# Using HTML5 and Javascript to embed a google map based on your current location

## What is Geolocation?

Geolocation is the principal of using your computer's IP and other data to determine where abouts you are. It's used by many websites to personalise results for your area in particular. Most browsers warn the user they are being tracked, and various laws apply to various countries so it's worth taking note of, it's also worth noting that different browsers support this in different ways- for instance Chrome won't allow a geolocation request unless you're requesting from a secure site.

## How can I implement it?

Geolocation by itself just returns the latitude and longitude of your position. You can find more information about latitude and longitude here- http://whatis.techtarget.com/definition/latitude-and-longitude.w We then need to do something with that, so let's start with some JavaScript that handles this.

### Starting point in HTML

```
<doctype html>

<head>

        <script type="text/javascript">

        </script>

</head>

<title>Geolocation Tests<title>

<body>

        <p id="demo"> Lat and Long here </p>

        <div id="mapholder">Map Holder</div>

        <button onclick="getLocation()">Click me</button>

</body>

</html>
```

This basic HTML document is a mostly empty page with room to add a scripts in the head and a body that contains a blank paragraph with the id demo- where the lat and long are posted and a blank divider where the Google map will eventually go. Finally, there is a button linked to the Javascript code we're going to write.

Copy this into your own HTML document and read on to discover how to add Javascript to your page to add some geolocation.

## Basic Geolocation request

Geolocation is built into the HTML5 API, this means we actually don't need to write much, "getCurrentPosition()" is the method used to get the user's position.

Inside the script tags above add the following JavaScript code…

```
var x = document.getElementById("demo");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}
function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
```

The first line searches for the document to find the element called "demo" recall from the HTML we set up this is where lat and long will go.

The first function is getLocation() a function is a self contained bundle of code we can call any time by using the name.

Within this function we test if navigator.geolocation is true (this is a simple test to see if the browser supports geolocation). If it is we run the built in getCurrentPosition command using the "showPosition" parameter this may not seem straightforwards but it is essentially saying "run getCurrentPosition and send it straight to showPosition". If not, we change the innerHTML (what's inside the braces) of x we found earlier to the string shown.

The second function is showPosition with a parameter (position), this says for the position given, put in the innerHTML section the latitude and the longitude. We don't need to specify types here as Javascript switches between them easily, it doesn't matter what type of data it is.

Try it, you should get the lat and long of your position printed.

## Displaying the result in a map

Lat and long mean nothing to most people, so instead we can pass it to a map service like google.

```
function showPosition(position) {
    var latlon = position.coords.latitude + "," +
position.coords.longitude;

    var img_url = "http://maps.googleapis.com/maps/api/staticmap?center=
    "+latlon+"&zoom=14&size=400x300&sensor=false";

    document.getElementById("mapholder").innerHTML = "<img
src='"+img_url+"'>";
}
```

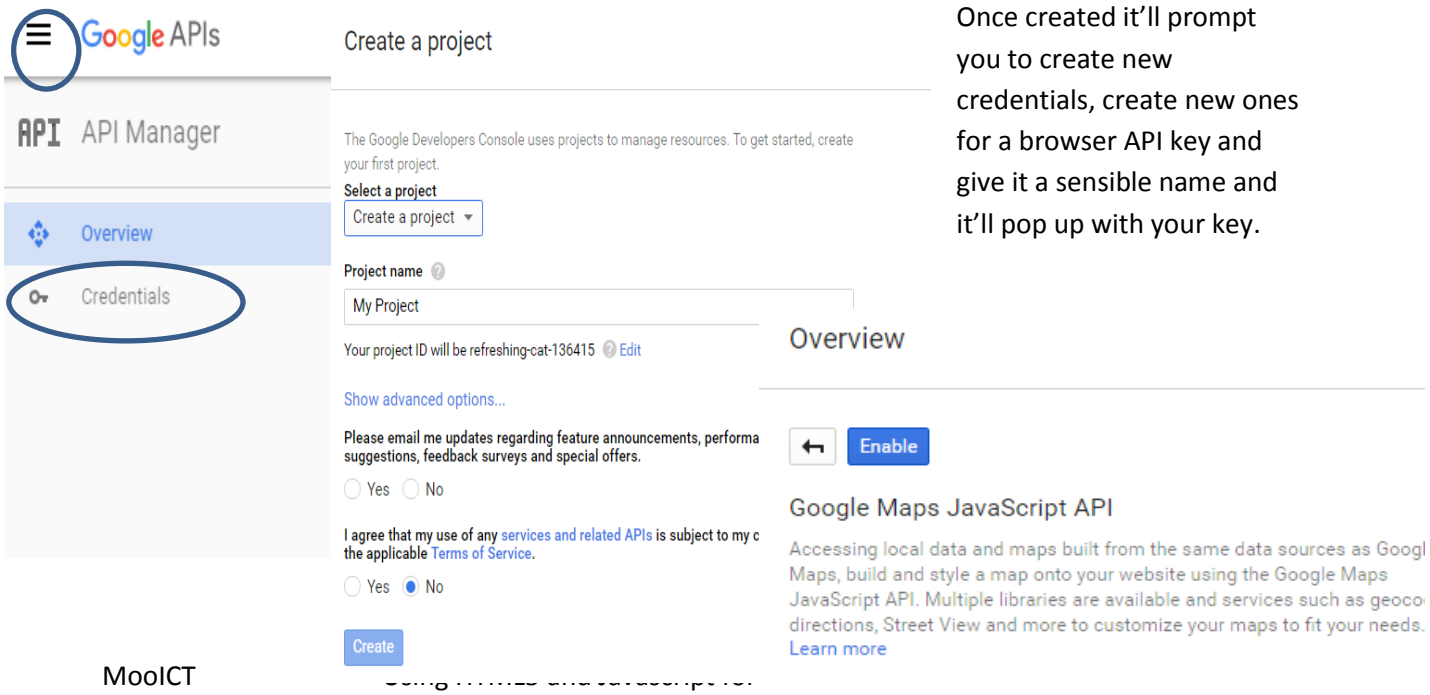
Replace your showPosition code with the one shown above which tells google to generate a static image and loads that img URL into a variable. Notice the +latlon+ that's where we pass google our latitude and longitude in the correct format, once we've got it, we print it out to the mapholder element and voila, we can see the map.

### Going one step further, directions from and too (More Advanced)

This bit is more complicated again; significantly, it uses the google maps API to query google for a live map showing where the user is that they can interact with. We'll then add to it using google directions API.

The first step is to use the developers.google.com website to sign in as a developer, if you've already got a google account/ gmail address you can use that to sign in.

Once you've signed in you now need to get your API key setup for a project, in the API Manager click on the three bars to ensure you're in the API manager. Find create a project, and create a project and once it's created click google maps javascript API, then enable to enable it. Now, click on Credentials, this will let you setup a new project, click the corresponding buttons- email requests and terms of service and click create.




Once created it'll prompt you to create new credentials, create new ones for a browser API key and give it a sensible name and it'll pop up with your key.

To bring and use the google API into your website you need to use the following tag and code;

**<script async defer src="https://maps.googleapis.com/maps/api/js?key=YOUR_KEY-HERE&callback=initMap" type="text/javascript"></script>**

Be sure to replace the YOUR_KEY-HERE with your API code you generated above. You may have noticed the "async defer" bit of that script tag. This means your javascript is loaded asynchronously, in other words it'll keep loading the rest of the page then go back to load and run your javascript result when it's ready. This means it doesn't hold up the rest of the page loading.

We can just get a basic map running fairly easily, the line from above that says "callback=initMap" this means when the javascript from google has finished loading it runs the Javascript function called "initMap"

## Navigatable map

```
function initMap() {
  var map;
  map = new google.maps.Map(document.getElementById('mapholder'), {
    center: {lat: 52.486243, lng: -1.890401},
    zoom: 8
  });
}
```

Add this code between a script tag somewhere in your HTML document before you import the api.

This is our Javascript function initMap that is called by the API import when it's finished loading. It's fairly simple, creates a new map by contacting google and puts it inside the html of the object called mapholder. Then, it sets the center of the map to the lat and long given (Birmingham) and zoom to the level set.

Notice with this new map a few things 1) it loads automatically when you load the page, 2) it's navigatable, you can move it around, street view, all the things you would for a normal map.

## Navigatable Map of local area

That's nice and all, but we could just use a static map of an area or a static location, what we really want is to combine this all together, get the user's location then send it to google to render a map of that location.

So, that's what we'll do.

## Navigatable Map of user position

```
function initMap()
{
    var map;
    if (navigator.geolocation)
    {
        navigator.geolocation.getCurrentPosition(showUserPosition);
    }
}

function showUserPosition(position) {


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

    var map = new google.maps.Map(document.getElementById("mapholder"), { center: coords, zoom: 15 });

}
```

Some changes made to initMap now, it still creates a map but then checks if geolocation is available, if it is, like before it runs the geolocation and sends the information to "showUserPosition". This is a different function that runs, "cords" sends out lat and long from geolocation to google and gets them back in the format google maps likes.

It then creates a new map object and like before finds the mapholder element and sticks the map in there with the center and zoom level shown.

We can also add an else to the if statement, saying if the statement fails so geolocation isn't available, just default to a map of a set location, that's fairly easy IF you read back so I'll leave it up to you. The second parameter of `getCurrentPosition` is used to direct some code to run if there's an error so you should also put that in. If you're totally stuck you can check the source on site.

## Directions

The last bit we will look at is directions. These are an additional service available from google and can be very helpful in providing directions to your shop or a location from wherever the user is. This brings together all the stuff we've seen so far, maps, geolocation and google services.

We are writing a new function, showDirections, this takes in the position data and sets up a few variables. DirectionsDisplay and DirectionsService are two google made functions we can pull in that lets us calculate and render onto screen the directions.

```
function showDirections(position) {
    var directionsDisplay = new google.maps.DirectionsRenderer();
    var directionsService = new google.maps.DirectionsService();

    var coords = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);
    var Birmingham = new google.maps.LatLng(52.486243, -1.890401);
    var map = new google.maps.Map(document.getElementById("mapholder"), { center: coords, zoom: 10 });
    directionsDisplay.setMap(map);

    var request = {
        origin: Birmingham,
        destination: coords,
        travelMode: google.maps.TravelMode.DRIVING
    };

    directionsService.route(request, function (result, status) {
        if (status == google.maps.DirectionsStatus.OK) {
            directionsDisplay.setDirections(result);
        }
    });
}
```

Next comes the coords, like before we can pass the position from our geolocation to google to get it in a format we like. This time we also pass our destination point, a point somewhere in Birmingham- the location of where we want to go, you can use plenty of sites to get the lat long of a location, a quick search will bring them up. Then as before we setup the map holder centered around the cords and finally tell the directions renderer what map we're using to render directions to.
Next up is the request to the directions service, we specify our origin and our destination then we specify the mode of travel. Here we've used driving as the default, we can change it if we'd like (see google's API docs for more information).

The only thing left to do is alter initmap to make sure showDirections is called when starting.

```
function initMap()
{
    var map;
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showDirections,showDefaultPosition);
    }
    else {
        showDefaultPosition();
    }
}
```

That's it for this tutorial, use it with other javascript functions and some CSS to produce interactive websites. For instance you could couple it with a javascript alert function to produce a simple popup saying "Congratulations winner in <location> you have won!" but we wouldn't do that would we. Those of you taking part in or aware of the pokemon go trend, this is also one of the technologies that it uses for it's game, tracking the user's location and overlaying stuff depending on where you are. It's also the same stuff the pokemon trackers use.

## Acknowledgements and further reading

Google Developers. (2016). *Geolocation*. [online] Available at: https://developers.google.com/maps/documentation/javascript/examples/map-geolocation [Accessed 14 Aug. 2016].

Google Developers. (2016). *Google Maps Directions API | Google Developers*. [online] Available at: https://developers.google.com/maps/documentation/directions/ [Accessed 14 Aug. 2016].

Mozilla Developer Network. (2016). *Using geolocation*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/Using_geolocation [Accessed 14 Aug. 2016].

W3schools.com. (2016). *HTML5 Geolocation*. [online] Available at: http://www.w3schools.com/html/html5_geolocation.asp [Accessed 14 Aug. 2016].