

We then use `route` method to issue a request to the Directions service. If status is OK, we display directions.

If you refresh the page now and click the button or change a select box, the directions will be displayed, which means we're done with the map.



## Additional: Obtaining location coordinates

You might say, getting the coordinates manually and then entering them into the database is not very fun. And as we seen earlier, Google Geocoding service can be used to geocode addresses into latitudes and longitudes. After that we can use PHP/MySQL to update those coordinates for locations.

If you decide to go that way, here's a piece of JavaScript code you can use to send a Ajax request to update latitude and longitude (insert it into `displayLocation` function - if geocoding succeeds):



```
// Save geocoding result to the Database
var url = 'set_coords.php?id=' + location.id
+ '&lat=' + results[0].geometry.location.lat()
+ '&lon=' + results[0].geometry.location.lng();

makeRequest(url, function(data) {
if (data.responseText == 'OK') {
// Success
}
});
```

The PHP file (**set\_coords.php**) to update latitude and longitude would go something like this:

```
<?php

require 'config.php';

try {

$db = new PDO($dsn, $username, $password);
$db->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

$sth = $db->prepare("UPDATE locations SET lat = ?, lon = ? WHERE id = ?");
if ($sth->execute(array($_GET['lat'], $_GET['lon'], $_GET['id'])))
echo "OK";

} catch (Exception $e) {
echo $e->getMessage();
}
```

That's basically it. Let's repeat entire JavaScript again, so that you have everything in one place:

```

var map;

// Ban Jelačić Square - City Center
var center = new google.maps.LatLng(45.812897, 15.97706);

var geocoder = new google.maps.Geocoder();
var infowindow = new google.maps.InfoWindow();

var directionsService = new google.maps.DirectionsService();
var directionsDisplay = new google.maps.DirectionsRenderer();

function init() {

var mapOptions = {
  zoom: 13,
  center: center,
  mapTypeId: google.maps.MapTypeId.ROADMAP
}

map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);

directionsDisplay.setMap(map);
directionsDisplay.setPanel(document.getElementById('directions_panel'));

// Detect user location
if(navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(function(position) {

var userLocation = new google.maps.LatLng(position.coords.latitude,position.coords.longitude);

geocoder.geocode( { 'latLng': userLocation }, function(results, status) {
  if (status == google.maps.GeocoderStatus.OK) {
    document.getElementById('start').value = results[0].formatted_address;
  }
});

}, function() {
  alert('Geolocation is supported, but it failed');
});
}

makeRequest('get_locations.php', function(data) {

var data = JSON.parse(data.responseText);
var selectBox = document.getElementById('destination');

for (var i = 0; i < data.length; i++) {
  displayLocation(data[i]);
  addOption(selectBox, data[i]['name'], data[i]['address']);
}
});

function displayLocation(location) {

var content = '<div class="infoWindow"><strong>' + location.name + '</strong>'
+ '<br/>' + location.address
+ '<br/>' + location.description + '</div>';

if (parseInt(location.lat) == 0) {
  geocoder.geocode( { 'address': location.address }, function(results, status) {
    if (status == google.maps.GeocoderStatus.OK) {

var marker = new google.maps.Marker({
  map: map,
  position: results[0].geometry.location,
  title: location.name
});

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

/* Save geocoding result to the Database
var url = 'set_coords.php?id=' + location.id
+ '&lat=' + results[0].geometry.location.lat()
+ '&lon=' + results[0].geometry.location.lng();

makeRequest(url, function(data) {
  if (data.responseText == 'OK') {
    // Success
  }
});*/
}
});
} else {

var position = new google.maps.LatLng(parseFloat(location.lat), parseFloat(location.lon));
var marker = new google.maps.Marker({
  map: map,
  position: position,
  title: location.name
});

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

```

```
}
}

function addOption(selectBox, text, value) {
var option = document.createElement("OPTION");
option.text = text;
option.value = value;
selectBox.options.add(option);
}

function calculateRoute() {

var start = document.getElementById('start').value;
var destination = document.getElementById('destination').value;

if (start == '') {
start = center;
}

var request = {
origin: start,
destination: destination,
travelMode: google.maps.DirectionsTravelMode.DRIVING
};
directionsService.route(request, function(response, status) {
if (status == google.maps.DirectionsStatus.OK) {
directionsDisplay.setDirections(response);
}
});
});

function makeRequest(url, callback) {
var request;
if (window.XMLHttpRequest) {
request = new XMLHttpRequest(); // IE7+, Firefox, Chrome, Opera, Safari
} else {
request = new ActiveXObject("Microsoft.XMLHTTP"); // IE6, IE5
}
request.onreadystatechange = function() {
if (request.readyState == 4 && request.status == 200) {
callback(request);
}
}
request.open("GET", url, true);
request.send();
}
```

So, there you have it. A simple map with locations displayed from the database, directions to those locations and info windows displaying info about locations. We even have simple means for getting coordinates and storing them into the database.

NOTE: This is by no means a complete production-ready solution and it basically serves to show how easy it's to start using Google Maps API. Also, note that proposed solution to the problem of not having coordinates for some locations is not optimal. The best solution would be to use a Google Maps Geocoding Service with PHP to update coordinates in the database, so we have them ready. To do that, I suggest you take a look at the [Geocoding Addresses with PHP/MySQL](#) article.


I hope you found something useful in this tutorial. Thanks for visiting this site and until next time I wish you smooth work days without stress and nervousness.

Share this Post:

⇐ [Create a Registration and Login System using Facebook Registration Plugin](#)


[Create a simple Responsive Design for a blog site using CSS media queries](#) ⇒

## 2 Comments



RAYMOND

31 March 2020 21:30:50



hi,  
// Save geocoding result to the Database < This doesnt work. Any ideas?  
Also, if I add a date columns in the DB and each records and would like to select dates to display pointers.. instead of displaying all as the command shows in "get\_locations.php" is that possible? Meaning a php form with from\_date and to\_date and when submit, it displays records of those days only.  
  
Would really appreciate your help.



**Akinwumi Ayotunde**

13 May 2020 01:18:38

pls can help me with the source code in zip



**Logged in users can add comments.**

 LOGIN

Leave a **new Comment**

COMMENT

Please login to comment



SUBMIT COMMENT





CATEGORIES

- Personal (2)
- Development (22)
- IT (2)
- Networking (3)
- Dynamics CRM (1)

Logged in users can **comment**, **vote** and **add posts to Favourites**.  
Login or create your account in a matter of seconds.

 LOGIN

Tweets

RECENT POSTS

- 

MS Dynamics 365: Some useful JavaScript Functions  
*27. July, 2020.*
- 

Three Web.config tricks - force HTTPS, force yes/no www subdomain  
*28. October, 2019.*
- 

Migration of the old posts  
*03. November, 2018.*

TAG CLOUD

- SEO
- Ubuntu
- JavaScript
- Dynamics 365

