# Requerimientos generales para proyectos

Cada proyecto debe incluir la siguiente documentación

* Diagrama de Casos de Uso
* Diagrama de Base de Batos
* Diagrama de Clases

El Product Owner podrá agregar más requerimientos en el BackLog de cada proyecto. El Scrum Master en conjunto con el TEAM deben analizar los requerimientos y transformarlos en tareas dentro del Sprint Backlog

**Los sprint deben ser de una semana cada uno.**

El primer Sprint comienza la semana del 26 de diciembre y debe incluir de forma obligatoria los Casos de uso.

Segundo Sprint debe incluir de forma obligatoria el diagrama de base de datos

El diagrama de clases puede ser entregado en el 3° o 4° sprint.

El examen es la entrega del proyecto funcional en la semana del 23 de enero.

* Notas
* Unidad I
* N1: Presentación inicial.
* N2: Nota trabajo en equipo (es un 1 o un 7)
* Unidad II
* N1: Promedio Sprint 1 y Sprint 2
* N2: Promedio Sprint 3 y Sprint 4
* N3: Nota Prueba Servicios
* N4: Nota sorpresa

Proyecto Alojamiento

* Crear sistema para arriendo por noche de habitación, pieza, casa, cabaña, depto., para cualquier tipo de alojamiento.
* Debe quedar un historial de personas que realizan el arriendo.
* La aplicación debe dejar buscar alojamientos cercanos de acuerdo a su geolocalización.
* Las personas que ingresar el alojamiento disponible deben subir fotos e indicar si es necesario abono para reservar y su valor.
* De ser necesario el sistema debe permitir abonar un monto para realizar la reserva.
* Debe existir un sistema de calificación entre ambas partes, tanto para el alojamiento y su dueño como para el huesped.

# Subir Imagen con MVC

**Creamos nuestro controlador**  
El cual contiene 2 acciones, una que la vista que contendra al formulario de subida y la otra es la acción encargada de subir el archivo.

public class HomeController : Controller

{

*//*

*// GET: /Home/*

public ActionResult Index()

{

return View();

}

[HttpPost]

public void Subir(HttpPostedFileBase file)

{

if (file == null) return;

string archivo = (DateTime.Now.ToString("yyyyMMddHHmmss") + "-" + file.FileName).ToLower();

file.SaveAs(Server.MapPath("~/Uploads/" + archivo));

}

}

Como pueden ver, el código de la acción **Subir** no es nada del otro mundo y contiene la etiqueta **[HttpPost]** para indicarle que solo reciba peticiones del tipo POST.

## Creando la vista

@using (Html.BeginForm("subir", "home", FormMethod.Post, new { enctype = "multipart/form-data" }))

{

<input type="file" name="file" /> <button type="submit">Subir archivo</button>

}

Creamos una etiqueta de formulario y le especificamos que la información a enviar va a contener archivos a subir **"multipart/form-data"**. El nombre del control, en este caso el input file debe ser igual que el parametro que recibe nuestra acción **name="file"**.

**Analizando el código de la acción "Subir"**

string archivo = (DateTime.Now.ToString("yyyyMMddHHmmss") + "-" + file.FileName).ToLower();

file.SaveAs(Server.MapPath("~/Uploads/" + archivo));

Guardo el nombre del archivo actual concatenado la **fecha y hora**, porque si vuelvo a subir el mismo archivo este lo **va a querer remplazar**.

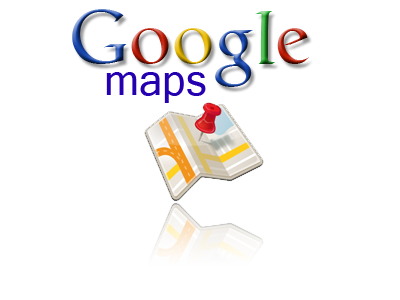
Luego le indico que mueva el archivo actual a la siguiente ruta. No te olvides **crear una carpeta Uploads** dentro de tu proyecto.

EJEMPLO



# Google Maps API v.3.0 y ASP.NET MVC 3 con Razor

[**ASP.NET MVC 3**](https://www.returngis.net/tag/asp-net-mvc-3/), [**Google Maps API**](https://www.returngis.net/tag/google-maps-api/) [**31 Comments**](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/#disqus_thread)



Gracias a todas las [APIs](http://code.google.com/intl/es-ES/more/" \o "Directorio de APIs de Google" \t "_blank) que nos facilita Google, podemos hacer uso prácticamente de todos los servicios que ofrece la compañía a día de hoy. En este post en concreto me gustaría hablaros de cómo utilizar el API de Google Maps en su versión 3.0 para generar mapas dinámicos con una enorme cantidad de funciones. Además, para rizar más el rizo utilizaré [ASP.NET MVC 3 con Razor](https://www.returngis.net/2011/02/razor-view-engine-para-asp-net-mvc/) para crear una llamada síncrona y otra asíncrona con la ayuda de [JQuery](http://jquery.com/" \o "Página oficial de JQuery" \t "_blank).

## REFERENCIAS

Para utilizar el API de Google Maps son necesarias las siguientes referencias:

[?](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/)

|  |  |
| --- | --- |
| 1  2 | <script src="@Url.Content("~/Scripts/jquery-1.4.4.min.js")" type="text/javascript"></script>  <script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script> |

***Nota****: Cuando utilizas Google Maps****siempre debe especificarse si vas a utilizar un sensor****, por ejemplo un GPS, para determinar la ubicación del usuario (importante para los dispositivos móviles). En este caso estableceremos el valor del parámetro sensor a false ya que no entra este escenario dentro del post*

## ¿POR DÓNDE EMPIEZO?

Lo primero que debemos tener en cuenta es que podemos**ubicar marcas de dos formas distintas**: **Indicando las coordenadas del sitio o recuperando las coordenadas a través de una dirección postal**. Sin embargo, en el segundo caso tenemos una limitación por parte de la API, la cual sólo nos permite solicitar unas 10 direcciones por consulta, lo cual limita nuestro campo de acción. En este post utilizaremos coordenadas para poder agregar tantas marcas como sea necesario  Para ello, he creado un objeto de prueba que almacenará todas las marcas y un repositorio,  también de prueba, que nos devolverá tres objetos dentro de una lista para la demo.

## OBJETO DE PRUEBA

[?](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | namespace GoogleMapsASPNETMVC3.Models  {      public class GoogleMarker      {          public string SiteName { get; set; }          public double Latitude { get; set; }          public double Longitude { get; set; }          public string InfoWindow { get; set; }      }  } |

## REPOSITORIO DE PRUEBA

[?](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | using System.Collections.Generic;    namespace GoogleMapsASPNETMVC3.Models  {      public class MarkerRepository      {            public IList<GoogleMarker> GetMarkers()          {              var googleMarkers = new List<GoogleMarker>                                      {                                          new GoogleMarker                                              {                                                  SiteName = "Jardines de Sabatini",                                                  Latitude = 40.421749,                                                  Longitude = -3.713994,                                                  InfoWindow = "InfoWindow de los Jardines de Sabatini"                                              },                                          new GoogleMarker                                              {                                                  SiteName = "Campo del Moro",                                                  Latitude = 40.419658,                                                  Longitude = -3.718801,                                                  InfoWindow = "InfoWindow del Campo del Moro"                                              },                                          new GoogleMarker                                              {                                                  SiteName = "Parque de la Cornisa",                                                  Latitude = 40.413254,                                                  Longitude = -3.716483,                                                  InfoWindow = "InfoWindow del Parque de la Cornisa"                                              }                                      };                return googleMarkers;          }      }  } |

## LLAMADA SÍNCRONA

Para la llamada síncrona lo que vamos a hacer es solicitar la lista de marcas y pasarla a la vista para poder asociarla a la propiedad Model.

[?](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | [HttpGet]  public ActionResult Sync()  {       return View(\_markerRepository.GetMarkers());  } |

Una vez en la vista agregaremos el siguiente código:

[?](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | @model IEnumerable<GoogleMapsASPNETMVC3.Models.GoogleMarker>  @using System.Threading;  @using System.Globalization;  @{      ViewBag.Title = "Sync";      Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");  }  <script language="javascript" type="text/javascript">      $(document).ready(function () {      var bounds = new google.maps.LatLngBounds();      var options = {          zoom : 14,          mapTypeId: google.maps.MapTypeId.TERRAIN      };      var googleMap = new google.maps.Map($("#map")[0],options);      var infoWindow = new google.maps.InfoWindow({ content: "Cargando..." });      @foreach (var marker in Model)      {      <text>          var point = new google.maps.LatLng(@marker.Latitude, @marker.Longitude);          bounds.extend(point);            var marker = new google.maps.Marker({                  position: point,                  map: googleMap,                  icon:'/Content/images/cloud\_marker.png',                  html: '@marker.InfoWindow'              });            google.maps.event.addListener(marker, "click", function () {                  infoWindow.setContent(this.html);                  infoWindow.open(googleMap, this);              });      </text>      }      googleMap.fitBounds(bounds);      });  </script>  <h2>      Sync</h2>  <div id="map" style="width: 800px; height: 500px;"></div> |

Lo primero que hacemos en la parte javascript es la inicialización de unas variables bases como son **bounds** donde vamos a insertar cada uno de los puntos de coordenadas, **googleMap** que asignará el espacio dedicado al mapa a un div con id “map” situado abajo del todo e **infoWindow** que nos servirá para asociar un pop up a cada marca del mapa.

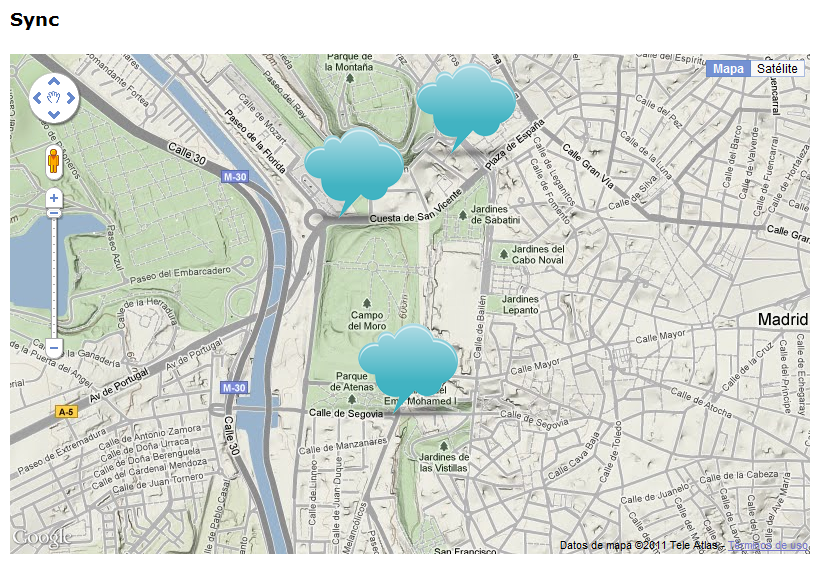
Posteriormente utilizamos la @ para inicializar un foreach, ya que estamos utilizando el motor de vistas Razor, y recorremos cada una de las marcas ubicadas en el Model del tipo GoogleMarker. Como vamos a mezclar código javascript con variables de servidor, debemos hacer el uso de <text></text> para que no interprete todo el código dentro del foreach como código de servidor.

Los pasos que vamos a realizar en cada iteración es la creación de cada conjunto de coordenadas en el mapa a través de **google.maps.LatLng** para acto seguido usar el resultado como la posición de la marca en el objeto **google.maps.Marker**en el mapa asociado. En este último podemos asociar distintos tipos de opciones como por ejemplo el icono que queremos mostrar para dicha marca, sombreados, etcétera. En este caso he modificado la imagen por una nube

Además podemos asociar un evento a cada marca para mostrar el típico pop up que aparece cuando haces clic sobre una de ellas. Para ello utilizaremos **google.maps.event.Listener** donde crearemos la asociación con la clase InfoWindow instanciada al inicio del script.

Por otro lado, si os fijáis de nuevo en el código anterior, he cambiado **la cultura a en-US**. Esto es debido a que es necesario respetar el signo de separación de los decimales en las coordenadas ya que, de lo contrario, la latitud y longitud recuperadas de la propiedad Model se pintarían con comas y no se mostraría correctamente.

Como resultado obtendríamos la siguiente imagen



## LLAMADA ASÍNCRONA

Para la llamada asíncrona el escenario cambia:

[?](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | @{      ViewBag.Title = "Async";  }  <script language="javascript" type="text/javascript">      $(document).ready(function () {            var bounds = new google.maps.LatLngBounds();          var options = {              zoom: 14,              mapTypeId: google.maps.MapTypeId.TERRAIN          };          var googleMap = new google.maps.Map($("#map")[0], options);          var infoWindow = new google.maps.InfoWindow({ content: "Cargando..." });            $.ajax({              type: "POST",              url: "GetMarkersAsync",              datatype: "json",              success: function (data) {                  for (var i = 0; i < data.length; i++) {                        var point = new google.maps.LatLng(data[i].Latitude, data[i].Longitude);                        bounds.extend(point);                        var marker = new google.maps.Marker({                          position: point,                          map: googleMap,                          html: data[i].InfoWindow                      });                        google.maps.event.addListener(marker, "click", function () {                          infoWindow.setContent(this.html);                          infoWindow.open(googleMap, this);                      });                  }              }          });            googleMap.fitBounds(bounds);      });  </script>  <h2>      Async</h2>  <div id="map" style="width: 800px; height: 500px;">  </div> |

En este segundo caso estamos utilizando JQuery para la llamada Ajax a una acción llamada **GetMarkersAsync**, la cual nos devolverá un listado de objetos en [JSON](http://es.wikipedia.org/wiki/JSON) con el mismo formato que nuestro objeto de prueba. Para ello, crearemos una acción HttpPost que retorne dicho listado basado en la clase GoogleMarker.

[?](https://www.returngis.net/2011/02/google-maps-api-v-3-y-asp-net-mvc-3-con-razor/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | [HttpPost]  public ActionResult GetMarkersAsync()  {      return Json(\_markerRepository.GetMarkers());  } |

En este caso el resultado es el mismo que en la llamada síncrona a excepción de los iconos utilizados en las marcas, ya que en el segundo ejemplo he omitido la propiedad icon para que utilice el icono por defecto de Google.

# HTML 5] Trabajando con la geolocalización

Otra de las cosas nuevas que trae [HTML5](http://dev.w3.org/html5/spec/Overview.html) es la geolocalización, con ella podemos determinar en qué lugar (coordenadas) se encuentra un usuario.

Determinar la posición de un usuario es bastante sencillo, y en este post les voy a mostrar como lo podemos hacer, y como complemento, mostraremos dicha ubicación en un mapa utilizando [Bing Maps](http://www.bing.com/maps/).

a sintaxis general para trabajar con la geolocalización es:

navigator.geolocation.getCurrentPosition(GetPosition,funcionError, options);

Entonces se utiliza el método **getCurrentPosition** de **geolocation**, donde el primer parámetro es la función que se llama si todo nos sale bien, la segunda será la función que se ejecuta cuando algo falla y por último podemos definir un set de opciones.

Algunas de las opciones que podemos definir son:

* enableHighAccuracy: De tipo booleano, indica si se debe obtener la posición más acertada.
* timeout: indica el tiempo permitido en milisegundos para intentar obtener la posición.

Una vez obtenida la posición accedemos a las coordenadas (latitud y longitud) mediante un objeto position, el cual contiene dicha información.

Ahora que ya tenemos algo (no toda) de teoría manos a la obra, la idea como comente antes es mostrar en donde estamos en un mapa, y por otro lado vamos a tener dos campos de texto para mostrar la latitud y la longitud.

**Escribir el HTML:**

De una manera muy simple tenemos los dos campos de texto para mostrar la latitud y la longitud, y por otro lado un div el cual será nuestro mapa:

<p>Geolocalizacion en HTML </p>

<p>Latitud: <input type="text" id="txtLatitud" /> </p>

<p>Longitud: <inputtype="text"id="txtLongitud" /> </p>

<hr/>

<div id='mapDiv' style="position:relative; width:600px; height:600px;"></div>

**Referenciar los JavaScript**

Ahora se debe referenciar el archivo js de [jQuery](http://jquery.com/) y el [API](http://www.microsoft.com/maps/developers/web.aspx) de [Bing Maps](http://www.bing.com/maps/):

<script type="text/javascript" src="js/jquery-1.7.1.min.js"></script>

<script type="text/javascript" src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=7.0"></script>

**Definir las opciones:**

Definir en un variable las opciones que vamos a utilizar:

var options = {

enableHighAccuracy: true,

timeout: 45000

};

**Obtener posición:**

navigator.geolocation.getCurrentPosition(GetPosition,funcionError, options);

Luego definimos la función funcionError:

function funcionError(error)

{

alert(error.message);

}

Y la función **GetPosition**:

function GetPosition(posicion)

{

var latitud = posicion.coords.latitude;

var longitud = posicion.coords.longitude;

$("#txtLatitud").val(latitud);

$("#txtLongitud").val(longitud);

GetMap(latitud,longitud);

}

En esta función lo interesante es que tenemos la variable posicion, y es por medio de ella que accedemos a los valores de latitud y longitud respectivamente, finalmente se observa el llamado a la función GetMap, la cual recibe la latitud y la longitud para poder ubicarse en el mapa, en resumen la función GetMap sería:

function GetMap(a,b)

{

var map = new Microsoft.Maps.Map(document.getElementById("mapDiv"),

{

credentials: "######",

center: new Microsoft.Maps.Location(a, b),

mapTypeId: Microsoft.Maps.MapTypeId.road,

zoom: 8

});

}

Pero esa parte ya nos la da es el [API de Bing](http://www.microsoft.com/maps/developers/web.aspx), y en donde dice ### debemos colocar el número dado por el API.

Listo, ahora un tema importante es que cuando se intente obtener la localización del usuario, siempre el usuario debe permitir realizar dicha acción, es por eso que al cargar la página vemos que se muestra un mensaje como (mensaje tomado de [Internet Explorer 9](http://windows.microsoft.com/es-ES/internet-explorer/products/ie/home)):

Y allí el usuario puede decidir aceptar o rechazar la petición de localización, en caso afirmativo lo que se va a obtener es:

