Manual de Google Maps JavaScript API v3

Rubén Alcaraz Martínez

Departament de Biblioteconomia, Documentació i Comunicació Audiovisual

juliol 2019

UNIVERSITAT DE BARCELONA

Sumario

Resumen

Manual de referencia de carácter introductorio para el uso de la versión 3 de la API de Google Maps. El manual está dirigido a diseñadores y desarrolladores web, así como a estudiantes que deseen iniciarse en el uso de esta API.

Se explica cómo acceder al servicio, la personalización de la interfaz y los controles del mapa, añadir uno o varios marcadores a través del código JavaScript o de información contenida en un fichero JSON externo, el uso del servicio de rutas y direcciones, la aplicación de la API de geolocalización de HTML5, la creación de polígonos y otras formas sobre el mapa, así como algunas estrategias para mejorar la accesibilidad de los productos resultantes.

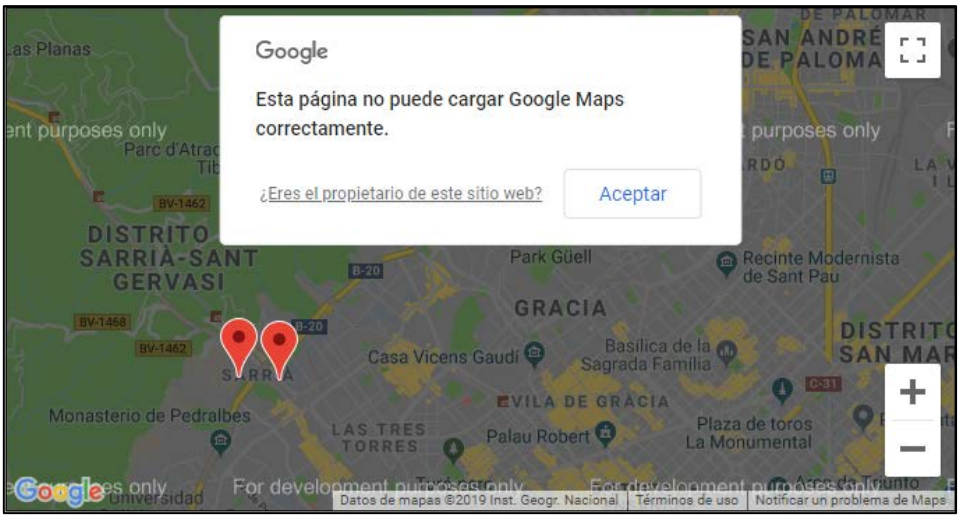
El manual se encuentra acompañado por el código expuesto como ejemplo en los diferentes apartados del documento.

0. Antes de empezar

0.1. Conseguir una clave de API

Desde octubre de 2016, es necesario obtener una clave de API para poder acceder a los servicios de la API de Google Maps. Podemos conseguir una clave de manera gratuita en la Consola de desarrolladres de Google[1], pero es necesario introducir los datos de una tarjeta de crédito, en la cual Google efectuaría los cargos correspondientes en el caso de que excediéramos el uso gratuito. A continuación, se explica el proceso de obtención de la clave, pero para entornos de preproducción o en el aula, se puede utilizar la siguiente llamada que también funcionará, aunque mostrando los mapas creados con un sombreado y un mensaje que indica que no estamos utilizando ninguna clave.

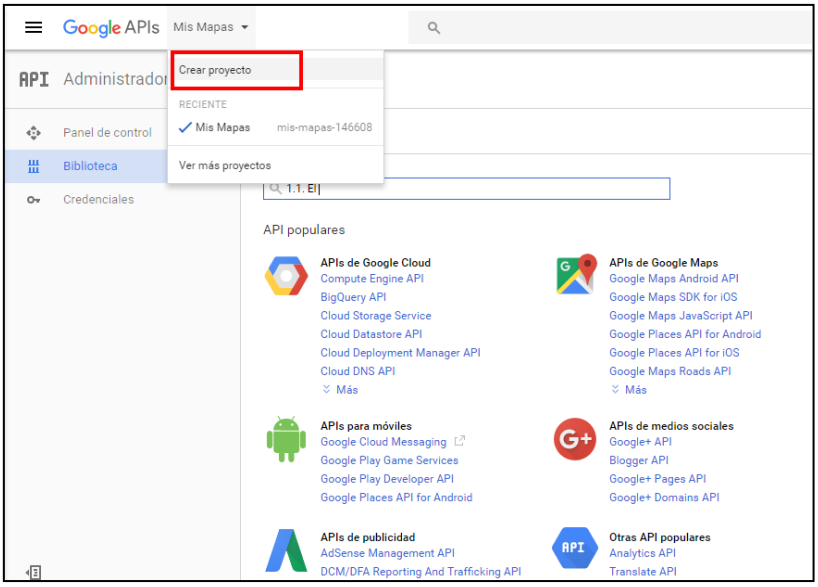
```
<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js"></script>
```



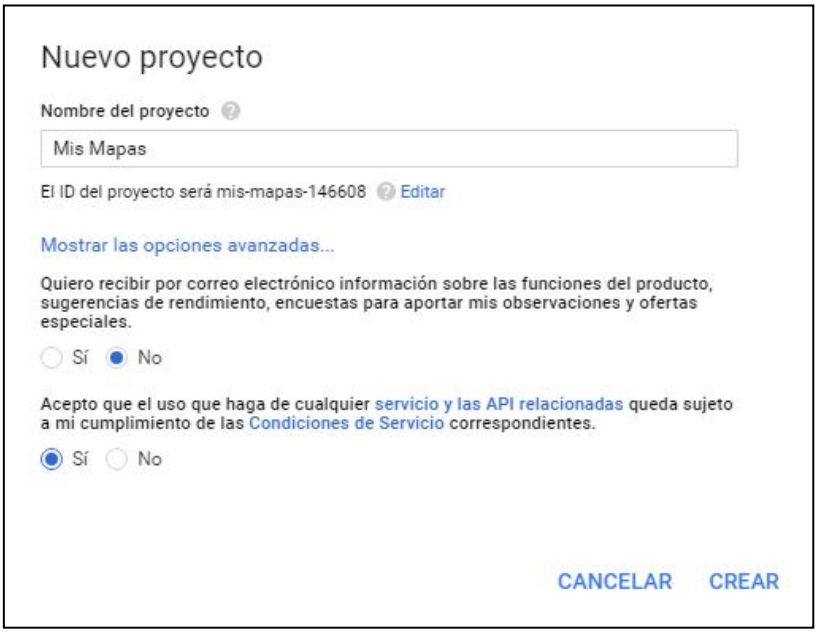
Mapa creado sin una clave API. Al pulsar sobre el botón aceptar el mensaje desaparece.

[1] https://console.developers.google.com/apis

Si necesitamos disponer de una clave API, una vez dentro de la Consola de desarrolladores, debemos crear un nuevo proyecto y aceptar los términos de uso.



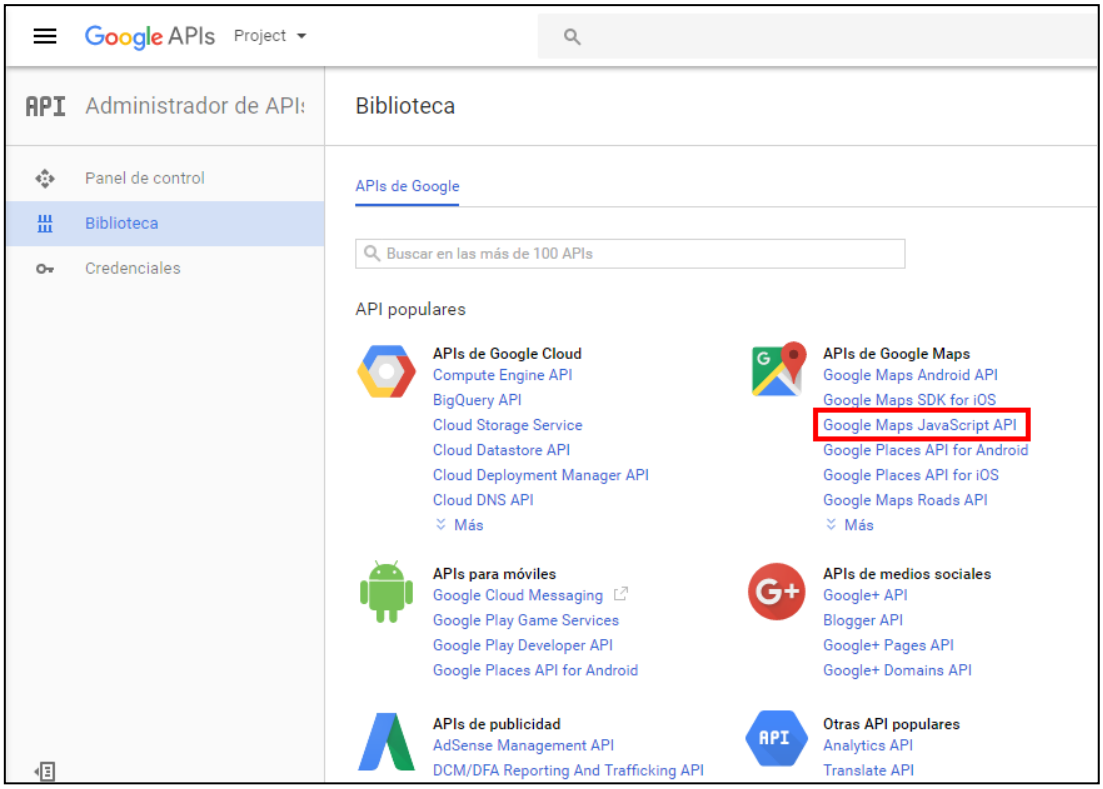
Creación del proyecto.



Asignación del nombre del proyecto y aceptación de los términos de uso.

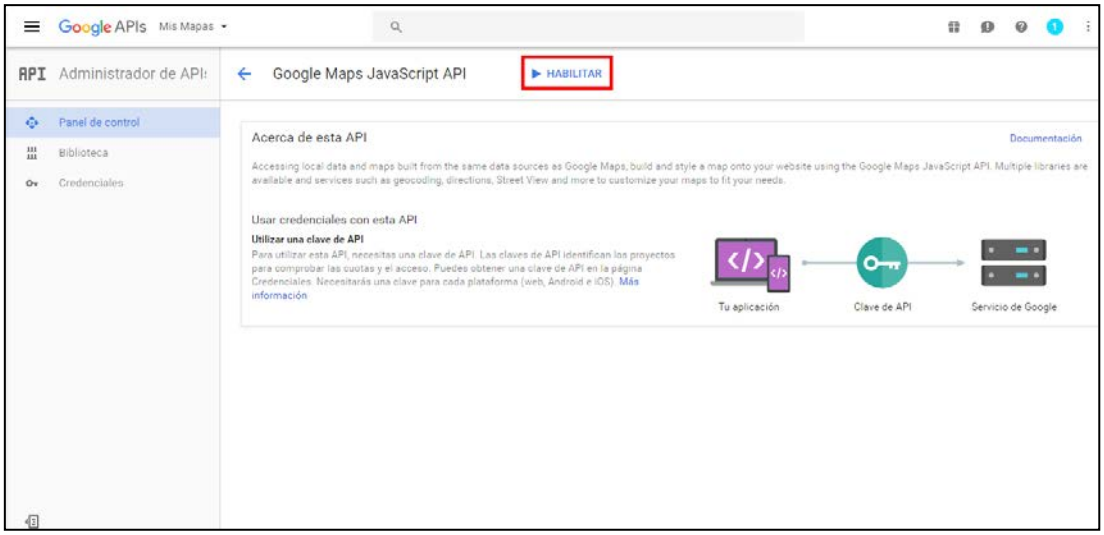
A continuación, seleccionaremos la API que deseamos utilizar. En nuestro caso, como mínimo la API de JavaScript de Google Maps. Si deseamos añadir más APIs, deberemos seguir los mismos pasos que se muestran en este apartado una

vez finalizado el proceso que nos ocupa y teniendo seleccionado el proyecto que acabamos de crear.



Selección de la **API de JavaScript de Google Maps.**

Una vez seleccionada, debemos habilitarla.

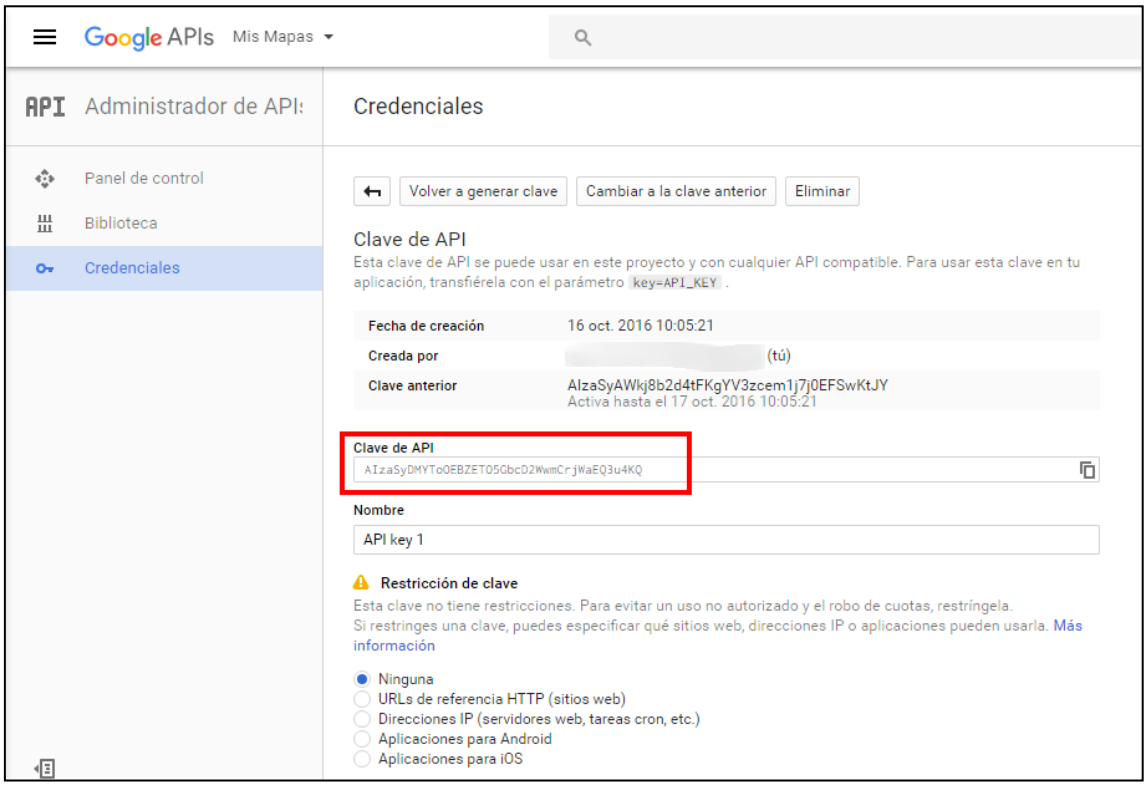


Habilitando la API.

A continuación, creamos las credenciales.

Creando las credenciales.

Una vez finalizado el proceso, se nos asignará una clave que debemos incorporar a la llamada a la API tal y como se indica a continuación.



Clave API asignada.

<script type="text/javascript" src="http://maps.google.com/maps/api/js?key=Aqui-mi-clave"></script>

1. Fundamentos

En este apartado se explican los fundamentos o elementos básicos necesarios para empezar a trabajar con la API de Google Maps: los ficheros HMTL, CSS i JS, así como la referencia necesaria y parámetros de la API de este servicio de Google.

1.1. El contenedor del mapa: el documento HTML y CSS

Para crear un mapa con la API de Google Maps, en primer lugar, debemos crear un documento HTML con una estructura mínima similar a la reproducida a continuación.

```
<!DOCTYPE html>
<html lang="es">
	<head>
		<meta charset="utf-8">
		<link type="text/css" href="css/estilo.css" rel="stylesheet" media="all" />
		<script type="text/javascript"
		src="http://maps.google.com/maps/api/js?key=aqui-miclave&sensor=false"></script>
		<title>GoogleMaps APIv3</title>
	</head>
	<body>
		<div id="mapa"></div>
	</body>
</html>
```

En el ejemplo anterior y en los siguientes, utilizaremos HTML5. El primero de los elementos es la declaración de tipo de documento (DOCTYPE) y se utiliza para explicarle al navegador cómo interpretar la página. El segundo de los elementos es la declaración de idioma (<html lang="es">) en el elemento HTML. El tercero de los elementos es la etiqueta <head>. Esta sección de la página contiene una serie de elementos muy importantes. En primer lugar, encontramos la etiqueta <meta charset> que dice qué tipo de codificación de caracteres estamos utilizando. En este caso utilizaremos UTF-8, ya que incluye caracteres especiales para todos los idiomas. A continuación, podemos observar la etiqueta <title>, que

establece el título de la página web. Ambos son elementos obligatorios que debemos incluir en nuestro código para poder validar el documento, según los requisitos del W3C.

Para fijar el tamaño del mapa debemos establecer el estilo del elemento que lo contendrá. El tamaño de este contenedor define las dimensiones del mapa. Una buena práctica consiste en mantener separados el código HTML y el CSS, por lo tanto, crearemos otro archivo con extensión .css llamado "estilo.css". Además, para mantener una estructura ordenada, crearemos una carpeta con el nombre css en la que guardaremos las diferentes hojas de estilo que vayamos creando.

Vamos a trabajar con un mapa que ocupe todo el ancho de la página y 500 píxeles de alto. Además, agregaremos un borde de 1 píxel de color negro a nuestro contenedor.

```
#mapa {
width: 100%;
height: 500px;
border: 1px solid #000;
}
```

Una vez creado el fichero CSS con los estilos, es necesario hacer una referencia en el archivo HTML apuntando al archivo CSS. Esta referencia se hace con el elemento <link> dentro de la sección <head>.

```
<link type="text/css" href="css/estilo.css" rel="stylesheet" media="all" />
```

En ocasiones, podemos tener problemas si el tamaño del mapa no se encuentra definido en el mismo fichero HTML. En ese caso, podemos probar alguna de las dos siguientes alternativas:

- Proporcionar el estilo del elemento en la cabecera del documento HTML (dentro de la sección <head>):

```
<style>
	#mapa {
	width: 100%;
	height: 500px;
	border: 1px solid #000;
	}
</style>
```

- Indicar el estilo del elemento en el propio elemento mediante el atributo "style":

```
<div id="map" style="height: 500px; width:100%; border:1px solid #000;"></div>
```

1.2. La API de Google Maps

La API de Google Maps se encuentra alojada en los servidores de Google. Para poder cargar esta API, debemos hacer una referencia desde nuestro archivo HTML hacia el lugar en el que se encuentra. La referencia se ha de incluir en la sección <head> del documento. La cargaremos con el elemento <script>. Este elemento tiene dos atributos que debemos utilizar. El primero es el tipo de script que deseamos utilizar y el otro es el URL que apunta a la API.

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

1.3. Parámetros de la API de Google Maps

1.3.1. Sensor

La API de Google Maps requiere que se indique si la aplicación que estamos creando utiliza algún tipo de sensor, por ejemplo, para determinar la ubicación del usuario a través de un localizador de GPS. Los posibles valores son "true" y "false".

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js=aqui-mi-clave&?sensor=false"></script>
```

1.3.2. Localización (idioma)

La API de Google Maps intentará determinar automáticamente qué idioma utiliza la interfaz del usuario. No obstante, también podemos indicarlo en el <script> que hemos utilizado para referenciar la API. Para hacerlo, debemos añadir el parámetro (opcional) &language="codigo_del_idioma" al final de la cadena.

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?=aqui-
miclave&sensor=false&language=es"></script>
```

1.2.3. Localización (región)

Al cargar la API de Google Maps desde maps.googleapis.com, se aplica un sesgo por defecto hacia los Estados Unidos. Si deseamos modificar este comportamiento predeterminado, lo podemos hacer mediante la adición de un nuevo parámetro a la etiqueta <script>: "region". Los valores válidos para este parámetro son los de la *ISO 3166-1 alpha-2 code*. El valor para España es "ES".

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js? =aqui-
miclave&sensor=false&region=ES"></script>
```

1.3. El código JavaScript

Ahora que ya tenemos nuestro fichero HTML vinculado a la API de Google Maps y a la hoja de estilos, ya podemos empezar a escribir código JavaScript para inicializar nuestro mapa. De la misma manera que con las hojas de estilo, también es una buena práctica separar el código JavaScript de nuestro HTML. Así que crearemos un fichero llamado "mapa.js" y lo guardaremos en una carpeta con nombre "js".

Una vez creada esa estructura, debemos referenciar el fichero "mapa.js" desde nuestra página HTML. Para ello, utilizaremos de nuevo la etiqueta <script>, pero esta vez referenciando un fichero ubicado en nuestro PC o servidor.

```
<script type="text/javascript" src="js/mapa.js"></script>
```

La posición de esta nueva etiqueta no es trivial. Debe estar ubicada a continuación de la referencia a la API de Google Maps, y no antes. De esta manera nos aseguramos de que la API de Google Maps sea cargada antes de que la página HTML intente usarla.

1.4. El contenedor del mapa

Llegados a este punto, inicializaremos un mapa dentro del que anteriormente habíamos creado en nuestra página HTML. Para ello, haremos una referencia desde nuestro fichero "mapa.js", utilizando el método "document.getElementById()". Este método busca el ID de un elemento HTML y devuelve una referencia a ese mismo elemento. Es importante tener en cuenta que un mismo ID sólo puede ser utilizado una vez por página. En el caso de que no se pueda encontrar el ID indicado, el valor devuelto será "null" o lo que es lo mismo, nada.

Crearemos una variable llamada mapDiv usando el método "getElementById()" referenciando nuestro .

```
var mapDiv = document.getElementById('mapa');
```

El objeto "mapOptions" es necesario y define las propiedades de visualización del mapa. Para utilizarlo, crearemos una variable llamada "mapOptions" con las tres propiedades mínimas necesarias para que el mapa funcione:

- **center:** Define el centro del mapa mediante unas coordenadas.
- **zoom:** Define el nivel inicial de zoom del mapa. Debe ser un número comprendido entre 1 (zoom mínimo) y 23 (zoom máximo).
- **mapTypeId:** Define el tipo de mapa que será cargado inicialmente. Los diferentes tipos de mapas los encontramos en el objeto "google.maps.MapTypeId". Por ejemplo:

google.maps.MapTypeId.ROADMAP o google.maps.MapTypeId.SATELLITE.

- Las coordenadas para la propiedad "center" pueden facilitarse directamente como valor de la propiedad, o pueden estar almacenadas en alguna variable que hayamos definido anteriormente. Por ejemplo:

```
center: new google.maps.LatLng(41.652393,1.691895),
o
var catalunya = new google.maps.LatLng(41.652393,1.691895);
center: catalunya,
```

En el primer caso, hemos utilizado el objeto "google.maps.LatLng" para proporcionar las coordenadas del centro de nuestro mapa. En el segundo caso hemos grabado las coordenadas de Cataluña en una variable mediante la cual centraremos posteriormente nuestro mapa en esa posición.

Todo lo visto anteriormente junto:

```
var mapDiv = document.getElementById('mapa');
var catalunya = new google.maps.LatLng(41.652393,1.691895);
var mapOptions = {
        center: catalunya,
        zoom: 8,
        mapTypeId: google.maps.MapTypeId.ROADMAP
};
```

Ahora que ya hemos hecho referencia al elemento mediante el "mapDiv" y hemos establecido las diferentes opciones, sólo falta pasarlas al objeto "Map".

```
var mapa = new google.maps.Map(mapDiv, mapOptions);
```

Una vez lo tenemos todo, sólo nos queda poner en marcha el código cuando la página HTML se haya cargado completamente. Para ello, utilizaremos el evento DOM "window.onload" al que le asociaremos una función.

```
  window.onload = function() {
        var mapDiv = document.getElementById('mapa');
        var catalunya = new google.maps.LatLng(41.652393,1.691895);
        var mapOptions = {
        center: catalunya,
        zoom: 8,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
        var mapa = new google.maps.Map(mapDiv, mapOptions);
}
```

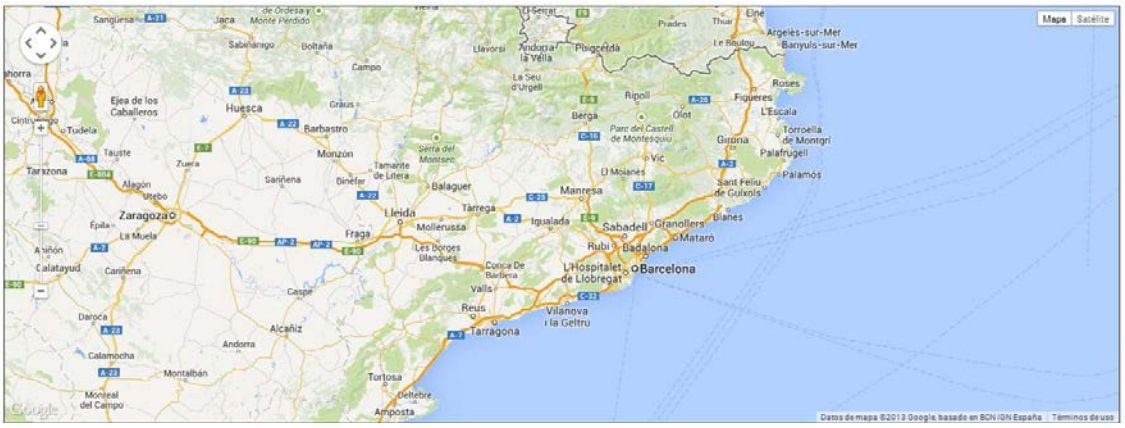
La referencia al elemento y la creación del objeto "Map" se pueden resumir de la siguiente manera:

```
map = new google.maps.Map(document.getElementById('mapa'), mapOptions);
```

Todo junto:

```
 window.onload = function() {
        var catalunya = new google.maps.LatLng(41.652393,1.691895);
        var mapOptions = {
        center: catalunya,
        zoom: 8,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
        map = new google.maps.Map(document.getElementById('mapa'), mapOptions);
}
```

Vista del mapa



Un mapa simple de tipo Roadmap con centro en Cataluña.

1.5. JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos basado en un subconjunto del lenguaje de programación JavaScript. Entre sus principales características destaca su mayor sencillez y ligereza frente a XML y su independencia de cualquier tipo de lenguaje.

```
[
{
"title": "Barcelona",
"lat": 41.385064,
"lng": 2.173404,
"description": "Barcelona"
},
{
"title": "Girona",
"lat": 41.9794,
"lng": 2.821426,
"description": "Girona"
}
]
```

Un fichero JSON se puede crear mediante dos estructuras diferentes:

- Una lista ordenada de valores (en la mayoría de los lenguajes se realiza en forma de array, vector, lista o secuencia).
- Un *array* de objetos, consistente en una colección de pares nombre/valor.

El valor puede tomar cualquiera de los siguientes tipos de datos:

- Objetos (object)
- Matrices o arreglos (colecciones de valores) (array)
- Cadenas de texto (string)
- Números (number)
- CaracteresUnicode válidos (char)
- Valores verdadero o falso (boolean)
- Valores nulos (null)

La forma más habitual de mostrar datos en JSON es a través de *arrays* o colecciones de valores. Cada *array* empieza y acaba con un corchete de inicio y otro de final ([ ]), y representauna lista ordenada de objetos. A su vez, cada objeto consiste en una colección de pares nombre/valor separados por comas. Cada valor puede contener nuevos objetos, con lo cual se pueden crear estructuras más complejas. A continuación, se muestra un ejemplo.

```
[{
       "nombre": "Ateneu Barcelonès",
       "coords": {
        "lng": 2.1716089115143404,
        "lat": 41.3846241973006
    }
},
…
]
```

En el contexto que nos ocupa, podemos añadir datos de forma dinámica a nuestros mapas proporcionados por diferentes servicios o proveedores que utilicen este formato para distribuir información (servicios meteorológicos,

información sobre servicios municipales como escuelas o transporte público proporcionados por gobiernos, etc.). También lo podemos utilizar para definir las propiedades de ciertos elementos de nuestro mapa como los marcadores y su información asociada.

2. Propiedades de control

Los mapas que podemos crear mediante la API de Google Maps disponen de diferentes elementos que permiten a los usuarios interactuar con el mapa. Esta colección de elementos se denominan controles. Por defecto, estos controles se comportan de una manera determinada, pero es posible alterar este comportamiento accediendo a las diferentes propiedades de control.

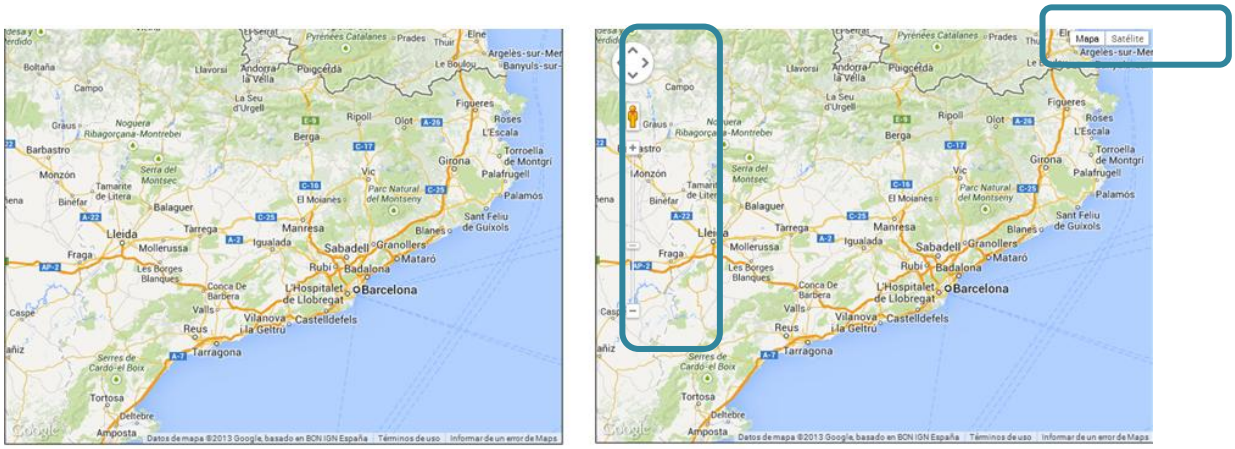
2.1. Propiedades de control sobre la interfaz de usuario

En este apartado se describen diferentes propiedades para el control de la interfaz de usuario.

2.1.1. Botones de tipo de mapa y zoom: disableDefaultUI

El valor por defecto de esta propiedad es "false" o lo que es lo mismo, la interfaz de usuario (UI) por defecto se muestra si nadie dice lo contrario. Al establecer esta propiedad como verdadera ("true") podemos desactivar la interfaz. Desactivar la interfaz significa dejar de mostrar el control de zoom y el selector de tipo de mapa, Como veremos más adelante, también es posible desactivar la interfaz y posteriormente activar o desactivar los diferentes controles individualmente.

```
disableDefaultUI: true
```



A la izquierda, un mapa con la interfaz de usuario desactivada y a la derecha, otro con la interfaz por defecto.

2.1.2. Botón de tipo de mapa: mapTypeControl

Con esta propiedad se controla la aparición del"mapTypeControl". Este elemento aparece ubicado por defecto en la esquina superior derecha del mapa. Se utiliza para escoger el tipo de mapa a mostrar. El valor predeterminado es "true", es decir que si no hacemos nada el "mapTypeControl" siempre se mostrará. Si le asignamos el valor "false", dejará de mostrarse.

```
mapTypeControl: false
```

El mapTypeControl nos permite escoger entre los diferentes tipos de mapa disponibles.

2.1.2.1. mapTypeControlOption

Si decidimos dejar activado el control de tipo de mapa, podemos personalizar su estilo, posición y los tipos de mapa entre los que nuestros usuarios podrán escoger. Para poder personalizar el control de tipo de mapa, es imprescindible utilizar el valor "true" para la propiedad "mapTypeControl".

2.1.2.2. Estilo

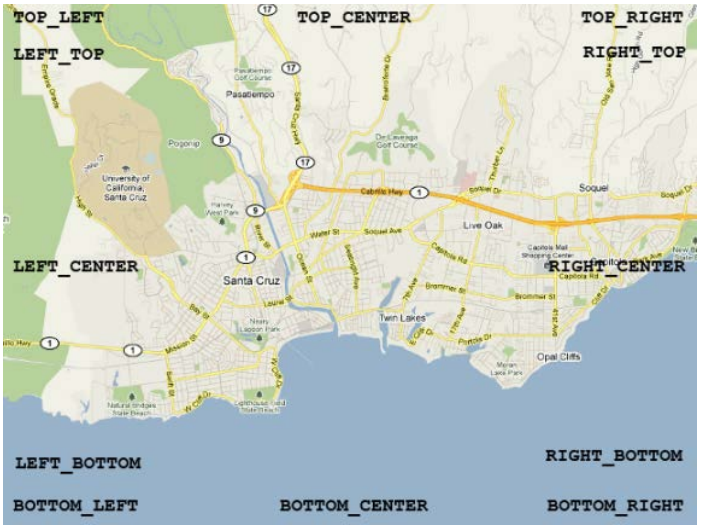
La propiedad "Style" determina la apariencia del objeto "mapTypeControl". Esta propiedad reside en el objeto "google.maps.MapTypeControlStyle". Los diferentes valores aplicables son:

- **DEFAULT:** el valor DEFAULT modifica la apariencia del "mapTypeControl" según el tamaño de la pantalla y otros factores. Si la pantalla es suficientemente grande, el control se mostrará en una barra horizontal, en caso contrario, se utilizará en forma de desplegable.
- **HORIZONTAL_BAR:** Este valor mostrará siempre nuestro "mapTypeControl" en forma de barra horizontal, por muy pequeño que sea el tamaño de la pantalla.
- **DROPDOWN_MENU:** Esta opción muestra siempre el control en forma de desplegable.

```
mapTypeControlOptions: {
      style: google.maps.MapTypeControlStyle.HORIZONTAL_BAR
}
```

2.1.2.3. Posición

La posición por defecto del objeto "mapTypeControl" es en la parte superior derecha del mapa. Mediante la clase "google.maps.ControlPosition" es posible posicionar este control en una ubicación diferente.



Mapa con las diferentes posiciones disponibles para los elementos de la interfaz.

A pesar de poder seleccionar cualquiera de las ubicaciones mostradas en la figura anterior, no todas funcionan correctamente. Las posiciones LEFT y RIGHT se muestran en la misma zona que las posiciones TOP_LEFT y TOP_RIGHT. La posición BOTTOM_RIGHT no se muestra. Finalmente, la posición BOTTOM_LEFT no se muestra en la esquina inferior izquierda, sino que, para evitar tapar el logo de Google, lo hace algo más centrada.

```
mapTypeControlOptions: {
      position: google.maps.ControlPosition.BOTTOM_CENTER
}
```

2.1.2.4. Listado de tipos: mapTypeIds

La propiedad "mapTypeIds" controla los diferentes tipos de mapas disponibles para los usuarios. Es posible controlar que tipos de mapa aparecerán como seleccionables. Los mapas disponibles por defecto son:

- MapTypeId.ROADMAP
- MapTypeId.SATELLITE
- MapTypeId.HYBRID
- MapTypeId.TERRAIN

```
var mapOptions = {
	mapTypeControlOptions: {
		mapTypeIds: [
			google.maps.MapTypeId.ROADMAP,
			google.maps.MapTypeId.HYBRID
		]
	}
}
```

2.1.3. Control de zoom: zoomControl

La herramienta de zoom se controla mediante el "zoomControl". Podemos mostrar una barra larga o un mini control más adecuado, por ejemplo, en el caso de pequeños mapas. El "zoomControl" se modifica al alterar de manera apropiada el campo "zoomControlOptions" dentro del objeto "mapOptions". El control del zoom se puede presentar según los siguientes estilos:

- google.maps.ZoomControlStyle.SMALL
- google.maps.ZoomControlStyle.LARGE
- google.maps.ZoomControlStyle.DEFAULT (se adapta según el tamaño del mapa)

```
var mapOptions = {
	zoomControl: true,
	zoomControlOptions: {
		style: google.maps.ZoomControlStyle.SMALL
	},
}
```

2.1.3.1. Zoom con doble clic: disableDoubleClickZoom

Hacer doble clic dentro del mapa equivale a aumentar el zoom. Si deseamos desactivar este comportamiento por defecto, podemos utilizar la propiedad "disableDoubleClickZoom" mediante el valor "true".

```
	disableDoubleClickZoom: true
```

2.1.3.2. Zoom con rueda ratón: scrollwheel

Por defecto, podemos aumentar o disminuir el zoom de nuestro mapa mediante la rueda del ratón. Si queremos desactivar este comportamiento debemos establecer la propiedad "scrollwheel" como "false".

```
	scrollwheel: false
```

2.1.4. Control de desplazamiento: panControl

El "panControl" es el control que nos permite mover la ventana de visualización del mapa de izquierda a derecha, de arriba a abajo y viceversa, y que aparece por defecto en la esquina superior izquierda. Mediante las "panControlOptions", podemos ubicar este control en otras posiciones.

```
var mapOptions = {
	panControl: true,
		panControlOptions: {
		position: google.maps.ControlPosition.TOP_RIGHT
},
```

2.1.4.1. Control de desplazamiento con el cursor: draggable

Por defecto podemos movernos por el mapa, arrastrando el cursor. Si por alguna razón deseamos desactivar este comportamiento debemos establecer como "false" la propiedad "draggable".

```
	draggable: false
```

2.1.5. Teclado: keyboardShortcuts

Esta propiedad permite activar o desactivar la posibilidad de navegar por el mapa con el teclado. Los atajos de teclado disponibles por defecto para navegar

son las cuatro teclas de dirección y las teclas +/- para el zoom. El valor por defecto es true.

```
keyboardShortcuts: false
```

2.1.6. Vista a pie de calle: streetViewControl

La propiedad "streetViewControl" permite activar y desactivar la funcionalidad *Street View* en nuestro mapa.

```
streetViewControl: true
```

También podemos modificar la posición de este control de la misma manera que en casos anteriores:

```
streetViewControl: true,
        streetViewControlOptions: {
        position: google.maps.ControlPosition.RIGHT_TOP
        }
```

2.1.7. Todo junto. Control de la interfaz

```
window.onload = function() {
    var catalunya = new google.maps.LatLng(41.652393,1.691895);
    var mapOptions = {
        center: catalunya,
        zoom: 8,
        zoomControl: true,
            zoomControlOptions: {
                style: google.maps.ZoomControlStyle.SMALL
        },
        keyboardShortcuts: true,
        disableDoubleClickZoom: true,
        draggable: true,
        scrollwheel: false,
        panControl: true,
        panControlOptions: {
```

```
            position: google.maps.ControlPosition.TOP_RIGHT
            },
            streetViewControl: true,
                  streetViewControlOptions: {
                        position: google.maps.ControlPosition.BOTTOM_CENTER
                  },
            mapTypeControl: true,
                  mapTypeControlOptions: {
                        style: google.maps.MapTypeControlStyle.DROPDOWN_MENU,
                        position: google.maps.ControlPosition.TOP_CENTER,
                        mapTypeIds: [
                              google.maps.MapTypeId.ROADMAP,
                              google.maps.MapTypeId.HYBRID
                        ]
      }
};
map = new google.maps.Map(document.getElementById('mapa'), mapOptions);
}
```

Vista del mapa

Mapa con diferentes elementos de la interfaz cambiados de ubicación y apariencia.

2.2. Propiedades de control sobre el contenedor del mapa

El contenedor del mapa es el elemento que contiene nuestro mapa dentro de la página HTML. El objeto "MapOptions" dispone de algunas propiedades que nos permiten cambiar el comportamiento de este contenedor.

2.2.1. Borrar contenido anterior: noClear

Por defecto, cuando el mapa se carga en la página HTML, éste borra cualquier otro contenido ubicado en el contenedor. Si por alguna razón nos interesase mantener ese otro contenido, podemos utilizar la propiedad "noClear" con el valor "true".

```
var mapOptions = {
	noClear: true
};
```

2.2.2. Color de fondo: backgroundColor

La propiedad "backgroundColor" también afecta al contenedor del mapa. En este caso podemos definir el color de fondo del contenedor. El valor de esta propiedad puede ser un valor hexadecimal o alguno de los colores básicos en HTML y CSS (red, green, yellow, etc.). Cuando cargamos la página HTML es posible ver durante el tiempo de carga el color de fondo seleccionado.

```
var mapOptions = {
	backgroundColor: '#000'
};
```

2.2.3. Todo junto: Control contenedor

```
window.onload = function() {
	var catalunya = new google.maps.LatLng(41.652393,1.691895);
	var mapOptions = {
		center: catalunya,
		zoom: 8,
		zoomControl: true,
			zoomControlOptions: {
				style: google.maps.ZoomControlStyle.SMALL
		},
		noClear: true,
		backgroundColor: '#000000',
		mapTypeControl: true,
			mapTypeControlOptions: {
				style: google.maps.MapTypeControlStyle.DROPDOWN_MENU,
				position: google.maps.ControlPosition.TOP_RIGHT,
				mapTypeIds: [
					google.maps.MapTypeId.ROADMAP,
					google.maps.MapTypeId.HYBRID
				]
	}
};
map = new google.maps.Map(document.getElementById('mapa'), mapOptions);
}
```

2.3. Propiedades de control sobre el cursor

Las propiedades de control sobre el cursor nos permiten controlar la apariencia de este elemento en determinadas situaciones.

2.3.1. Aspecto del cursor de objeto: draggableCursor

Con esta propiedad es posible controlar la apariencia del cursor cuando se sitúa sobre algún objeto "arrastable" dentro del mapa. Podemos establecer el aspecto del cursor mediante alguno de los iconos definidos en la API de Google Maps, o bien con una imagen propia, que referenciaremos con su URL.

auto	n/a
crosshair	
default	
e-resize	
help	
move	
n-resize	
ne-resize	
nw-resize	
pointer	
s-resize	
se-resize	
sw-resize	
text	
url(*url*)	n/a
w-resize	
wait	

Algunos de los iconos disponibles para establecer el aspecto del cursor.

```
var mapOptions = {
      draggableCursor: 'crosshair'
};
```

2.3.2. Aspecto del cursor de arrastre: draggingCursor

La propiedad "draggingCursor" es similar a la anterior. La diferencia es que, en este caso, el cursor sólo cambia cuando arrastramos un objeto en el mapa, no al situarnos encima.

```
var mapOptions = {
      draggingCursor: 'crosshair'
};
```

2.4. Accesibilidad de los controles de la interfaz

La imposibilidad de utilizar los diferentes controles del mapa mediante el teclado provoca que la mayoría de los mapas creados con la API de Google Maps no sean accesibles. El problema radica en que los controles del mapa están creados con elementos que no pueden recibir el foco. Existen dos alternativas para solucionar este problema de accesibilidad. La primera de ellas consiste en sacar fuera del mapa los diferentes controles de la interfaz, utilizando elementos que si puedan captar el foco (button, etc.) para ubicarlos en el código HTML. La segunda de las opciones pasa por insertar en el código HTML generado por Google Maps, es decir, dentro del mismo mapa, elementos que si capten el foco en lugar de los generados por defecto.

2.4.1. Ubicar los controles fuera del mapa: el selector de tipo de mapa

Ahora que conocemos los diferentes controles que actúan sobre la interfaz de Google Maps, vamos a crear unos botones accesibles fuera del mapa que nos permitan interactuar con la aplicación. Anteriormente, hemos visto que la propiedad "mapTypeIds" controla los diferentes tipos de mapas disponibles en la API. Para cambiar esta propiedad de manera dinámica utilizaremos la función:

```
map.setMapTypeId(MapTypeId-al-que-cambiará);
```

La función la asociaremos a un evento de teclado (onkeypress) y uno de clic (onclick) dentro de un botón en el código HTML. Así, por ejemplo, para cambiar el tipo de mapa a "Terrain" utilizaremos el siguiente código:

```
<button onkeypress="map.setMapTypeId(google.maps.MapTypeId.TERRAIN);"
onclick="map.setMapTypeId(google.maps.MapTypeId.TERRAIN);">TERRAIN</button>
```

2.4.1.1. Código de ejemplo

Javascript

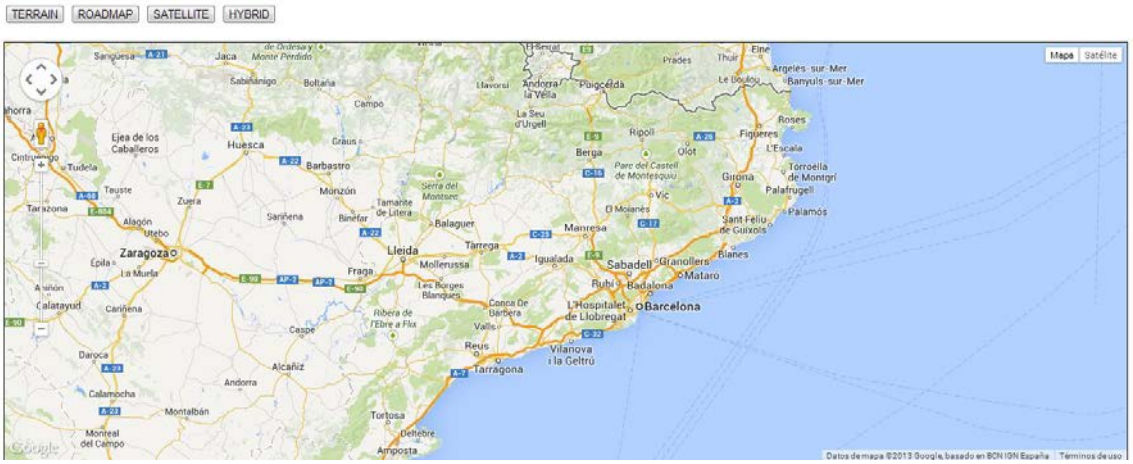
```
window.onload = function() {
	var mapOptions = {
		center:new google.maps.LatLng(41.652393,1.691895),
		zoom:8,
		mapTypeId:google.maps.MapTypeId.ROADMAP
	};
	map = new google.maps.Map(document.getElementById("mapa"),mapOptions);
}
```

HTML

```
<body>
<button onkeypress="map.setMapTypeId(google.maps.MapTypeId.TERRAIN);"
onclick="map.setMapTypeId(google.maps.MapTypeId.TERRAIN);">TERRAIN</button>
<button onkeypress="map.setMapTypeId(google.maps.MapTypeId.ROADMAP);"
onclick="ap.setMapTypeId(google.maps.MapTypeId.ROADMAP);">ROADMAP</button>
<button onkeypress="map.setMapTypeId(google.maps.MapTypeId.SATELLITE);"
onclick="map.setMapTypeId(google.maps.MapTypeId.SATELLITE);">SATELLITE</butto
n>
<button onkeypress="map.setMapTypeId(google.maps.MapTypeId.HYBRID);"
onclick="map.setMapTypeId(google.maps.MapTypeId.SATELLITE);">HYBRID</button>
```

```
<br><br>

<div id="mapa"></div>

</body>
```

Vista del mapa



Controles de tipo de mapa fuera de la interfaz del mapa.

2.4.2. Ubicar los controles fuera del mapa: el control de zoom

Para poder llevar fuera del mapa el control de zoom, en primer lugar debemos utilizar el método "map.getZoom();" para que la aplicación pueda determinar cuál es el nivel de zoom actual en el mapa. Para ello podemos utilizar una función como:

```
function Zoomactual() {
        return map.getZoom();
}
```

El siguiente paso es crear otras dos funciones a las que llamaremos "subirZoom" y "bajarZoom" que cogerán como argumento el "nivel" de zoom. Las variables "maszoom" y "menoszoom" almacenarán el valor resultado del método "map.getZoom()" (es decir, el zoom actual del mapa) y le sumarán el valor del argumento "nivel" de sendas funciones.

```
function subirZOOM(nivel) {
	var maszoom = Zoomactual() + nivel;
	map.setZoom(maszoom);
}
function bajarZOOM(nivel) {
	var menoszoom = Zoomactual() - nivel;
	map.setZoom(menoszoom);
}
```

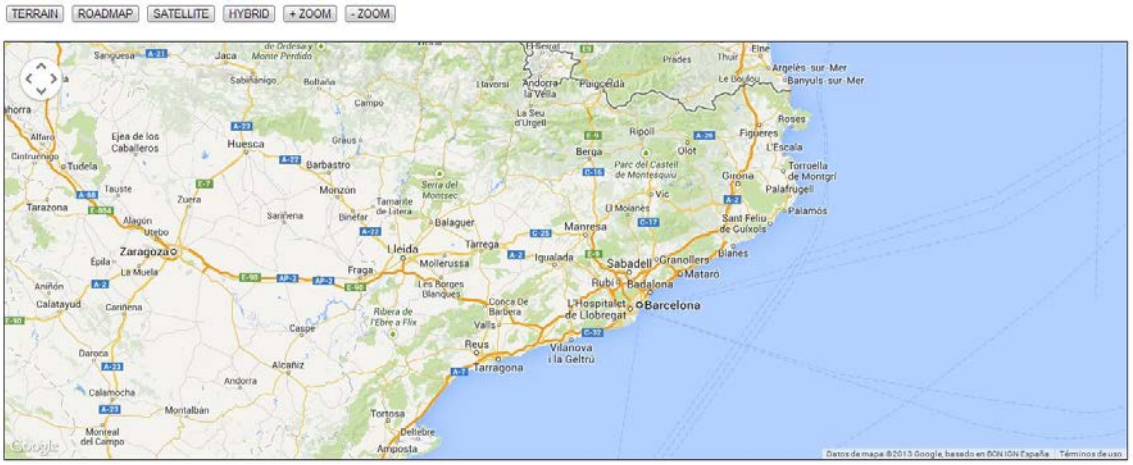
Para que funcione correctamente, en el código HTML debemos indicar el valor del argumento "nivel". Este valor determinará el incremento o disminución del nivel de zoom. Así, por ejemplo, si el valor del argumento "nivel" es 1, cada vez que pulsemos sobre el botón +Zoom o -Zoom la función se encargará se subir o bajar el nivel del zoom en 1 punto respecto al valor de zoom actual del mapa.

```
<button onkeypress="subirZOOM(1)" onclick="subirZOOM(1)">+ ZOOM</button>
<button onkeypress="bajarZOOM(1)" onclick="bajarZOOM(1)">- ZOOM</button>
```

Todo junto

```
window.onload = function() {
	var mapOptions = {
		center:new google.maps.LatLng(41.652393,1.691895),
		zoom:8,
		mapTypeControl: false,
		zoomControl: false,
		panControl: true,
		streetViewControl: false,
		mapTypeId:google.maps.MapTypeId.ROADMAP
	 };
	map = new
	google.maps.Map(document.getElementById("mapa"),mapOptions);
}
function Zoomactual() {
	return map.getZoom();
}
function subirZOOM(nivel) {
	var maszoom = Zoomactual() + nivel;
	map.setZoom(maszoom);
}
function bajarZOOM(nivel) {
	var menoszoom = Zoomactual() - nivel;
	map.setZoom(menoszoom);
}
```

Vista del mapa



Controles de tipo de mapa y control de zoom fuera de la interfaz del mapa.

2.4.3. Hacer accesibles los controles dentro del mapa

Para hacer accesibles los controles internos del mapa, debemos generar elementos que sean capaces de recibir el foco, en lugar de los que se generan automáticamente. En este caso utilizaremos el elemento "button".

En primer lugar, debemos crear una nueva capa sobre el objeto "mapa", mediante una variable.

```
var capaAccesible = 'mapa'
```

A continuación, crearemos otras dos variables para los elementos "button" y para su estilo "button_estilo", a los que les asociaremos el estilo que deseamos.

```
var button, button_estilo =
'width:100%;height:100%;padding:2px;margin:2px;background:transparent;
border-width:0px;border-
style:solid;cursor:pointer;overflow:hidden;text-indent:-
100em;position:absolute;top:-2px;left:-2px;';
```

Enuna nueva variable ("divs") almacenaremos los resultados de los métodos "document.getElementById()" y "document.getElementsByTagName()" para que obtener el elemento con el id "CapaAccesible" y la lista entera (array) de elementos "div".

```
var divs = document.getElementById(capaAccesible).getElementsByTagName('div');
```

A continuación, crearemos un bucle *for*[2] para recorrer los diferentes elementos de la variable "divs".

```
for (var i = 0; i < (divs.length); i++) {
}
```

Utilizaremos una estructura condicional *if* dentro del bucle para obtener el valor de cada uno de los atributos de los diferentes , en concreto el "title". Si la condición se cumple, recuperamos el "title" de ese elemento.

```
if ( divs[i].getAttribute('title')) {
}
```

Dentro de la estructura condicional crearemos una variable llamada "button" asociada al método "document.createElement()", mediante el cual crearemos para cada un elemento de tipo "button".

```
button = document.createElement("button");
```

Utilizaremos el método "setAttribute()" para agregar nuevos atributos al elemento "button", aprovechando los valores de i y del atributo "title" recuperado. También añadimos el atributo "tabindex" para ordenar la tabulación, evitando así que los

[2] En el apartado 3.2.2, se explican con más detalle los bucles y las estructuras condicionales.

primeros elementos recuperados sean el logotipo de Google y el enlace a los términos de uso.

```
button.setAttribute('id','boton-mapa-'+i);
button.setAttribute('tabindex', 0+i);
button.setAttribute('value',divs[i].getAttribute('title'));
```

Asociaremos los estilos definidos anteriormente con nuestros nuevos elementos "button".

```
// Para la mayoría de navegadores
button.setAttribute('style',button_estilo);
// Para IE
button.style.cssText = button_estilo;
```

Finalmente, mediante el método "appendChild()" agregaremos los elementos "button" al mapa.

```
divs[i].appendChild(button);
```

Toda la función debe llamarse con el método "setTimeout" para que el navegador tenga tiempo de cargar el mapa.

Todo junto:

```
window.onload = function() {
var catalunya = new google.maps.LatLng(41.652393,1.691895);
	// Iniciar Google Maps
	var mapDiv = document.getElementById('mapa');
	var myOptions = {
		center: catalunya,
		zoom: 8,
		streetViewControl: false,
		mapTypeId: google.maps.MapTypeId.ROADMAP
		}
	map = new google.maps.Map(document.getElementById("mapa"), myOptions);
	setTimeout('gmaps_teclado();',500);
}
function gmaps_teclado() {
	var capaAccesible = 'mapa' // crear una capa sobre el objeto "mapa"
	var button, button_estilo =
'width:100%;height:100%;padding:2px;margin:2px;background:transparent;border-
width:0px;border-style:solid;cursor:pointer;overflow:hidden;text-indent:-
100em;position:absolute;top:-2px;left:-2px;';
	var divs =
document.getElementById(capaAccesible).getElementsByTagName('div');
	for (var i = 0; i < (divs.length); i++) {
		if ( divs[i].getAttribute('title')) {
			// Crear el elemento botón
			button = document.createElement("button");
			button.setAttribute('id','boton-mapa-'+i);
			button.setAttribute('tabindex', 0+i);
```

```
            button.setAttribute('value',divs[i].getAttribute('title'));

                        // estilos

                        button.setAttribute('style',button_estilo); // Para la
mayoría                                 de navegadores

                        button.style.cssText = button_estilo; // Para IE

                        // Se añade el button

                        divs[i].appendChild(button);

                }

        }

}
```

Vista del mapa

En la imagen se observa como el Pan Control recibe el foco.

3. Indicar un punto de interés (POI)

Para crear un marcador que señalice un punto de interés (POI) es necesario utilizar el objeto "google.maps.Marker". El parámetro "google.maps.MarkerOptions" tiene diversas propiedades que podemos utilizar para modificar el aspecto y comportamiento del marcador. Las dos únicas propiedades obligatorias son: "position" y "map".

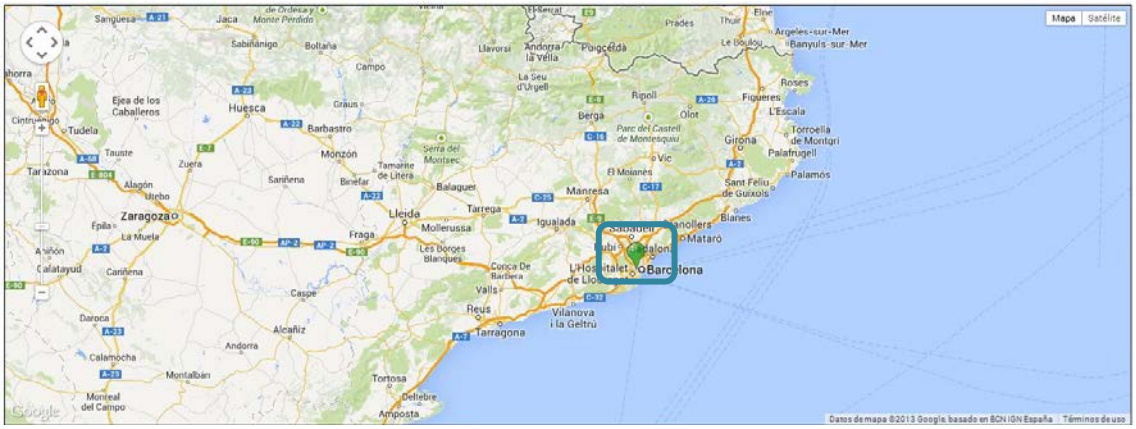
- **Position:** esta propiedad define las coordenadas en las que el marcador será ubicado. El argumento a utilizar es idéntico al del objeto google.maps.LatLng.
- **Map:** esta propiedad es una referencia al mapa en el que deseamos añadir el marcador.
- **Title:** Muestraun texto al pasar el cursor sobre el marcador.
- **Icon:** Permite utilizar imágenes personalizadas para los iconos que representan los POI. Es posible utilizar los iconos creados por Google[3] o una imagen propia que referenciaremos con su URL.

Ejemplo

```
var marker = new google.maps.Marker({
      position: new google.maps.LatLng(41.381269, 2.138956),
      map: map,
      title: 'Facultat de Biblioteconomia',
      icon: 'iconos/green.png'
});
```

[3] http://gmaps-samples.googlecode.com/svn/trunk/markers

Vista del mapa



Mapa con un POI para el cual se ha utilizado un icono personalizado.

3.1. Añadir más información a los marcadores: los InfoWindow

Si necesitamos mostrar más información acerca del marcador que hemos creado, podemos utilizar un "InfoWindow", una especie de bocadillo que aparece al pulsar sobre el marcador en cuestión. De la misma manera que el objeto "Marker", el objeto "InfoWindow" reside en el *namespace* de "google.maps", y presenta un único argumento, el objeto "InfoWindowOptions". Este objeto tiene diferentes propiedades. La más importante de ellas es la propiedad "content", el valor de la cual será mostrada dentro del bocadillo. El valor de la propiedad "content" puede ser texto plano o código HTML.

```
var infowindow = new google.maps.InfoWindow({
      content:'Facultat de Biblioteconomia i Documentaci&oacute;'
});
```

Una vez creado el objeto "InfoWindow", tenemos que conectarlo con el marcador para que al pulsar sobre él, el bocadillo aparezca. Para ello añadiremos un evento de clic al marcador. El controlador del evento "google.maps.events.addListener()" nos permite asociar un determinado evento a un objeto dentro del DOM, y a una función que se ejecuta cuando se activa el evento.

```
google.maps.event.addListener(objeto, 'evento', función()
google.maps.event.addListener(marker, 'click', function()
```

El objeto google.maps.Marker es compatible con los siguientes eventos:

- 'click'
- 'dblclick'
- 'mouseup'
- 'mousedown'
- 'mouseover'
- 'mouseout'

A continuación, debemos llamar al método "open" del objeto "InfoWindow".

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

Todo junto

```
window.onload = function() {
	var catalunya = new google.maps.LatLng(41.652393,1.691895);
	var mapOptions = {
		center: catalunya,
		zoom: 8,
		mapTypeId: google.maps.MapTypeId.ROADMAP
	};
map = new google.maps.Map(document.getElementById('mapa'), mapOptions);

	var marker = new google.maps.Marker({
			position: new google.maps.LatLng(41.381269, 2.138956),
			map: map,
			title: 'Facultat de Biblioteconomia',
			icon: 'iconos/green.png'
		});

	var infowindow = new google.maps.InfoWindow({
		content:'<h1>Facultat de Biblioteconomia i
	Documentaci&oacute;</h1><p>Adre&ccedil;a: Carrer de Melcior de Palau,
140, 08014   Barcelona.</p><p>Tel&egrave;fon: 934 03 57 70.</p><img
```

```
src="imagenes/bid.png"    alt="Facultat de Biblioteconomia i
Documentaci&oacute;">'
        });

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

Vista del mapa

InfoWindow asociado a nuestro marcador.

3.2. Añadir varios puntos de interésa un mapa

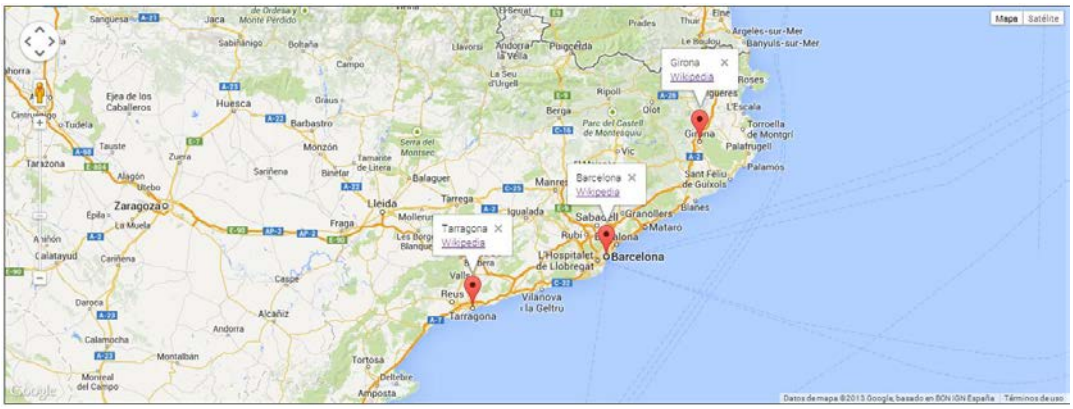
De nuevo, necesitamos utilizar el método "google.maps.events.addListener()" para ejecutar una función en el momento en que se active un evento asociado a los marcadores. En este caso no podemos utilizar el mismo código que cuando sólo teníamos un único marcador en el mapa. Si lo hacemos, al pulsar sobre cualquiera de los marcadores se abrirá el "InfoWindow" del marcador creado en último lugar, y el texto que aparecerá tampoco coincidirá con el que le tocaría según el orden establecido. A continuación, vemos como solucionarlo.

3.2.1. Marcadores e Infowindows independientes

Para generar los marcadores, una posibilidad consiste en definir una variable diferente para cada marcador y cada "InfoWindow", que posteriormente se relacionará mediante el método "open" desde cada "InfoWindow".

```
// marcador 1
var marker1 = new google.maps.Marker({
position: new google.maps.LatLng(41.385064, 2.173404),
map: map,
    });

    // InfoWindow para el marcador 1
var infowindow1 = new google.maps.InfoWindow({
content: 'Barcelona<br /><a
href="http://es.wikipedia.org/wiki/Barcelona"
       target="_blank">Wikipedia</a>'
    });

    // Añadimos un evento de clic al marcador

google.maps.event.addListener(marker1, 'click', function() {

// Llamamos el método open del InfoWindow
infowindow1.open(map, marker1);
    });
    // final del marcador 1

// marcador 2
var marker2 = new google.maps.Marker({
...
```

Vista del mapa



Varios marcadores con sus respectivos infoWindows.

3.2.2. Uso de vectores

La otra posibilidad consiste en la utilización de vectores. Un vector es básicamente una secuencia de valores para una variable. También necesitaremos utilizar un bucle. Existen dos tipos de bucles en JavaScript. Los que se ejecutan un determinado número de veces, llamados bucles *for*, y los que se ejecutan mientras una determinada condición sea verdadera, o bucles *while*. En este caso utilizaremos un bucle *for* para ejecutar el mismo código varias veces, pero con diferentes datos en cada ocasión.
En primer lugar, crearemos un vector que contendrá el título, las coordenadas y un texto que usaremos en los "infoWindow" de cada uno de los POI.

```
var marcadores = [
            ['Barcelona',41.385064,2.173404,'Barcelona'],
            ['Tarragona',41.119019,1.245212,'Tarragona'],
            ['Girona',41.9794,2.821426,'Girona'],
            ['Lleida',41.60034,0.609762,'Lleida'],
];
```

Una vez tenemos el vector con los datos que necesitamos para crear los marcadores, tenemos que extraer esos datos del vector mediante un bucle:

```
for (var i = 0; i < marcadores.length; i++) {
       var myLatLng = new google.maps.LatLng(marcadores[i][1],
       marcadores[i][2]);
       var marker = new google.maps.Marker({
       position: myLatLng,
       map: map,
       title: marcadores[i][0],
 });
```

El código anterior recorre el vector y crea un marcador en cada nueva iteración. El valor de i se inicializa con 0, en tanto que la condición se verifica como verdadera (i es menor que el número de "marcadores") se ejecuta el bloque del *for*:

- Obtenemos las coordenadas almacenadas para el primer marcador (var myLatLng = new google.maps.LatLng(marcadores[i][1], marcadores[i][2]);) que después pasaremos a "Position".
- Asociamos el marcador a nuestro mapa (map: map).
- Mostramos como "title" de nuestro marcador el primer valor de cada elemento del vector. Para acceder al primer elemento escribiremos el nombre del vector (marcadores) y la posición en el índice del valor al que deseamos acceder. En este caso al ser el primer valor es la posición 0 (marcadores[i][0],).

Después de ejecutarse el bloque del *for*, el bucle pasa a su tercer argumento. En este caso el operador ++ incrementando en 1 el valor de la variable i, y el bucle vuelve a empezar. El proceso acaba cuando i es superior a 4, momento en que la condición deja de ser verdadera.

Ahora necesitamos crear un "infoWindow" global con acceso al último de los valores del vector, que si recordamos era el texto que deseábamos que apareciera en el bocadillo.

```
google.maps.event.addListener(marker,'click',function() {
          if (!infowindow) {
                infowindow = new google.maps.InfoWindow();
                }
          infowindow.setContent(marcadores[i][3]);
          infowindow.open(map, marker);
    });
```

En el código anterior, creamos el "InfoWindow" y lo conectamos con su correspondiente marcador para que, al pulsar sobre él, el bocadillo aparezca. Como en el apartado 3.1, lo hacemos mediante el método "google.maps.events.addListener()".

```
google.maps.event.addListener(marker,'click',function() {
      if (!infowindow) {
            infowindow = new google.maps.InfoWindow();
      }
```

A continuación, utilizamos el método "infowindow.setContent" para indicar cuál es el valor que debe mostrar. En este caso, el cuarto valor de cada vector (posición 3 en el índice).

```
infowindow.setContent(marcadores[i][3]);
```

Para acabar, llamamos el método "open" del objeto "InfoWindow".

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

Para evitar que el último valor de "i" al finalizar el bucle sea el valor asignado a todos los marcadores por el *event handler*, debemos aislar la "i" de esta función, de la "i" del bucle, usando un *closure*. Así, pondremos el *event listener* dentro de una función anónima, pasándole las variables del bucle, de "i", del marcador y sus parámetros. Los cambios en la "i" del bucle no afectarán a los valores ya asignados a los infoWindows creados dentro del *event handler*.

```
		(function(i, marker) {
			//código
		})(i, marker);
```

Todo junto

```
window.onload = function() {
function initialize() {
		var latlng = new google.maps.LatLng(41.652393,1.691895);
	var mapOptions = {
		zoom: 8,
		center: latlng,
		mapTypeId: google.maps.MapTypeId.ROADMAP,
		}
		var map = new google.maps.Map(document.getElementById('mapa'),
	mapOptions);
		setMarkers(map, marcadores);
	}

	var marcadores = [
			['Barcelona',41.385064,2.173404,'Barcelona'],
			['Tarragona',41.119019,1.245212,'Tarragona'],
			['Girona',41.9794,2.821426,'Girona'],
			['Lleida',41.60034,0.609762,'Lleida'],
	];

	var infowindow;
	function setMarkers(map, marcadores) {

		for (var i = 0; i < marcadores.length; i++) {
			var myLatLng = new google.maps.LatLng(marcadores[i][1],
	marcadores[i][2]);
			var marker = new google.maps.Marker({
				position: myLatLng,
				map: map,
				title: marcadores[i][0],
			});
			(function(i, marker) {
				google.maps.event.addListener(marker,'click',function() {
					if (!infowindow) {
						infowindow = new google.maps.InfoWindow();
```

```
                                    }
                                    infowindow.setContent(marcadores[i][3]);
                            infowindow.open(map, marker);
                            });
                    })(i, marker);
            }
    };
    initialize();}
```

3.2.3. Vectores con JSON

Empezamos por los datos en formato JSON. Creamos un vector con una colección de pares nombre/valor, que nos proporcionarán el título, latitud, longitud y descripción (texto del InfoWindow) de cada uno de los marcadores.

```
var json = [
    {
    "title": "Barcelona",
    "lat": 41.385064,
    "lng": 2.173404,
    "description": "<h1>Barcelona</h1><p>Lorem ipsum dolor sit amet,
    consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et
    dolore magna aliqua.</p>"
    },
    {
    "title": "Girona",
    "lat": 41.9794,
    "lng": 2.821426,
    "description": "<h1>Girona</h1><p>Lorem ipsum dolor sit amet,
    consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et
    dolore magna aliqua.</p>"
    },
    {
    "title": "Tarragona",
    "lat": 41.119019,
    "lng": 1.245212,
    "description": "<h1>Tarragona</h1><p>Lorem ipsum dolor sit amet, consectetur
    adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
    aliqua.</p>"
    },
    {
```

```
        "title": "Lleida",
        "lat": 41.60034,
        "lng": 0.609762,
        "description": "<h1>Lleida</h1><p>Lorem ipsum dolor sit amet, consectetur
        adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
        aliqua.</p>"
        }
]
```

Creamos un nuevo mapa con centro en Catalunya.

```
var map = new google.maps.Map(document.getElementById("mapa"), {
center: new google.maps.LatLng(41.652393, 1.691895),
zoom: 8,
mapTypeId: google.maps.MapTypeId.ROADMAP
});
```

Para crear los marcadores a partir de los datos del fichero JSON, lo recorremos con un bucle *for*, con el objetivo de extraer la información de cada marcador. La variable "marker" agregará los marcadores al mapa y a cada uno de ellos se le asignará su propio título a partir de la información contenida en el fichero JSON ("data.title").

```
for (var i = 0, length = json.length; i < length; i++) {
        var data = json[i],
        latLng = new google.maps.LatLng(data.lat, data.lng);

        var marker = new google.maps.Marker({
                position: latLng,
                map: map,
                title: data.title
        });
```

Para asignar a cada marcador su propio "InfoWindow", en primer lugar, debemos crear un "InfoWindow" global vacío que reutilizaremos para todos los marcadores. Se ha de crear fuera del bucle anterior, justo antes.

```
var infoWindow = new google.maps.InfoWindow();
```

Ahora utilizamos un evento de clic para cada marcador. El código que se ejecuta toma el valor de cada nombre "description" del JSON para llenar el "InfoWindow" correcto.

```
google.maps.event.addListener(marker, "click", function(e) {
      infoWindow.setContent(data.description);
      infoWindow.open(map, marker);
  });
```

Como en el apartado anterior, debemos utilizar un *clousure* para evitar que todos los "InfoWindow" recojan el contenido de la última iteración del bucle. Encapsulamos el código anterior dentro de una función anónima.

```
      (function(marker, data) {

      google.maps.event.addListener(marker, "click", function(e) {
      infoWindow.setContent(data.description);
      infoWindow.open(map, marker);
        });

      })(marker, data);
```

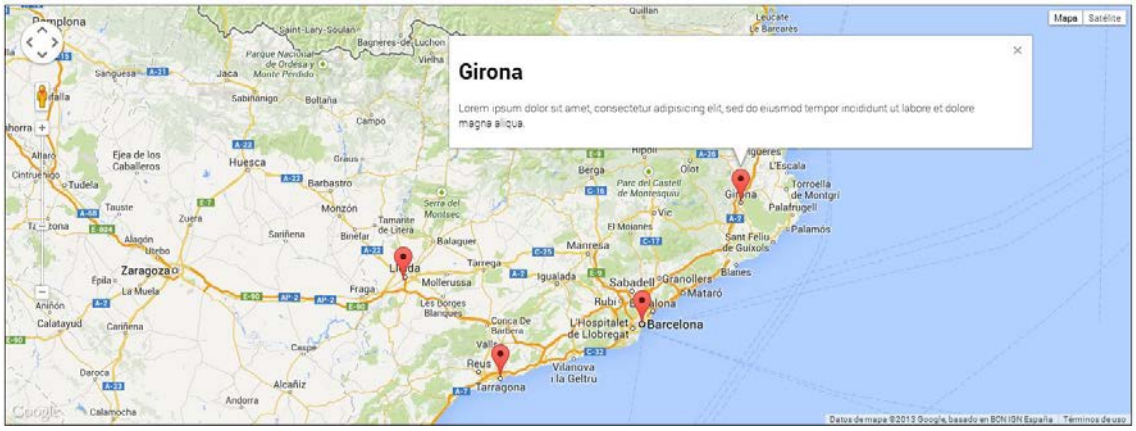
Todo junto

```
      window.onload = function() {

            var json = [
                {
                    "title": "Barcelona",
                    "lat": 41.385064,
            "lng": 2.173404,
                   "description": "<h1>Barcelona</h1><p>Lorem ipsum dolor sit amet,
                   consectetur adipisicing elit, sed do eiusmod tempor incididunt
                   ut labore et dolore magna aliqua.</p>"
                },
                {
```

```
                    "title": "Girona",
                    "lat": 41.9794,
                    "lng": 2.821426,
                   "description": "<h1>Girona</h1><p>Lorem ipsum dolor sit amet,
                   consectetur adipisicing elit, sed do eiusmod tempor incididunt
                   ut labore et dolore magna aliqua.</p>"
               },
               {
                    "title": "Tarragona",
                    "lat": 41.119019,
                    "lng": 1.245212,
                   "description": "<h1>Tarragona</h1><p>Lorem ipsum dolor sit amet,
                   consectetur adipisicing elit, sed do eiusmod tempor incididunt
                   ut labore et dolore magna aliqua.</p>"
               },
                   {
                    "title": "Lleida",
                    "lat": 41.60034,
                    "lng": 0.609762,
                   "description": "<h1>Lleida</h1><p>Lorem ipsum dolor sit amet,
                   consectetur adipisicing elit, sed do eiusmod tempor incididunt
                   ut labore et dolore magna aliqua.</p>"
               }
           ]

      var map = new google.maps.Map(document.getElementById("mapa"), {
center: new google.maps.LatLng(41.652393, 1.691895),
zoom: 8,
mapTypeId: google.maps.MapTypeId.ROADMAP
       });
            var infoWindow = new google.maps.InfoWindow();

            for (var i = 0, length = json.length; i < length; i++) {
                var data = json[i],
                        latLng = new google.maps.LatLng(data.lat, data.lng);

                var marker = new google.maps.Marker({
                        position: latLng,
                        map: map,
                        title: data.title
                });

                (function(marker, data) {
```

```
                        google.maps.event.addListener(marker, "click",
function(e) {
                                infoWindow.setContent(data.description);
                                infoWindow.open(map, marker);
                        });
                    })(marker, data);
            }
        }
```

Vista del mapa



Mapa con marcadores y infoWindows obtenidos de un vector JSON.

3.2.4. Importar información de los marcadores desde un fichero JSON externo

Nota: Algunas restricciones en el uso de consultas HTTP en local con Chrome y Opera, provocan que este ejemplo no funcione con estos navegadores. La alternativa es utilizar Firefox, Safari o un servidor HTTP como Apache[4].

Para este ejemplo utilizaremos la librería JQuery. jQuery es una librería de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el DOM, manejar eventos, desarrollar animaciones y agregar interacción mediante AJAX a páginas web. jQuery, al igual que otras bibliotecas,

[4] http://www.apachefriends.org/es/xampp.html

ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta librería se logran grandes resultados en menos tiempo y espacio.

Para utilizar Jquery debemos hacer una referencia desde nuestro archivo HTML hacia el lugar en el que se encuentra esa biblioteca (en un directorio en nuestro servidor, o en Internet). La referencia se ha de incluir en la sección <head> del documento. La cargaremos con el elemento <script>. Este elemento tiene dos atributos que debemos utilizar. El primero es el tipo de script que deseamos utilizar y el otro es el URL que apunta a la API.

```
<script type="text/javascript" src="js/jquery.min.js"></script>
o
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.min.js"></scri
pt>
```

En esta ocasión, la información en formato JSON estará en un archivo independiente al código javascript.

```
[{
      "nombre": "Ateneu Barcelonès",
      "coords": {
        "lng": 2.1716089115143404,
        "lat": 41.3846241973006
    }
}, {
      "nombre": "Biblioteca Arús",
      "coords": {
        "lng": 2.1786953077316866,
        "lat": 41.39295110192035
    }
}, {
      "nombre": "Centre Excursionista de Catalunya",
      "coords": {
        "lng": 2.177043066978513,
```

```
        "lat": 41.38346100200511
    }
}]
```

El archivo con nombre "bibliotecas.json" estará ubicado en el directorio "json", al que haremos referencia posteriormente desde nuestro fichero javascript mediante el método "getJSON()". Este método carga un archivo en formato JSON de nuestro servidor usando una petición de tipo GET HTTP.

```
$.getJSON('json/bibliotecas.json',insertar_marcador);
```

Para recorrer el archivo JSON e insertar los marcadores en el mapa, usaremos el método ".each()". Este método realiza una iteración sobre un objeto JQuery, ejecutando una función para cada uno de los elementos (en este caso marcadores).

```
var insertar_marcador = function (data){
            $.each(data, function(i, obj){
                  var marcador = new google.maps.Marker({
                        title: obj.nombre,
                        position : new

      google.maps.LatLng(obj.coords.lat,obj.coords.lng),
                        map : map,
                  });
            });
```

Como se puede observar en el código anterior, las propiedades de cada marcador se extraen de los valores de cada elemento del fichero JSON ("obj.nombre", obj.coords.lat", etc.). Finalmente, usaremos la función ".push()" para añadir los marcadores.

```
            marcadores.push(marcador);
```

Todo junto

```
window.onload = function() {
	var catalunya = new google.maps.LatLng(41.387325,2.17332);
	var opciones = {
	zoom : 12,
	center: catalunya,
	mapTypeId: google.maps.MapTypeId.ROADMAP
	};
	var div = document.getElementById('mapa');
	var map = new google.maps.Map(div, opciones);
	var marcadores = [];

	var insertar_marcador = function (data){
		$.each(data, function(i, obj){
			var marcador = new google.maps.Marker({
				title: obj.nombre,
				position : new

	google.maps.LatLng(obj.coords.lat,obj.coords.lng),
				map : map,
			});
			marcadores.push(marcador);
		});
	};

	$.getJSON('json/bibliotecas.json',insertar_marcador);
}
```

3.3. Crear listas con los marcadores del mapa

Una de las maneras de conseguir que los marcadores del mapa puedan tener el foco y así ser accesibles, es disponer de una lista externa al mapa con los diferentes puntos de interés.

Para crear los enlaces que apuntarán a los marcadores del mapa usaremos el método "trigger()", que se encargará de activar el evento "onkeypress" asociado al primer valor del vector de cada marcador.

Código HTML

```
<a href="javascript:google.maps.event.trigger(markers['Ateneu'],'onkeypress');">Ateneu
Barcelon&egrave;s</a>
```

Código JavaScript

```
for (var i = 0; i < marcadores.length; i++) {
       markers[marcadores[i][0]] =
       createMarker(new google.maps.LatLng(marcadores[i][2],
       marcadores[i][3]), marcadores[i][0] + "<br>" + marcadores[i][1]);
}
```

Todo junto

Javascript

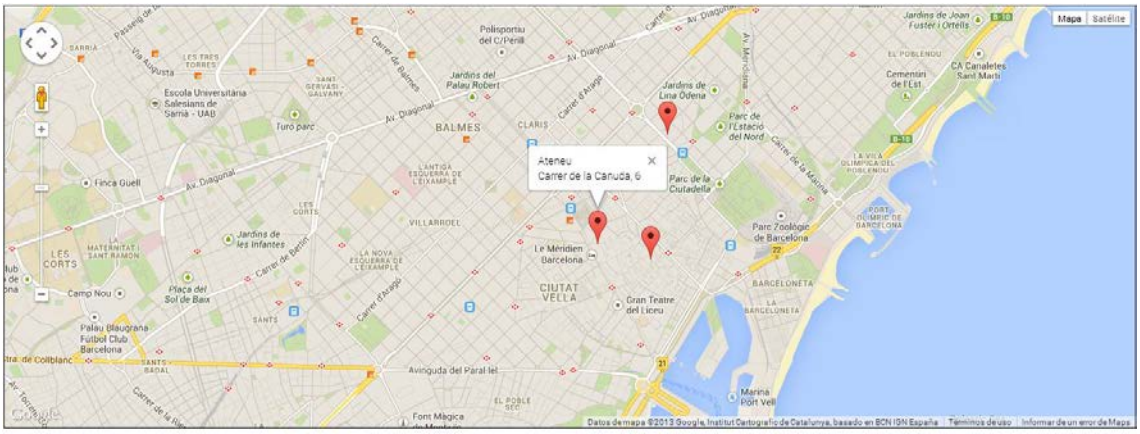
```
window.onload = function() {

var marcadores = [
    ["Ateneu","Carrer de la Canuda, 6","41.3846241973006","2.1716089115143404"],
    ["CEC","Carrer Parad&iacute;s, 10-12","41.38346100200511","2.177043066978513"],
    ["Biblioteca Arus","Passeig de Sant Joan,
26","41.39295110192035","2.1786953077316866"]
    ];

markers = [];

var map = new google.maps.Map(document.getElementById('mapa'), {
zoom: 14,
center: new google.maps.LatLng(41.38210861038204, 2.1690608129501925),
mapTypeId: google.maps.MapTypeId.ROADMAP
});

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


function createMarker(latlng, html) {
var marker = new google.maps.Marker({
position: latlng,
map: map
    });
```

```
google.maps.event.addListener(marker, 'onkeypress', function() {
infowindow.setContent(html);
infowindow.open(map, marker);
    });
return marker;
}

for (var i = 0; i < marcadores.length; i++) {
markers[marcadores[i][0]] =
createMarker(new google.maps.LatLng(marcadores[i][2], marcadores[i][3]),
marcadores[i][0] + "<br>" + marcadores[i][1]);
}

};
```

HTML

```
<!DOCTYPE html>
<html lang="es">
       <head>
              <meta charset="utf-8">
              <link type="text/css" href="css/estilo.css" rel="stylesheet"
media="all" />
              <script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
              <script type="text/javascript" src="js/mapa.js"></script>
              <title>Lista externa con los marcadores</title>
       </head>
       <body>

       <div id="mapa"></div>
       <ul>
<li><a
href="javascript:google.maps.event.trigger(markers['Ateneu'],'onkeypress');">Ateneu
Barcelon&egrave;s</a></li>
<li><a
href="javascript:google.maps.event.trigger(markers['CEC'],'onkeypress');">Centre
Excursionista de Catalunya</a></li>
<li><a href="javascript:google.maps.event.trigger(markers['Biblioteca
Arus'],'onkeypress');">Biblioteca Ar&uacute;s</a></li>
</ul>
       </body>
</html>
```

Vista del mapa



- Ateneu Barcelonès
- Centre Excursionista de Catalunya
- Biblioteca Arús

En la parte inferior del mapa se observa la lista con acceso a los marcadores.

4. Servicio de rutas y direcciones

4.1. Indicar una ruta desde un punto dado a otro punto dado

Como vimos en el punto 1.2.3, es recomendable añadir el parámetro "region" a la etiqueta <script> con la que cargamos la API de Google Maps, para evitar el sesgo por defecto hacia los Estados Unidos que aplica la interfaz. Los valores válidos para este parámetro son los de la *ISO 3166-1 alpha-2 code*. En el caso de España es "ES".

```
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false&region=ES"></script>
```

En el documento HTML encontramos dos elementos nuevos además del mapa: los desplegables que nos permitirán seleccionar el punto de inicio y el punto final de la ruta, y el panel donde aparecerán las indicaciones.

- El panel para las indicaciones irá dentro de un como el mapa:

```
<div id="directions-panel"></div>
```

- Los desplegables para seleccionar las rutas en una etiqueta <select> con sus respectivas opciones (<option>), que contendrán la dirección o coordenadas de los lugares de inicio y fin de las rutas. Para que cada vez que seleccionemos una ubicación, la API de Google Maps sea capaz de recalcular la ruta, hemos de asociar la función "calcRoute();" como evento "onchange" de la etiqueta <select>. Todo ello lo incluiremos dentro de un nuevo .

```
<div id="control">
<strong>Start:</strong>
<select id="start" onchange="calcRoute();">
<option value="Carrer de Melcior de Palau, 140, 08014 Barcelona">Facultat
de Biblioteconomia (UB)</option>
<option value="Facultad de Comunicación y Documentación @37.1931,-
3.596606">Facultad de Comunicaci&oacute;n y Documentaci&oacute;n
(UM)</option>
<option value="Calle Madrid, 126, 28903, Getafe">Facultad de Humanidades,
Comunicaci&oacute;n y Documentaci&oacute;n (UC3)</option>
<option value="Calle Campus de Cartuja, 0, 18011 Granada"> Facultad de
Comunicaci&oacute;n y Documentaci&oacute;n (UGR)</option>
</select>
<strong>End:</strong>
<select id="end" onchange="calcRoute();">
<option value="Carrer de Melcior de Palau, 140, 08014 Barcelona">Facultat
de Biblioteconomia (UB)</option>
<option value="Facultad de Comunicación y Documentación @37.1931,-
3.596606">Facultad de Comunicaci&oacute;n y Documentaci&oacute;n
(UG)</option>
<option value="Calle Madrid, 126, 28903, Getafe">Facultad de Humanidades,
Comunicaci&oacute;n y Documentaci&oacute;n (UC3)</option>
<option value="Calle Campus de Cartuja, 0, 18011 Granada"> Facultad de
Comunicaci&oacute;n y Documentaci&oacute;n (UGR)</option>
</select>
</div>
<div id="directions-panel"></div>
<div id="mapa"></div>
```

A continuación, debemos definir la ruta, o mejor dicho, asociar el origen y destino de la ruta a los desplegables que hemos creado en la página HTML (#start y

#end). Usaremos el objeto "google.maps.DirectionsService" para calcular la ruta. Este objeto nos devolverá otro objeto, el "DirectionsResult", con toda la información de la ruta. Para que el objeto "DirectionsService" pueda calcular la ruta, además del punto de origen y el punto de destino, debemos indicarle el medio de transporte. Actualmente, la API de Google Maps admite los siguientes medios de transporte:

- **google.maps.TravelMode.DRIVING** (predeterminado): proporciona rutas estándar para llegar en coche a través de la red de carreteras.
- **google.maps.TravelMode.BICYCLING** solicita rutas para llegar en bicicleta a través de carriles bici y vías preferenciales para bicicletas.
- **google.maps.TravelMode.TRANSIT** solicita rutas de transporte público.
- **google.maps.TravelMode.WALKING** solicita rutas a pie a través de aceras y rutas peatonales.

```
function calcRoute() {
      var start = document.getElementById('start').value;
      var end = document.getElementById('end').value;
      var request = {
            origin: start,
            destination: end,
            travelMode: google.maps.TravelMode.DRIVING
      };
```

Ahora pasaremos la información de la ruta al objeto "DirectionsService", mediante un objeto "DirectionsRequest". Para ello ejecutamos el método ".route()".

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

El resultado del método ".route()" devuelve el "DirectionsResult" dentro del atributo "response". Una vez tenemos el resultado podemos acceder a la información que contiene y representarlo sobre el mapa. Para poder representar esa información sobre el mapa usaremos el objeto "google.maps.DirectionsRenderer". Para configurarlo utilizaremos el método ".setMap()" para indicar cuál es el mapa, y el método ".setPanel()" para indicarle dónde ha de mostrar el texto de la ruta.

```
directionsDisplay = new google.maps.DirectionsRenderer();
var mapOptions = {
	zoom: 8,
	mapTypeId: google.maps.MapTypeId.ROADMAP,
	center: new google.maps.LatLng(41.652393,1.691895)
};
var map = new google.maps.Map(document.getElementById('mapa'),
	mapOptions);
directionsDisplay.setMap(map);
directionsDisplay.setPanel(document.getElementById('directions-panel'));
```

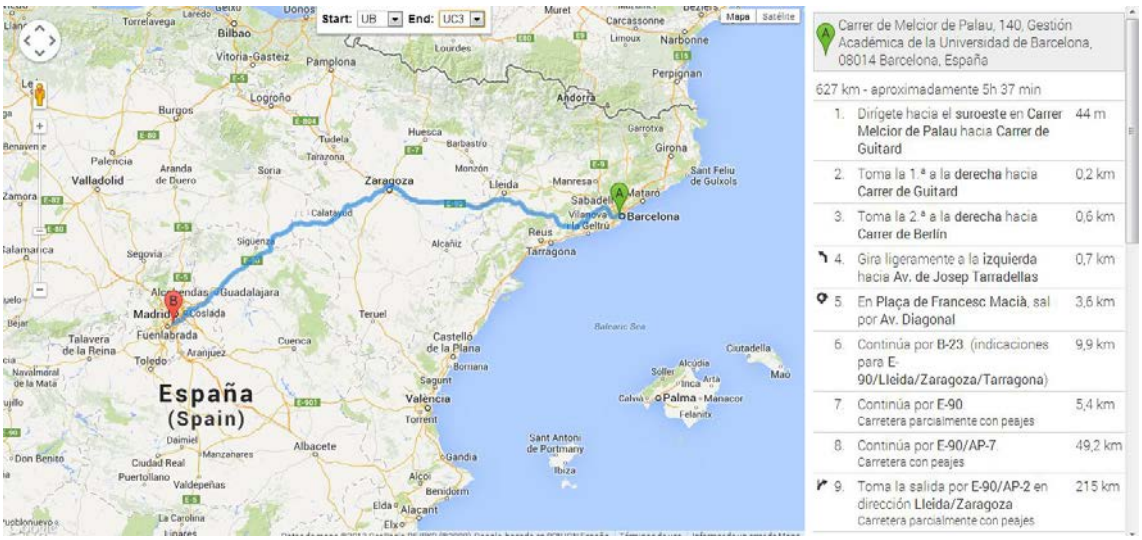
Si queremos que los desplegables aparezcan dentro del mapa, debemos crearlos mediante una variable, asociarlos a su ID correspondiente en el código HTML (#control) e indicar su posición en el mapa.

```
var control = document.getElementById('control');
control.style.display = 'block';
map.controls[google.maps.ControlPosition.TOP_CENTER].push(control);
```

Todo junto: controles y función CalcRoute

```
var directionsDisplay;
var directionsService = new google.maps.DirectionsService();
function initialize() {
directionsDisplay = new google.maps.DirectionsRenderer();
var mapOptions = {
zoom: 7,
mapTypeId: google.maps.MapTypeId.ROADMAP,
center: new google.maps.LatLng(41.652393,1.691895)
  };
var map = new google.maps.Map(document.getElementById('map-canvas'),
mapOptions);
directionsDisplay.setMap(map);
directionsDisplay.setPanel(document.getElementById('directions-panel'));
var control = document.getElementById('control');
control.style.display = 'block';
map.controls[google.maps.ControlPosition.TOP_CENTER].push(control);
}
function calcRoute() {
var start = document.getElementById('start').value;
var end = document.getElementById('end').value;
var request = {
origin: start,
destination: end,
travelMode: google.maps.TravelMode.DRIVING
  };
directionsService.route(request, function(response, status) {
if (status == google.maps.DirectionsStatus.OK) {
directionsDisplay.setDirections(response);
    }});
}
google.maps.event.addDomListener(window, 'load', initialize);
```

Vista del mapa



Servicio de rutas entre dos puntos dados.

4.2. Detectar la ubicación del usuario

Existen diferentes formas de acceder a la ubicación del usuario. La más precisa de ellas es cuando el usuario utiliza un dispositivo como un GPS. El problema en los ordenadores de sobremesa es que los navegadores generalmente no tienen acceso a un GPS. A pesar de ello, están conectados a una red, y tienen una IP que nos puede ofrecer una aproximación de la localización del usuario.

4.2.1. Detectar la ubicación del usuario a partir de su IP

Para tener acceso a la ubicación del usuario a través de su IP, debemos cargar la API de Google Maps de una manera diferente a la que lo estábamos haciendo hasta ahora. En primer lugar cargaremos la API de Google Maps desde "google.load", que forma parte de la API de Google AJAX.

```
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
```

Para poder utilizar la API de Google AJAX, o cualquier otra API adicional, tenemos que utilizar el método "google.load()". De la siguiente manera podemos cargar la versión 3 de la API de Google Maps.

```
google.load('maps', 3, {'other_params': 'sensor=false'});
```

Los parámetros que pasamos son el nombre de la API (maps), la versión (3), y un objeto de opción que contiene configuraciones adicionales (sensor, etc.).
Ahora ya podemos tener acceso aproximado a la ubicación del usuario a través de su IP. Esta información se encuentra en el objeto "google.loader.ClientLocation". Sus propiedades son:

- **latitude:** devuelve la latitud asociada a la IP del cliente.longitude: devuelve la longitud asociada a la IP del cliente.
- **address.city:** devuelve el nombre de la ciudad asociado a la IP del clienteaddress.country: devuelve el nombre del país asociado a la IP del cliente.address.country_code: devuelve el código ISO 3166-1 de país asociado a la IP del cliente.
- **address.region:** devuelve la región del país asociada a la IP del cliente

Empezamos cargando la API de Google Maps y a continuación creamos un controlador (handler) para el “window.onload”.

```
(function() {
	google.load('maps', 3, {
	'other_params': 'sensor=false&language=en'
});
window.onload = function() {
	//aquí el resto de código

	}
})();
```

A continuación, intentaremos obtener la posición del usuario mediante el objeto "google.loader.ClientLocation". Primero proporcionaremos una ubicación predeterminada en (0,0) y crearemos el contenido del "InfoWindow", que consistirá en el nombre de la ciudad y el país.

```
window.onload = function() {
	if (google.loader.ClientLocation.latitude &&
google.loader.ClientLocation.longitude) {

	var latLng = new
	google.maps.LatLng(google.loader.ClientLocation.latitude,
	google.loader.ClientLocation.longitude);

var location = 'Te encuentras en: '
location += google.loader.ClientLocation.address.city + ', ';
location += google.loader.ClientLocation.address.country;

} else {

	var latLng = new google.maps.LatLng(0, 0);
	var location = 'No es posible geolocalizarte';
}
}
```

Una vez tenemos la ubicación del usuario, ya podemos crear el mapa y el marcador con su "InfoWindow".

```
var options = {
	zoom: 2,
	center: latLng,
	mapTypeId: google.maps.MapTypeId.ROADMAP
	};
map = new google.maps.Map(document.getElementById('map'), options);

var marker = new google.maps.Marker({
	position: latLng,
	map: map
	});

var infoWindow = new google.maps.InfoWindow({
	content: location
	});
infoWindow.open(map, marker);
}
```

Todo junto: carga de la API, consulta de posición por IP y creación de un marcador

```
(function() {
google.load('maps', 3, {
'other_params': 'sensor=false&language=en'
});
window.onload = function() {
if (google.loader.ClientLocation.latitude && google.loader.ClientLocation.longitude)
{
var latLng = new google.maps.LatLng(google.loader.ClientLocation.latitude,
google.loader.ClientLocation.longitude);
var location = 'Te encuentras en '
location += google.loader.ClientLocation.address.city + ', ';
location += google.loader.ClientLocation.address.country;
} else {
var latLng = new google.maps.LatLng(0, 0);
var location = 'No es posible geolocalizarte';
}
var options = {
        zoom: 2,
        center: latLng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
        };
map = new google.maps.Map(document.getElementById('mapa'), options);

var marker = new google.maps.Marker({
        position: latLng,
        map: map
        });
var infoWindow = new google.maps.InfoWindow({
content: location
});
infoWindow.open(map, marker);
}
})();
```

Vista del mapa

El servicio ha detectado nuestra ubicación en Barcelona.

4.2.2. Detectar la ubicación del usuario mediante la API de geolocalización de HMTL5

A partir de la API de Geolocalización de HTML5 es posible detectar la ubicación geográfica de un usuario. Empezamos creando un mapa sin definir su centro:

```
var mapOptions = {
zoom: 18,
mapTypeId: google.maps.MapTypeId.ROADMAP
};
map = new google.maps.Map(document.getElementById('mapa'),
mapOptions);
```

A continuación, comprobamos si el navegador soporta la API de geolocalización HTML5. Si el resultado de la condición es verdadero, se activa el método "getCurrentPosition()". Aprovechamos los valores de la función anterior para almacenarlo en la variable "posicionUsuario".

```
if(navigator.geolocation) {
navigator.geolocation.getCurrentPosition(function(position) {
var posicionUsuario = new
google.maps.LatLng(position.coords.latitude,position.coords.longitude);
```

Ahora que tenemos las coordenadas almacenadas en la variable "posicionUsuario", vamos a crear un marcador centrado en esa posición, y un "InfoWindow" asociado a él mediante un evento de clic. Estos elementos sólo se activarán si la primera condición se cumple, es decir, si el navegador es compatible y el usuario permite la geolocalización.

```
var marker = new google.maps.Marker({
      position: posicionUsuario,
      map: map,
      title: 'Geolocalizaci&oacute;n HTML5',
});

var infowindow = new google.maps.InfoWindow({
content: 'Ubicaci&oacute;n obtenida a partir de HTML5.'
});
google.maps.event.addListener(marker, 'click', function() {
infowindow.open(map,marker);
});
```

Creamos una función que centrará la posición del usuario en las coordenadas almacenadas en la variable "posicionUsuario" si es verdadera.

```
map.setCenter(posicionUsuario);
    }, function() {
handleNoGeolocation(true);
});
  } else {
    // el navegador no es compatible o no lo permite
handleNoGeolocation(false);
  }
}
```

La función "handleNoGeolocation(errorFlag)" tiene asociados dos textos diferentes que aparecerán en el "InfoWindow" según si el servicio de

geolocalización ha fallado, o el usuario no lo ha permitido ("if (errorFlag)") o en el resto de los casos (navegador no compatible principalmente).

```
function handleNoGeolocation(errorFlag) {
if (errorFlag) {
var content = 'Error: El servicio de geolocalizaci&oacute;n ha
fallado.';
  } else {
var content = 'Error: Tu navegador no soporta el servicio de
geolocalizaci&oacute;n.';
}
```

Por último, definimos las características del mapa que se mostrará en los dos casos anteriores y el "InfoWindow" que contiene el mensaje de error.

```
var options = {
map: map,
position: new google.maps.LatLng(0,0),
content: content
  };

var infowindow = new google.maps.InfoWindow(options);
map.setCenter(options.position);
}
```

Todo junto: consulta de posición GPS y creación de un marcador

```
var map;
function initialize() {
var mapOptions = {
zoom: 18,
mapTypeId: google.maps.MapTypeId.ROADMAP
  };
map = new google.maps.Map(document.getElementById('mapa'),
mapOptions);

  // Intenta utilizar geolocalización a partir de HTML5
```

```
if(navigator.geolocation) {
navigator.geolocation.getCurrentPosition(function(position) {
var posicionUsuario = new
google.maps.LatLng(position.coords.latitude,position.coords.longitude)
;
       var marker = new google.maps.Marker({
                    position: posicionUsuario,
                    map: map,
                    title: 'Geolocalizaci&oacute;n HTML5',
             });

var infowindow = new google.maps.InfoWindow({
content: 'Ubicaci&oacute;n obtenida a partir de HTML5.'
});
             google.maps.event.addListener(marker, 'click', function()
{
infowindow.open(map,marker);
       });
map.setCenter(posicionUsuario);
}, function() {
handleNoGeolocation(true);
    });
  } else {
    // el navegador no es compatible o no lo permite
handleNoGeolocation(false);
  }
}
function handleNoGeolocation(errorFlag) {
if (errorFlag) {
var content = 'Error: El servicio de geolocalizaci&oacute;n ha
fallado.';
  } else {
var content = 'Error: Tu navegador no soporta el servicio de
geolocalizaci&oacute;n.';
}
var options = {
map: map,
position: new google.maps.LatLng(0,0),
content: content
```

```
  };

var infowindow = new google.maps.InfoWindow(options);
map.setCenter(options.position);
}
google.maps.event.addDomListener(window, 'load', initialize);
```

4.2.3. Detectar la ubicación GPS del usuario mediante el plugin geo.js

Si queremos implementar un mapa con esta funcionalidad y deseamos evitar la posible incompatibilidad con determinados navegadores podemos utilizar "geo.js" una biblioteca de JavaScript de código abierto[5]. En primer lugar agregamos el vínculo a la biblioteca dentro del <head>:

```
<script type="text/javascript" src="js/geo.js"></script>
```

Ahora creamos un mapa en el que se vea todo el mundo:

```
(function() {
var map;
window.onload = function() {

      var options = {
            zoom: 1,
            center: new google.maps.LatLng(31.35, 3.51),
            mapTypeId: google.maps.MapTypeId.ROADMAP
      };
map = new google.maps.Map(document.getElementById('mapa'), options);
// aquí irá la llamada a la librería geo.js
}
function handleError() {
}
function setPosition() {
}
})();
```

[5] https://code.google.com/p/geo-location-javascript/

A continuación, intentaremos determinar la localización del usuario. Todos los métodos de la biblioteca geo.js residen en el objeto "geo_position_js". En primer lugar inicializamos el objeto "geo_position_js" con el método "init()". Utilizaremos un *else* para que en caso de que el valor del método "init()" sea falso, devuelva un mensaje de error.

```
if (geo_position_js.init()) {
} else {
alert('No es posible geolocalizar');
}
```

A continuación utilizamos el método "getCurrentPosition()" para determinar la localización del dispositivo. Este método tiene tres argumentos. El primero es una función que será devuelta en caso de que la API sea capaz de determinar la posición, otra que lo hará si la API no es capaz de determinar la posición, y el último argumento es un objeto de configuración. Sólo utilizaremos la opción "enableHighAccuracy" con el valor "true", para conseguir la posición más precisa posible.

```
if (geo_position_js.init()) {
var settings = {
enableHighAccuracy: true
};
geo_position_js.getCurrentPosition(setPosition, handleError,
settings);
} else {
alert('No es posible geolocalizar');
}
```

Ahora tenemos que crear dos funciones de devolución de llamada: "setPosition()" y " handleError()"

```
function handleError(error) {
alert('Error = ' + error.message);
}
```

El método "setPosition ()" se utilizará para pasar la posición a la función encargada de crear un marcador y una ventana de información.

```
function setPosition(position) {
var latLng = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
var marker = new google.maps.Marker({
position: latLng,
map: map
});
var infoWindow = new google.maps.InfoWindow({
content: 'Estás aquí'
});
infoWindow.open(map, marker);
map.setZoom(6);
}
```

Todo junto

```
(function() {
var map;
window.onload = function() {

// Creamos el mapa
var options = {
zoom: 1,
center: new google.maps.LatLng(31.35, 3.51),
mapTypeId: google.maps.MapTypeId.ROADMAP
};
map = new google.maps.Map(document.getElementById('mapa'), options);

// Comprobamos si la geolocalización es posible
if (geo_position_js.init()) {

// Creamos el objeto settings
```

```
var settings = {
enableHighAccuracy: true
};

// Intentamos determinar la ubicación del usuario
geo_position_js.getCurrentPosition(setPosition, handleError, settings);
} else {
alert('No es posible geolocalizar ');
}
};
function handleError(error) {
alert('Error = ' + error.message);
}
function setPosition(position) {

// Creamos la variable latLng con la posición del usuario
var latLng = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);

// Añadimos un marcador al mapa
var marker = new google.maps.Marker({
position: latLng,
map: map
});

// Creamos un InfoWindow
var infoWindow = new google.maps.InfoWindow({
content: 'Estás aquí'
});

// Añadimos el InfoWindow al mapa
infoWindow.open(map, marker);

// Zoom sobre el mapa
map.setZoom(6);
}
})();
```

4.4. Indicar una ruta tomando como punto de partida la ubicación GPS del usuario

Para este ejemplo volveremos a utilizar la biblioteca jQuery. En primer lugar, almacenaremos en la cache los diferentes selectores.

```
$Selectors = {
        mapCanvas: jQuery('#mapCanvas')[0],
        dirPanel: jQuery('#directionsPanel'),
        dirInputs: jQuery('.directionInputs'),
        dirSrc: jQuery('#dirSource'),
        dirDst: jQuery('#dirDestination'),
        getDirBtn: jQuery('#getDirections'),
        dirSteps: jQuery('#directionSteps'),
        paneToggle: jQuery('#paneToggle'),
        useGPSBtn: jQuery('#useGPS'),
        paneResetBtn: jQuery('#paneReset')
        },
```

Un selector en jQuery es similar a uno en CSS, pero aplicado a los elementos HTML con un id. Es la forma con que jQuery nos permite elegir un elemento de nuestro documento HTML, para posteriormente aplicarle diferentes funciones o eventos.

4.4.1. Configurar el servicio Directions

El acceso al servicio de rutas tiene lugar de forma asíncrona, ya que la API de Google Maps debe hacer una llamada a un servidor externo. Por ese motivo, debemos incluir un método de devolución de llamada para que se ejecute al completar la solicitud. Este método de devolución de llamada procesará los resultados. La solicitud la ejecutaremos con el método "DirectionsService.route()". Este servicio genera un JSON como salida. Para interpretarlo podemos utilizar la opción por defecto en la API de Google Maps, "DirectionsRenderer".

```
directionsSetup = function() {
        directionsService = new google.maps.DirectionsService();
        directionsDisplay = new google.maps.DirectionsRenderer({
        });
        directionsDisplay.setMap(map);
        directionsDisplay.setPanel($Selectors.dirSteps[0]);
},
```

El objetivo es generar las direcciones a partir de los valores de origen y destino introducidos en sendos campos del código HTML o en su defecto a partir de la geolocalización del usuario. Estos valores se pasarán al Directions Service, que en caso de éxito devolverá una respuesta. Debemos representar el en el que aparecerá la respuesta. Lo debemos definir mediante:

```
directionsDisplay.setPanel($Selectors.dirSteps[0])
```

Además del "setPanel" también necesitamos el "setDirections" para mostrar la respuesta dentro del

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

La respuesta del servicio contiene el objeto "route", que contendrá diferentes informaciones como las coordenadas de inicio y destino, o los puntos de interés, entre otros detalles.

También crearemos una variable (selectedMode) que cogerá como valor, una de las tres opciones del cuadro desplegable con id "mode" disponible en la página HTML (WALKING, DRIVE, TRANSIT). Este valor lo pasaremos después al campo "traveMode", campo obligatorio donde se especifica el medio de transporte que se utilizará al calcular las rutas.

```
directionsRender = function(source, destination) {
	var selectedMode = document.getElementById('mode').value;
	var request = {
		origin: source,
		destination: destination,
		provideRouteAlternatives: false,
		travelMode: google.maps.TravelMode[selectedMode]
	};
	directionsService.route(request, function(response, status) {
		if (status == google.maps.DirectionsStatus.OK) {
			directionsDisplay.setDirections(response);
		}
	});
},
```

4.4.2. Autocompletar con la API de Google Places

La API de autocompletado de Google Places es un servicio web que ofrece información sobre un sitio basada en términos de búsqueda de texto y, opcionalmente, en límites geográficos. Esta API se puede utilizar para proporcionar funciones de autocompletado para búsquedas geográficas basadas en texto mostrando sitios como empresas, direcciones y lugares de interés a medida que el usuario escribe. Para llamar a la API ubicaremos en la cabecera del documento el siguiente script:

```
<script type="text/javascript"
src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=true&libraries=pl
aces"></script>
```

El código para utilizar el servicio:

```
autoCompleteSetup = function() {
	autoSrc = new google.maps.places.Autocomplete($Selectors.dirSrc[0]);
	autoDest = new google.maps.places.Autocomplete($Selectors.dirDst[0]);
},
```

4.4.3. Geolocalización (HTML5 Geolocation y Google Geocoding API integration)

Como hemos visto anteriormente, la API de geolocalización de HTML5 permite compartir la ubicación (latitud y longitud) siempre que el usuario permita a la API rastrear su ubicación. Una vez obtenida, es posible pasar esas coordenadas a la API de geocodificación de Google para obtener una dirección formateada (por ejemplo, Carrer Melcior de Palau, 140, Barcelona, Espanya).

Invocar la API de Geolocalización de HTML5

```
$Selectors.useGPSBtn.on('click', function(e) {
	if (navigator.geolocation) {
		navigator.geolocation.getCurrentPosition(function(position) {
			fetchAddress(position);
		});
	}
});
```

Luego usamos la API de geocodificación para transformar las coordenadas en una dirección.

```
fetchAddress = function(p) {
	var Position = new google.maps.LatLng(p.coords.latitude, p.coords.longitude),
		Locater = new google.maps.Geocoder();
	Locater.geocode({'latLng': Position}, function (results, status) {
		if (status == google.maps.GeocoderStatus.OK) {
			var _r = results[0];
			$Selectors.dirSrc.val(_r.formatted_address);
		}
	});
}
```

Todo junto

El código JavaScript

```
(function(mapDemo, $, undefined) {
	mapDemo.Directions = (function() {
		function _Directions() {
			var map,
				directionsService, directionsDisplay,
				autoSrc, autoDest,
	$Selectors = {
		mapCanvas: jQuery('#mapCanvas')[0],
		dirPanel: jQuery('#directionsPanel'),
		dirInputs: jQuery('.directionInputs'),
		dirSrc: jQuery('#dirSource'),
		dirDst: jQuery('#dirDestination'),
		getDirBtn: jQuery('#getDirections'),
```

```
                dirSteps: jQuery('#directionSteps'),
                paneToggle: jQuery('#paneToggle'),
                useGPSBtn: jQuery('#useGPS'),
                paneResetBtn: jQuery('#paneReset')
                },
        autoCompleteSetup = function() {
                autoSrc = new google.maps.places.Autocomplete($Selectors.dirSrc[0]);
                autoDest = new google.maps.places.Autocomplete($Selectors.dirDst[0]);
                },
        directionsSetup = function() {
                directionsService = new google.maps.DirectionsService();
                directionsDisplay = new google.maps.DirectionsRenderer({
                });
        directionsDisplay.setPanel($Selectors.dirSteps[0]);

        },
        trafficSetup = function() {
                var controlDiv = document.createElement('div'),
                controlUI = document.createElement('div'),
                trafficLayer = new google.maps.TrafficLayer();
                jQuery(controlDiv).addClass('gmap-control-
container').addClass('gmnoprint');
                jQuery(controlUI).text('Traffic').addClass('gmap-control');
                jQuery(controlDiv).append(controlUI);
        google.maps.event.addDomListener(controlUI, 'click', function() {
                if (typeof trafficLayer.getMap() == 'undefined' ||
trafficLayer.getMap() === null) {
                jQuery(controlUI).addClass('gmap-control-active');
```

```
            trafficLayer.setMap(map);
            } else {
            trafficLayer.setMap(null);
            jQuery(controlUI).removeClass('gmap-control-active');
            }
            });

map.controls[google.maps.ControlPosition.TOP_RIGHT].push(controlDiv);

},
mapSetup = function() {
            map = new google.maps.Map($Selectors.mapCanvas, {
            zoom: 9,
            center: new google.maps.LatLng(41.652393,1.691895),

            mapTypeControl: true,
            mapTypeControlOptions: {
                        style: google.maps.MapTypeControlStyle.DEFAULT,
                        position: google.maps.ControlPosition.TOP_RIGHT
            },
            panControl: true,
            panControlOptions: {
                        position: google.maps.ControlPosition.RIGHT_TOP
            },
            zoomControl: true,
            zoomControlOptions: {
                        style: google.maps.ZoomControlStyle.LARGE,
                        position: google.maps.ControlPosition.RIGHT_TOP
```

```
                },
                scaleControl: true,
                streetViewControl: true,
                overviewMapControl: true,
                mapTypeId: google.maps.MapTypeId.ROADMAP
                        });
        autoCompleteSetup();
        directionsSetup();
        trafficSetup();
        },
directionsRender = function(source, destination) {
        $Selectors.dirSteps.find('.msg').hide();
        directionsDisplay.setMap(map);
var request = {
        origin: source,
        destination: destination,
        provideRouteAlternatives: false,
        travelMode: google.maps.DirectionsTravelMode.DRIVING
        };
directionsService.route(request, function(response, status) {
        if (status == google.maps.DirectionsStatus.OK) {
                directionsDisplay.setDirections(response);
                }
        });
},
fetchAddress = function(p) {
```

```
            var Position = new google.maps.LatLng(p.coords.latitude,
p.coords.longitude),
            Locater = new google.maps.Geocoder();


            Locater.geocode({'latLng': Position}, function (results, status) {
                  if (status == google.maps.GeocoderStatus.OK) {
                        var _r = results[0];
                              $Selectors.dirSrc.val(_r.formatted_address);
                              }
                  });
      },
      invokeEvents = function() {
            $Selectors.getDirBtn.on('click', function(e) {
                  e.preventDefault();
                  var src = $Selectors.dirSrc.val(),
                        dst = $Selectors.dirDst.val();
                  directionsRender(src, dst);
            });
            $Selectors.paneResetBtn.on('click', function(e) {
                  $Selectors.dirSteps.html('');
                  $Selectors.dirSrc.val('');
                  $Selectors.dirDst.val('');
                  directionsDisplay.setMap(null);
            });
            $Selectors.paneToggle.toggle(function(e) {
                  $Selectors.dirPanel.animate({'left': '-=305px'});
                  jQuery(this).html('&gt;');
```

```
		}, function() {
			$Selectors.dirPanel.animate({'left': '+=305px'});
			jQuery(this).html('&lt;');
		});
		$Selectors.useGPSBtn.on('click', function(e) {
			if (navigator.geolocation) {

	navigator.geolocation.getCurrentPosition(function(position) {
					fetchAddress(position);
				});
			}
		});
	},
	_init = function() {
		mapSetup();
		invokeEvents();
	};
	this.init = function() {
		_init();
		return this;
	}
	return this.init(); // Refers to: mapDemo.Directions.init()
	}
	return new _Directions(); // Creating a new object of _Directions rather than
a funtion
	}());
})(window.mapDemo = window.mapDemo || {}, jQuery);
```

El código HTML

```
<!DOCTYPE html>

<html>

        <head>

        <title>Buscar direcci&oacute;n desde mi localizaci&oacute;n</title>

        <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

        <script type="text/javascript" src="js/jquery.min.js"></script>

        <script type="text/javascript"
        src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=true&libraries=places"></scr
        ipt>

        <link href="css/estilo.css" type="text/css" rel="stylesheet" />

        </head>

        <body>

        <div id="mapCanvas"> </div>

        <div id="directionsPanel">

        <a href="#geoLocation" id="useGPS">Usar m&iacute; localizaci&oacute;n<img
        src="geo.png"></a>

<p class="or">O</p>

        <div class="directionInputs">

        <form>

                <p><label>Origen</label><input type="text" value="" id="dirSource" /></p>

                <p><label>Destino</label><input type="text" value="" id="dirDestination" /></p>

                <p><label>Medio de transporte</label>

                <select id="mode">

                        <option value="DRIVING">En coche</option>

                        <option value="WALKING">A pie</option>

                        <option value="TRANSIT">En transporte p&uacute;blico</option>

                </select></p>

                <a href="#getDirections" id="getDirections">C&oacute;mo llegar</a>

<a href="#reset" id="paneReset">Borrar</a>
```

```
        </form>
        </div>
        <div id="directionSteps">
                <p class="msg">La ruta se mostrar&aacute; aqu&iacute;</p>
        </div>
        <a href="#toggleBtn" id="paneToggle" class="out">&lt;</a>
        </div>
        <script type="text/javascript" src="js/mapa.js"></script>
        </body>
</html>
```

5. Dibujar sobre el mapa

5.1. Polígonos

En primer lugar, debemos definir las coordenadas de la línea o polígono. Para ello debemos crear un array (vector) de elementos del tipo “google.maps.LatLng”, indicando para cada uno de ellos sus respectivas coordenadas. En el siguiente ejemplo, podemos ver la definición de un polígono de tres lados entre las ciudades de Barcelona, Reus y Manresa.

```
var poligono = [
        new google.maps.LatLng(41.3788696258852, 2.1752934374999313),
        new google.maps.LatLng(41.14763810202612, 1.1041264453124313),
        new google.maps.LatLng(41.72828039782993, 1.8237309374999313)
];
```

A continuación, debemos definir el estilo de la línea o, en el ejemplo que nos ocupa, del polígono. Para ello nos podemos valer de diferentes propiedades. Es importante que el valor del atributo "paths" sea el nombre de la variable en la que hemos almacenado nuestro array con todas las coordenas. Mediante el resto de

las propiedades que se muestran a continuación podemos establecer el color y grosor de la línea, así como del relleno.

```
miPoligono = new google.maps.Polygon({
  paths: poligono,
  strokeColor: "#FF0000",
  strokeOpacity: 0.8,
  strokeWeight: 3,
  fillColor: "#FF0000",
  fillOpacity: 0.4
});
```

Una vez definidos los puntos que formarán la línea o polígono y su estilo, sólo nos falta añadirlo al mapa. Para ello utilizaremos:

```
miPoligono.setMap(mapa);
```

A continuación, se muestra un ejemplo completo:

```
window.onload = function() {
	var poligono = [
		new google.maps.LatLng(41.3788696258852, 2.1752934374999313),
		new google.maps.LatLng(41.14763810202612, 1.1041264453124313),
		new google.maps.LatLng(41.72828039782993, 1.8237309374999313)
	];
	var mapDiv = document.getElementById('map');
	var catalunya = new google.maps.LatLng(41.652393,1.691895);
	var options = {
		center: catalunya,
```

```
            zoom: 8,
            mapTypeId: google.maps.MapTypeId.ROADMAP
        };
        miPoligono = new google.maps.Polygon({
          paths: poligono,
          strokeColor: "#FF0000",
          strokeOpacity: 0.8,
          strokeWeight: 3,
          fillColor: "#FF0000",
          fillOpacity: 0.4
        });
        var mapa = new google.maps.Map(mapDiv, options);
        miPoligono.setMap(mapa);
}
```

El resultado sería el siguiente:

Vista del polígono sobre el mapa.

5.2. Círculos

En primer lugar, crearemos un array de objetos que contengan las coordenadas y la población de cada ciudad.

```
var citymap = {
  Barcelona: {
    center: {lat: 41.381601, lng: 2.172852},
    population: 5524000
  },
  Tarragona: {
    center: {lat: 41.117280, lng: 1.250000},
    population: 800962
  },
  Lérida: {
    center: {lat: 41.624323, lng: 0.623779},
    population: 438001
  },
  Gerona: {
    center: {lat: 41.984658, lng: 2.821045},
    population: 706185
  }
};
```

Una vez creadas todas las localizaciones, nos faltará generar el mapa.

```
window.onload = function() {
var mapa = new google.maps.Map(document.getElementById('mapa'), {
zoom: 8,
center: {lat: 41.652393, lng: 1.691895},
mapTypeId: google.maps.MapTypeId.ROADMAP
});
// aquí el resto del código ...
}
```

Con un bucle *for in* creamos un círculo para cada localización, basándonos en la población para establecer su diámetro.

```
for (var city in citymap) {
      var cityCircle = new google.maps.Circle({
      strokeColor: '#0000FF',
      strokeOpacity: 0.8,
      strokeWeight: 2,
      fillColor: '#5882FA',
      fillOpacity: 0.35,
      map: mapa,
      center: citymap[city].center,
      radius: Math.sqrt(citymap[city].population) * 10
      });
}
```

El resultado final es

Vista de los círculos generados.

Bibliografía

"Arrays". En: *jQuery: write less, do more*. <http://learn.jquery.com/javascript-101/arrays/>. [Consulta: 19/07/2019].

"Autocompletado de sitios". En: *Google Places API Web Service.* <https://developers.google.com/places/documentation/autocomplete?hl=es>. [Consulta: 12/02/2016].

"Directions service". En: *Google Maps JavaScript API.* <https://developers.google.com/maps/documentation/javascript/directions>. [Consulta: 19/07/2019].

".each()". En: *jQuery: write less, do more*. <http://api.jquery.com/each/>. [Consulta: 19/07/2019].

"Geolocation". En: *Google Maps API.* <https://developers.google.com/maps/articles/geolocation>. [Consulta: 19/07/2019].

geo-location-javascript: javascript geo location framework for the mobile web. <https://code.google.com/p/geo-location-javascript/>. [Consulta: 12/02/2016].

Jeedigunta, Giri. "Custom Directions Panel with Google Maps API v3". *theWebStoreByGJ!*. <http://thewebstorebyg.wordpress.com/2013/01/11/custom-directions-panel-with-google-maps-api-v3/>. [Consulta: 19/07/2019].

"Google Maps: accesibilidad de teclado y alternativa en ausencia de Javascript". *Outbook*. <http://blog.outbook.es/desarrollo-web/google-maps-accesibilidad-de-teclado-y-alternativa-en-ausencia-de-javascript>. [Consulta: 19/07/2019].

"HTML5 geolocation". En: *w3schools.com*. <http://www.w3schools.com/html/html5_geolocation.asp>. [Consulta: 19/07/2019].

"jQuery.getJSON()". En: *jQuery: write less, do more*. <http://api.jquery.com/jQuery.getJSON/>. [Consulta: 19/07/2019].

Jquery: write less, do more. <http://jquery.com/>. [Consulta: 19/07/2019].

" Map Types ". En: *Google Maps JavaScript API*. <https://developers.google.com/maps/documentation/javascript/maptypes?hl=es>. [Consulta: 19/07/2019].

"Maps API getZoom() Method". En: *w3schools.com*. <http://www.w3schools.com/googleAPI/ref_getzoom.asp>. [Consulta: 19/07/2019].

"Maps API setMapTypeId() Method". En: *w3schools.com*. <http://w3schools.com/googleapi/ref_setmaptypeid.asp>. [Consulta: 19/07/2019].

Svennerberg, Gabriel. *Beginning Google Maps API 3*. [Berkeley, CA]: Apress; cop. 2010.

"The Google Maps Geocoding API". En: *Google Maps Geocoding API* . <https://developers.google.com/maps/documentation/geocoding/?hl=es>. [Consulta: 19/07/2019].