Creación del Modelo

En este capítulo crearemos el Modelo de nuestra aplicación. Recordemos que sus componentes representan los datos del dominio, son responsables de aportar la lógica de negocio y deben aportar los mecanismos de persistencia del sistema.

Por simplificar el código y nos desviarnos mucho de nuestros objetivos, almacenaremos los posts de nuestro blog en una base de datos LocalDb, y utilizaremos *Entity Framework* para el mapeo E/R y la gestión de los datos. Adicionalmente, crearemos un componente de gestión de más alto nivel que incluya la lógica de negocio y aísle al resto de componentes del sistema de persistencia elegido.

> Podemos implementar el Modelo de las aplicaciones orientadas a datos usando cualquier tecnología: Entity Framework, NHibernate, micro-ORMs, ADO.NET clásico, etc. ASP.NET MVC no establece ningún requisito ni directriz al respecto.

Dado que el objetivo de este curso no es profundizar en Entity Framework, lo utilizaremos de la forma más simple y directa posible: **el enfoque denominado "code-first"**. Como su nombre sugiere, este enfoque permite que nos centremos en la definición de las entidades de datos desde el código fuente, obviando todos los detalles relativos a la persistencia. Es decir, definiremos las entidades de datos que vamos a necesitar, y, con muy poca información adicional, **Entity Framework se encargará de crear la base de datos por nosotros** y gestionar automáticamente la persistencia.

Así, los artilugios que necesitamos añadir a nuestro proyecto relativos a Entity Framework son:

- Las entidades de datos que necesitamos. En nuestro caso, sólo necesitaremos de momento la entidad "Post".
- El contexto de datos de Entity Framework, que es la clase que nos abstrae del almacén de persistencia subyacente y nos ofrece mecanismos para obtener y almacenar entidades en él de forma muy sencilla y transparente.
- Para facilitarnos la vida durante el desarrollo, crearemos también una clase de inicialización de datos. Veremos más adelante lo que es.

Creación de las entidades

Ya hemos comentado que utilizando el enfoque "code-first" de Entity Framework debemos centrarnos en el código de las entidades de datos que modelan el dominio de nuestra aplicación. Normalmente se trata de clases POCO (*Plain Old CLR Objects*, u objetos planos), como la siguiente, que cumple los requisitos expresados en el capítulo anterior:

```
public class Post
{
   public int PostId { get; set; }
   public string Title { get; set; }
   public string Code { get; set; }
   public string Text { get; set; }
   public DateTime Date { get; set; }
   public string Author { get; set; }
}
```

Atendiendo a las convenciones de ubicación de archivos propuesta por ASP.NET MVC, un buen sitio para esta clase sería la carpeta /Models del proyecto.

El contexto de datos de Entify framework

Muy básicamente, el contexto de datos es la clase que nos permitirá acceder fácilmente al almacén de persistencia para leer o grabar entidades. Un contexto de datos simple para nuestro proyecto podría ser el siguiente:

```
public class BlogContext: DbContext
{
   public BlogContext(): base("DefaultConnection") { }

   public DbSet<Post> Posts { get; set; }
}
```

De esta forma tan sencilla estamos creando un contexto de datos propio (heredando de `DbContext`, una clase propia de Entity Framework) que contiene un conjunto de objetos de tipo `Post` (`DbSet<Post>`) al que accederemos utilizando la propiedad llamada `Posts` del contexto. Los `DbSet` actúan de forma parecida a otras colecciones en memoria de .NET: podemos consultarlas mediante sentencias LINQ y manipular su contenido

usando métodos como `Add()` o `Remove()`. Aunque en este caso nuestro dominio es muy simple y sólo necesitamos un conjunto de datos, lo normal es que en el contexto de datos de una aplicación tengamos definidos varios `DbSet`, uno por cada tipo de entidad a la que necesitaremos acceder.

El constructor utilizado indica que la cadena de conexión a la base de datos a utilizar se encuentra definida en el web.config bajo la denominación "DefaultConnection". Esta cadena de conexión viene de serie en proyectos MVC, y apunta la instancia de LocalDb de la máquina local, pero podríamos modificarla fácilmente en el archivo de configuración, o bien indicar otro valor (o incluso directamente otra cadena de conexión) en el constructor del contexto de datos.

Inicializadores y datos de prueba

La primera vez que intentemos acceder al contenido de un `DbSet` del contexto de datos, ya sea para consultarlo o modificarlo, Entify Framework comprobará si existe la base de datos que actuará como almacén para estos datos, y si es necesario la creará de forma automática atendiendo a la estructura de datos que le hemos ordenado gestionar. En el ejemplo anterior, EF es lo suficientemente inteligente como para determinar que necesita crear una tabla para almacenar entidades de tipo `Post` , e incluso los campos (tipos incluidos) que debe introducir en ella para que se puedan almacenar entidades de este tipo.

Sin embargo, si la base de datos existía, Entity Framework comprueba si su estructura coincide con la que sería necesaria atendiendo a la estructura del contexto de datos y entidades, fallando estrepitosamente en tiempo de ejecución si esto no ocurre.

Para evitar estos problemas y dar un poco de flexibilidad a este proceso, en Entity Framework existen los **inicializadores**, que son clases mediante las cuales podemos indicar qué queremos que haga Entity Framework durante su inicialización, y podemos aprovechar para añadir datos de prueba, algo que resulta bastante útil durante el desarrollo. El siguiente código muestra un inicializador personalizado que indica que la base de datos debe ser eliminada y creada de nuevo cuando cambie la estructura de las entidades o el contexto, y aporta datos de inicialización para que la base de datos aparezca ya con alguna información:

```
public class BlogInitializer :
DropCreateDatabaseIfModelChanges<BlogContext>
{
   protected override void Seed(BlogContext context)
   {
```

```
        context.Posts.Add(new Post() {
            Author = "José M. Aguilar",
            Title = "Hello, world!",
            Code = "hello-world",
            Date = new DateTime(2005, 8, 12),
            Text = "Hi, everybody, this is my first blog post!"
        });
        context.Posts.Add(new Post() {
            Author = "José M. Aguilar",
            Title = "Second post",
            Code = "second-post",
            Date = new DateTime(2005, 8, 22),
            Text = "Well, it's time to start writing... :)"
        });
        context.Posts.Add(new Post() {
            Author = "José M. Aguilar",
            Title = "Today is my birthday",
            Code = "today-is-my-birthday",
            Date = new DateTime(2005, 9, 1),
            Text = "Today is my birthday! I accept gifts ;)"
        });

        // Other seed data

        base.Seed(context);
    }
}
```

La clase `DropCreateDatabaseIfModelChanges` es uno de los inicializadores que acompañan de serie a Entity Framework, y cuya intencionalidad queda totalmente clara simplemente viendo su nombre. Existen otros, como `DropCreateDatabaseAlways` o `CreateDatabaseIfNotExists` (el usado por defecto por EF) que también podríamos haber empleado, heredando de ellos, si estas estrategias fueran más apropiadas para nuestras necesidades.

En cualquier caso, para que un inicializador se tenga en cuenta, **debemos añadir el siguiente código para que se ejecute durante el arranque del sistema**. Un buen sitio podría ser en el evento `Application_Start` en el archivo Global.asax.cs

```
public class MvcApplication : System.Web.HttpApplication
{
   protected void Application_Start()
   {
      Database.SetInitializer(new BlogInitializer()); // Set
default initializer
      AreaRegistration.RegisterAllAreas();
      FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
      RouteConfig.RegisterRoutes(RouteTable.Routes);
      BundleConfig.RegisterBundles(BundleTable.Bundles);
   }
}
```

Componente de gestión

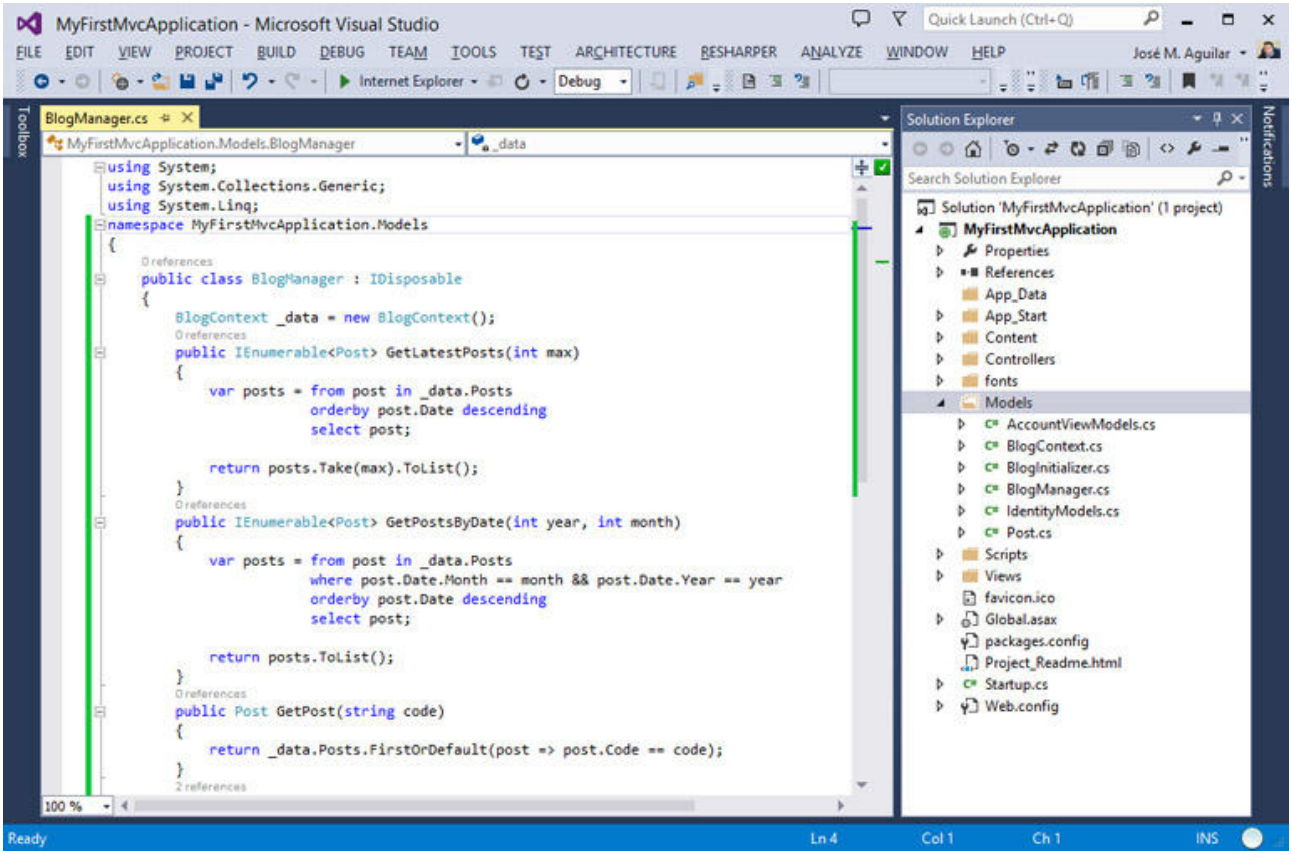
Ya tenemos resuelta la infraestructura de acceso a datos. Ahora crearemos el componente del Modelo que aportará las operaciones de alto nivel que podremos utilizar desde el resto del sistema, y que vienen definidas por los requisitos que hemos especificado anteriormente. Para ello, crearemos, de nuevo dentro de la carpeta /Models, la clase `BlogManager`.

Como se puede observar, instanciamos a nivel de clase el modelo de datos de Entity Framework, e implementamos los tres únicos métodos necesarios; el primero de ellos para obtener los últimos posts, el segundo para obtenerlos según el mes y año de publicación, y el tercero para obtener un post dado su código. La implementación del interfaz `IDisposable` permitirá la liberación de los recursos usados por el contexto (por ejemplo, internamente utiliza una conexión física a la base de datos).

```
public class BlogManager: IDisposable
{
   BlogContext _data = new BlogContext();

   public IEnumerable<Post> GetLatestPosts(int max)
   {
      var posts = from post in _data.Posts
                  orderby post.Date descending
                  select post;
      return posts.Take(max).ToList();
   }
```

```
   public IEnumerable<Post> GetPostsByDate(int year, int month)
   {
      var posts = from post in _data.Posts
                  where post.Date.Month == month &&
post.Date.Year == year
                  orderby post.Date descending
                  select post;
      return posts.ToList();
   }

   public Post GetPost(string code)
   {
      return _data.Posts.FirstOrDefault(post => post.Code ==
code);
   }

   public void Dispose()
   {
      _data.Dispose();
   }
}
```

La siguiente captura de pantalla muestra el resultado de las tareas realizadas hasta el momento en el Modelo. Como se puede observar, se han creado en la carpeta /Models los archivos BlogContext.cs, BlogInitializer.cs, BlogManager.cs, y Post.cs:



Con esto tendríamos completamente definidos los componentes de nuestro Modelo. Si te fijas, hasta este momento no hemos introducido ninguna particularidad del framework MVC; de hecho, estos mismos componentes podrían ser útiles en una aplicación ASP.NET WebForms, o incluso de escritorio, sin ninguna modificación.

> **Los componentes del Modelo deben ser ajenos a la tecnología sobre la que está construido el sistema al que sirven.**

A continuación pasaremos a definir las rutas, estableciendo las URLs que entenderá nuestra aplicación y definiendo en qué controladores y acciones se implementará su procesamiento.
