Práctica 2: Añadiendo funcionalidades

Objetivo

El objetivo de esta práctica es que profundices en la estructura de las aplicaciones y proyectos ASP.NET MVC, mediante la observación e investigación del proyecto que hemos ido desarrollando a lo largo de este módulo.

1. Enlazar el blog con el sitio web

1. Hasta ahora hemos accedido a las distintas funcionalidades del motor de blogs introduciendo directamente la URL en el navegador. Esta primera parte de la práctica consiste en crear una nueva opción en el menú principal de la aplicación (donde se encuentran los enlaces a las páginas "Home", "About" y "Contact") y hacer que enlace con la visualización del listado de últimos posts publicados.

2. En primer lugar, abre el Layout del sitio web (`/Views/Shared/_Layout.cshtml`) y observa que existe una lista sin orden (`<ul>`) en la que se encuentran definidas las opciones del menú. Fíjate en la forma en que se están creando los enlaces llamando a `Html.ActionLink()`. Se trata de un método helper que nos ayuda a crear etiquetas `<a>` dirigidas hacia controladores y acciones concretas; ya lo estudiaremos más adelante, de momento simplemente intenta adivinar qué significan cada uno de los parámetros utilizados.

3. Incluye un nuevo elemento en la lista, e introduce en él un tag `<a>` con el atributo `href` apuntando hacia la URL en la que se puede encontrar la funcionalidad de visualización del listado de los últimos posts publicados (/blog). Ejecuta la aplicación para comprobar que todo funciona correctamente.

4. Prueba a continuación, aunque sea de forma intuitiva en este momento, a utilizar el helper `Html.ActionLink()` para generar el enlace. Reflexiona sobre la ventaja de utilizar este método sobre la codificación manual del enlace que hiciste previamente.

   (Pista: ¿qué ocurriría si modificamos la tabla de rutas y hacemos que cambie la URL de acceso a las funcionalidades del blog?)

2. Enlazar el listado de posts con la visualización del post

1. En esta sección vamos a enlazar el listado de posts (al que ya debes poder acceder desde el menú principal de la aplicación), con la visualización de cada post, es decir, con la

acción `ViewPost()` que hemos ido implementando a lo largo del módulo.

2. Accede a la vista */Views/Blog/Index.cshtml*, y modifícala para que en cada elemento se incluya un enlace hacia la página donde se puede visualizar el post completo (la acción "ViewPost"), de forma que ya tengamos la navegación completa de nuestro pequeño motor de blogs. La fórmula más sencilla de hacerlo sería incluir algo como lo siguiente:

```
<a href="/blog/@post.Code">Read post...</a>
```

3. Observa que, de nuevo, estamos introduciendo URLs en nuestro código, lo cual introduce en el código de la vista una dependencia muy rígida respecto al contenido de la tabla de rutas actual. Es decir, si cambiamos la ruta de acceso a la acción "ViewPost", nuestros usuarios no podrían visualizar los posts completos, puesto que la URL especificada en la vista no se modificaría de forma automática.

   Comprueba que el sistema funciona de forma correcta, y después realiza este pequeño cambio en la tabla de rutas:

```
routes.MapRoute(
   name: "ViewPost",
   url: "view/{code}",
   defaults: new { controller = "Blog", action = "ViewPost" }
);
```

   Ejecuta de nuevo la aplicación, y verás cómo el link entre el listado de posts y la visualización individual ha dejado de funcionar.

4. Sustituye la etiqueta `<a>` que has introducido anteriormente por el siguiente código:

```
@Html.ActionLink("Read post...", "ViewPost", new { code = post.Code} )
```

   En él, estamos generando un enlace de forma sensible al contenido de la tabla de rutas de nuestra aplicación, por lo que cualquier cambio en ésta será tenido en cuenta. La sobrecarga de `Html.ActionLink()` que estamos utilizando asume que el controlador será el mismo que está siendo ejecutado en este momento (Blog), y observa que estamos suministrando el parámetro que necesita la acción ViewPost utilizando un objeto anónimo.

   Si vuelves a probar la aplicación, verás que los cambios en la tabla de rutas ahora se ven reflejados de forma inmediata en los enlaces.

3. Añadir visualización de comentarios al blog

En esta tercera parte de la práctica vamos a plantearnos un objetivo algo más ambicioso: añadir a la vista "ViewPost" (donde se visualizan los posts completos) los comentarios asociados al artículo que esté siendo visualizado.

Los pasos que debes seguir son los descritos a continuación:

1. Añade al modelo de datos de la aplicación una nueva entidad, llamada `Comment`, definida de la siguiente forma:

```
public class Comment
{
   public int Id { get; set; }
   public string Text { get; set; }
   public string Author { get; set; }
   public DateTime Date { get; set; }
}
```

2. Actualiza la entidad `Post` para añadirle una propiedad que permita acceder a la colección de objetos `Comment` asociados al artículo. El resultado debe ser algo así:

```
public class Post
{
   public Post()
   {
      Comments = new HashSet<Comment>();
   }
   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; }
   public ICollection<Comment> Comments { get; set; }
}
```

3. Añade algunos comentarios a los posts de prueba que introdujimos en el inicializador de Entity Framework. Puedes seguir, por ejemplo, el siguiente patrón para añadir artículos con sus correspondientes comentarios:

```
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!",
      Comments = new[] {
         new Comment() { Author = "John Doe", Date = new
DateTime(2005, 8,13), Text="Hey, this is great!" },
         new Comment() { Author = "Peter Petersen", Date =
new DateTime(2005, 8,14), Text="Good news! Keep writing,
please :)" }
      }
   }
);
```

4. A continuación, debemos modificar ligeramente la clase gestora para asegurar que cuando se obtenga un objeto de tipo `Post`, se incluya en él la colección de comentarios asociados al mismo. Los cambios a realizar en BlogManager.cs son:

```
using System.Data.Entity;
...
public class BlogManager: IDisposable
{
   ...
   public Post GetPost(string code)
   {
      return _data.Posts.Include
(p=>p.Comments).FirstOrDefault(post => post.Code == code);
   }
}
```

5. Por último, actualiza la vista "ViewPost" para que muestre los comentarios asociados al Post actual, por ejemplo insertando este código:

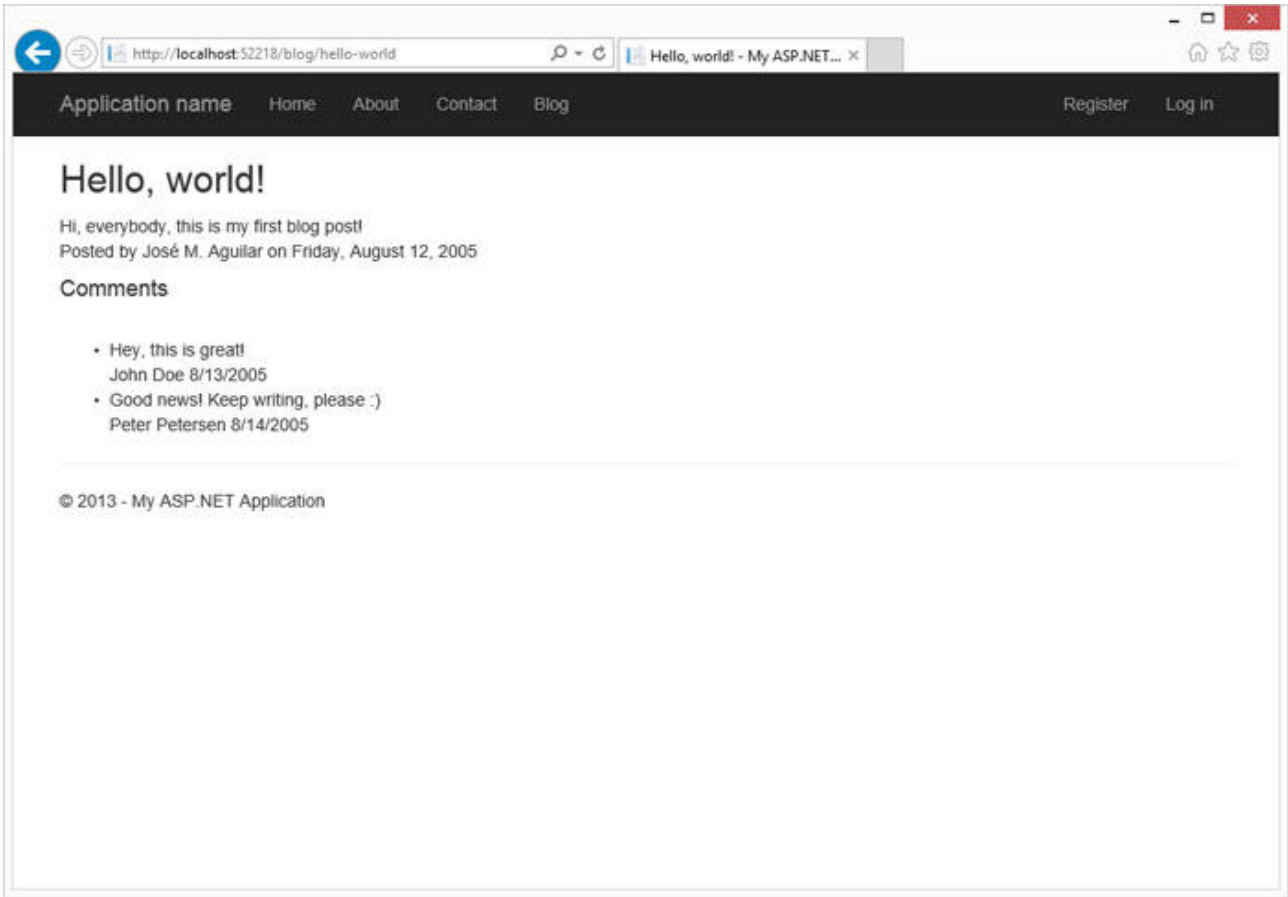
```
@if(Model.Comments.Any())
{
        <h4>Comments</h4>
   <ul>

        @foreach(var comment in Model.Comments)
   {
      <li>

        <div>@comment.Text</div>
        <div>@comment.Author
        @comment.Date.ToShortDateString()</div>
      </li>
   }
```

```
    </ul>
}
```

El resultado obtenido debería ser aproximadamente como el mostrado en la siguiente captura de pantalla:



Encontrarás el proyecto completo para Visual Studio 2015 y 2013 en el área de recursos. En este proyecto, además, se ha implementado una funcionalidad adicional, el acceso desde el menú principal al archivo del blog, que puedes estudiar para comprender aún mejor la forma de programar con ASP.NET MVC.