|  |  |
| --- | --- |
| Geo Parking | Documentación de Capacitación  2014 |

Documentación de capacitación utilizada en el desarrollo del producto Geo Parking.

# Control de la documentación

### Control de la Configuración.

|  |  |
| --- | --- |
| **Título:** | Documentación de Capacitación |
| Referencia: | GeoP\_Proyecto\_Doc\_Capacitacion.docx |
| Autores: | Lucas Toneatoo |
| Fecha: | 02/10/2014 |

### Histórico de Versiones.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versión** | **Fecha** | **Estado** | **Responsable** | **Cambios** |
| 1.0 | 02/10/2014 | Pendiente de Aprobación | Lucas Toneatto | - |
|  |  |  |  |  |
|  |  |  |  |  |

Contenido

[Control de la documentación 2](#_Toc400051988)

[Control de la Configuración. 2](#_Toc400051989)

[Histórico de Versiones. 2](#_Toc400051990)

[Introducción 5](#_Toc400051991)

[Entity Framework (Code First) 6](#_Toc400051992)

[1. Crear la aplicación 6](#_Toc400051993)

[2. Crear el modelo 7](#_Toc400051994)

[3. Crear un contexto 7](#_Toc400051995)

[4. Leer y escribir datos 9](#_Toc400051996)

[¿Dónde están mis datos? 10](#_Toc400051997)

[5. Tratar los cambios en el modelo 11](#_Toc400051998)

[Anotaciones de datos 13](#_Toc400051999)

[7. API fluida 14](#_Toc400052000)

[Resumen 16](#_Toc400052001)

[Referencias 16](#_Toc400052002)

[API Google Maps V3 17](#_Toc400052003)

[Cómo obtener una clave de API 17](#_Toc400052004)

[Creación de un simple mapa (ver uso de Api key) 18](#_Toc400052005)

[COMO CARGAR EL API EN FORMA ASINCRONA 18](#_Toc400052006)

[Tipos de Mapas 19](#_Toc400052007)

[El objeto "Map" 20](#_Toc400052008)

[Eventos 20](#_Toc400052009)

[Eventos de la Interfaz de Usuario 21](#_Toc400052010)

[Cambios de Estado 21](#_Toc400052011)

[Ejemplo de Eventos y cambios de Estado 21](#_Toc400052012)

[Descripción general de los controles del mapa 22](#_Toc400052013)

[Cómo añadir controles al mapa 23](#_Toc400052014)

[Un ejemplo de un control personalizado 23](#_Toc400052015)

[Marcadores 25](#_Toc400052016)

[Animaciones 26](#_Toc400052017)

[Poligonos 27](#_Toc400052018)

[Info Windows (Información de marcadores) 28](#_Toc400052019)

[Calculo de Ruta a través de dos puntos 29](#_Toc400052020)

[Calculo de ruta con indicaciones al costado 31](#_Toc400052021)

[Referencias 33](#_Toc400052022)

[Ajax 34](#_Toc400052023)

[Funciones para AJAX en JQuery 37](#_Toc400052024)

[Referencias 38](#_Toc400052025)

[Ejemplos 38](#_Toc400052026)

[Json 39](#_Toc400052027)

[Sintaxis JSON 39](#_Toc400052028)

[Objetos 39](#_Toc400052029)

[Array 39](#_Toc400052030)

[Herramientas de Desarrollo 39](#_Toc400052031)

[Formas de Generación de JSON 40](#_Toc400052032)

[Método ToJSON() con JavaScriptSerializer 40](#_Toc400052033)

[Serializacion utilizando JSON.Net 41](#_Toc400052034)

[JSON en Entity Framework 41](#_Toc400052035)

[Aplicación práctica en GeoParking 42](#_Toc400052036)

[JSON y JavaScript 44](#_Toc400052037)

[Referencias 45](#_Toc400052038)

[Unit Test 46](#_Toc400052039)

[Introducción 46](#_Toc400052040)

[Investigación 46](#_Toc400052041)

[Resultados 46](#_Toc400052042)

[Bibliografia 49](#_Toc400052043)

# Introducción

El siguiente documento es un resumen de toda la documentación producida por el equipo como medio de capacitación en las distintas herramientas y tecnologías utilizadas en el desarrollo del producto GeoParking.

Aquí se hará referencia a cada una de las distintas capacitaciones que ha implementado el equipo, y de las mismas se abordaran los temas más importantes y por sobre todo los que fueron implementados en el proyecto.

Este no intenta ser un manual profundo de cada tecnología sino una guía de conocimientos teóricos y prácticos aplicables al desarrollo del sistema GeoParking.

# Entity Framework (Code First)

Entity Framework es un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos. Los arquitectos y programadores de aplicaciones orientadas a datos se han enfrentado a la necesidad de lograr dos objetivos muy diferentes. Deben modelar las entidades, las relaciones y la lógica de los problemas empresariales que resuelven, y también deben trabajar con los motores de datos que se usan para almacenar y recuperar los datos. Los datos pueden abarcar varios sistemas de almacenamiento, cada uno con sus propios protocolos; incluso las aplicaciones que funcionan con un único sistema de almacenamiento deben equilibrar los requisitos del sistema de almacenamiento con respecto a los requisitos de escribir un código de aplicación eficaz y fácil de mantener.

Entity Framework permite a los desarrolladores trabajar con datos en forma de objetos y propiedades específicos del dominio, como clientes y direcciones de cliente, sin tener que preocuparse por las tablas y columnas de la base de datos subyacente donde se almacenan estos datos. Con Entity Framework, los desarrolladores pueden trabajar en un nivel mayor de abstracción cuando tratan con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código que en las aplicaciones tradicionales. Dado que Entity Framework es un componente de .NET Framework, las aplicaciones de Entity Framework se pueden ejecutar en cualquier equipo en el que esté instalado .NET Framework a partir de la versión 3.5 SP1.

Antes que nada se necesita tener instalado la extensión o complemento “Entity Framework”, lo cual se realiza de la siguiente manera:

1. **Menú “Herramientas”**
2. **Actualizaciones y Extensiones**
3. **Buscar “Entity Framework”**
4. **Instalar**

Con eso ya tendríamos habilitado todas las funciones que nos provee Entity Framwork; la versión que trae Visual Studio 2012 es la 6.0.

Ahora realizaremos un ejemplo de practica donde se mostrara todo el funcionamiento del framework, para ello necesitaremos crear primero una base de datos, la cual llamaremos “EntityFramework”. Habiendo hecho esto comenzamos a crear la aplicación.

## 1. Crear la aplicación

Para mantener la simplicidad, vamos a generar una aplicación de consola básica que use Code First para el acceso a los datos.

1. Abra Visual Studio
2. **Archivo -> Nuevo -> Proyecto**
3. Seleccione **Windows** en el menú de la izquierda y **aplicación de consola**
4. Escriba **CodeFirstNewDatabaseSample** como nombre
5. Seleccione **Aceptar**

## 2. Crear el modelo

Vamos a definir un modelo muy sencillo con clases. Solo vamos a definirlas en el archivo Program.cs pero en una aplicación real las clases se dividirían en archivos independientes y, posiblemente, también en un proyecto independiente.

Debajo de la definición de clase Program, en Program.cs, agregue las dos clases siguientes.

**public** **class** Blog   
{   
    **public** **int** BlogId { get; set; }   
    **public** **string** Name { get; set; }   
   
    **public** **virtual** List<Post> Posts { get; set; }   
}   
   
**public** **class** Post   
{   
    **public** **int** PostId { get; set; }   
    **public** **string** Title { get; set; }   
    **public** **string** Content { get; set; }   
   
    **public** **int** BlogId { get; set; }   
    **public** **virtual** Blog Blog { get; set; }   
}

Observará que vamos a hacer las dos propiedades de navegación (Blog.Posts y Post.Blog) virtuales. Esto habilita la característica de carga diferida de Entity Framework. La carga diferida significa que el contenido de estas propiedades se cargará automáticamente desde la base de datos al intentar tener acceso.

## 3. Crear un contexto

Ahora es el momento de definir un contexto derivado que representa una sesión con la base de datos, lo que nos permite consultar y guardar los datos. Definimos un contexto que deriva de System.Data.Entity.DbContext y expone un DbSet<TEntity> con tipo para cada clase en nuestro modelo.

Ahora vamos a comenzar a usar tipos de Entity Framework de modo que necesitamos agregar el paquete de EntityFramework NuGet. (ESTO ES SI EL PROYECTO ES NUEVO, USTEDES YA LO TIENEN CONFIGURADO CUANDO LE HACEN UPDATE, PERO LO INSTALARIN ANTES.)

1. **Proyecto –> Administrar paquetes de NuGet**  
   Nota: si no dispone de la opción **Administrar paquetes de NuGet**, debe instalar la [versión más reciente de NuGet](http://visualstudiogallery.msdn.microsoft.com/27077b70-9dad-4c64-adcf-c7cf6bc9970c)
2. Seleccione la pestaña **En línea**
3. Seleccione el paquete **EntityFramework**
4. Haga clic en **Instalar**

Agregue una instrucción using para System.Data.Entity al principio de Program.cs.

**using** System.Data.Entity;

Debajo de la clase Post en Program.cs, agregue el siguiente contexto derivado.

**public** **class** BloggingContext : DbContext   
{

/\*Esta linea permite que el constexto se relacione con la base de datos

\* que nosotros le digamos, en este caso la conexxion del webconfig se

\* va a llamar "BDprueba"

\*/

public BloggingContext() : base("entityFramework") { }

    **public** DbSet<Blog> Blogs { get; set; }   
    **public** DbSet<Post> Posts { get; set; }   
}

Esta es una lista completa de lo que ahora debe contener Program.cs.

**using** System;   
**using** System.Collections.Generic;   
**using** System.Linq;   
**using** System.Text;   
**using** System.Threading.Tasks;   
**using** System.Data.Entity;   
   
**namespace** CodeFirstNewDatabaseSample   
{   
    **class** Program   
    {   
        **static** **void** Main(**string**[] args)   
        {   
        }   
    }   
   
    **public** **class** Blog   
    {   
        **public** **int** BlogId { get; set; }   
        **public** **string** Name { get; set; }   
   
        **public** **virtual** List<Post> Posts { get; set; }   
    }   
   
    **public** **class** Post   
    {   
        **public** **int** PostId { get; set; }   
        **public** **string** Title { get; set; }   
        **public** **string** Content { get; set; }   
   
        **public** **int** BlogId { get; set; }   
        **public** **virtual** Blog Blog { get; set; }   
    }   
   
    **public** **class** BloggingContext : DbContext   
    {   
        **public** DbSet<Blog> Blogs { get; set; }   
        **public** DbSet<Post> Posts { get; set; }   
    }   
}

Ese es todo el código que necesitamos para empezar a almacenar y recuperar datos. Obviamente, hay mucho más detrás y le echaremos un vistazo en un momento aunque primero vamos a verlo en acción.

## 4. Leer y escribir datos

Implemente el método Main en Program.cs como se muestra a continuación. Este código crea una nueva instancia de nuestro contexto y la usa para insertar un nuevo blog. Entonces, usa una consulta LINQ para recuperar todos los blogs de la base de datos ordenados alfabéticamente por el título.

**class** Program   
{   
    **static** **void** Main(**string**[] args)   
    {   
        **using** (var db = **new** BloggingContext())   
        {   
            // Create and save a new Blog   
            Console.Write("Enter a name for a new Blog: ");   
            var name = Console.ReadLine();   
   
            var blog = **new** Blog { Name = name };   
            db.Blogs.Add(blog);   
            db.SaveChanges();   
   
            // Display all Blogs from the database   
            var query = from b **in** db.Blogs   
                        orderby b.Name   
                        select b;   
   
            Console.WriteLine("All blogs in the database:");   
            **foreach** (var item **in** query)   
            {   
                Console.WriteLine(item.Name);   
            }   
   
            Console.WriteLine("Press any key to exit...");   
            Console.ReadKey();   
        }   
    }   
}

Ahora puede ejecutar la aplicación y probarla.

|  |
| --- |
| Escriba un nombre para el nuevo blog: *Blog de ADO.NET* Todos los blogs de la base de datos: Blog de ADO.NET Presione cualquier tecla para salir. |

### ¿Dónde están mis datos?

Por convención, DbContext ha creado una base de datos en su lugar.

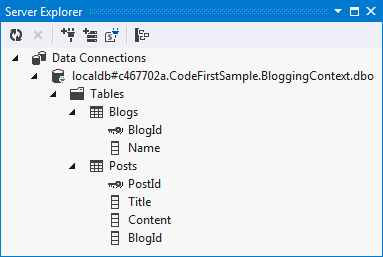
Si una instancia local de SQL Express está disponible (se instala de forma predeterminada con Visual Studio 2010), Code First ha creado la base de datos en esa instancia

Si SQL Express no está disponible, Code First lo intentará y usará [LocalDb](http://msdn.microsoft.com/library/hh510202(v=SQL.110).aspx) (se instala de forma predeterminada con Visual Studio 2012)

Esta base de datos se denomina después del nombre completo del contexto derivado, que en nuestro caso es**CodeFirstNewDatabaseSample.BloggingContext**

Estas son solo las convenciones predeterminadas y hay varias maneras de cambiar la base de datos que Code First usa; también hay más información disponible en el tema **Cómo detecta DbContext el modelo y la conexión de base de datos** .

Puede conectarse a esta base de datos mediante el Explorador de servidores en Visual Studio

1. **Ver -> Explorador de servidores**

DbContext determinó las clases que se incluyen en el modelo examinando las propiedades de DbSet que definimos. Después, usa el conjunto predeterminado de convenciones de Code First para determinar los nombres de tabla y columna, determinar los tipos de datos, buscar las claves principales, etc. Más adelante en este tutorial examinaremos cómo puede invalidar estas convenciones.

## 5. Tratar los cambios en el modelo

Es el momento de realizar algunos cambios en el modelo; cuando realicemos dichos cambios, también necesitaremos actualizar el esquema de la base de datos. Para ello, vamos a usar una característica denominada migraciones de Code First, o migraciones para abreviar.

Las migraciones nos permiten tener un conjunto ordenado de pasos que describen cómo actualizar (y degradar) nuestro esquema de la base de datos. Cada uno de estos pasos, conocido como migración, contiene código que describe los cambios que se aplicarán.

El primer paso es habilitar Migraciones de Code First en nuestro BloggingContext.

**Herramientas -> Administrador de paquetes de la biblioteca -> Consola del Administrador de paquetes**

Ejecute el comando **enable-migrations** en la Consola del Administrador de paquetes

Una carpeta de migraciones se ha agregado a nuestro proyecto y contiene dos elementos:

**Configuration.cs**: este archivo contiene las opciones que las migraciones usarán para migrar BloggingContext. No necesitamos cambiar nada para este tutorial pero aquí es donde puede especificar los datos de inicialización, registrar los proveedores para otras bases de datos, cambiar el espacio de nombres en que las migraciones se generan, etc.

**<timestamp>\_InitialCreate.cs**: se trata de la primera migración, representa los cambios que ya se han aplicado a la base de datos para llevarla de una base de datos vacía a una que incluye las tablas de blogs y posts. Aunque permitiéramos que Code First creara automáticamente estas tablas en nuestro lugar, ahora que hemos optado por las migraciones, se han convertido en una migración. Code First también ha registrado en nuestra base de datos local que esta migración ya se ha aplicado. La marca de tiempo en el nombre de archivo se usa para la clasificación.

Ahora vamos a realizar un cambio en nuestro modelo, agregando una propiedad Url a la clase Blog:

**public** **class** Blog   
{   
    **public** **int** BlogId { get; set; }   
    **public** **string** Name { get; set; }   
    **public** **string** Url { get; set; }   
   
    **public** **virtual** List<Post> Posts { get; set; }   
}

Ejecute el comando **Add-Migration AddUrl** en la Consola del Administrador de paquetes.  
  
El comando Add-Migration comprueba si hay cambios desde la última migración y aplica scaffold a una nueva migración con los cambios encontrados. Podemos dar un nombre a las migraciones; en este caso, le denominamos "AddUrl".

**(OPCIONAL) NO APLICA A LO NUESTRO**  
El código al que se ha aplicado scaffold indica que tenemos que agregar una columna Url, que puede contener datos de cadena, a la tabla dbo.Blogs. Si es necesario, podríamos modificar dicho código pero no se requiere en este caso.

**namespace** CodeFirstNewDatabaseSample.Migrations   
{   
    **using** System;   
    **using** System.Data.Entity.Migrations;   
       
    **public** partial **class** AddUrl : DbMigration   
    {   
        **public** **override** **void** Up()   
        {   
            AddColumn("dbo.Blogs", "Url", c => c.String());   
        }   
           
        **public** **override** **void** Down()   
        {   
            DropColumn("dbo.Blogs", "Url");   
        }   
    }   
}

**MIGRACIONES AUTOMATICAS**

**Para activar las migraciones automáticas y no ejecutar cada vez el comando “add migrations…” deben escribir y ejecutar el siguiente comando en la ventana de consola**

* ***enable-migrations –enableautomaticmigrations***

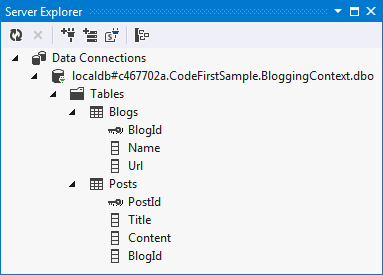
**IMPORTANTE**

***“Si bien se activó la migración automática, se debe si o si realizar un update a la base de datos, la migración no incluye actualizar los nuevos cambios del código en la BD sino solo en el entity framework, es decir cuando vaya a hacer la migración, el objeto entity va a tener el nuevo campo (por la migración automática) pero va a lanzar error por no encontrar el campo en la BD”***

***“Después de cualquier cambio en el código que signifique modificación en la estructura de las tablas, se requiere hacer un update-database antes de correrlo.”***

Ejecute el comando **Update-Database** en la Consola del Administrador de paquetes. Este comando aplicará las migraciones pendientes a la base de datos.

La nueva columna Url se agrega ahora a la tabla de blogs de la base de datos:



## Anotaciones de datos

(Algo para tener en cuenta por si en algún momento nos hace falta)

Hasta ahora solo hemos dejado que EF detecte el modelo usando sus convenciones predeterminadas pero habrá ocasiones en que nuestras clases no sigan las convenciones y necesitemos realizar una configuración adicional. Hay dos opciones para ello; examinaremos las anotaciones de datos en esta sección y luego la API fluida en la sección siguiente.

Vamos a agregar una clase User a nuestro modelo

**public** **class** User   
{   
    **public** **string** Username { get; set; }   
    **public** **string** DisplayName { get; set; }   
}

También tenemos que agregar un conjunto a nuestro contexto derivado

**public** **class** BloggingContext : DbContext   
{   
    **public** DbSet<Blog> Blogs { get; set; }   
    **public** DbSet<Post> Posts { get; set; }   
    **public** DbSet<User> Users { get; set; }   
}

Si intentáramos agregar una migración, obtendríamos un error que indicaría que “*El usuario EntityType no tiene ninguna clave definida. Defina la clave para este EntityType.”* dado que EF no tiene ninguna manera de saber que el nombre de usuario debe ser la clave principal del usuario.

Ahora vamos a usar anotaciones de datos de modo que necesitamos agregar una instrucción using al principio de Program.cs

**using** System.ComponentModel.DataAnnotations;

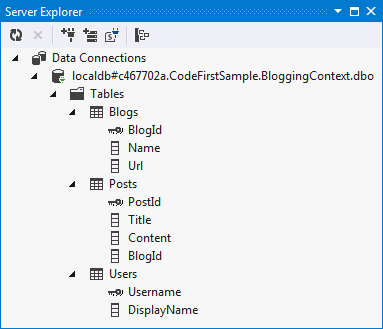
Ahora anote la propiedad Username para identificar que es la clave principal

**public** **class** User   
{   
    [Key]   
    **public** **string** Username { get; set; }   
    **public** **string** DisplayName { get; set; }   
}

Use el comando **Add-Migration AddUser** para scaffold una migración para aplicar estos cambios a la base de datos

Ejecute el comando **Update-Database** para aplicar la nueva migración a la base de datos

La nueva tabla se agrega ahora a la base de datos:



## 7. API fluida

En la sección anterior examinamos el uso de anotaciones de datos para complementar o invalidar qué se detecta por convención. Otra forma de configurar el modelo es a través de la API fluida de Code First.

La mayor parte de la configuración del modelo se puede hacer con anotaciones de datos simples. La API fluida es una forma más avanzada de especificar la configuración del modelo que abarca todo lo que las anotaciones de datos pueden hacer además de otras configuraciones avanzadas que no son posibles con las anotaciones de datos. Las anotaciones de datos y la API fluida se pueden usar juntas.

Para tener acceso a la API fluida, invalide el método OnModelCreating en DbContext. Supongamos que desea cambiar el nombre de la columna en que User.DisplayName está almacenada por display\_name.

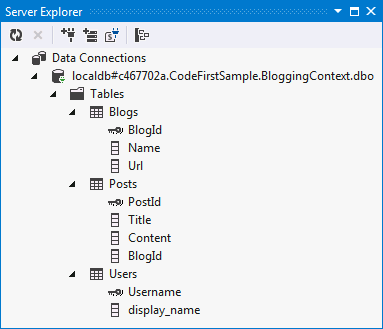
Invalide el método OnModelCreating en BloggingContext con el siguiente código

**public** **class** BloggingContext : DbContext   
{   
    **public** DbSet<Blog> Blogs { get; set; }   
    **public** DbSet<Post> Posts { get; set; }   
    **public** DbSet<User> Users { get; set; }   
   
    **protected** **override** **void** OnModelCreating(DbModelBuilder modelBuilder)   
    {   
        modelBuilder.Entity<User>()   
            .Property(u => u.DisplayName)   
            .HasColumnName("display\_name");   
    }   
}

Use el comando **Add-Migration ChangeDisplayName** para aplicar scaffold a una migración y aplicar estos cambios a la base de datos.

Ejecute el comando **Update-Database** para aplicar la nueva migración a la base de datos.

El nombre de la columna DisplayName se cambia por display\_name:



## Resumen

En este tutorial, examinamos el desarrollo de Code First con una nueva base de datos. Definimos un modelo con clases y después usamos ese modelo para crear una base de datos y almacenar y recuperar los datos. Una vez que se creó la base de datos, usamos las Migraciones de Code First para cambiar el esquema a medida que evolucionó nuestro modelo. También vimos cómo configurar un modelo mediante anotaciones de datos y la API fluida.

## Referencias

* [**http://chanmingman.wordpress.com/2012/10/08/how-to-install-entity-framework-5-in-visual-studio-2012/**](http://chanmingman.wordpress.com/2012/10/08/how-to-install-entity-framework-5-in-visual-studio-2012/)
* [**http://msdn.microsoft.com/es-es/data/jj193542**](http://msdn.microsoft.com/es-es/data/jj193542)
* [**http://geeks.ms/blogs/omarvr/archive/2011/07/14/ef-4-1-code-first-191-d-243-nde-est-225-la-base-de-datos.aspx**](http://geeks.ms/blogs/omarvr/archive/2011/07/14/ef-4-1-code-first-191-d-243-nde-est-225-la-base-de-datos.aspx)
* [**http://www.avanet.org/entityframework-code-first-creando-nuestra-base-de-datos-basados-en-nuestro-contexto.aspx**](http://www.avanet.org/entityframework-code-first-creando-nuestra-base-de-datos-basados-en-nuestro-contexto.aspx)
* [**http://msdn.microsoft.com/es-ar/data/jj554735.aspx**](http://msdn.microsoft.com/es-ar/data/jj554735.aspx)
* [**http://blogs.msdn.com/b/adonet/archive/2012/02/09/ef-4-3-automatic-migrations-walkthrough.aspx**](http://blogs.msdn.com/b/adonet/archive/2012/02/09/ef-4-3-automatic-migrations-walkthrough.aspx)
* [**http://stackoverflow.com/questions/17922945/how-to-enable-migration-to-update-my-database-in-mvc4**](http://stackoverflow.com/questions/17922945/how-to-enable-migration-to-update-my-database-in-mvc4)

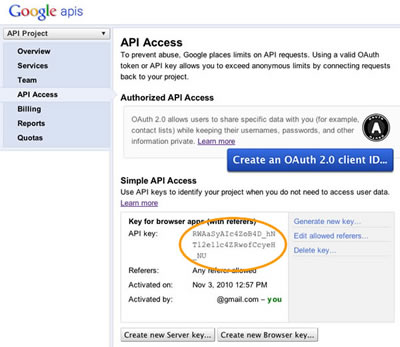
# API Google Maps V3

Google Maps dispone de un amplio conjunto de APIs que te permiten trasladar la gran funcionalidad y la utilidad diaria de Google Maps a tu propio sitio web y a tus propias aplicaciones, así como superponer tus datos.

## Cómo obtener una clave de API

Para crear tu clave de API:

1. Accede a la página de la consola de las API (<https://code.google.com/apis/console>) e inicia sesión con tu cuenta de Google.
2. Haz clic en el enlace de **servicios** en el menú de la izquierda.
3. Activa el servicio de la **versión 3 del API de Google Maps**.
4. Haz clic en el enlace de **acceso al API** en el menú de la izquierda. Tu clave de API está disponible desde la página de **acceso al API**, en la sección de **acceso al API sencilla**. Las aplicaciones del API de Google Maps utilizan la **clave para aplicaciones del navegador**.



Antes era necesario usar esta Key en la siguiente línea:

src="http://maps.googleapis.com/maps/api/js?key=*YOUR\_API\_KEY*&sensor=*SET\_TO\_TRUE\_OR\_FALSE*">

Pero ahora ya no es necesario como se ve en el código de ejemplo, igual se puede seguir utilizando la Key si es que la tienes.

## Creación de un simple mapa (ver uso de Api key)

<!DOCTYPE html>  
<html>  
  <head>  
    <title>Simple Map</title>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  
    <meta charset="utf-8">  
    <style>  
      html, body, #map-canvas {  
        height: 100%;  
        margin: 0px;  
        padding: 0px  
      }  
    </style>  
    <script src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false"></script>  
    <script>  
var map;  
function initialize() {  
  var mapOptions = {  
    zoom: 8,  
    center: new google.maps.LatLng(-34.397, 150.644)  
  };  
  map = new google.maps.Map(document.getElementById('map-canvas'),  
      mapOptions);  
}  
  
google.maps.event.addDomListener(window, 'load', initialize);  
  
    </script>  
  </head>  
  <body>  
    <div id="map-canvas"></div>  
  </body>  
</html>

**CONSIDERACIONES:** Ten en cuenta que algunas de las CSS que funcionan en el modo chapucero no son válidas en el modo estándar. En concreto, todos los tamaños basados en porcentajes deben heredarse de los elementos de bloque principales, y si cualquiera de estos antecesores no puede especificar un tamaño, se supondrá un tamaño de 0 x 0 píxeles. Por esta razón, incluimos la declaración <style>:

### COMO CARGAR EL API EN FORMA ASINCRONA

Esto hace que el api de Java Script de Google Maps se cargue después de toda la página. Es una recomendación que nos hace Google.

<!DOCTYPE html>  
<html>  
  <head>  
    <title>Asynchronous Loading</title>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  
    <meta charset="utf-8">  
    <style>  
      html, body, #map-canvas {  
        height: 100%;  
        margin: 0px;  
        padding: 0px  
      }  
    </style>  
    <script>  
function initialize() {  
  var mapOptions = {  
    zoom: 8,  
    center: new google.maps.LatLng(-34.397, 150.644)  
  };  
  
  var map = new google.maps.Map(document.getElementById('map-canvas'),  
      mapOptions);  
}  
  
function loadScript() {  
  var script = document.createElement('script');  
  script.type = 'text/javascript';  
  script.src = 'https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false&' +  
      'callback=initialize';  
  document.body.appendChild(script);  
}  
  
window.onload = loadScript;  
  
    </script>  
  </head>  
  <body>  
    <div id="map-canvas"></div>  
  </body>  
</html>

### Tipos de Mapas

También debes establecer expresamente un tipo de mapa inicial en este momento.

mapTypeId: google.maps.MapTypeId.ROADMAP

Se admiten los siguientes tipos de mapas:

* ROADMAP, que muestra los mosaicos normales en 2D predeterminados de Google Maps.
* SATELLITE muestra imágenes de satélite.
* HYBRID muestra una mezcla de mosaicos fotográficos y una capa de mosaicos para los elementos del mapa más destacados (carreteras, nombres de ciudades, etc.).
* TERRAIN muestra mosaicos de relieve físico para indicar las elevaciones del terreno y las fuentes de agua (montañas, ríos, etc.).

## El objeto "Map"

var map = new google.maps.Map( document.getElementById("map\_canvas"),    mapOptions);

La clase de JavaScript que representa a los mapas es Map. Cada objeto de esta clase define un único mapa en una página. (Puedes crear más de una instancia de esta clase; cada objeto definirá un mapa independiente en la página). Creamos una nueva instancia de esta clase mediante el operador new de JavaScript.

Al crear una nueva instancia de mapa, se especifica un elemento HTML <div> en la página como contenedor para el mapa. Los nodos HTML son elementos secundarios del objeto document de JavaScript. Se obtiene una referencia a este elemento mediante el método document.getElementById().

Este código permite definir una variable (denominada map) y asignar dicha variable a un nuevo objeto Map, además de transmitir opciones definidas en el objeto mapOptionsliteral. Estas opciones se utilizarán para inicializar las propiedades del mapa. A continuación se muestra la definición de la función Map(), conocida como *constructor*:

|  |  |
| --- | --- |
| **Constructor** | **Descripción** |
| Map(mapDiv:Node, opts?:[*MapOptions*](https://developers.google.com/maps/documentation/javascript/tutorial?hl=es#MapOptions)) | Crea un mapa nuevo dentro del contenedor HTML en cuestión,  que suele ser un elemento DIV, mediante los parámetros  (opcional) que se especifiquen. |

### Eventos

JavaScript en el navegador está orientado a eventos, lo que significa que JavaScript responde a las interacciones generando eventos y espera que un programa detecte los eventos interesantes. El modelo de evento de la versión 3 del API de Google Maps es similar al utilizado en la versión 2 del API, aunque se han modificado muchos aspectos de su funcionamiento. Hay dos tipos de eventos:

* Los eventos de usuario (como, por ejemplo, los eventos de ratón "click") se propagan desde el DOM hasta el API de Google Maps. Estos eventos son distintos e independientes de los eventos DOM estándar.
* Las notificaciones de cambio de estado de MVC reflejan los cambios en los objetos del API de Google Maps y se denominan mediante una convenciónproperty\_changed.

Cada objeto del API de Google Maps exporta una determinada cantidad de eventos con nombres. Los programas interesados en determinados eventos registran **detectores de eventos** de JavaScript para estos eventos y ejecutan código al recibirlos mediante el registro de controladores de eventos addListener() en el espacio de nombregoogle.maps.event. Los desarrolladores de la versión 2 del API de Google Maps están familiarizados con este uso.

Para obtener una lista completa de eventos, consulta la [Referencia del API de Google Maps](https://developers.google.com/maps/documentation/javascript/reference?hl=es). Los eventos aparecen en una sección independiente para cada objeto que contiene eventos.

### Eventos de la Interfaz de Usuario

Algunos objetos del API de Google Maps están diseñados para responder a los eventos de usuario como los de ratón o los de teclado. Por ejemplo, un objetogoogle.maps.Marker puede detectar los siguientes eventos de usuario:

* 'click'
* 'dblclick'
* 'mouseup'
* 'mousedown'
* 'mouseover'
* 'mouseout'

Estos eventos pueden tener el aspecto de eventos DOM estándar pero, realmente, forman parte del API de Google Maps. Dado que los distintos navegadores implementan distintos modelos de eventos DOM, el API de Google Maps ofrece estos mecanismos que permiten detectar eventos DOM y responder a ellos sin necesidad de gestionar las diversas peculiaridades de cada navegador. Estos eventos suelen incluir argumentos en el evento que destacan algún estado de la interfaz de usuario (como, por ejemplo, la posición del ratón).

### Cambios de Estado

Los objetos MVC suelen contener estado. Cuando cambia la propiedad de un objeto, el API activa un evento que la propiedad ha cambiado. Por ejemplo, el API activa un eventozoom\_changed en un mapa cuando cambia el nivel de zoom del mismo. Para interceptar estos cambios de estado, registra también los controladores de eventosaddListener() en el método de espacio de nombre event.

Los eventos de usuario y los cambios de estado de MVC pueden parecer similares pero, generalmente, se recomienda tratarlos de forma diferente en el código. Los eventos MVC, por ejemplo, no incluyen argumentos en sus eventos. Se recomienda inspeccionar la propiedad que se ha modificado en un cambio de estado de MVC mediante la ejecución del método getProperty adecuado de ese objeto.

### Ejemplo de Eventos y cambios de Estado

<!DOCTYPE html>  
<html>  
  <head>  
    <title>Simple click event</title>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  
    <meta charset="utf-8">  
    <style>  
      html, body, #map-canvas {  
        height: 100%;  
        margin: 0px;  
        padding: 0px  
      }  
    </style>  
    <script src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false"></script>  
    <script>  
function initialize() {  
  var mapOptions = {  
    zoom: 4,  
    center: new google.maps.LatLng(-25.363882, 131.044922)  
  };  
  
  var map = new google.maps.Map(document.getElementById('map-canvas'),  
      mapOptions);  
  
  var marker = new google.maps.Marker({  
    position: map.getCenter(),  
    map: map,  
    title: 'Click to zoom'  
  });  
  
  google.maps.event.addListener(map, 'center\_changed', function() {  
    // 3 seconds after the center of the map has changed, pan back to the  
    // marker.  
    window.setTimeout(function() {  
      map.panTo(marker.getPosition());  
    }, 3000);  
  });  
  
  google.maps.event.addListener(marker, 'click', function() {  
    map.setZoom(8);  
    map.setCenter(marker.getPosition());  
  });  
}  
  
google.maps.event.addDomListener(window, 'load', initialize);  
  
    </script>  
  </head>  
  <body>  
    <div id="map-canvas"></div>  
  </body>  
</html>

## Descripción general de los controles del mapa

Los mapas que se muestran mediante el API de Google Maps contienen elementos de interfaz de usuario que permiten interactuar con el mapa. Estos elementos se denominan*controles* y puedes incluir variaciones de ellos en la aplicación que hayas creado con el API de Google Maps. También puedes optar por no hacer nada y dejar que el API de Google Maps se ocupe del comportamiento de todos los controles.

El API de Google Maps dispone de varios controles integrados que puedes emplear en tus mapas:

* El ***control de zoom*** muestra un control deslizante (para mapas grandes) o pequeños botones "+/-" (para mapas pequeños) que permiten controlar el nivel de zoom del mapa. Este control aparece en la esquina superior izquierda del mapa de forma predeterminada en dispositivos no táctiles o en la esquina inferior izquierda del mapa en los dispositivos táctiles.
* El ***control de desplazamiento*** muestra botones para desplazarse por el mapa. Este control aparece en la esquina superior izquierda del mapa de forma predeterminada en dispositivos no táctiles. El control de desplazamiento también te permite girar imágenes de 45º, si están disponibles.
* El ***control de escala*** muestra un elemento de escala del mapa. Este control está inhabilitado de forma predeterminada.
* El ***control MapType*** permite al usuario alternar entre los diferentes tipos de mapas, como ROADMAP (mapa de carreteras) y SATELLITE (satélite). Este control aparece en la esquina superior derecha del mapa de forma predeterminada.
* El ***control de Street View*** incluye un icono de hombrecito naranja que se puede arrastrar hasta el mapa para habilitar la función de Street View. Este control aparece en la esquina superior izquierda del mapa de forma predeterminada.
* El ***control de giro*** incluye un icono circular pequeño que te permite girar mapas que contienen imágenes oblicuas. Este control aparece en la esquina superior izquierda del mapa de forma predeterminada (para obtener más información, consulta la documentación sobre [imágenes de 45º](https://developers.google.com/maps/documentation/javascript/maptypes?hl=es#45DegreeImagery)).
* El ***control de mapa general*** muestra un mapa general en miniatura que refleja la ventana gráfica del mapa actual dentro de una zona más amplia. Este control aparece en la esquina inferior derecha del mapa y se muestra contraído de forma predeterminada.

### Cómo añadir controles al mapa

Si quieres, puedes confeccionar tu interfaz mediante la eliminación, la adición y la modificación del comportamiento y de los controles de la interfaz de usuario y asegurarte de que las futuras actualizaciones no modifiquen este comportamiento. Si solo quieres añadir o modificar comportamientos existentes, deberás asegurarte de que el control se haya añadido explícitamente a tu aplicación.

Algunos controles aparecen en el mapa de forma predeterminada, mientras que otros no se mostrarán a menos que lo solicites explícitamente. La adición o la eliminación de controles del mapa se especifica mediante los siguientes campos del objeto Map options; campos que deberás establecer en true para mostrar los controles o en false para ocultarlos:

{  
  panControl: boolean,  
  zoomControl: boolean,  
  mapTypeControl: boolean,  
  scaleControl: boolean,  
  streetViewControl: boolean,  
  overviewMapControl: boolean  
}

### Un ejemplo de un control personalizado

El control que se muestra a continuación es sencillo (aunque no especialmente útil) y combina los patrones indicados anteriormente. Este control responde a los eventos 'click'DOM centrando el mapa en una ubicación predeterminada:

var map;  
var chicago = new google.maps.LatLng(41.850033, -87.6500523);  
  
/\*\*  
 \* The HomeControl adds a control to the map that simply  
 \* returns the user to Chicago. This constructor takes  
 \* the control DIV as an argument.  
 \*/  
  
function HomeControl(controlDiv, map) {  
  
  // Set CSS styles for the DIV containing the control  
  // Setting padding to 5 px will offset the control  
  // from the edge of the map.  
  controlDiv.style.padding = '5px';  
  
  // Set CSS for the control border.  
  var controlUI = document.createElement('div');  
  controlUI.style.backgroundColor = 'white';  
  controlUI.style.borderStyle = 'solid';  
  controlUI.style.borderWidth = '2px';  
  controlUI.style.cursor = 'pointer';  
  controlUI.style.textAlign = 'center';  
  controlUI.title = 'Click to set the map to Home';  
  controlDiv.appendChild(controlUI);  
  
  // Set CSS for the control interior.  
  var controlText = document.createElement('div');  
  controlText.style.fontFamily = 'Arial,sans-serif';  
  controlText.style.fontSize = '12px';  
  controlText.style.paddingLeft = '4px';  
  controlText.style.paddingRight = '4px';  
  controlText.innerHTML = '<strong>Home</strong>';  
  controlUI.appendChild(controlText);  
  
  // Setup the click event listeners: simply set the map to Chicago.  
  google.maps.event.addDomListener(controlUI, 'click', function() {  
    map.setCenter(chicago)  
  });  
}  
  
function initialize() {  
  var mapDiv = document.getElementById('map\_canvas');  
  var mapOptions = {  
    zoom: 12,  
    center: chicago,  
    mapTypeId: google.maps.MapTypeId.ROADMAP  
  }  
  map = new google.maps.Map(mapDiv, mapOptions);  
  
  // Create the DIV to hold the control and call the HomeControl() constructor  
  // passing in this DIV.  
  var homeControlDiv = document.createElement('div');  
  var homeControl = new HomeControl(homeControlDiv, map);  
  
  homeControlDiv.index = 1;  
  map.controls[google.maps.ControlPosition.TOP\_RIGHT].push(homeControlDiv);  
}

[Ver ejemplo (control-custom.html)](https://developers.google.com/maps/documentation/javascript/examples/control-custom?hl=es)

## Marcadores

Los marcadores identifican ubicaciones en el mapa. De manera predeterminada, utilizan un icono estándar, aunque puedes establecer un icono personalizado dentro del constructor del marcador o mediante la ejecución de setIcon() en el marcador. El constructor google.maps.Marker toma un único objeto literal *Marker options* que especifica las propiedades iniciales del marcador. A continuación se indican algunos campos especialmente importantes que se suelen definir al crear un marcador.

* position (obligatorio) especifica un valor de LatLng que identifica la ubicación inicial del marcador.
* map (opcional) especifica el objeto Map en el que se sitúa el marcador.

Ten en cuenta que debes especificar el mapa en el que vas a añadir el marcador dentro del constructor Marker. Si no especificas este argumento, el marcador se creará, pero no se añadirá al mapa (o no mostrará). Puedes añadir el marcador más tarde mediante la ejecución del método setMap() del marcador. Para eliminar un marcador, ejecuta el método setMap() y transmite null como el argumento.

Los marcadores están diseñados para ser interactivos. De forma predeterminada, reciben eventos 'click', por ejemplo, y se suelen utilizar dentro de detectores de eventos para abrir ventanas de información. Puedes establecer la propiedad draggable de un marcador en true para que [los usuarios puedan editar](https://developers.google.com/maps/documentation/javascript/overlays?hl=es#editable) el marcador en el mapa.

En el siguiente ejemplo, se añade un marcador simple a un mapa de Uluru, en el centro de Australia:

  var myLatlng = new google.maps.LatLng(-25.363882,131.044922);  
  var mapOptions = {  
    zoom: 4,  
    center: myLatlng,  
    mapTypeId: google.maps.MapTypeId.ROADMAP  
  }  
  var map = new google.maps.Map(document.getElementById("map\_canvas"), mapOptions);  
  
  var marker = new google.maps.Marker({  
      position: myLatlng,  
      map: map,  
      title:"Hello World!"  
  });

Este título de Marker se mostrará como información sobre la herramienta. ES DECIR COMO UN TOOL TIPS

Si no quieres incluir ninguna opción del marcador (*Marker options*) en el constructor del marcador, incluye un objeto {} vacío en el último argumento del constructor.

[Ver ejemplo (marker-simple.html)](https://developers.google.com/maps/documentation/javascript/examples/marker-simple?hl=es)

### Animaciones

También puedes animar los marcadores de forma que muestren un movimiento dinámico en numerosas circunstancias diferentes. La propiedad animation, del tipogoogle.maps.Animation, especifica la forma en que está animado un marcador. Actualmente, se admiten los siguientes valores Animation:

* DROP indica que el marcador debe caer desde la parte superior del mapa hasta su ubicación definitiva en la que se colocase primero en el mapa. La animación finalizará una vez el marcador empiece a quedarse quieto y animation se restablezca a null. Este tipo de animación se suele especificar durante la creación del marcador (Marker).
* BOUNCE indica que el marcador debe "rebotar" en el mismo sitio. Un marcador que rebota seguirá haciéndolo hasta que la propiedad animation se defina explícitamente en null.

Puedes iniciar una animación en un marcador existente ejecutando setAnimation() en el objeto Marker.

El siguiente ejemplo crea un marcador en Estocolmo (Suecia) con una animación DROP. Al hacer clic en el marcador, se alterna entre una animación BOUNCE o ninguna animación:

var stockholm = new google.maps.LatLng(59.32522, 18.07002);  
var parliament = new google.maps.LatLng(59.327383, 18.06747);  
var marker;  
var map;  
  
function initialize() {  
  var mapOptions = {  
    zoom: 13,  
    mapTypeId: google.maps.MapTypeId.ROADMAP,  
    center: stockholm  
  };  
  
  map = new google.maps.Map(document.getElementById("map\_canvas"),  
      mapOptions);  
  
  marker = new google.maps.Marker({  
    map:map,  
    draggable:true,  
    animation: google.maps.Animation.DROP,  
    position: parliament  
  });  
  google.maps.event.addListener(marker, 'click', toggleBounce);  
}  
  
function toggleBounce() {  
  
  if (marker.getAnimation() != null) {  
    marker.setAnimation(null);  
  } else {  
    marker.setAnimation(google.maps.Animation.BOUNCE);  
  }  
}

[Ver ejemplo (marker-animations.html)](https://developers.google.com/maps/documentation/javascript/examples/marker-animations?hl=es)

### Poligonos

<!DOCTYPE html>  
<html>  
  <head>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  
    <meta charset="utf-8">  
    <title>Circles</title>  
    <style>  
      html, body, #map-canvas {  
        height: 100%;  
        margin: 0px;  
        padding: 0px  
      }  
    </style>  
    <script src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false"></script>  
    <script>  
// This example creates circles on the map, representing  
// populations in North America.  
  
// First, create an object containing LatLng and population for each city.  
var citymap = {};  
citymap['chicago'] = {  
  center: new google.maps.LatLng(41.878113, -87.629798),  
  population: 2714856  
};  
citymap['newyork'] = {  
  center: new google.maps.LatLng(40.714352, -74.005973),  
  population: 8405837  
};  
citymap['losangeles'] = {  
  center: new google.maps.LatLng(34.052234, -118.243684),  
  population: 3857799  
};  
citymap['vancouver'] = {  
  center: new google.maps.LatLng(49.25, -123.1),  
  population: 603502  
};  
  
var cityCircle;  
  
function initialize() {  
  // Create the map.  
  var mapOptions = {  
    zoom: 4,  
    center: new google.maps.LatLng(37.09024, -95.712891),  
    mapTypeId: google.maps.MapTypeId.TERRAIN  
  };  
  
  var map = new google.maps.Map(document.getElementById('map-canvas'),  
      mapOptions);  
  
  // Construct the circle for each value in citymap.  
  // Note: We scale the area of the circle based on the population.  
  for (var city in citymap) {  
    var populationOptions = {  
      strokeColor: '#FF0000',  
      strokeOpacity: 0.8,  
      strokeWeight: 2,  
      fillColor: '#FF0000',  
      fillOpacity: 0.35,  
      map: map,  
      center: citymap[city].center,  
      radius: Math.sqrt(citymap[city].population) \* 100  
    };  
    // Add the circle for this city to the map.  
    cityCircle = new google.maps.Circle(populationOptions);  
  }  
}  
  
google.maps.event.addDomListener(window, 'load', initialize);  
  
    </script>  
  </head>  
  <body>  
    <div id="map-canvas"></div>  
  </body>  
</html>

### Info Windows (Información de marcadores)

<!DOCTYPE html>  
<html>  
  <head>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  
    <meta charset="utf-8">  
    <title>Info windows</title>  
    <style>  
      html, body, #map-canvas {  
        height: 100%;  
        margin: 0px;  
        padding: 0px  
      }  
    </style>  
    <script src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false"></script>  
    <script>  
// This example displays a marker at the center of Australia.  
// When the user clicks the marker, an info window opens.  
  
function initialize() {  
  var myLatlng = new google.maps.LatLng(-25.363882,131.044922);  
  var mapOptions = {  
    zoom: 4,  
    center: myLatlng  
  };  
  
  var map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);  
  
  var contentString = '<div id="content">'+  
      '<div id="siteNotice">'+  
      '</div>'+  
      '<h1 id="firstHeading" class="firstHeading">Uluru</h1>'+  
      '<div id="bodyContent">'+  
      '<p><b>Uluru</b>, also referred to as <b>Ayers Rock</b>, is a large ' +  
      'sandstone rock formation in the southern part of the '+  
      'Northern Territory, central Australia. It lies 335&#160;km (208&#160;mi) '+  
      'south west of the nearest large town, Alice Springs; 450&#160;km '+  
      '(280&#160;mi) by road. Kata Tjuta and Uluru are the two major '+  
      'features of the Uluru - Kata Tjuta National Park. Uluru is '+  
      'sacred to the Pitjantjatjara and Yankunytjatjara, the '+  
      'Aboriginal people of the area. It has many springs, waterholes, '+  
      'rock caves and ancient paintings. Uluru is listed as a World '+  
      'Heritage Site.</p>'+  
      '<p>Attribution: Uluru, <a href="http://en.wikipedia.org/w/index.php?title=Uluru&oldid=297882194">'+  
      'http://en.wikipedia.org/w/index.php?title=Uluru</a> '+  
      '(last visited June 22, 2009).</p>'+  
      '</div>'+  
      '</div>';  
  
  var infowindow = new google.maps.InfoWindow({  
      content: contentString  
  });  
  
  var marker = new google.maps.Marker({  
      position: myLatlng,  
      map: map,  
      title: 'Uluru (Ayers Rock)'  
  });  
  google.maps.event.addListener(marker, 'click', function() {  
    infowindow.open(map,marker);  
  });  
}  
  
google.maps.event.addDomListener(window, 'load', initialize);  
  
    </script>  
  </head>  
  <body>  
    <div id="map-canvas"></div>  
  </body>  
</html>

## Calculo de Ruta a través de dos puntos

<!DOCTYPE html>  
<html>  
  <head>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  
    <meta charset="utf-8">  
    <title>Directions service</title>  
    <style>  
      html, body, #map-canvas {  
        height: 100%;  
        margin: 0px;  
        padding: 0px  
      }  
      #panel {  
        position: absolute;  
        top: 5px;  
        left: 50%;  
        margin-left: -180px;  
        z-index: 5;  
        background-color: #fff;  
        padding: 5px;  
        border: 1px solid #999;  
      }  
    </style>  
    <script src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false"></script>  
    <script>  
var directionsDisplay;  
var directionsService = new google.maps.DirectionsService();  
var map;  
  
function initialize() {  
  directionsDisplay = new google.maps.DirectionsRenderer();  
  var chicago = new google.maps.LatLng(41.850033, -87.6500523);  
  var mapOptions = {  
    zoom:7,  
    center: chicago  
  }  
  map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);  
  directionsDisplay.setMap(map);  
}  
  
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);  
  
    </script>  
  </head>  
  <body>  
    <div id="panel">  
    <b>Start: </b>  
    <select id="start" onchange="calcRoute();">  
      <option value="chicago, il">Chicago</option>  
      <option value="st louis, mo">St Louis</option>  
      <option value="joplin, mo">Joplin, MO</option>  
      <option value="oklahoma city, ok">Oklahoma City</option>  
      <option value="amarillo, tx">Amarillo</option>  
      <option value="gallup, nm">Gallup, NM</option>  
      <option value="flagstaff, az">Flagstaff, AZ</option>  
      <option value="winona, az">Winona</option>  
      <option value="kingman, az">Kingman</option>  
      <option value="barstow, ca">Barstow</option>  
      <option value="san bernardino, ca">San Bernardino</option>  
      <option value="los angeles, ca">Los Angeles</option>  
    </select>  
    <b>End: </b>  
    <select id="end" onchange="calcRoute();">  
      <option value="chicago, il">Chicago</option>  
      <option value="st louis, mo">St Louis</option>  
      <option value="joplin, mo">Joplin, MO</option>  
      <option value="oklahoma city, ok">Oklahoma City</option>  
      <option value="amarillo, tx">Amarillo</option>  
      <option value="gallup, nm">Gallup, NM</option>  
      <option value="flagstaff, az">Flagstaff, AZ</option>  
      <option value="winona, az">Winona</option>  
      <option value="kingman, az">Kingman</option>  
      <option value="barstow, ca">Barstow</option>  
      <option value="san bernardino, ca">San Bernardino</option>  
      <option value="los angeles, ca">Los Angeles</option>  
    </select>  
    </div>  
    <div id="map-canvas"></div>  
  </body>  
</html>

[Ver ejemplo (directions-simple.html)](https://developers.google.com/maps/documentation/javascript/examples/directions-simple?hl=es)

## Calculo de ruta con indicaciones al costado

<!DOCTYPE html>  
<html>  
  <head>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  
    <meta charset="utf-8">  
    <title>Displaying text directions with <code>setPanel()</code></title>  
    <style>  
      html, body, #map-canvas {  
        height: 100%;  
        margin: 0px;  
        padding: 0px  
      }  
      #panel {  
        position: absolute;  
        top: 5px;  
        left: 50%;  
        margin-left: -180px;  
        z-index: 5;  
        background-color: #fff;  
        padding: 5px;  
        border: 1px solid #999;  
      }  
    </style>  
    <style>  
      #directions-panel {  
        height: 100%;  
        float: right;  
        width: 390px;  
        overflow: auto;  
      }  
  
      #map-canvas {  
        margin-right: 400px;  
      }  
  
      #control {  
        background: #fff;  
        padding: 5px;  
        font-size: 14px;  
        font-family: Arial;  
        border: 1px solid #ccc;  
        box-shadow: 0 2px 2px rgba(33, 33, 33, 0.4);  
        display: none;  
      }  
  
      @media print {  
        #map-canvas {  
          height: 500px;  
          margin: 0;  
        }  
  
        #directions-panel {  
          float: none;  
          width: auto;  
        }  
      }  
    </style>  
    <script src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false"></script>  
    <script>  
var directionsDisplay;  
var directionsService = new google.maps.DirectionsService();  
  
function initialize() {  
  directionsDisplay = new google.maps.DirectionsRenderer();  
  var mapOptions = {  
    zoom: 7,  
    center: new google.maps.LatLng(41.850033, -87.6500523)  
  };  
  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);  
  
    </script>  
  </head>  
  <body>  
    <div id="control">  
      <strong>Start:</strong>  
      <select id="start" onchange="calcRoute();">  
        <option value="chicago, il">Chicago</option>  
        <option value="st louis, mo">St Louis</option>  
        <option value="joplin, mo">Joplin, MO</option>  
        <option value="oklahoma city, ok">Oklahoma City</option>  
        <option value="amarillo, tx">Amarillo</option>  
        <option value="gallup, nm">Gallup, NM</option>  
        <option value="flagstaff, az">Flagstaff, AZ</option>  
        <option value="winona, az">Winona</option>  
        <option value="kingman, az">Kingman</option>  
        <option value="barstow, ca">Barstow</option>  
        <option value="san bernardino, ca">San Bernardino</option>  
        <option value="los angeles, ca">Los Angeles</option>  
      </select>  
      <strong>End:</strong>  
      <select id="end" onchange="calcRoute();">  
        <option value="chicago, il">Chicago</option>  
        <option value="st louis, mo">St Louis</option>  
        <option value="joplin, mo">Joplin, MO</option>  
        <option value="oklahoma city, ok">Oklahoma City</option>  
        <option value="amarillo, tx">Amarillo</option>  
        <option value="gallup, nm">Gallup, NM</option>  
        <option value="flagstaff, az">Flagstaff, AZ</option>  
        <option value="winona, az">Winona</option>  
        <option value="kingman, az">Kingman</option>  
        <option value="barstow, ca">Barstow</option>  
        <option value="san bernardino, ca">San Bernardino</option>  
        <option value="los angeles, ca">Los Angeles</option>  
      </select>  
    </div>  
    <div id="directions-panel"></div>  
    <div id="map-canvas"></div>  
  </body>  
</html>

## Referencias

* <https://developers.google.com/maps/documentation/javascript/examples/?hl=es>

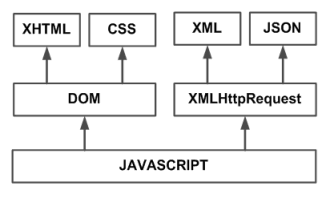
# Ajax

AJAX es un acrónimo de *Asynchronous JavaScript + XML*, que se puede traducir como "JavaScript asíncrono + XML".

*“Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes.”*

Las tecnologías que forman AJAX son:

* XHTML y CSS, para crear una presentación basada en estándares.
* DOM, para la interacción y manipulación dinámica de la presentación.
* XML, XSLT y JSON, para el intercambio y la manipulación de información.
* XMLHttpRequest, para el intercambio asíncrono de información.
* JavaScript, para unir todas las demás tecnologías.

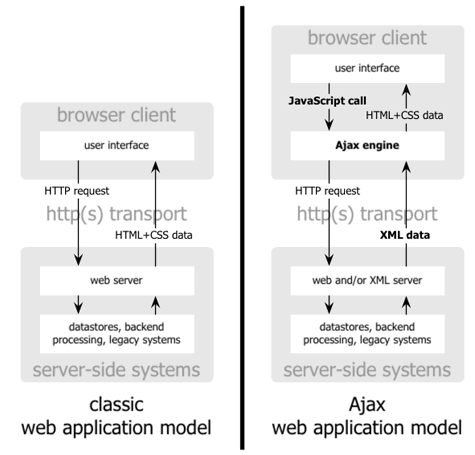


**Figura** Tecnologías agrupadas bajo el concepto de AJAX

Desarrollar aplicaciones AJAX requiere un conocimiento avanzado de todas y cada una de las tecnologías anteriores.

En las aplicaciones web tradicionales, las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenan llamadas al servidor. Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

En el siguiente esquema, la imagen de la izquierda muestra el modelo tradicional de las aplicaciones web. La imagen de la derecha muestra el nuevo modelo propuesto por AJAX:



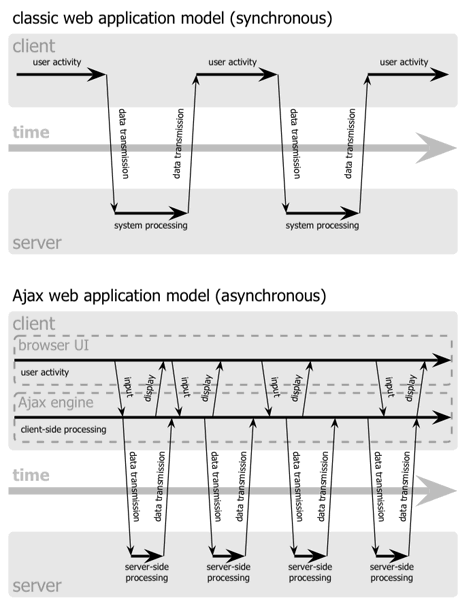
**Figura 1.2** Comparación gráfica del modelo tradicional de aplicación web y del nuevo modelo propuesto por AJAX.

Esta técnica tradicional para crear aplicaciones web funciona correctamente, pero no crea una buena sensación al usuario. Al realizar peticiones continuas al servidor, el usuario debe esperar a que se recargue la página con los cambios solicitados. Si la aplicación debe realizar peticiones continuas, su uso se convierte en algo molesto

AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.

El siguiente esquema muestra la diferencia más importante entre una aplicación web tradicional y una aplicación web creada con AJAX. La imagen superior muestra la interación síncrona propia de las aplicaciones web tradicionales. La imagen inferior muestra la comunicación asíncrona de las aplicaciones creadas con AJAX.



**Figura 1.3** Comparación entre las comunicaciones síncronas de las aplicaciones web tradicionales y las comunicaciones asíncronas de las aplicaciones.

Las peticiones HTTP al servidor se sustituyen por peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor.

Desde su aparición, se han creado cientos de aplicaciones web basadas en AJAX. En la mayoría de casos, AJAX puede sustituir completamente a otras técnicas como Flash. Además, en el caso de las aplicaciones web más avanzadas, pueden llegar a sustituir a las aplicaciones de escritorio.

## Funciones para AJAX en JQuery

Las funciones y utilidades relacionadas con AJAX son parte fundamental de jQuery. El método principal para realizar peticiones AJAX es $.ajax(). A partir de esta función básica, se han definido otras funciones relacionadas, de más alto nivel y especializadas en tareas concretas: $.get(), $.post(), $.load(), etc.

La sintaxis de $.ajax() es muy sencilla:

$.ajax(opciones);

La URL que se solicita también se incluye dentro del array asociativo de opciones. A continuación se muestra un ejemplo básico que se utiliza con $.ajax():

$.ajax({

url: '/ruta/hasta/pagina.php',

type: 'POST',

async: true,

data: 'parametro1=valor1&parametro2=valor2',

success: procesaRespuesta,

error: muestraError

});

La siguiente tabla muestra todas las opciones que se pueden definir para el método $.ajax():

|  |  |
| --- | --- |
| **Opción** | **Descripción** |
| **async** | Indica si la petición es asíncrona. Su valor por defecto es true, el habitual para las peticiones AJAX |
| **beforeSend** | Permite indicar una función que modifique el objeto XMLHttpRequest antes de realizar la petición. El propio objeto XMLHttpRequest se pasa como único argumento de la función |
| **complete** | Permite establecer la función que se ejecuta cuando una petición se ha completado (y después de ejecutar, si se han establecido, las funciones de success o error). La función recibe el objeto XMLHttpRequest como primer parámetro y el resultado de la petición como segundo argumento |
| **contentType** | Indica el valor de la cabecera Content-Type utilizada para realizar la petición. Su valor por defecto es application/x-www-form-urlencoded |
| **data** | Información que se incluye en la petición. Se utiliza para enviar parámetros al servidor. Si es una cadena de texto, se envía tal cual, por lo que su formato debería ser parametro1=valor1&parametro2=valor2. También se puede indicar un array asociativo de pares clave/valor que se convierten automáticamente en una cadena tipo *query string* |
| **dataType** | El tipo de dato que se espera como respuesta. Si no se indica ningún valor, jQuery lo deduce a partir de las cabeceras de la respuesta. Los posibles valores son: xml (se devuelve un documento XML correspondiente al valor responseXML), html (devuelve directamente la respuesta del servidor mediante el valor responseText), script (se evalúa la respuesta como si fuera JavaScript y se devuelve el resultado) y json (se evalúa la respuesta como si fuera JSON y se devuelve el objeto JavaScript generado) |
| **error** | Indica la función que se ejecuta cuando se produce un error durante la petición. Esta función recibe el objeto XMLHttpRequest como primer parámetro, una cadena de texto indicando el error como segundo parámetro y un objeto con la excepción producida como tercer parámetro |
| **ifModified** | Permite considerar como correcta la petición solamente si la respuesta recibida es diferente de la anterior respuesta. Por defecto su valor es false |
| **processData** | Indica si se transforman los datos de la opción data para convertirlos en una cadena de texto. Si se indica un valor de false, no se realiza esta transformación automática |
| **success** | Permite establecer la función que se ejecuta cuando una petición se ha completado de forma correcta. La función recibe como primer parámetro los datos recibidos del servidor, previamente formateados según se especifique en la opción dataType |
| **timeout** | Indica el tiempo máximo, en milisegundos, que la petición espera la respuesta del servidor antes de anular la petición |
| **type** | El tipo de petición que se realiza. Su valor por defecto es GET, aunque también se puede utilizar el método POST |
| **url** | La URL del servidor a la que se realiza la petición |

## Referencias

* <http://librosweb.es/ajax/>
* <http://www.uco.es/~lr1maalm/manualdeajax.pdf>
* <http://www.w3schools.com/ajax/default.ASP>
* <http://api.jquery.com/jquery.ajax/>
* <http://www.w3schools.com/jquery/jquery_ajax_intro.asp>

## Ejemplos

* <http://www.esasp.net/2009/11/ajax-jquery-en-aspnet.html>
* <http://aspsnippets.com/Articles/Call-ASPNet-Page-Method-using-jQuery-AJAX-Example.aspx>
* <http://codehero.co/jquery-desde-cero-ajax/>

# Json

JSON (JavaScript Object Notation) es un formato para el intercambios de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. JSON nació como una alternativa a XML, el fácil uso en JavaScript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

## Sintaxis JSON

JSON utiliza lo que se denomina una sintaxis de pares “nombre/valor”. Para asignar a un nombre un valor, se utilizan los “:”, lo cual sería comparable con el “=” de cualquier otro lenguaje.

***"Nombre”: "Geeky Theory"***

Los valores que podemos manejar en JSON son los siguientes:

1. Numero
2. String
3. Booleano
4. Array
5. Objeto
6. Null

### Objetos

Los objetos en JSON deben estar entre llaves (“{}”) como por ejemplo

***{ "NombreFruta":"Manzana" , "Cantidad":20 }***

### Array

El contenido de un array en JSON debe de ir entre corchetes (“[]”) como por ejemplo

**{**

**"Frutas": [**

**{"NombreFruta":"Manzana" , "cantidad":10 },**

**{"NombreFruta":"Pera" , "cantidad":20 },**

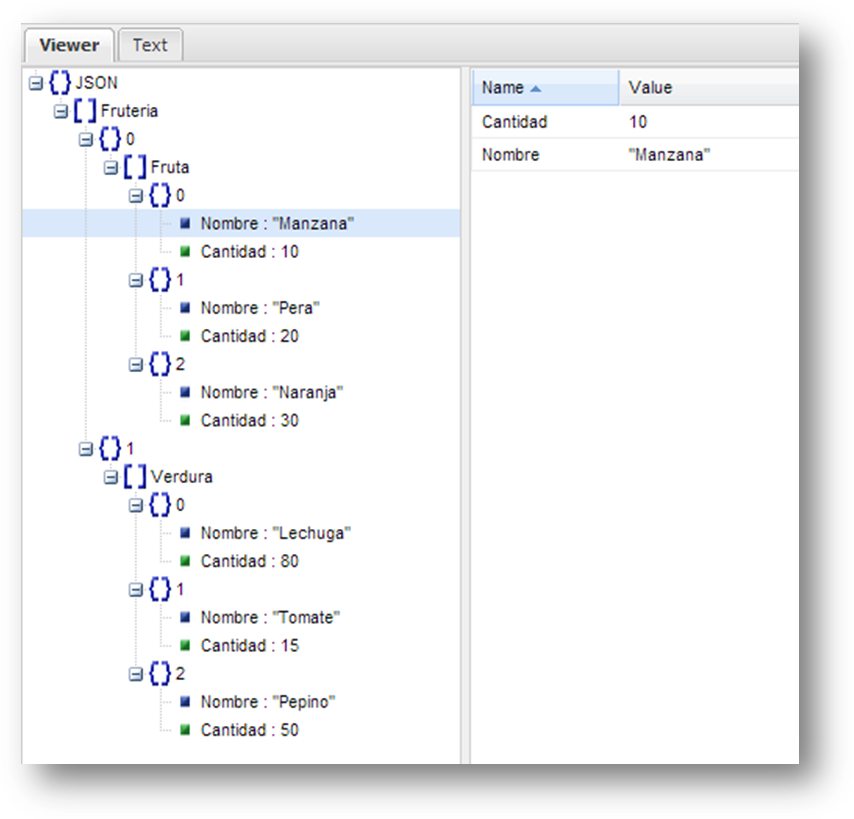
**{"NombreFruta":"Naranja" , "cantidad":30 }**

**]**

**}**

## Herramientas de Desarrollo

Existen herramientasonline que ayudan a visualizar mejor un JSON. Una de las mejores herramientas que he visto es la página [JSON Viewer](http://jsonviewer.stack.hu/). Si introducimos nuestro ejemplo observamos lo siguiente:

[](http://i2.wp.com/geekytheory.com/wp-content/uploads/2013/10/Captura11.png)

## Formas de Generación de JSON

### Método ToJSON() con JavaScriptSerializer

class DatosPersona

{

public string Nombre { get; set; }

public string Apellidos { get; set; }

public string TipoDocumento { get; set; }

public string Documento { get; set; }

*public string ToJSON()*

*{*

*System.Web.Script.Serialization.JavaScriptSerializerjsonSerializer*

*= new System.Web.Script.Serializaion.JavaScriptSerializer();*

*return jsonSerializer.Serialize(this);*

*}*

}

### Serializacion utilizando JSON.Net

class Persona {

public Guid Id { get; set; }

public string Nombre { get; set; }

public string Apellidos { get; set; }

public string FechaNacimiento { get; set; }

}

public void serializar{

var devjoker = new Persona() {

Id = Guid.NewGuid(),

Nombre = "Pedro",

Apellidos = "Herrarte",

FechaNacimiento = new DateTime (1975,01,13).ToString("d")

};

var json = Newtonsoft.Json.JsonConvert.SerializeObject(devjoker);

}

## JSON en Entity Framework

Todo lo explicado anteriormente es aplicable a clases simples o clases con relaciones simples; en esos casos la serialziacion y deserializacion se puede hacer de manera automática con JSON.Net.

Pero en el caso de Entity Framework cuando existen relaciones bidireccionales entre dos o mas clases, a la hora de serializar el objeto u entidad, se produce lo que se de denomina una excepción por existencia de referencia circular, lo cual JSON no lo maneja.

Es por ello que existen varias formas para solucionar esto, acá se detallaran a modo de ejemplo solo tres de ellas.

1. Clases simples para las vitas: consta de crear clases que solo contengan atributos simples que no generen la excepción de la referencia circular. Si bien es muy fácil de aplicar es tener una duplicación de información.
2. Exclusión de atributos: consta de marcar aquellos atributos que causan la excepción para que no sean serializados. Si bien es efectivo, se esta solucionando modificando el modelo para solucionar algo que es problema de la vista.
3. Proyección o método personalizado de socialización: consta de que cada clase genere la estructura de su JSON. Es similar a tener el método toString(), en este caso seria toJSON().

## Aplicación práctica en GeoParking

En esta sección se mostrara a modo de ejemplo como se trabaja la serializaron JSON de las entidades del proyecto. Aquí se mostrara la aplicación sobre la entidad “PlayaDeEstacionamiento”.

/// <summary>

/// retorna el objeto en formato JSON

/// </summary>

/// <returns>ObjetoJSON</returns>

public String ToJSONRepresentation()

{

StringBuilder sb = new StringBuilder();

JsonWriter jw = new JsonTextWriter(new StringWriter(sb));

jw.Formatting = Formatting.Indented;

//comienzo a escribrir el objeto

jw.WriteStartObject();

jw.WritePropertyName("Id");

jw.WriteValue(this.Id);

jw.WritePropertyName("Nombre");

jw.WriteValue(this.Nombre);

jw.WritePropertyName("Mail");

jw.WriteValue(this.Mail);

jw.WritePropertyName("Telefono");

jw.WriteValue(this.Telefono);

jw.WritePropertyName("TipoPlaya");

jw.WriteValue(this.TipoPlayaStr);

jw.WritePropertyName("Latitud");

jw.WriteValue(this.Direcciones[0].Latitud);

jw.WritePropertyName("Longitud");

jw.WriteValue(this.Direcciones[0].Longitud);

//DIRECCIONES

jw.WritePropertyName("Direcciones");

jw.WriteStartArray();

int i;

i = 0;

for (i = 0; i < this.Direcciones.Count; i++)

{

jw.WriteStartObject();

jw.WritePropertyName("Calle");

jw.WriteValue(Direcciones[i].Calle);

jw.WritePropertyName("Numero");

jw.WriteValue(Direcciones[i].Numero);

jw.WriteEndObject();

}

jw.WriteEndArray();

//SERVICIOS

jw.WritePropertyName("Servicios");

jw.WriteStartArray();

int j;

j = 0;

for (j = 0; j < this.Servicios.Count; j++)

{

jw.WriteStartObject();

jw.WritePropertyName("TipoVehiculo");

jw.WriteValue(Servicios[j].TipoVehiculoStr);

jw.WritePropertyName("Capacidad");

jw.WriteValue(Servicios[j].Capacidad);

jw.WriteEndObject();

}

jw.WriteEndArray();

//HORARIOS

jw.WritePropertyName("Horarios");

jw.WriteStartArray();

int k;

k = 0;

for (k = 0; k < this.Horarios.Count; k++)

{

jw.WriteStartObject();

jw.WritePropertyName("Dia");

jw.WriteValue(Horarios[k].DiaAtencionStr);

jw.WritePropertyName("HoraDesde");

jw.WriteValue(Horarios[k].HoraDesde);

jw.WritePropertyName("HoraHasta");

jw.WriteValue(Horarios[k].HoraHasta);

jw.WriteEndObject();

}

jw.WriteEndArray();

//PRECIOS

jw.WritePropertyName("Precios");

jw.WriteStartArray();

int l;

l = 0;

for (l = 0; l < this.Precios.Count; l++)

{

jw.WriteStartObject();

jw.WritePropertyName("TipoVehiculo");

jw.WriteValue(Precios[l].TipoVehiculoStr);

jw.WritePropertyName("Dia");

jw.WriteValue(Precios[l].DiaAtencionStr);

jw.WritePropertyName("Tiempo");

jw.WriteValue(Precios[l].TiempoStr);

jw.WritePropertyName("Monto");

jw.WriteValue(Precios[l].Monto);

jw.WriteEndObject();

}

jw.WriteEndArray();

jw.WriteEndObject();

return sb.ToString();

}

## JSON y JavaScript

En esta sección se explicara como es el manejo de JSON con JavaScript y su interacción, dando métodos y ejemplo utilizados.

* En el ejemplo siguiente se usa **JSON.parse** para convertir una cadena JSON en un objeto.

var jsontext = {"firstname":"Jesper","surname":"Aaberg","phone":["555-0100","555-0120"]}';

var contact = JSON.parse(jsontext);

document.write(contact.surname + ", " + contact.firstname);

* En el ejemplo siguiente se usa **JSON.stringify** para convertir un objeto en una cadena JSON

var nuevaCadena = JSON.stringify(libros);

alert(nuevaCadena);

**Se vera**

'[{"Titulo": "El señor de los anillos", "Autor": "J.R.R. Tolkien"},

{"Titulo": "Cancion de hielo y fuego", "Autor": "George RR Martin"},

{"Titulo": "Los Pilares de la Tierra", "Autor": "Ken Follett"}]';

## Referencias

* <http://json.org/>
* <http://www.json.org/js.html>
* <http://jhernandz.es/noticia/uso-json-con-javascript>
* <http://www.devjoker.com/contenidos/catss/459/Serializacion-JSON-con-NET.aspx>
* <http://james.newtonking.com/json>
* <http://www.devjoker.com/contenidos/catss/505/JSONNET-Mejorando-la-serializacion-JSON-con-NET.aspx>
* <http://stackoverflow.com/questions/6201529/turn-c-sharp-object-into-a-json-string-in-net-4>
* <http://www.brian-driscoll.com/2012/11/just-code-tojson-and-toxml-object.html?goback=.gde_136510_member_185144755>
* <http://speakingin.net/2007/10/04/trucos-creando-un-metodo-de-extension-tojson-con-net-35/>

# Unit Test

**Aplicado a Desarrollo con capas utilizando EF como Capa de datos**

## Introducción

En este documento, se mostrara una de las formas posibles para realizar código pensando en el test unitario. El test unitario es importante en el desarrollo de un sistema para verificar rápidamente que los cambios introducidos, no generaron bugs en las implementaciones de los métodos.

Para realizar tests unitarios, se debe sacrificar tiempo de desarrollo a dicha tarea, para reducir las horas de desarrollo de pruebas unitarias es necesario desarrollar siempre con el concepto de código testeable.

Código testeable es aquel que es fácil de observar, dados ciertas entradas es fácil determinar la salida, por ejemplo el siguiente método

public int Add(int x, int y)

{

    return x + y;

}

La otra cualidad que debe tener el código testeable, es que debe estar aislado, es decir cada método debe centrarse en una acción específica y dejar el resto a otros métodos.

## Investigación

La investigación fue realizada en internet, se busco en diversos foros y sitios especializados en la tecnología, encontrando la información mas completa en el paper ¨Testability and Entity Framework 4.0¨ de Scott Allen. Si bien el paper habla sobre la versión 4.0 de EF, sirvió de base para entender bien el contexto de las pruebas unitarias, luego se amplio esa información con la obtenida en el articulo “Using Microsoft Fakes to Unit Test Entity Framework” y se buscaron ejemplos para consolidar lo investigado.

## Resultados

Es necesario contar con una arquitectura que implemente el patrón repositorio, para definir los métodos a utilizar por los distintos repositorios y reutilizar el código, permitiendo en las clases de testing trabajar sobre estos métodos definidos en la interfaz. Se puede ver un ejemplo de este patrón en el proyecto “Empleados”

A la hora de realizar las pruebas unitarias sobre la capa de datos, se debe crear una estructura falsa de datos para aislar el código de las distintas acciones que pueden fallar, como por ejemplo el acceso a la base de datos. Para crear esta estructura, es necesario contar con una Interfaz IRepositorio la cual define los los métodos para acceder a los datos, para una mejor comprensión, hago un ejemplo:

public interface IRepositorio<TEntity> where TEntity : class

{

TEntity FindById(int id);

IList<TEntity> FindAll();

IList<TEntity> FindWhere(Expression<Func<TEntity, bool>> predicate);

TEntity Create(TEntity t);

int Delete(Expression<Func<TEntity, bool>> predicate);

int Update(TEntity t);

}

Esta interfaz debe ser implementada por los repositorios, y por la clase que creara la estructura de datos falsa para usar en las pruebas unitarias, (también es posible crear una interfaz especifica que implemente esta interfaz por cada repositorio que contenga un tipo de entidades).

En la capa de reglas, que serán las encargadas de instanciar los repositorios, se debe definir un constructor con parámetros del tipo IRepositorio, los cuales se usaran en el momento de hacer el test para usar la estructura de datos falsa (el constructor por defecto instanciara los repositorios persistentes en la base de datos).

Para crear la estructura de datos falsa, se debe crear una clase de repositorio que implemente IRepositorio y traiga datos por defecto, por ejemplo:

public class FakeProductsRepository : IProductsRepository

{

public List<Product> FindAll()

{

return new List<Product>

{

new Product { Name = "PASTE" }

new Product { Name = "BRUSH" }

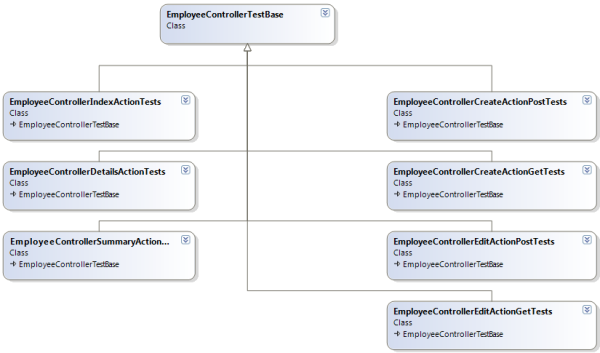
},

}

}

De esta forma se salta la conexión con la base de datos y se prueba únicamente la lógica del método.

Para probar los métodos, se debe crear un proyecto de unit test, y dentro de este las clases necesarias para las estructuras de datos falsas, y las pruebas necesarias. Es recomendable crear una clase base por cada tipo de datos que se va a testear, la cual va a instanciar a la clase correpondiente de la capa de negocio, enviando por parámetro el repositorio falso previamente creado; y luego tener clases que hereden de esta para cada conjunto de operaciones relacionadas al objeto en si (se muestra un diag. De clases a modo de ejemplo).



Si se realiza todo lo aquí comentado, las clases de los tests individuales serán simples, ya que se dedicaran a probar una pequeña porción de código aislado, por lo que no deberán fallar por cuestiones no relacionadas a la lógica del método.

Agrego ejemplos de clases de tests:

[TestClass]

public class EmployeeControllerCreateActionPostTests

           : EmployeeControllerTestBase {

    [TestMethod]

    public void ShouldAddNewEmployeeToRepository() {

        \_controller.Create(\_newEmployee);

        Assert.IsTrue(\_repository.Contains(\_newEmployee));

    }

    [TestMethod]

    public void ShouldCommitUnitOfWork() {

        \_controller.Create(\_newEmployee);

        Assert.IsTrue(\_unitOfWork.Committed);

    }

    // ... more tests

    Employee \_newEmployee = new Employee() {

        Name = "NEW EMPLOYEE",

        HireDate = new System.DateTime(2010, 1, 1)

    };

}

[TestClass]

public class EmployeeControllerIndexActionTests

           : EmployeeControllerTestBase {

    [TestMethod]

    public void ShouldBuildModelWithAllEmployees() {

        var result = \_controller.Index();

        var model = result.ViewData.Model

                      as IEnumerable<Employee>;

        Assert.IsTrue(model.Count() == \_employeeData.Count);

    }

    [TestMethod]

    public void ShouldOrderModelByHiredateAscending() {

        var result = \_controller.Index();

        var model = result.ViewData.Model

                     as IEnumerable<Employee>;

        Assert.IsTrue(model.SequenceEqual(

                       \_employeeData.OrderBy(e => e.HireDate)));

    }

    // ...

}

## Bibliografia

* http://msdn.microsoft.com/en-us/library/ff714955.aspx
* http://vaideeswaranr.blogspot.com.ar/2013/02/using-microsoft-fakes-to-unit-test.html
* http://stackoverflow.com/questions/19287055/unit-testing-and-entity-framework