Tema 10. Geolocalización

Contenido

1) Introducción

Otra de las novedades que el nuevo estándar HTML5 ofrece, es la opción de detectar la geolocalización del usuario de una página o aplicación web de una forma muy sencilla. Esto, unido a la gran expansión del uso de internet en dispositivos móviles que se está produciendo en los últimos años, abre nuevas e interesantes posibilidades para el desarrollo de aplicaciones Web. Así, con el uso de HTML5 y JavaScript, es posible ofrecer al usuario información sobre productos, servicios o eventos de forma personalizada, en función de la ubicación física en la que se encuentre.

La detección de la geolocalización del usuario, como la mayoría de las novedades que presenta HTML5, necesita de un lenguaje de script, en este caso JavaScript, para funcionar. La utilización de esta API, combinada con el uso de Google Maps, hace posible que los desarrolladores web puedan situar al usuario sobre un mapa y guiarle con precisión o simplemente mostrarle lugares cercanos que puedan ser de su interés.

Conocer la localización de un usuario no es algo nuevo, hay servicios que ofrecen la posibilidad de conocerla utilizando la dirección IP y haciendo una consulta a una base de datos en la que estarán relacionadas dichas direcciones con sus localizaciones. Esto a veces, puede no ser muy eficaz ni funcional porque requiere mantener constantemente actualizadas las bases de datos de ip y no siempre muestra la ubicación con exactitud.

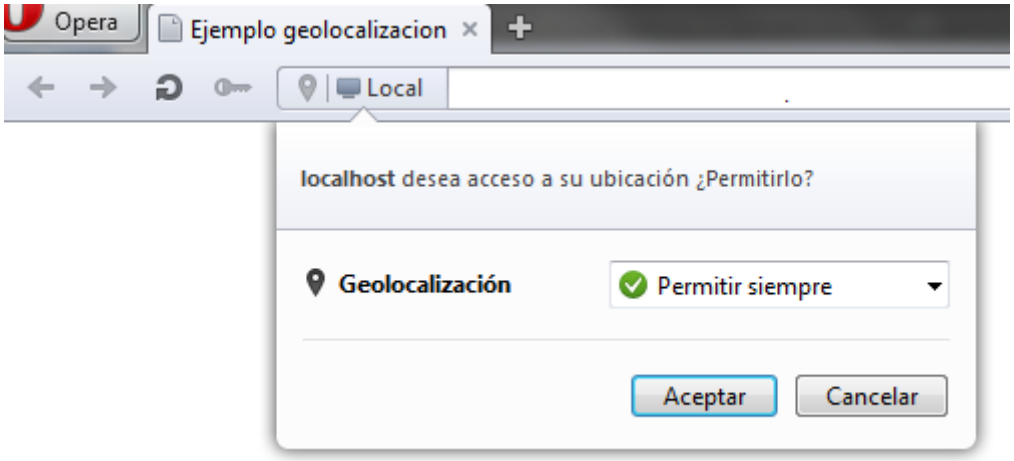
El objetivo, es obtener la información de localización de los diferentes dispositivos también mediante Wifi o GPS. Hoy en día, cualquier dispositivo móvil dispone de estas tecnologías, por lo que tampoco es tan difícil obtener la información de localización a partir de ellas. Para esto, necesitamos que nuestro navegador pueda acceder a la información de ubicación que estas tecnologías proporcionen.

¡IMPORTANTE!
W3C ha elaborado las especificaciones para la API de geolocalización de

JavaScript. Básicamente quien recoge la información de localización no es un servicio web o una base de datos como las aplicaciones existentes, sino que es el propio navegador quien detecta la latitud y longitud.

2) Cómo detectar la ubicación

Como hemos dicho, el navegador será el encargado de detectar la ubicación de cualquier usuario que lo desee, pero para ello, este deberá de tener permiso para acceder a los datos de posicionamiento correspondientes. Por lo tanto, siempre que una página o aplicación Web quiera trabajar con la ubicación del usuario, antes deberá de pedir permiso, y el usuario aceptar, ya que se trata de datos privados.



Una vez el usuario da permiso para conocer su ubicación, la API de JavaScript para esta parte de geolocalización, proporciona los datos correspondientes mediante la llamada al método *getCurrentPosition()*. Esta función recibe como parámetros las funciones que se deben ejecutar en caso de que todo vaya bien, y en caso de que suceda un error, así como tres parámetros opcionales más que veremos a continuación:

La función quedaría de la siguiente manera: ***getCurrentPosition(exito, error, { timeout: valor, maximumAge: valor, enableHighAccuracy: boolean})***

- **exito():** Función que se ejecutará en caso de que todo vaya bien.
- **error():** Función que se ejecutará en el caso de que suceda algún error.
- **timeout:** Indica cuál es el tiempo máximo de espera para obtener las coordenadas del usuario, en milisegundos.
- **maximumAge:** Indica la máxima antigüedad de la posición cacheada del usuario, en milisegundos.
- **enableHighAccuracy:** Indica si permitimos la detección mediante GPS en caso de estar disponible, es un valor *true* o *false*.

Una vez llamada la función, los datos son recibidos desde el proveedor de localización y los tenemos disponibles mediante las propiedades del objeto *position* devuelto por el método *getCurrentPosition()*. Las propiedades de dicho objeto son las siguientes:

coords.latitude: Latitud, en grados decimales

coords.longitude: Longitud, en grados decimales

coords.altitude: Altitud, en metros

coords.accuracy: Nivel de precisión de las coordenadas, en metros.

coords.altitudeAccuracy: Nivel de precisión de la altitud, en metros.

coords.heading: Dirección en la que se desplaza el aparato que proporciona las coordenadas, en grados, comenzando desde el norte y contando en el sentido de las agujas del reloj.

coords.speed: Velocidad de desplazamiento del aparato, en metros por segundo.

timestamp: Momento en que la posición fue adquirida, en formato timestamp.

Anotación

Aparte del método *getCurrentPosition()*, disponemos de otros dos métodos más que son muy similares, pero que se utilizan más en aplicaciones que van siguiendo una ruta o que necesitan saber la ubicación constantemente. Estos métodos son los siguientes:

watchPosition(): Método similar a *getCurrentPosition()*, recibe los mismos parámetros, pero en este caso en el momento que lo invocamos comienza a consultar la ubicación de manera periódica según se va moviendo el usuario.

clearWatch(): Método que detiene el proceso lanzado por *watchPosition()*. El parámetro que recibe, es el identificador del proceso *watchPosition()* que queremos detener.

Estas funciones, están más orientadas a dispositivos móviles que necesitan conocer la ubicación del dispositivo en todo momento.

Ejemplo

En el siguiente ejemplo vamos a crear una página que nos devolverá toda la información disponible sobre la ubicación del usuario que la ejecute. El código será el siguiente:

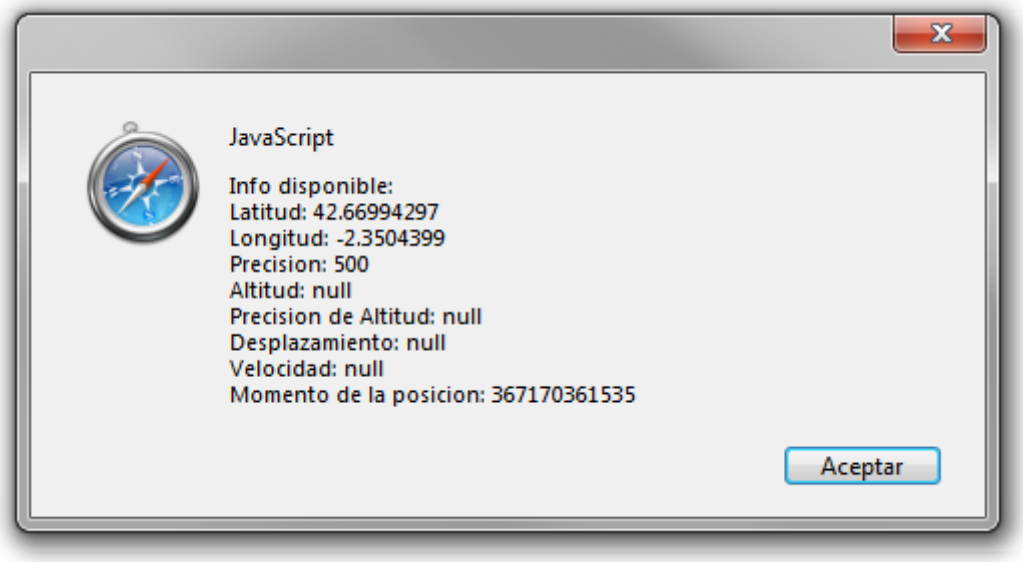
EjemploGeo.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo geolocalizacion</title>
</head>
<script type="text/javascript">

    function exito(position) {
        var cadena = "Info disponible:\n";

        cadena += 'Latitud: ' + position.coords.latitude + '\n';
        cadena += 'Longitud: '+position.coords.longitude + '\n';
        cadena += 'Precision: '+position.coords.accuracy + '\n';
        cadena += 'Altitud: '+position.coords.altitude + '\n';
        cadena += 'Precision de Altitud: '+position.coords.altitudeAccuracy + '\n';
        cadena += 'Desplazamiento: '+position.coords.heading + '\n';
        cadena += 'Velocidad: '+position.coords.speed + '\n';
        cadena += 'Momento de la posicion: '+position.timestamp + '\n';

        alert(cadena);
    }

    function error() {
        alert('Ha ocurrido un error');
    }

    if(navigator.geolocation)
    {
        navigator.geolocation.getCurrentPosition(exito, error);
    }
    else
    {
        alert('Lo sentimos, el navegador no soporta Geolocalizacion');
    }

</script>
<body>
</body>
</html>
```

Vemos como lo primero que hacemos es comprobar si el navegador soporta o no la opción de geolocalización mediante la propiedad *navigator.geolocation*. En caso de que no lo acepte, se mostrará una alerta comunicándolo, pero en caso de que sí se acepte, se invocará al método *getCurrentPosition()*. En este caso solo le hemos indicado cuáles serán las funciones que deberá ejecutar en caso de que todo vaya bien, y en caso de que ocurriera un error, no hemos definido ninguno de los parámetros opcionales.

Por lo tanto, si ocurre algún error, se ejecutará la función *error()* que simplemente mostrará una alerta y en el caso de que todo vaya bien, se ejecutará la función *exito()*. Esta función, recibe como parámetro el objeto *position* que contiene la información sobre la ubicación del usuario, y lo que hacemos es extraer dicha información mediante las propiedades disponibles y mostrarlas en una alerta.

El resultado que veremos en pantalla si permitimos mostrar nuestra ubicación será el siguiente:



Vemos como la alerta nos muestra todos los datos disponibles sobre la ubicación del usuario que lo ha permitido. Algunos de los datos aparecen como *null*, eso es porque en alguna localización y depende con que dispositivos, no es posible recoger algunos de los datos.

3) Cómo diferenciar el error ocurrido

En el apartado anterior, hemos visto que cuando ocurre un error en la geolocalización, se lanzaba una función que mostraba una alerta comunicándolo. Pues bien, existe una propiedad que define de manera más concreta el error ocurrido, y de esta manera nos será más fácil solucionarlo o saber por qué no funciona. El objeto *error*, dispone de la propiedad *code*, que devuelve un valor u otro en función de cuál sea el error ocurrido, dispone de cuatro tipos de error diferentes:

error.code = 0: Error desconocido, si ha fallado por cualquier otra cosa.

error.code = 1: Permiso denegado, cuando el usuario no comparte la posición y por tanto deniega el acceso a su localización.

error.code = 2: Posición no disponible, si se ha perdido la conexión o no se puede contactar con los satélites que realizan el posicionamiento.

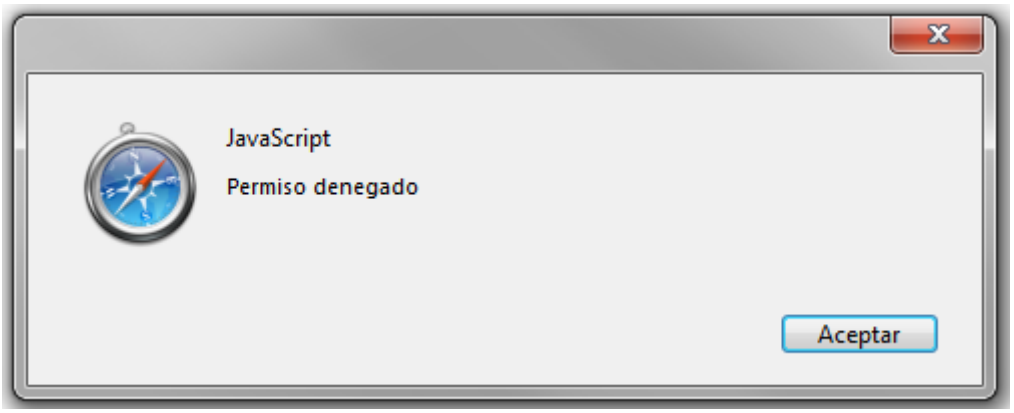
error.code = 3: Timeout, si se ha tardado demasiado en calcular la posición del usuario.

La siguiente función, será la que se ejecute en el momento que suceda cualquier error, y mostrará una alerta con el error concreto que suceda. El código sería el siguiente:

```
23    function error(error) {
24        switch (error.code)
25        {
26            case 0:
27                alert("Error deconocido");
28                break;
29            case 1:
30                alert("Permiso denegado");
31                break;
32            case 2:
33                alert("Posicion no disponible");
34                break;
35            case 3:
36                alert("Se ha superado el tiempo de espera");
37                break;
38            default:
39                alert("Error deconocido");
40                break;
41
42        }
43    }
```

Vemos como la función recibe el objeto *error*, y luego mostramos una alerta u otra en función de cuál sea el valor de la propiedad *error.code*.

Por ejemplo, en el caso de que no se acepte que se conozca la ubicación, el sistema lanzaría esta alerta:



En lugar de una alerta común para todos los errores, de esta manera tendremos más definido el motivo del error.

4) Posicionarnos con Google Maps

Hemos visto ya, cómo obtener la información de la ubicación del usuario conectado, y cómo controlar los errores que puedan generarse. Lo que nos falta ahora, es utilizar dichos datos en nuestra página para poder ofrecerle al usuario algo más que unos simples números. En este caso, vamos a ver cómo hacer que aparezca la localización del usuario conectado sobre un mapa de Google Maps.

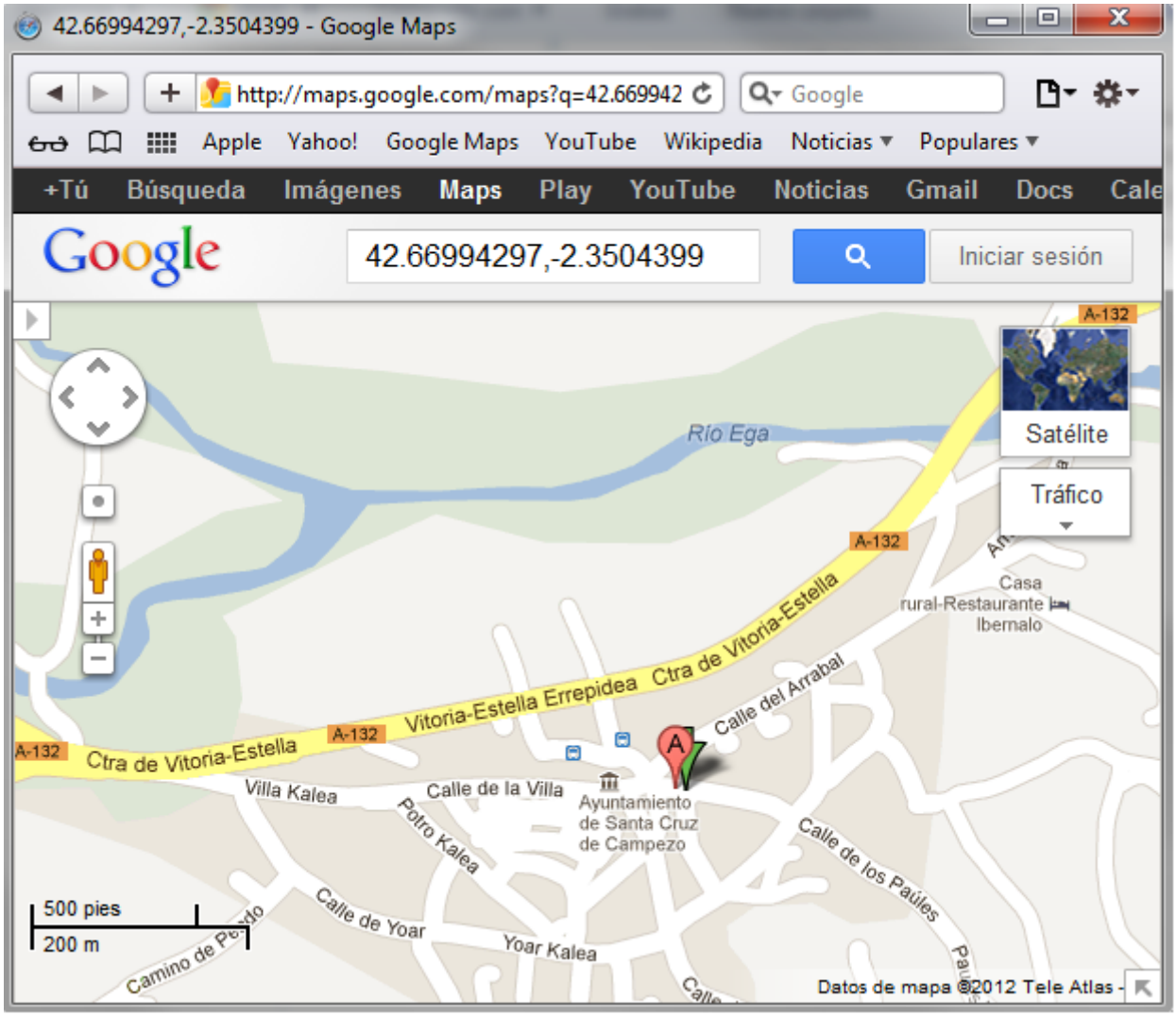
En primer lugar vamos a ver la forma más sencilla que tenemos para conseguirlo. Lo que hacemos es cargar la página de Google Maps, añadiéndole los parámetros de posición, que serán las

coordenadas que nos devuelva el objeto *position* como hemos visto anteriormente. El código sería el siguiente:

EjemploGeoM.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo geolocalización</title>
</head>
<script type="text/javascript">
    function exito(position) {
        var longitud = position.coords.longitude;
        var latitud = position.coords.latitude;
        document.location = 'http://maps.google.com/maps?q='+latitud+','+longitud;
    }

    function error(error) {
        switch (error.code)
        {
            case 0:
                alert("Error deconocido");
                break;
            case 1:
                alert("Permiso denegado");
                break;
            case 2:
                alert("Posicion no disponible");
                break;
            case 3:
                alert("Se ha superado el tiempo de espera");
                break;
            default:
                alert("Error deconocido");
                break;
        }
    }

    if(navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(exito, error);
    }
    else
    {
        alert('Lo sentimos, el navegador no soporta Geolocalizacion');
    }
</script>
<body>
</body>
</html>
```

Como vemos, si el navegador soporta geolocalización y no se produce ningún error, se recogen las coordenadas de longitud y latitud y se hace referencia a la página de Google Maps pasándole dichos valores en la propia URL como vemos en la imagen. De esta manera, detectamos la ubicación y la cargamos sobre un mapa de Google como vemos a continuación:



Vemos como aparece la ubicación del usuario marcada en un mapa de Google Maps utilizando las coordenadas proporcionadas por el sistema de localización.

Aparte de esta opción, **Google Maps cuenta con una API propia para JavaScript** que nos ofrece más posibilidades a la hora de cargar el mapa con la ubicación detectada. Vamos a ver su funcionamiento básico.

En primer lugar, necesitamos invocar a la librería JavaScript de Google Maps desde el cabecero de la página de la siguiente manera:

```
<script language="JavaScript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

Una vez invocada, ya podemos hacer uso de sus métodos y propiedades. El primer paso será crear un objeto de tipo *LatLng* que definirá el centro del mapa con las coordenadas que nos proporciona el localizador, el código será el siguiente:

```
21          var longitud = position.coords.longitude;
22          var latitud = position.coords.latitude;
23
24          var centro = new google.maps.LatLng(latitud, longitud);
```

Una vez creado, vamos a definir las características del mapa haciendo uso de las propiedades que nos ofrece la librería, el código será el siguiente:

```
26        var opciones = {
27            zoom: 15,
28            center: centro,
29            mapTypeId: google.maps.MapTypeId.ROADMAP
30        };
```

Vemos que hemos definido el zoom para el mapa en 15, puede ir desde 1 hasta 18, el centro del mismo será la ubicación que hemos definido con anterioridad y que será la del usuario conectado, y hemos definido también el tipo de mapa.

El siguiente paso será crear el elemento mapa con las opciones que hemos definido, para ello, crearemos un objeto de tipo *Map*, y lo cargaremos sobre el *DIV* que tenemos en nuestra página y que tiene como identificador "mapa", el código será el siguiente:

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

Con esto ya tendríamos nuestro mapa con las características indicadas cargado sobre el elemento *DIV* que hemos creado para ello. Aparte de esto, tenemos la opción de colocar marcadores sobre dicho mapa, y lo haremos de la siguiente manera:

```
34        var marcador = new google.maps.Marker({
35            position: centro,
36            map: map,
37            title:"Usted esta aqui"
38        });
```

Vemos como hemos creado un marcador que estará colocado justamente en la ubicación del usuario conectado, y que tendrá como título "Usted está aquí", que se mostrará al colocarnos sobre él. Para ello, hemos creado un objeto de tipo *Marker* y mediante el atributo *map* le hemos indicado sobre qué mapa debe de colocarlo.

El código completo de la función que se ejecutaría si todo funciona bien quedaría así:

```
19    function exito(position) {
20
21        var longitud = position.coords.longitude;
22        var latitud = position.coords.latitude;
23
24        var centro = new google.maps.LatLng(latitud, longitud);
25
26        var opciones = {
27            zoom: 15,
28            center: centro,
29            mapTypeId: google.maps.MapTypeId.ROADMAP
30        };
31
32        var map = new google.maps.Map(document.getElementById("mapa"), opciones);
33
34        var marcador = new google.maps.Marker({
35            position: centro,
36            map: map,
37            title:"Usted esta aqui"
38        });
39    }
```

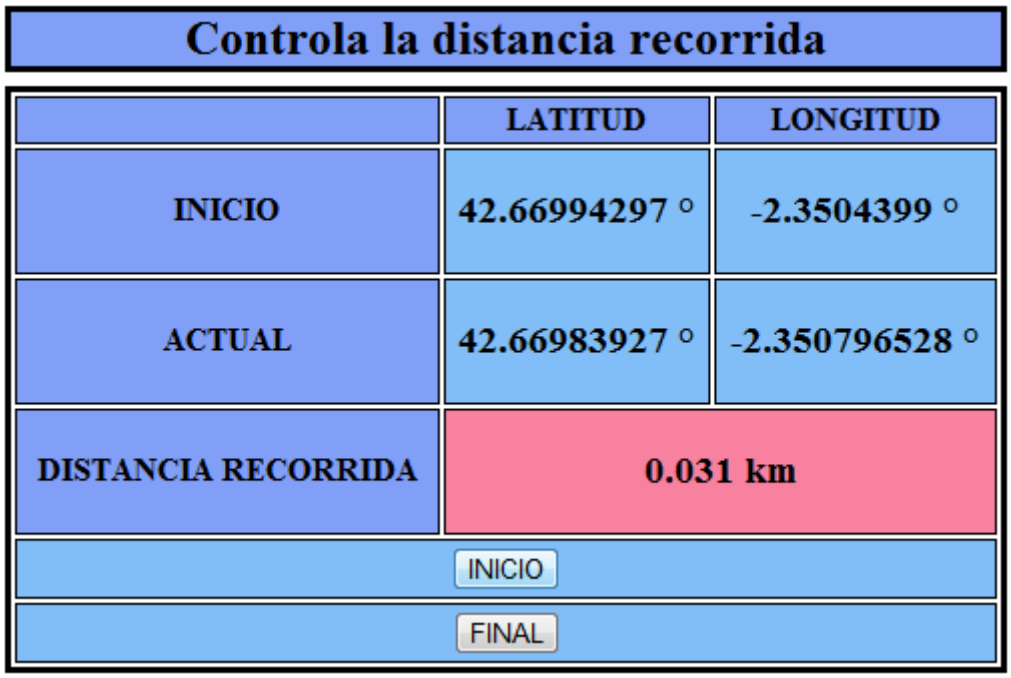
El resultado en el navegador sería el siguiente:

Vemos que el mapa se ha generado con las características indicadas sobre nuestro elemento *DIV* con borde negro.

Ejercicios

Ejercicio 1: Crear un sistema que nos vaya calculando la distancia que llevamos recorrida mientras nos desplazamos

Vamos a crear una página, la cual, mientras nos desplazamos, nos indicará las coordenadas de inicio, las coordenadas actualizadas en cada momento y la distancia que llevamos recorrida en función de la diferencia entre las coordenadas iniciales y las actuales. Lo que queremos conseguir es algo similar a lo que vemos en la imagen:



Pasos a seguir

1. En primer lugar, crearemos la tabla donde iremos actualizando los datos.
2. En el momento que pulsemos el botón de INICIO, se mostrarán las coordenadas de inicio, y se pondrá en marcha el proceso para ir actualizando las coordenadas actuales.
3. La distancia recorrida, se mostrará en kilómetros, y se actualizará a la vez que las coordenadas. La función que calcula la distancia en kilómetros partiendo de las coordenadas, la tenéis disponible para descargar.
4. En el momento que pulsemos el botón de FINAL, se dejarán de actualizar las coordenadas y la distancia recorrida.