MVC

MVC es el significado de:

- Modelo

- Información de la aplicación y el comportamiento en términos del dominio de su problema y es independiente de la interfaz gráfica.

- Solo representa la información y no está ligada a la interfaz gráfica.

- En el modelo se incluyen entidades como película, cliente, renta, transacción, pero ninguna relacionada a la interfaz.

- Son representados como POCOs (Plain Old CLR Objects).

- Vista

- Representa la parte gráfica de la aplicación, en programación web el código html y css.

- Controlador

- Es el responsable de manejar la petición http para consultar el modelo y de ahí mandar la información a la vista.

- Es un patrón de arquitectura para implementar interfaces gráficas.

- Diseñada en 1970.

- Muy usada en la web.

- Se encuentra en ASP.NET, Ruby on Rails y Express entre otros.

- Otorga la característica de dar mejor mantenimiento al proyecto y una mejor separación de responsabilidades.

Al crear el proyecto en ASP.NET con MVC tenemos:

App\_Data:

- Lugar donde se guarda el archivo de base de datos.

App\_Start:

- Incluye las clases que se van a llamar cuando la aplicación se inicializa.

Archivo RouteConfig.cs:

- Contiene información de las rutas a usar.

- Por defecto contiene una ruta con el nombre default, con un patrón de url asignado, si se empareja con ese patrón coincidiera conforme lo indicado.

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute( // Realiza el routing para mandarlo a un controlador.

name: "Default", // Nombre de patrón a usar.

url: "{controller}/{action}/{id}", // Url con el formato del patrón.

// Asigna la primera parte de la ruta a la variable controller.

// Asigna la segunda parte de la ruta a la variable action.

// Asigna la tercera parte de la ruta a la variable id.

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Ejecuta un método por defecto si no cumplió con el patrón de la ruta, asignando valores default para los casos no dados.

);

}

- Llama a la clase controller indicada, action ejecuta el método de la clase controller indicado y pasa el parámetro si se asignó.

- En la opción defaults asigna los parámetros si no se pasaron, si solo se pasó el controlador, por defecto llama a la acción indicada en action.

- Ruta: /movies/popular

- movies (controller): Coincide a MoviesController.

- popular (action): Coincide al método Popular de la clase MoviesController.

- Ruta: /movies/edit/1

- movies (controller): Coincide a MoviesController.

- edit (action): Coincide al método Edit(int id) de la clase MoviesController.

- 1 (id): Pasa el parámetro id al método Edit(int id), como el tipo declarado en el método.

- Ruta: /movies:

- movies (controller): Coincide a MoviesController.

- Index (action): Llama a la acción por defecto MoviesController.Index().

- Opcional (id): Al ser UrlParameter.Optional no asigna nada pues el parámetro id es declarado como opcional.

- Content:

- En esta carpeta se guardan los archivos css para dar estilos y formato a las vistas.

- También se acostumbra guardar imágenes necesarias, contenido multimedia de la página y otros assets.

- Controllers:

- Aquí se guardan todos los controladores que servirán de medio para obtener la información de los métodos http y procesarla para mostrar una respuesta en la vista.

- Por defecto tiene el HomeController el cual nos va a enviar a la página home del sitio.

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

}

- Fonts:

- Carpeta donde se guardan los distintos estilos de letra a usar en la aplicación, es recomendable moverlo a la carpeta de contenido.

- Models:

- Aqui se guardan todas las clases de nuestra aplicación de dominio como entidades o cálculos de la lógica de negocio.

- Scripts:

- Carpeta donde se guardan los archivos de JavaScript necesarios para el funcionamiento de la aplicación.

- Views:

- Carpeta donde se guardan las distintas vistas (html) de la aplicación.

- Por cada controlador debe haber una carpeta de vistas con el mismo nombre y dentro de cada carpeta una vista por cada método de la aplicación.

- Para HomeController está la carpeta Controller (dentro de Views) y las vistas que hacen referencia a los métodos del controlador:

- About.cshtml

- Contact.cshtml

- Index.cshtml

- Además contiene la carpeta shared la cual sirve para compartir un diseño base entre varias vistas.

- \_Layout.cshtml

- Error.cshtml

- Contiene un archivo favicon con el cual se presentará la aplicación.

- Contiene archivo viewstart.cshtml el cual llama por defecto a \_Layout.cshtml

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

- Diseño por defecto de \_Layout.cshtml

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - Mi aplicación ASP.NET</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Nombre de la aplicación", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Inicio", "Index", "Home")</li>

<li>@Html.ActionLink("Acerca de", "About", "Home")</li>

<li>@Html.ActionLink("Contacto", "Contact", "Home")</li>

</ul>

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - Mi aplicación ASP.NET</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

- Global.asax.cs

- Archivo de incialización de la aplicación.

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFiltersFilters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

- package.config

- Archivo de configuración para saber los paquetes de dependencias de los archivos.

Para crear un nuevo modelo se agrega a la carpeta modelos una nueva clase.

public class Movie

{

public int Id { get; set; }

public string Name { get; set; }

}

Controladores

Para crear controladores se pueden usar plantillas predefinidas para agilizar el desarrollo, sin embargo, también se pueden crear controladores vacíos.

public class MovieController : Controller

{

// Get Movies/Random

public ActionResult Random()

{

var movie = new Movie() { Name = "Shrek!"}; // Instancia el modelo.

return View(); // Llama a la vista de nombre Random.

}

}

Vistas

Para crear la vista se debe verificar la carpeta en la que se creará (con respecto al controlador), si se elige vista parcial la vista actuará como un componente que se pueda llamar a través de varias vistas.

Además, se debe configurar el Layout que la vista usará como base al momento que sea llamada, esto para no repetir código.

- Código de Vista:

@{

ViewBag.Title = "Random";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Random</h2>

Con @Model podemos leer desde la vista el modelo pasado por el controlador, el cual es dinámico por defecto a menos que se especifique su tipo.

- Vista con modelo

@model Vidly.Models.Movie

@{

ViewBag.Title = "Random";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Random</h2>

<h2>@Model.Name</h2>

Cambio de tema

Para cambiar el tema por default de la página se puede navegar hacia bootswatch.com y sustituir el tema de bootstrap por defecto que tiene la plantilla.

En la página se busca el tema deseado y ahí se descarga el archivo css, posteriormente se guarda en la carpeta content en este caso con el nombre bootstrap-lumen.css .

- Se abre en App\_Start el archivo BundleConfig para cambiar la referencia al archivo css en StyleBundle.

public class BundleConfig

{

// Para obtener más información sobre las uniones, visite https://go.microsoft.com/fwlink/?LinkId=301862

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

// Utilice la versión de desarrollo de Modernizr para desarrollar y obtener información. De este modo, estará

// para la producción, use la herramienta de compilación disponible en https://modernizr.com para seleccionar solo las pruebas que necesite.

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap-lumen.css",

"~/Content/site.css")); // Se cambia el archivo de referencia a css.

}

}

- En bundle config se puede empaquetar varios archivos de js que se manejen juntos, esto para solo hacer una petición http.

Action Result

- ActionResult es la clase base que se regresa en todas las peticiones que se le hacen al controlador.

- Dependiendo de lo que realice el controlador puede devolver distintos tipos de respuestas.

- El método View(movie) es el método base heredado de la clase Controller, el cual nos permite regresar una vista por defecto.

// Get Movies/Random

public ActionResult Random()

{

var movie = new Movie() { Name = "Shrek!"}; // Instancia el modelo.

return View(movie); // Llama a la vista de nombre Random.

// return new ViewResult(); // Llama a la vista por medio de la creación de una instancia.

}

- Cuando se llama al método View(), en realidad se está regresando un ViewResult, por lo cual en el método de regreso también sería válido colocar:

public ViewResult Random(){return View();}

- Colocar ViewResult es útil en los test para evitar una conversión de dato Cast.

- Existen varios tipos de respuestas que se pueden regresar para los cuales también existe un método derivado de controller que nos crea el objeto y devuelve el tipo de respuesta que deseamos.

- Tipo: ViewResult, Método: View()

- Tipo: PartialViewResult, Método: PartialView()

- Tipo: ContentViewResult, Método: ContentView()

- Tipo: RedirectViewResult, Método: RedirectView()

- Tipo: RedirectToRouteResult, Método: RedirectToAction()

- Tipo: JsonResult, Método: Json()

- Tipo: FileResult, Método: File()

- Tipo: HttpNotFoundResult, Método: HttpNotFound()

- Tipo: EmptyResult

public ActionResult Random()

{

var movie = new Movie() { Name = "Shrek!"}; // Instancia el modelo.

//return View(movie); // Llama a la vista de nombre Random.

//return Content("Hello World"); // Respuesta con texto plano

//return HttpNotFound(); // Error 404.

// return new EmptyResult(); // Devuelve página en blanco.

return RedirectToAction("Index", "Home", new { page = 1, sortby="name"}); // Redirecciona al método indicado con los parámetros indicados.

}

Action Parameters

- Son las entradas que tienen los action methods.

- ASP.NET automáticamente convierte los parámetros de las peticiones http en parámetros que reciban los action methods.

- Los formatos en los que ASP.NET puede recibir los parámetros son:

- En la URL: movies/edit/1

- En una consulta tipo string: /movies/edit?id=1

- En tipo form data: id = 1

- Método que acepta parámetro distinto a id.

public ActionResult Edit(int movieId)

{

return Content("id=" + movieId);

}

- Para declarar parámetros opcionales y evitar que la página se caiga se usan tipo nullable y una validación:

// Declaración de parámetros tipo nullable para poder ser opcionales.

public ActionResult Index(int? pageIndex, string sortBy)

{

if (!pageIndex.HasValue)

{

pageIndex = 1;

}

if (String.IsNullOrWhiteSpace(sortBy))

{

sortBy = "Name";

}

return Content($"pageIndex={pageIndex} & sortBy{sortBy}");

}

Rutas Personalizadas

Las rutas personalizadas se añaden en:

App\_Start/RouteConfig.cs

- Añadir una ruta personalizada:

routes.MapRoute( // Creación de ruta personalizada.

name: "MoviesByReleaseDate",

url: "movies/released/{year}/{month}",

defaults: new { controller = "Movies", action = "ByReleaseDate"},

constraints: new { year = @"\d{4}", month = @"\d{2}" } // Expresión regular a cumplir 4 dígitos para el año y 2 dígitos para el mes.

);

- Acción para recibir los parámetros de la ruta personalizada.

public ActionResult ByReleaseDate(int year, int month)

{

return Content($"year: {year}, month: {month}");

}

Attribute Routing

- Si se usa la configuración de routing con las "magic string", se tendrán problemas como:

- Tener muchas rutas declaradas convirtiéndolas en un laberinto.

- Estar cambiando entre archivos para agregar las rutas.

- Al realizar una refactorización automática sobre la acción, no se altera el magic string y muestra el error hasta que se trata de invocar la rutas.

- Attribute routing permite definir la ruta por encima del método haciéndolo más descriptivo y fácil de identificar.

- Para activar attribute routing se debe ejecutar en RegisterRoutes()

routes.MapMvcAttributeRoutes();

- Para asignar una ruta se usa:

[Route("movies/release/{year}/{month:regex(\\d{2}:range(1,12)}")]

public ActionResult ByReleaseYear(int year, int month)

{

return Content($"year: {year}, month: {month}");

}

- En attribute routing se define la ruta con [Route(url)].

- Para asignar restricciones a las variables se usa ":" y el tipo de restricción a usar.

- :regex(\\d{2}) : Función para concordar con la expresión regular.

- :range(1,12) : Función que acepta dentro de un rango de números.

- Algunas restricciones para poder aplicar son:

- alpha, {x:alpha} : Coincide si es una letra del alfabeto (a-z) (A-Z).

- bool, {x:bool} : Coincide si es un dato booleano.

- datetime, {x:datetime} : Coincide si es un valor Datetime.

- decimal, {x:decimal} : Coincide si es un valor decimal.

- double, {x:double} : Coincide si es un valor de punto flotante de 64 bits.

- float, {x:float} : Coincide si es un valor de punto flotante de 32 bits.

- guid, {x:guid} : Coincide si es un valor GUID.

- int, {x:int} : Coincide si es un entero de 32 bits.

- length, {x:length(6)} o {x:length(1,20)} : Coincide si una cadena tiene la longitud especificada.

- long, {x:long} : Coincide si es un entero de 64 bits.

- max, {x:max(10)} : Coincide si el entero está dentro del máximo indicado.

- maxlength, {x:maxlength(10)} : Coincide si la cadena está dentro del máximo indicado.

- min, {x:min(10)} : Coincide si el entero está dentro del mínimo indicado.

- minlength, {x:minlength(10)} : Coincide si la cadena está dentro del mínimo indicado.

- range, {x:range(10,50)} : Coincide si el entero está dentro del rango.

- regex, {x:regex(^\d{3}-\d{3}-\d{4}$)} : Coincide si la cadena cumple con la expresión regular.

- Documentación: https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2

Pasar información a la vista.

- Todos las vistas y controladores tienen un diccionario llamado ViewData que nos permite pasar valores a la vista, View Data es de tipo ViewDataDictionary, ViewData tiene el problema de magic string.

- Método del controlador.

public ActionResult Random()

{

var movie = new Movie() { Name = "Shrek!"}; // Instancia el modelo.

ViewData["Movie"] = movie; //Guarda en el diccionario la variable movie.

return View(); // Llama a la vista de nombre Random.

}

- Vista con ViewBag (se necesita importar el modelo con using y hacer un cast para hacer uso del diccionario).

@using Vidly.Models

@{

ViewBag.Title = "Random";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>@(((Movie)ViewData["Movie"]).Name)</h2>

- Las vistas también cuentan con ViewBag que permite agregar nuevas propiedades dinámicamente, por lo cual si hay un error se sabrá hasta la ejecución.

- Método del controlador.

public ActionResult Random()

{

var movie = new Movie() { Name = "Shrek!"}; // Instancia el modelo.

ViewBag.myMovie = movie;

return View(); // Llama a la vista de nombre Random.

}

- Vista con ViewBag.

@{

ViewBag.Title = "Random";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>@ViewBag.myMovie.Name</h2>

- Implementación de model en las vistas - controladores.

- Modelo en el controlador.

public ActionResult Random()

{

var movie = new Movie() { Name = "Shrek!"}; // Instancia el modelo.

ViewBag.myMovie = movie; // Guarda dinámicamente el objeto (no necesita casting).

var viewResult = new ViewResult();

viewResult.ViewData.Model = movie;

return viewResult; // Llama a la vista de nombre Random.

}

- Modelo en la vista.

@{

ViewBag.Title = "Random";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>@Model.Name</h2>

View Model

- Son modelos para un tipo de datos específico como el realizar un join entre movie y customer para saber que películas han comprado determinadas personas.

- Estos modelos se cargan en la definición de las listas de la misma manera, mediante el código razor se pueden ir realizando las operaciones necesarias para visualizar la información.

// Código controlador.

public ActionResult Random()

{

var movie = new Movie() { Name = "Shrek!"}; // Instancia la película.

var customers = new List<Customer>()

{

new Customer { Id = 1, Name = "Miranda" },

new Customer { Id = 2, Name = "Merritt" },

new Customer { Id = 3, Name = "Tad" },

new Customer { Id = 4, Name = "Galvin" },

new Customer { Id = 5, Name = "Allen" }

}; // Inicializa los compradores.

var viewModel = new RandomMovieViewModel

{

Movie = movie,

Customers = customers

}; // Inicializa view model.

return View(viewModel); // Llama a la vista de nombre Random con viewModel.

}

@model Vidly.ViewModels.RandomMovieViewModel

@{

ViewBag.Title = "Random";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>@Model.Movie.Name</h2>

@if (Model.Customers.Count == 0)

{

<p> No one hast rented this movie before.</p>

}

else

{

<ul>

@foreach (var customer in Model.Customers)

{

<li>@customer.Name</li>

}

</ul>

}

- Sintaxis condicional con razor (Dependiendo del modelo asigna la clase popular al header o null para que no aparezca nada).

@{

var className = Model.Customers.Count() > 5 ? "popular" : null;

}

<h2 class="@className">@Model.Movie.Name</h2>

Partials View

- Son las vistas que se colocan dentro de su vista padre.

- Se acostumbra a usar vistas parciales para procesar una pequeña parte de html y que así sea más legible el documento.

- Las vistas parciales se añaden en shared.

- Por convención las vistas parciales se nombran usando un guion bajo, \_NombreVista.

- Las vistas parciales no tienen Layout por defecto, pues no es la página completa a mostrar sino un segmento o componente que se agregará.

- Para crear una vista parcial se va a “agregar vistas” se selecciona “vista parcial” y se asigna un nombre y ubicación a la vista.

- Vista parcial del navbar:

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Nombre de la aplicación", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Inicio", "Index", "Home")</li>

<li>@Html.ActionLink("Acerca de", "About", "Home")</li>

<li>@Html.ActionLink("Contacto", "Contact", "Home")</li>

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

- Para renderizar la vista se llama ya sea en una vista, layout o u otra vista parcial con el nombre de la vista y con un modelo como parámetro opcional:

@Html.Partial("\_NombreVista")

@Html.Partial("\_NombreVista", modelo)

Sintaxis Razor con elementos html

<table class=“table table-striped table-hover” style="width: 100%">

<thead>

<tr>

<th scope="col">Customer</th>

</tr>

</thead>

<tbody class="table-striped">

@foreach (var customer in Model)

{

<tr class="table-light">

<td> <a href="~/Customers/Details/@customer.Id">@customer.Name</a></td>

</tr>

}

</tbody>

</table>

Creando links

- Un link puede ser declarado con la etiqueta a como comunmente se hace en html.

- Link con etiqueta a y ruta relativa a la página (.).

<a href="./Customers/Details/@customer.Id">@customer.Name</a>

- Link con etiqueta a y ruta desde la raiz (~).

<a href="~/Customers/Details/@customer.Id">@customer.Name</a>

- Además de usar link por medio de la etiqueta a se puede usar el Html helper proveido por el framework, el cual nos posibilita referenciar directamente al controlador y la acción al que se redirigirá.

- Con @Html.ActionLink se pasan los parámetros y clase(css) de la etiqueta por medio de un objeto anónimo.

- Prototipos de @Html.ActionLink

public static MvcHtmlString ActionLink(this HtmlHelper htmlHelper, string linkText, string actionName, string controllerName);

public static MvcHtmlString ActionLink(this HtmlHelper htmlHelper, string linkText, string actionName, string controllerName, object routeValues, object htmlAttributes);

public static MvcHtmlString ActionLink(this HtmlHelper htmlHelper, string linkText, string actionName, string controllerName, RouteValueDictionary routeValues, IDictionary<string, object> htmlAttributes);

- Link con @Html.ActionLink sin parámetro y sin clase css.

@Html.ActionLink("Vidly", "Index", "Home")

- Link con @Html.ActionLink sin parámetro y con clase css.

@Html.ActionLink("Movies", "Index", "Movies", new { area = "" }, new { @class = "nav-link" })

- Link con @Html.ActionLink sin parámetro y con clase css.

@Html.ActionLink("Movies", "Index", "Movies", new { area = "" }, new { @class = "nav-link" })

- Link con @Html.ActionLink pasando parámetro id y sin clase css.

@Html.ActionLink((string)movie.Name, "MovieDetails", "Movies", new { id = movie.Id }, null)

Entity framework

- Es una herramienta para poder acceder acceder a una base de datos, formalmente un ORM (Object Relational Mapper), el cual mapea los datos de sql en objetos de c#.

- Si no se usará un mapeador de datos se tendría que abrir manualmente la conexión, ejecutar la sentencia sql, leer la información, guardarla en un objeto y cerrar la conexión.

- Entity framework nos provee una clase de tipo DbContext que es el acceso a nuestra base de datos, de ahi se escojen los dbsets(tablas) a leer en la consulta.

- Para leer los DbSet se usa sintaxis LinQ el cual se encarga de general el SQL necesario para obtener los datos de la base de datos.

- Al leerse los datos se guardan en el DbSet, al hacer adiciones, modificaciones o borrados en el DbSet se guardan los cambios y cuando se indica modificar entonces ejecuta los cambios en la base de datos.

Flujos de trabajo en Entity Framework

- DbFirst: Se crea primero la base de datos y conforme a esta se generan los modelos para relacionar la base de datos con el orm.

- CodeFirst: Se crea primero el código de los modelos y conforme a estos se crea la base de datos, se puede ir actualizando si es necesario.

- Es mejor usar CodeFirst

- Incrmenta la productividad (no se tienen que crear tablas manualmente).

- Versionado total de la base de Datos.

- Es más fácil de ralizar pruebas de integración a la base de datos.

- Cosas que se no son verdaderas de Code-First:

- Solo se puede usar para proyectos nuevos o en blanco.

- No nos da el control total de la base de datos.

Pasos para integrar Entity Framework.

- Ir a la consola en NuGet Package Manager, NuGet Package Manager Console.

- En la consola ejecutar:

enable-migrations

- Con enable-migrations se configura poder usar migraciones para tener un historial del diseño de la base de datos.

- Se crea la primera migración con:

add-migration InitialModel

- Los conjuntos para crear el DbSet los toma de Models/IdentityModels, en la clase ApplicationDbContext la cual deriva de DbContext, por eso la toma para realizar las migraciones.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext()

: base("DefaultConnection", throwIfV1Schema: false)

{

}

public static ApplicationDbContext Create()

{

return new ApplicationDbContext();

}

}

- En esa misma clase se agregaran los modelos que serán considerados para ser tablas de la base de datos, en este caso la tabla Customers.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public DbSet<Customer> Customers { get; set; }

public ApplicationDbContext()

: base("DefaultConnection", throwIfV1Schema: false)

{

}

public static ApplicationDbContext Create()

{

return new ApplicationDbContext();

}

}

- Ya que no se han ejecutado cambios sobre la base de datos y no se incluyeron todas las tablas necesarias se puede rehacer el código de la migración con el parámetro -force:

add-migration InitialModel -force

- Crear la base de datos:

update-database

- En el explorador de soluciones dar clic en mostrar todos los archivos.

- En la carpeta App\_Data abrir el archivo .mdf generado para ver la base de datos.

Migraciones

Las migraciones deben ser tratadas como pequeñas modificaciones que se hacen y no una modificación masiva a la estructura de la base de datos.

- Se aconseja tener migraciones cortas y asociarles nombres referentes a lo modificación realizada.

Agregar nueva migración:

- add-migration AddIsSuscribedToCustomer

Reflejar cambios en la base de datos:

- update-database

Modelos

- Creacion de los modelos con llaves foráneas, por convención se escribe MembershipTypeId en donde debe terminar con Id.

public class Customer

{

public int Id { get; set; }

public string Name { get; set; }

public bool IsSubscribedToNewsletter { get; set; }

public MembershipType MembershipType { get; set; }

public int MembershipTypeId { get; set; }

}

public class MembershipType

{

public int Id { get; set; }

public float SignUpFee { get; set; }

public byte DurationInMonths { get; set; }

public double DiscountRate { get; set; }

}

Sembrando la base de datos.

Para realizar la siembra de los datos en la base de datos, entity framework nos permite agregar sentencias SQL en las migraciones con el Comando SQL:

public partial class PopulateMembershipTypes : DbMigration

{

public override void Up()

{

Sql("INSERT INTO MembershipTypes (Id, SignUpFee, DurationInMonths, DiscountRate) VALUES (1, 0, 0, 0)");

Sql("INSERT INTO MembershipTypes (Id, SignUpFee, DurationInMonths, DiscountRate) VALUES (2, 30, 1, 10)");

Sql("INSERT INTO MembershipTypes (Id, SignUpFee, DurationInMonths, DiscountRate) VALUES (3, 90, 3, 15)");

Sql("INSERT INTO MembershipTypes (Id, SignUpFee, DurationInMonths, DiscountRate) VALUES (4, 300, 12, 20)");

}

public override void Down()

{

}

}

- Guardar los cambios y ejecutar: update-database.

- Los datos que se siembren en las migraciones deben ser los pensados cuando se despliega la aplicación de manera inicial, no los datos de prueba para verificar su correcta operación.

Uso de DataAnotations

Para definir si un campo es requerido o no, o fijar distintas condiciones de la tabla se deben usar DataAnotations:

public class Customer

{

public int Id { get; set; }

[Required]

[StringLength(255)]

public string Name { get; set; }

public bool IsSubscribedToNewsletter { get; set; }

public MembershipType MembershipType { get; set; }

public int MembershipTypeId { get; set; }

}

https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/data-annotations

Consulta de Datos

Para consultar los datos se debe instanciar DbContext, este es un ejemplo sin inyección de dependencias en un controlador.

private ApplicationDbContext \_context;

public CustomersController()

{

\_context = new ApplicationDbContext();

}

protected override void Dispose(bool disposing)

{

\_context.Dispose(); // Limpia dbContext de la memoria.

}

public ActionResult Index()

{

var customers = \_context.Customers.ToList();

if (customers.Count == 0)

return View();

return View(customers);

}

Eager Loading

Para traer todas las propiedades del objeto consultado, como los propiedades que también son objetos se usa eager loading, trayendo mas campos en la consulta.

Uso de propiedad de otro objeto (join)

<td> @customer.MembershipType.DiscountRate</td>

Consulta con Eager Loading para traer todos los miembros indicados.

var customers = \_context.Customers.Include(c => c.MembershipType).ToList();

Espacio de nombre para incluir método de Eager Loading.

using System.Data.Entity;

Agregar nuevo campo con datos

- Si el tipo de dato no es nulo por defecto y se requiere que sea nulo debe ser declarado en la entidad como nullable.

public DateTime? Birthdate { get; set; }

public override void Up()

{

AddColumn("dbo.Customers", "Birthdate", c => c.DateTime());

}

public override void Up()

{

AddColumn("dbo.MembershipTypes", "Name", c => c.String());

Sql("UPDATE dbo.MembershipTypes SET Name = 'Pay as You Go' WHERE Id=1;");

Sql("UPDATE dbo.MembershipTypes SET Name = 'Monthly' WHERE Id=2;");

Sql("UPDATE dbo.MembershipTypes SET Name = 'Quarterly' WHERE Id=3;");

Sql("UPDATE dbo.MembershipTypes SET Name = 'Annual' WHERE Id=4;");

}

- Definición de Entidad con elementos foráneos.

public class Movie

{

public int Id { get; set; }

[Required]

[StringLength(255)]

public string Name { get; set; }

[Required]

public Genre Genre { get; set; }

public int GenreId { get; set; }

public DateTime ReleaseDate { get; set; }

public DateTime DateAdded { get; set; }

public int NumberInStock { get; set; }

}

- Declaración de Modelos tipo IEnumerable:

public ActionResult Index()

{

IEnumerable<Movie> movies = db.Movies.Include(m => m.Genre);

if (movies.Count() != 0)

return View(movies);

return View();

}

@model IEnumerable<Vidly.Models.Movie>

@foreach (var movie in Model)

{

<tr class="table-light">

<td> @movie.Genre.Name</td>

</tr>

}

Formularios

- Para hacer formularios ASP.NET tiene un Html Helper que se encarga de crear las etiquetas de formulario <form> </form>, limpiar el objeto de memoria (usando using) y asociarlo a una acción declarada en el controlador.

- Formulario que pasa la información por POST en la acción Create del controlador Customers.

@using (@Html.BeginForm("Create", "Customers"))

{

}

- Para darle estilo de bootstrap se usa dentro de using:

<div class="form-group"> </div>

- Para declarar la etiqueta del nombre de un campo se puede usar.

- Etiqueta por html por defecto.

<label for="Name"> </label>

- Html helper (se le asigna el nombre de la propiedad sobre la cual está asignada).

@Html.LabelFor(m => m.Name)

Resultado en html: <label for="Name">Name</label>

- Para cambiar el texto del label se puede definir en la entidad un Data DataAnotation para ese campo.

[Display(Name = "Date of Birth")]

public DateTime? Birthdate { get; set; }

- Para declarar el campo de datos (textbox) se usa un HtmlHelper el cual nos ayuda con las validaciones definidas en las DataAnnotations

@Html.TextBoxFor(m => m.Name)

Resultado en html: <input data-val="true" data-val-length="El campo Name debe ser una cadena con una longitud máxima de 255." data-val-length-max="255" data-val-required="El campo Name es obligatorio." id="Name" name="Name" type="text" value="">

- En el html TextBoxFor se puede definir la clase de la siguiente manera:

@Html.TextBoxFor(m => m.Name, new { @class = "form-control" } )

- Para trabajar con formato de tipo fecha se puede usar la expresión de formato {0} indica el valor del primer parámetro.

@Html.TextBoxFor(m => m.Customer.Birthdate, "{0:d MMM yyyy}", new { @class = "form-control" })

@Html.TextBoxFor(m => m.Customer.Birthdate, "{0:dddd, dd MMMM yyyy}", new { @class = "form-control" })

- Para verificar casillas booleanas se usa el HtmlHelper CheckBoxFor

@Html.CheckBoxFor(m => m.IsSubscribedToNewsletter)

Dropdown List

- Para crear un Dropdown List por lo general se requiere llamar las opciones de otra tabla de la base de datos por lo cual es recomenable crear su propio modelo en la carpeta con todas las ViewModels.

- La función Html.DropdownList contiene los siguientes elementos:

- public static MvcHtmlString DropDownListFor<TModel, TProperty>(this HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TProperty>> expression, IEnumerable<SelectListItem> selectList, string optionLabel, object htmlAttributes);

- expresion: Función lambda con la propiedad que se referencia.

- selectList: Objeto SelectListItem con las opciones a pasar.

- SelectListItem: new SelectList(Model.MembershipTypes, "Id", "Name"):

1 arg: DataItems (Valores de la lista)

2 arg: Campo del valor (por lo general Id).

3 arg: Campo del valor Visible (por lo general Name).

- optionLabel: Texto por defecto del DropdownList.

- htmlAttributes: Objeto anónimo con los atributos html (clase id).

- Ejemplo de DropdownList:

@Html.DropDownListFor(m => m.Customer.MembershipTypeId, new SelectList(Model.MembershipTypes, "Id", "Name"), "Select Membership Type", new { @class = "form-control" })

Data Binding

- Al enviar la información a través de un formulario se envían los campos definidos en el formulario a traves de form data, las llaves de estos son pasados a través de POST y cuando los recibe el controlador (POST) este infiere todos los campos posibles a asignar al ojeto y los guarda.

- Declaración de Método POST para procesar la solicitud del formulario de agregar un usuario.

[HttpPost]

public ActionResult Create(Customer customer)

{

db.Customers.Add(customer); // Agregar al contexto.

db.SaveChanges(); // Realizar los cambios en la base de datos.

return RedirectToAction("Index","Customers");

}

Add-Update

Cuando se usa una vista con un modelo definido para poder agergar o actualizar dependiendo de la acción indicada por la url, se puede agregar o editar eligiendo 2 caminos:

- Mostrar una vista general y dependiendo si el Id no es nulo (al usar data binding es nulo para add y con valor para edit), en el controlador decidir si se agrega o se actualiza.

- En la vista revisar si el Id es nulo y conforme al valor decidir si mostrar una vista de agregar con acción agregar al controlador o una vista editar con acción editar en el controlador.

- Usando la primera implementación el código queda así:

- En el controlador se lee el valor de Id.

[HttpPost]

public ActionResult SaveCustomer(Customer customer)

{

if (customer.Id <= 0) // Sin Id asignado agregar customer.

{

db.Customers.Add(customer);

}

else // Con Id asignado actualizar customer.

{

var updated\_customer = db.Customers.FirstOrDefault(c => c.Id.Equals(customer.Id));

updated\_customer.Name = customer.Name;

updated\_customer.MembershipTypeId = customer.MembershipTypeId;

updated\_customer.IsSubscribedToNewsletter = customer.IsSubscribedToNewsletter;

//TryUpdateModel(updated\_customer, "", new string[] { "Name", "MembershipTypeId", "IsSubscribedToNewsletter" });

}

db.SaveChanges();

return RedirectToAction("Index","Customers");

}

- En la vista se agrega el campo del Id escondido.

@Html.HiddenFor(m => m.Customer.Id)

@Html.HiddenFor(m => m.Movie.Id, new { Value = idValue })

- Para actualizar la información con métodos automáticos de transferencia de datos se puede usar

TryUpdateModel(updated\_customer, "", new string[] { "Name", "MembershipTypeId", "IsSubscribedToNewsletter" });

Mapper.Map(customer, updated\_customer)

- Consejos para actualizar.

Cuando se trabaja con varias secciones de código y varias acciones como agregar y actualizar conviene hacer todo en pasos pequeños e ir validando parte por parte aun sabiendo que el método o la vista no contempla aun todos los casos.

En particular para una vista que actualice y agrege, primero hay que configurar todo para que agregue y luego para que actualice.

Validaciones con Data Annotations

- Las Data Annotations no solo son usadas para asignar las restricciones y propiedades de la entidad (tabla) o para usarlas con Display y mostrar el label deseado, sino que también son usadas para validar el modelo y mostrar los mensajes en la vista.

- Para usar las validaciones del modelo ingresado por el parámetro se llama a la función, con esto sabra si el parámetro ingresado cumple con las Data Annotations que se declararon:

ModelState.IsValid

- Para reflejar en la vista se usa el html helper ValidationMessagesFor, el cual al verificar el modelo se actualiza:

- Html Helper:

@Html.ValidationMessageFor(m => m.Customer.Name)

- Etiqueta generada sin error y con error.

<span class="field-validation-valid" data-valmsg-for="Customer.Name" data-valmsg-replace="true"></span>

<span class="field-validation-error" data-valmsg-for="Customer.Name" data-valmsg-replace="true">El campo Name es obligatorio.</span>

- Para las validaciones se pueden seguir 3 pasos.

- Declarar las Data Annotations en las entidades.

- Usar ModelState.IsValid en el controlador POST para verificar que la entidad sea válida.

- Agregar validaciones a la vista del formulario.

Agregar estílos CSS para cada toda la página.

- Se accede a contents y a site.css se añaden los estilos que se quieran usar, al ser una página que siempre requiere una vista se configura en site.css que es el archivo del \_Layout por defecto.

- Una forma de resaltar los campos es usando las clases que agrega Model.IsValid a los elementos.

Resalta el texto en rojo del mensaje.

.field-validation-error {

color: red;

}

Coloca las cajas de texto o de selección en rojo.

.input-validation-error {

border: 4px solid red;

}

DataAnnotations

- Algunas Data Annotations posible son:

[Required]

[StringLength(255)]

[Range(1,10)]

[Compare("Other Property")]

[Phone]

[EmailAddress]

[Url]

[RegularExpresion""]

- Para sobrescribir el mensaje de error de las validaciones se usa la propiedad ErrorMessage de cada validación.

[Required(ErrorMessage = "Please enter customer's name.")]

[StringLength(255, ErrorMessage = "The maximun string length can be 255 characters.")]

Validaciones Personalizadas

Para realizar nuestras propias validaciones podemos crear nuestra propia Data Annotation que recibe el modelo a validar y conforme a eso lo válida.

- Para usar la validación se hereda de la clase:

ValidationAttribute

- Se debe sobrescribir el método IsValid:

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

- Si la validación indica que el modelo es válido se envia:

ValidationResult.Success

- Si la validación indica que el modelo no es válido se envia:

new ValidationResult(ErrorMessage = "Mensaje de error.")

- Creación de validación personalizada heredando de Validation Attribute.

public class Min18YearsIfAMember : ValidationAttribute

{

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

Customer customer = (Customer)validationContext.ObjectInstance;

if (customer.MembershipTypeId == 0 || customer.MembershipTypeId == 1)

return ValidationResult.Success;

if (customer.Birthdate == null)

return new ValidationResult(ErrorMessage = "Birthdate is required.");

int age = CalculateAge(customer.Birthdate.Value);

bool HasMoreThan18Years = (age >= 18);

return HasMoreThan18Years

? ValidationResult.Success

: new ValidationResult("Customer should be at least 18 years old to go on a membership.");

}

private int CalculateAge(DateTime birth)

{

DateTime today = DateTime.UtcNow.Date;

var age = today.Year - birth.Year;

// Si es mayor en mes o si es mayor en dias pero del mismo mes.

if (birth.Month > today.Month ||

(birth.Month == today.Month && birth.Day > today.Day))

{

age--;

}

return age;

}

}

Manteneabilidad de la lógica del negocio.

- Cuando se usa código que va referente a la lógica del negocio, es preferente usar variables que expliquen el código y no solo una referencia númerica como en el caso de la membresía.

if (customer.MembershipTypeId == 0 || customer.MembershipTypeId == 1)

return ValidationResult.Success;

- Cuando se usan variables hard-code estan pueden ser vulnerables y no mantenibles a lo largo del ciclo de vida del código por lo cual es recomendabel asignar un enum o una variable de solo lectura estática.

- Para declarar una variable de lectura estática se asigna como un atributo dentro del código de clase del modelo.

public class MembershipType

{

public int Id { get; set; }

public string Name { get; set; }

public float SignUpFee { get; set; }

public byte DurationInMonths { get; set; }

public double DiscountRate { get; set; }

public static readonly int Unknown = 0;

public static readonly int PayAsYouGo = 1;

}

- Actualizando el código queda asignado de la siguiente manera.

if (customer.MembershipTypeId == MembershipType.Unknown ||

customer.MembershipTypeId == MembershipType.PayAsYouGo)

return ValidationResult.Success;

Validation Sumary

- Con validation summary podemos mostrar todos los campos del modelo que no cumplen las condiciones en un solo bloque con la instrucción:

@Html.ValidationSummary()

- Cuando se usa ValidationSummary en la vista se debe considerar cada campo del formulario a evaluar en especial el Id del formulario cuando se va agregar un nuevo registro, ya que también incluye la validación para ese campo.

- Una manera de evitar el mensaje de error para la validación del Id es inicializando el valor del Id en 0 desde el controlador de la vista esto se puede hacer asignando el objeto pertinente al modelo de la vista como en el siguiente caso.

public ActionResult AddCustomer()

{

var membershipTypes = db.MembershipTypes.ToList();

var viewmodel = new CustomerFormViewModel

{

Customer = new Customer(), //Asignación de un nuevo objeto con Id=0 por defecto.

MembershipTypes = membershipTypes

};

return View("CustomerForm", viewmodel);

}

- Otra solución es asignar un valor al id ya sea para cuando es agregar o actualizar, usando viewBag o usando razor para verificar el modelo:

@{

bool isUpdating = !(Model.Movie == null || Model.Movie.Id == 0);

string title = isUpdating ? "Update Movie" : "Add New Movie";

int idValue = isUpdating ? Model.Movie.Id : 0;

}

@Html.HiddenFor(m => m.Movie.Id, new { Value = idValue })

- Adicionalmente se puede configurar si se desea mantener los detalles de los errores en el resumen y si se quiere un mensaje general que indique que hay errores, estos se pasan en el primer y segundo parámetro de la función respectivamente.

- Muestra mensaje general y mensaje para cada error.

- Html Helper:

@Html.ValidationSummary(false, "Please fix the following errors:")

- Etiquetas generadas:

<div class="validation-summary-errors" data-valmsg-summary="true">

<span>Please fix the following errors:</span>

<ul>

<li>Please enter customer's name.</li>

<li>Please select the Membership Type Id.</li>

</ul>

</div>

- Solo muestra mensaje general muestra mensaje general.

- Html Helper:

@Html.ValidationSummary(true, "Please fix the following errors:")

- Etiquetas generadas:

<div class="validation-summary-errors">

<span>Please fix the following errors:</span>

<ul>

<li style="display:none"></li>

</ul>

</div>

Validaciones del lado del cliente.

- El usuario tiene una respuesta casi inmediata.

- No se gastan recursos del lado del servidor.

- En el layout la función RenderSection nos permite colocar código de js en la vista.

@RenderSection("scripts", required: false)

- Para usar código js en la vista se debe usar:

@section scripts

{

@Scripts.Render("~/bundles/jqueryval")

}

- Las Data Annotations al momento de compilarse pueden usar las validaciones unobstructiva (que no afectan al servidor) si se llama el código js correspondiente en la vista:

@section scripts

{

@Scripts.Render("~/bundles/jqueryval")

}

- Los validadores de js funcionan leyendo el contenido de los tags y conforme al contenido puede verificar si la validación es correcta o no.

- Las validaciones disponibles del lado del cliente en Data Annotation son:

[Required]

[StringLength(255)]

[Range(1, 10)]

[Compare("OtherProperty")]

[Phone]

[EmailAddress]

[Url]

[RegularExpresion("...")]

Anti-Forgery-Token

Son tokens para comprobar la identidad del usuario y que solo admina request válidas que fueron generadas desde la página.

- Cross-site Request Forgery: Es un método de ataque al usuario donde desde un sitio malicioso se hace una petición POST para modificar o acceder a los datos del usuario.

- Con un HtmlHelper podemos defender a nuestra aplicación de ese tipo de ataques usando [ValidateAntiForgeryToken] en la acción del controlador y @Html.AntiForgeryToken() en la vista para por medio de una cookie con el token correcto poder validar que la petición vino del sitio permitido.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult SaveCustomer(Customer customer)

{

...

}

@Html.AntiForgeryToken()

Programación con API

- Cuando se usa una API solo se devuelve la información desde el servidor sin tener que devolver todo el html de lo cual se encarga el cliente, esto nos ayuda en varios aspectos como:

- Menos recursos usados por el servidor.

- Menos transferencia de datos.

- Conectividad con varios clientes como web, escritorio y app.

- Algunas API que son públicas proveen sus datos para complementar la experiencia de navegación con sus funcionalidades y no tener que repetir todo el trabajo.

- El framework ASP.NET Web API es distinto al de ASP.NET MVC pero sigue los mismos principios de arquitectura.

Covención Restful

Son API que usan los verbos http en conjunto con un Endpoint para realizar distintas acciones, generalmente referentes al CRUD.

- GET /api/customers

- GET /api/customers/1

- POST /api/customers

- PUT /api/customers/1

- DELETE /api/customers/1

REST significa Representational State Transfer

Configurar API

Para agregar una API se agrega Web API en Blanco desde las acciones y se configura en el Archivo Global.asx.cs

protected void Application\_Start()

{

GlobalConfiguration.Configure(WebApiConfig.Register); // Agregar esta línea de código al método.

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

Declaración de la API

GET

// Obtener todos los recursos.

// Verbo GET: /api/customers

public IEnumerable<Customer> GetCustomers()

{

return db.Customers.ToList();

}

// Obtener un recurso específico.

// Verbo GET: /api/customers/1

public Customer GetCustomer(int id)

{

var customer = db.Customers.FirstOrDefault(c => c.Id.Equals(id));

if (customer == null)

{

throw new HttpResponseException(HttpStatusCode.NotFound);

}

return customer;

}

POST

// Crear un nuevo recurso

// Verbo POST: /api/customers

[HttpPost]

public Customer CreateCustomer(Customer customer)

{

if (!ModelState.IsValid)

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

db.Customers.Add(customer);

db.SaveChanges();

return customer;

}

PUT

// Modificar un recurso existente.

// Verbo PUT: /api/customers/1

[HttpPut]

public void UpdateCustomer(int id, Customer customer)

{

if (!ModelState.IsValid || id != customer.Id)

throw new HttpResponseException(HttpStatusCode.BadRequest);

var customerInDb = db.Customers.FirstOrDefault(c => c.Id == id);

if (customerInDb == null)

throw new HttpResponseException(HttpStatusCode.NotFound);

customerInDb.Name = customer.Name;

customerInDb.Birthdate = customerInDb.Birthdate;

customerInDb.IsSubscribedToNewsletter = customerInDb.IsSubscribedToNewsletter;

customerInDb.MembershipTypeId = customerInDb.MembershipTypeId;

db.SaveChanges();

}

DELETE

// Verbo Delete: /api/customers/1

[HttpDelete]

public void DeleteCustomer(int id)

{

var customerInDb = db.Customers.FirstOrDefault(c => c.Id == id);

if (customerInDb == null)

throw new HttpResponseException(HttpStatusCode.NotFound);

db.Customers.Remove(customerInDb);

db.SaveChanges();

}

Probar la API

- Si no se específica el tipo de contenido en la API entonces por defecto ASP.NET la envía en formato XML.

- Para probar la API por completo se debe probar cada endpoint en POSTMAN viendo los distintos casos.

Data Transfer Objects

- Cuando se contruye una API se debe considerar que no puede cambiarse tan abruptamente pues otras aplicaciones dependen de este servicio, por lo tanto es necesario tener un objeto que no vaya a cambiar a pesar que la aplicación se actualice.

- Otro aspecto importante es la seguridad de los datos del modelo pues no debemos pasar todo el modelo a través de la API, sino solo que lo sea necesario, considerando así la realización de un nuevo modelo el cual sea pasado y enviado a través de la API.

- Los Data Transfer Objects son los modelos de los objetos que se van a pasar a través de la API ya sea para lectura o escritura.

- Con los DTO podemos especificar que información va a ser actualizada y cual no, librándonos asi de un ataque a información sensible.

- Un DTO el cual debe contener las mismas validaciones que el modelo que representa,

- La construcción de un DTO es de la siguiente forma.

public class CustomerDto

{

public int Id { get; set; }

[Required(ErrorMessage = "Please enter customer's name.")]

[StringLength(255, ErrorMessage = "The maximun string length is 255 characters.")]

public string Name { get; set; }

[Display(Name = "Date of Birth")]

[Min18YearsIfAMember]

public DateTime? Birthdate { get; set; }

public bool IsSubscribedToNewsletter { get; set; }

[Display(Name = "Membership Type")]

[Required(ErrorMessage = "Please select the Membership Type Id.")]

public int MembershipTypeId { get; set; }

}

- Cuando se usa un DTO se tienen que copiar los datos, esto se puede realizar manualmente asignando cada campo o usando automapper que con la referencia de los objetos hace el copiado automáticamente.

- Instalar automapper:

install-package automapper -version 4.1

- Crear un perfil de mapeo (en una clase que hereda de Profile) para saber como un objeto se va a copiar a otro objeto.

public class MappingProfile : Profile

{

public MappingProfile()

{

Mapper.CreateMap<Customer, CustomerDto>(); // Asignar valores de Customer a CustomerDto.

Mapper.CreateMap<CustomerDto, Customer>(); // Asignar valores de CustomerDto a Customer.

}

}

- Automapper usa reflexión para hacer el mapeo, lee los nombres de las propiedades y los que coincidan los asigna.

- Para inicializar automapper y que escanee los valores se llama en el en método de inicialización de Global.asax.cs .

protected void Application\_Start()

{

Mapper.Initialize(conf => conf.AddProfile<MappingProfile>()); // Inicializar Mapeo.

...

}

- Para convertir se usa el metodo Map de Mapper.

- public static TDestination Map<TSource, TDestination>(TSource source); // Método que convierte del tipo TSource al tipo TDestination, recibe como parámetro de entrada el objeto source de tipo Tsource y regresa como salida una instancia (objeto) de tipo TDestination.

- public static TDestination Map<TSource, TDestination>(TSource source, TDestination destination); // Método que convierte del tipo TSource al tipo TDestination, recibe como parámetro de entrada el objeto source de tipo Tsource y el objeto destination de tipo TDestination y regresa como salida una instancia (objeto) de tipo TDestination, guardando además en el objeto destination los datos.

- GetCustomers con Automapper:

public IEnumerable<CustomerDto> GetCustomers()

{

return db.Customers.ToList().Select(Mapper.Map<Customer, CustomerDto>); // Pasa como delegado el método Map que recibe un objeto de tipo Customer como parámetro.

}

- GetCustomer con Automapper:

public CustomerDto GetCustomer(int id)

{

var customer = db.Customers.FirstOrDefault(c => c.Id.Equals(id));

if (customer == null)

throw new HttpResponseException(HttpStatusCode.NotFound);

return Mapper.Map<Customer, CustomerDto>(customer);// Regresa un objeto CustomerDto con los valores coincidentes de customer.

}

- CreateCustomer con Automapper

[HttpPost]

public CustomerDto CreateCustomer(CustomerDto customerDto)

{

if (!ModelState.IsValid)

{

throw new HttpResponseException(HttpStatusCode.BadRequest);

}

var customer = db.Customers.Add(Mapper.Map<CustomerDto, Customer>(customerDto));

db.SaveChanges();

customerDto.Id = customer.Id; // Asignar el Id dado desde la db.

return customerDto;

}

- Update Customer con Automapper

[HttpPut]

public void UpdateCustomer(int id, CustomerDto customerDto)

{

if (!ModelState.IsValid)

throw new HttpResponseException(HttpStatusCode.BadRequest);

var customerInDb = db.Customers.FirstOrDefault(c => c.Id == id);

if (customerInDb == null)

throw new HttpResponseException(HttpStatusCode.NotFound);

customerDto.Id = customerInDb.Id; // Asignar id pasado por parámetro.

// Se anulan los tipos porque el compilador los puede inferir.

Mapper.Map(customerDto, customerInDb); // Copiar los datos de customerDto a customer.

db.SaveChanges();

}

Configuración de camel Notation:

- Para habilitar que el objeto se regrese respetando las notación CamelCase se puede configurar en WebApiConfig (equivalente a routeConfig de MVC), si se quiere usar camel Notation.

- Original con solo las rutas:

public static void Register(HttpConfiguration config)

{

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

- Añadiendo configuración CamelCase.

public static void Register(HttpConfiguration config)

{

var settings = config.Formatters.JsonFormatter.SerializerSettings; // Obtiene el objeto de ajustes del objeto de configuración http.

settings.ContractResolver = new CamelCasePropertyNamesContractResolver(); // Resuelve las peticiones JSON con CamelCase.

settings.Formatting = Formatting.Indented; // Devuelve el objeto indentado correctamente.

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

IActionResult:

- Para la peticiones que adopten Restful se deben devolver diferentes estados de código que describen si la petición fue exitosa, si hbo un error en el servidor, si el recurso no fue encontrado o algún otro estado que haya ocurrido.

- https://tutorialshelper.com/web-api-helper-class/.

- https://docs.microsoft.com/en-us/dotnet/api/system.web.http.apicontroller.badrequest?view=aspnetcore-2.2

- Para validar un modelo se puede usar Validate<TEntity> (TEntity entity).

- Para IActionResult result se tienen varios métodos y distintos tipos de respuesta como los que se tenían en IActionResult:

BadRequest() : Error 400 Bad Request.

BadRequest (string message): 400 Bad Request.

BadRequest (Microsoft.AspNetCore.Mvc.ModelBinding.ModelStateDictionary modelState): 400 Bad Request.

ConflictResult () : Error 409 Conflict.

Content((System.Net.HttpStatusCode statusCode, T value)) : Contenido con Resultado.

Created (string location, object content) : 201 Created.

Created(Uri uri, object content) : 201 Created.

CreatedAtRoute (string routeName, object routeValues, object content): 201 Created.

InternalServerError () : 500 Internal Server Error.

Json<T> (T content, Newtonsoft.Json.JsonSerializerSettings serializerSettings, System.Text.Encoding encoding) : 200 OK.

Json<T> (T content, Newtonsoft.Json.JsonSerializerSettings serializerSettings) : 200 OK.

Json<T> (T content) : 200 OK.

NotFound () : 404 Not Found.

Ok () : 200 OK.

Ok<T> (T content): 200 OK.

Redirect (Uri location): 302 Found.

Redirect (string location): 302 Found.

RedirectToRoute (string routeName, object routeValues): 302 Found.

ResponseMessage (System.Net.Http.HttpResponseMessage response): Sin estado.

StatusCode (System.Net.HttpStatusCode status): Regresa el estado.

API Con IActionResult

- Obtener todas las películas.

// Verbo GET: /api/movies

public IHttpActionResult GetMovies()

{

var moviesDto = db.Movies.ToList().Select(Mapper.Map<Movie, MovieDto>);

return Ok(moviesDto);

}

- Obtener una determinada película.

// Verbo GET: /api/movies/1

public IHttpActionResult GetMovie(int id)

{

var movie = db.Movies.FirstOrDefault(m => m.Id.Equals(id));

if (movie == null)

return NotFound();

return Ok(Mapper.Map<Movie, MovieDto>(movie)); // Respuesta Código 200 y objeto.

}

- Crear una nueva película.

// Verbo POST: /api/movies

[HttpPost]

public IHttpActionResult CreateMovie(MovieDto movieDto)

{

if (!ModelState.IsValid)

return BadRequest($"The arguments are invalid: {movieDto}");

if (!db.Genres.Any(g => g.Id == movieDto.GenreId))

return BadRequest($"The genre don't exist.");

var movie = db.Movies.Add(Mapper.Map<MovieDto, Movie>(movieDto));

movie.DateAdded = DateTime.UtcNow; // Registrar la fecha de añadido.

db.SaveChanges();

movieDto.Id = movie.Id; // Asignar el Id desde la db.

movieDto.DateAdded = movie.DateAdded;// Regresar la fecha de añadido.

return Created(new Uri($"{Request.RequestUri}/{movieDto.Id}"), movieDto); // Crear recurso con estado 201 y pasar un objeto como respuesta.

}

- Actualizar una película existente.

// Verbo PUT: /api/movies/1

[HttpPut]

public IHttpActionResult UpdateMovie(int id, MovieDto movieDto)

{

var movieInDb = db.Movies.FirstOrDefault(m => m.Id == id);

if (movieInDb == null)

return NotFound();

if (!ModelState.IsValid)

return BadRequest($"The arguments are invalid, needed: {movieDto}");

if (!db.Genres.Any(g => g.Id == movieDto.GenreId))

return BadRequest($"The genre don't exist.");

movieDto.Id = movieInDb.Id; // Asignar id pasado por parámetro.

movieDto.DateAdded = movieInDb.DateAdded;// Asignar la fecha de añadido.

// Se anulan los tipos porque el compilador los puede inferir.

Mapper.Map(movieDto, movieInDb); // Copiar los datos de movieDto a movie.

db.SaveChanges();

return Ok(movieDto);

}

- Borrar una película.

// Verbo Delete: /api/movies/1

[HttpDelete]

public IHttpActionResult DeleteMovie(int id)

{

var movieInDb = db.Movies.FirstOrDefault(m => m.Id == id);

if (movieInDb == null)

return NotFound();

db.Movies.Remove(movieInDb);

db.SaveChanges();

return Ok("resource deleted");

}

JQuery como librería del lado del cliente.

En el desarrollo web se puede usar JQuery como una opción para desarrollo del lado del cliente y asi administrar el DOM de una manera mas uniforme que solo usando Javascript puro. JQuery representa una solución para este problema pero se debe usar con cuidado para no generar código confuso, desorganizado y complejo. Opciones mas modernas para reemmplazar a JQuery son Angular y React que ofrecen una solución mas organizada para el manejo del DOM.

Tutorial básico de JQuery: https://www.impressivewebs.com/jquery-tutorial-for-beginners/

Incluir jquery como biblioteca:

<script type="text/javascript" src="https://www.impressivewebs.com/wp-includes/js/comment-reply.min.js?ver=5.3.4"></script>

Ejecutando código unobtrusivo cuando el DOM ya se cargó: Para ejecutar código que pueda actualizar el DOM, es conveniente que el DOM inicial ya se haya cargado para asi tener un mejor control de las modificaciones que se harán en la vista sin tener errores de llamdo y que la aplicación sera unobstrusiva, contenido (HTML) separado de la presentación (CSS).

$(document).ready(function() {

// all jQuery code goes here

});

Seleccionar elementos usando $(""), el cual es un contenedor tipo j Query.

$("div"); // Selecciona todos los elementos div del HTML.

$("#myElement"); // Selecciona un elemento con el Id "myElement".

$(".myClass"); // Selecciona todos los elementos con la clase "myclass"

$("p#myElement"); // Selecciona los elementos párrafo (p) con el ID: "myElement".

$("ul li a.navigation"); // Selecciona los elementos enlace (a) con clase "navigation" que esten contendidos dentro de li y ul.

Uso de JQuery para selectores CSS.

$("p > a"); // Selecciona las etiquetas (a) que son hijos de un párrafo (p).

$("input[type=text]"); // Selecciona los elementos input con el tipo "text".

$("a:first"); // Selecciona la primera etiqueta en la página.

$("p:odd"); // Selecciona todos los párrafos impares.

$("li:first-child"); // Selecciona cada elemento de la lista que es el primer hijo de una lista.

Selectores personalizados CSS.

$(":animated"); // Selecciona los elementos con animación.

$(":button"); // Selecciona cualquier elemento tipo boton (inputs o buttons).

$(":radio"); // Selecciona radio buttons.

$(":checkbox"); // Selecciona checkboxes.

$(":checked"); // Selecciona los checkboxes o radio buttons marcados(✅).

$(":header"); // Selecciona elementos headers (h1, h2, h3, etc.)

- Selectores en JQuery: https://api.jquery.com/category/selectors/

- Manipulando y accediendo a los nombres de clases CSS:

$("div").addClass("content"); // Añade la clase content a todos los elementos <div>.

$("div").removeClass("content"); // Añade la clase content de todos los elementos <div>.

$("div").toggleClass("content"); // Intercambia la clase en los elementos div usando la clase content, si la tiene la quita, si no la tiene la agrega.

- Verificar si un elemento tiene la clase:

if ($("#myElement").hasClass("content")) // Verifica si el elemento con ID myElement tiene la clase content.

- Manipulando estílos CSS con jQuery.

$("p").css("width", "400px"); // Agrega un width de 400px a todos los elementos p.

$("#myElement").css("color", "blue") // Hace que el color del texto sea azul en el elemento #myElement

$("ul").css("border", "solid 1px #ccc") // Agrega un borde a todos los elementos ul (Todas las listas).

Documentación estílos css: https://api.jquery.com/category/css/

- Agregando y eliminando elementos y contenido.

let myElementHTML = $("#myElement").html(); // Variable que contiene todo el HTML (incluyendo texto) dentro del elemento con id #myElement.

let myElementHTML = $("#myElement").text(); // Variable que contiene todo el HTML (excluyendo texto) dentro del elemento con id #myElement.

$("#myElement").html("<p>This is the new content.</p>"); // El contenido dentro del elemento con id myElement será reemplazado por el contido pasado dentro de la función.

$("#myElement").text("This is the new content."); // El contenido de texto será reemplazado por el ingresado en el parámetro.

$("#myElement").append("<p>This is the new content.</p>"); // Concatena el texto pasado al elemento.

("p").append("<p>This is the new content.</p>"); // Concatena el texto a todos los elementos p.

Otros métodos usados son: appendTo(), prepend(), prependTo(), before(), insertBefore(), after(), insertAfter().

Documentación de agregar elementos y contenido: https://api.jquery.com/category/manipulation/

Trabajando con eventos en jQuery:

$("a").click(function() {

// Ejecutar código cuando se haga un clic en a.

});

Algunos eventos ejecutados comunmente son: blur, focus, hover, keydown, load, mousemove, resize, scroll, submit, select.

Documentación elementos: https://api.jquery.com/category/events/.

Enseñando u ocultando elementos con jQuery.

// El primer parámetro nos dice la velocidad del efecto, la velocidad se puede dar en milisegundos.

$("#myElement").hide("slow", function() {

// Código a ejecutar una vez que el elemento se ocultó.

} // Oculta el elemento.

$("#myElement").show("fast", function() {

// Código a ejecutar una vez que el elemento se mostró.

} // Muestra el elemento.

$("#myElement").toggle(1000, function() {

// Código a ejecutar una vez que el elemento cambió su estado.

} // Muestra el elemento si estaba oculto o lo oculta si se mostraba.

- Desvaneciendo elementos.

$("#myElement").fadeOut("slow", function() {

// Código a ejecutar una vez que el elemento se ocultó.

} // Reaparece el elemento.

$("#myElement").fadeIn("fast", function() {

// Código a ejecutar una vez que el elemento se mostró.

} // Desvanece al elemento.

$("#myElement").fadeTo(2000, 0.4, function() {

// Código a ejecutar.

} // Reaparece el código parcialmente.

- Efectos y animación de secuencias:

$("#myElement").slideDown("fast", function() {

// Código a ejecutar cuando el efecto terminó.

} // Efecto hacía abajo.

$("#myElement").slideUp("slow", function() {

// Código a ejecutar cuando el efecto terminó.

} // Efecto hacía arriba.

$("#myElement").slideToggle(1000, function() {

// Código a ejecutar cuando el efecto terminó.

} // Efecto intercambiable.

Documentación de efectos: https://api.jquery.com/category/effects/

Documentación de animaciones: https://api.jquery.com/animate/https://api.jquery.com/animate/

JQuery en la vista

- Para usar instrucciones de jQuery en la vista se debe usar la instrucción:

@section scripts

{

// Código Javascript.

}

- En la parte de html se configuran las propiedades identificadores y clases que trabajaran con jQuery.

Selector para la clase customers.

<table id="customers" class=“table table-striped table-hover” style="width: 100%">

Agrega el atributo data-customer-id con el valor del Id para leerlo en JS con jQuery.

Agrega la clase js-delete indicando que solo se usará para llamarla en javascript con jQuery.

<button data-customer-id="@customer.Id" class="btn-link js-delete">

Delete @customer.Name

</button>

Para dar comportamiento a las clases atributos e identificadores se usa jQuery.

@section scripts

{

<script>

$(document).ready(function () {

$("#customers .js-delete").on("click", function () {

var button = $(this); // Obtiene la referencia del boton.

let result = confirm("Are you sure you want to delete this customer?");

if (result) { // Si se eligió eliminar el recurso.

$.ajax({ // Llamado de ajax para obtener el recurso.

url: "/api/customers/" + button.attr("data-customer-id"), // url a consultar.

method: "DELETE", // Verbo http.

success: function () { // Función a ejecutar cuando se realiza la petición.

button.parents("tr").remove(); // Selecciona el elemento padre del boton y lo elimina.

}

});

};

});

});

</script>

}

Usar Bootbox.js como elemento para bootstrap.

Para poder integrar modales de una forma fácil se puede usar bootbox para js.

En la consola nuget se ingresa: install-package bootbox -version:4.3.0 .

Al instalarlo agrega bootbox a la carpeta Scripts, por lo cual se puede agregar el archivo js a un bundle para que se llame y lo minifique a la vez.

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js",

"~/Scripts/bootbox.js"));

Llamar al modal de bootbox.

$("#customers .js-delete").on("click", function () {

var button = $(this); // Obtiene la referencia del boton.

let delete\_resource = function (result) {

if (result) { // Si se eligió eliminar el recurso.

$.ajax({ // Llamado de ajax para obtener el recurso.

url: "/api/customers/" + button.attr("data-customer-id"), // url a consultar.

method: "DELETE", // Verbo http.

success: function () { // Función a ejecutar cuando se realiza la petición.

button.parents("tr").remove(); // Selecciona el elemento padre del boton y lo elimina.

}

});

};

}

bootbox.confirm("Are you sure you want to delete this customer?", delete\_resource); // Pasa como parámetro la función a ejecutar cuando se seleccione la opción, el parámetro de la función es boolean.

});

- Optimizar código usando un filtro de elementos.

$("#customers").on("click",".js-delete", function () {}

jQuery DataTable

- Nos ayuda a realizar la consulta desde código Javascript hacia una API del servidor, obteniendo asi todos los datos en memoria del cliente y ahí poder realizar el filtrado y orden de una manera mas rápida en pantalla.

- Instalar datatables (tablas con filtrado, orden y diseño):

install-package jquery.datatables -version:1.10.11

- Crear bundle con datatables de jquery.

bundles.Add(new ScriptBundle("~/bundles/lib").Include(

"~/Scripts/jquery-{version}.js",

"~/Scripts/bootstrap.js",

"~/Scripts/bootbox.js",

"~/Scripts/dataTables/jquery.dataTables.js",

"~/Scripts/dataTables/dataTables.bootstrap.js"

));

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap-lumen.css",

"~/Content/bootstrap.css",

"~/Content/datatables/css/dataTables.bootstrap.css",

"~/Content/site.css"));

}

- Llamar al bundle en el \_Layout.cshtml:

@Scripts.Render("~/bundles/lib")

- Usar datatable en la vista:

$("#customers").DataTable(); // Usar dataTable leyendo la tabla customers html.

- Usar dataTable con una fuente de Ajax:

$("#customers").DataTable({

ajax: {

url: "/api/customers", // URL del recurso a consultar.

dataSrc: "" // Del objeto json consultado elegir el objeto que se requiere.

},

columns: [

{

data: "name",

render: function (data, type, customer) {

return `<a class="btn btn-primary" href="/customers/edit/${customer.id}"> ${customer.name} </a>`;

}

},

{

data: "name"

},

{

data: "id",

render: function (data) {

return `<button data-customer-id="${data}" class="btn btn-secondary js-delete"> Delete </button>`;

}

}

]

});

- Definición de tabla sin usar jQuery dataTables:

<table id="customersA" class="table table-striped table-hover" style="width: 100%">

<thead class="thead-dark">

<tr>

<th scope="col">Customer</th>

<th scope="col">Membership Type</th>

<th scope="col">Delete</th>

</tr>

</thead>

<tbody>

@foreach (var customer in Model)

{

<tr class="table-light">

<td> @Html.ActionLink(customer.Name, "Edit", "Customers", new { id = customer.Id }, null)</td>

<td> @customer.MembershipType.Name</td>

<td>

<button data-customer-id="@customer.Id" class="btn btn-secondary js-delete">

Delete @customer.Name

</button>

</td>

</tr>

}

</tbody>

</table>

- Definicion de tabla usando jQuery dataTables:

<table id="customers" class="table table-striped table-hover" style="width: 100%">

<thead class="thead-dark">

<tr>

<th scope="col">Customer</th>

<th scope="col">Membership Type</th>

<th scope="col">Delete</th>

</tr>

</thead>

<tbody>

</tbody>

</table>

Agregar campo de otra tabla con javascript.

- En el campo de get Customers obtener MembershipType:

// Verbo GET: /api/customers

public IHttpActionResult GetCustomers()

{

var customersDto = db.Customers

.Include(c => c.MembershipType).ToList()

.Select(Mapper.Map<Customer, CustomerDto>); // Pasa como delegado el método Map que recibe un objeto de tipo Customer como parámetro.

return Ok(customersDto);

}

- En la clase CustomersDto definir el parámetro MembershipType de tipo MembershipTypeDto.

public class CustomerDto

{

public int Id { get; set; }

public string Name { get; set; }

...

public MembershipTypeDto MembershipType { get; set; }

}

- Ya que se agrego un nuevo párametro con el nombre MembershipType, automapper intentará convertir del tipo MembershipType a MembershipTypeDto, por lo cual es necesario declarar la conversión en MappingProfile.

Mapper.CreateMap<MembershipType, MembershipTypeDto>(); // Asignar valores de MembershipType a MembershipTypeDto.

Mapper.CreateMap<MembershipTypeDto, MembershipType>(); // Asignar valores de MembershipTypeDto a MembershipType.

- En Javascript se debe agregar el atributo:

var table = $("#customers").DataTable({

ajax: {

url: "/api/customers", // URL del recurso a consultar.

dataSrc: "" // Del objeto json consultado elegir el objeto que se requiere.

},

columns: [

{

data: "name", // Nombre del atributo a consultar.

render: function (data, type, customer) { // Contenido a desplegar por columna.

return `<a class="btn btn-primary" href="/customers/edit/${customer.id}"> ${customer.name} </a>`;

}

},

{

data: "membershipType", // Nombre del atributo a consultar.

render: function (data) { // Contenido a desplegar por columna.

return `${data.name}`;

}

},

{

data: "id", // Nombre del atributo a consultar.

render: function (data) { // Contenido a desplegar por columna.

return `<button data-customer-id="${data}" class="btn btn-secondary js-delete"> Delete </button>`;

}

}

]});

- Eliminar los datos de memoria:

table.row(button.parents("tr")).remove().draw();

Single Page Aplication

- Se genera la vista en el cliente y se consulta la API en el servidor, haciendo una respuesta mas rapida en la página pues solo se traen los datos necesarios y todo lo demás se procesa en el cliente.

- Se eliminan las vistas razor pues ya no se envía toda la vista generada desde el servidor sino el código JS para procesar la vista.

- Los beneficios son interaciones mas rápidas y suaves, pues se procesa desde el cliente sin tener que enviar muchos datos desde el servidor.

- Algunos frameworks SPA son:

- Angular.

- Backbone.js.

- Ember.

Autorización y Autenticación.

- Se usa para restingir el acceso a ciertas vistas o partes del sitio qeb que solo puede acceder un usuario en específico.

- Tipos de Autorización en MVC:

- Sin autenticación:

- Individual User Accounts: Para proveer un formulario para que el usuario se pueda registrar.

- Organizational Accounts: Para activar un registro para aplicaciones internas o en la nube, usando el directorio activo.

- Windows Authentication: Para registro en aplicaciones de intranet.

- Para acceder al login se configura el proyecto como Individual User Accounts y se accede a traves de la vista parcial generada Login Partial:

@using Microsoft.AspNet.Identity

@if (Request.IsAuthenticated)

{

using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id = "logoutForm", @class = "navbar-right" }))

{

@Html.AntiForgeryToken()

<ul class="nav navbar-nav navbar-right">

<li>

@Html.ActionLink("Hola " + User.Identity.GetUserName() + "!", "Index", "Manage", routeValues: null, htmlAttributes: new { title = "Manage" })

</li>

<li><a href="javascript:document.getElementById('logoutForm').submit()">Cerrar sesión</a></li>

</ul>

}

}

else

{

<ul class="nav navbar-nav navbar-right">

<li>@Html.ActionLink("Register", "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink" })</li>

<li>@Html.ActionLink("Log in", "Login", "Account", routeValues: null, htmlAttributes: new { id = "loginLink" })</li>

</ul>

}

- ASP.NET Identity ha pasado por diferentes versiones o nombres:

ASP.NET Membership.

Simple Membership.

Universal Providers.

ASP.NET Identity.

- Arquitectura base de ASP.NET Identity

- Se compone de 2 clases de Dominio las cuales son: IdentityUser, Role.

- Tiene programada una API para trabajar con cada una de estas clases de dominio:

- User Manager.

- Role Manager.

- SignInManager.

- Cada una de las clases hacen referencia a la base de datos por clases UserStore y RoleStore.

- El modelo conforme se crea la base de datos está en IdentityModels, la cual tiene 2 clases:

ApplicationUser: Deriva de IdentityUser y define la contrucción de los modelos en el contexto.

ApplicationDbContext: Deriva de IdentityDbContext la cual es una clase genérica donde se le pasan por parámetro los modelos contenidos dentro de ApplicationUser.

- El controlador AccountController es el encargado de exponer distintas acciones como registro, entrar, cerrar sesión entre otras.

- En particular la acción Register para Post realiza el registro de un nuevo usuario asíncronamente.

Authorize

- Para impedir el acceso a un método o controlador si no se está registrado se usa [Authorize] con lo cual si no se está registrado se redirigue a la página de registro e iniciar sesión.

- Authorize en método:

[Authorize]

public ActionResult Index()

{

return View();

}

- Authorize en clase:

[Authorize]

public class CustomersController : Controller

{

}

- Authorize global para la aplicación (FilterConfig.cs).

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new HandleErrorAttribute());

filters.Add(new AuthorizeAttribute());

}

}

Sembrando Usuarios y Roles

Para añadir un rol a un usuario, primero se debe crear el rol o asignarlo en la base de datos, para eso se puede modificar el registro para asignar el rol a un nuevo usuario.

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

//Añadir rol con usuario.

var roleStore = new RoleStore<IdentityRole>(new ApplicationDbContext()); // Se dice el tipo de rol (IdentityRole).

var roleManager = new RoleManager<IdentityRole>(roleStore); // Crea contenedor de RoleStore para validar cada rol.

await roleManager.CreateAsync(new IdentityRole("CanManageMovies")); // Crea el rol CanManageMovies.

await UserManager.AddToRoleAsync(user.Id, "CanManageMovies"); // Asigna al usuario el Rol recíen creado.

await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

// Para obtener más información sobre cómo habilitar la confirmación de cuentas y el restablecimiento de contraseña, visite https://go.microsoft.com/fwlink/?LinkID=320771

// Enviar correo electrónico con este vínculo

// string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

// var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Confirmar cuenta", "Para confirmar la cuenta, haga clic <a href=\"" + callbackUrl + "\">aquí</a>");

return RedirectToAction("Index", "Home");

}

AddErrors(result);

}

// Si llegamos a este punto, es que se ha producido un error y volvemos a mostrar el formulario

return View(model);

}

- Los nombres que se le asignen a los roles deben describir sus funciones en el sistema como CanManageMovies, luego se les podra asignar una variable estática para distinguirlas conforme a la aplicación.

- Para sembrar datos se crea una nueva migración y con el comando Sql y el comando Insert se añaden los nuevos registros, para aplicar esta técnica se crean los scripts de llos datos originales , luego estos se borran de la base de datos y se añaden con la migracion.

public override void Up()

{

Sql(@"

INSERT INTO[dbo].[AspNetUsers]([Id], [Email], [EmailConfirmed], [PasswordHash], [SecurityStamp], [PhoneNumber], [PhoneNumberConfirmed], [TwoFactorEnabled], [LockoutEndDateUtc], [LockoutEnabled], [AccessFailedCount], [UserName]) VALUES(N'18ac6234-c2c8-47d3-b9a3-1c5978394efb', N'admin@user.com', 0, N'AJSOAFbS4QFrYcti2KUTtBdtQ6nKfiSLrmxfU64hgBVWx/DsKfx1i2T+E5Ic6fds+A==', N'871c3211-6acc-49f9-85c0-282f4b887a3a', NULL, 0, 0, NULL, 1, 0, N'admin@user.com');

INSERT INTO[dbo].[AspNetUsers] ([Id], [Email], [EmailConfirmed], [PasswordHash], [SecurityStamp], [PhoneNumber], [PhoneNumberConfirmed], [TwoFactorEnabled], [LockoutEndDateUtc], [LockoutEnabled], [AccessFailedCount], [UserName]) VALUES(N'42f48dd2-7f9d-473a-82e1-e43468a9c523', N'correo1@microsoft.com', 0, N'AHmmNecEgHCXRyDfTopwIaTlxaB6l3ELoKOK8HWC3k+74ZqmHAzpfBPYSMcjoB0OYQ==', N'1be26e67-bc3f-49f6-aae8-78d708166937', NULL, 0, 0, NULL, 1, 0, N'correo1@microsoft.com');

INSERT INTO[dbo].[AspNetUsers] ([Id], [Email], [EmailConfirmed], [PasswordHash], [SecurityStamp], [PhoneNumber], [PhoneNumberConfirmed], [TwoFactorEnabled], [LockoutEndDateUtc], [LockoutEnabled], [AccessFailedCount], [UserName]) VALUES(N'f0749fdb-a9c3-4825-99bd-400a6756c962', N'guest@user.com', 0, N'AMNCmOCHppAPKfjj+RZQ8NHzUPhwu8Yjj8JQfA2HDbiJtfYocFCwt3aF5BEjJ0zlig==', N'ade614d9-8f92-4fc0-b85f-8df4f0fcd45c', NULL, 0, 0, NULL, 1, 0, N'guest@user.com');

INSERT INTO [dbo].[AspNetRoles] ([Id], [Name]) VALUES (N'68604c2f-5da1-4eb3-8cda-6e14bfe10f30', N'CanManageMovies');

INSERT INTO [dbo].[AspNetUserRoles] ([UserId], [RoleId]) VALUES (N'18ac6234-c2c8-47d3-b9a3-1c5978394efb', N'68604c2f-5da1-4eb3-8cda-6e14bfe10f30');

");

}

Restringir acceso por roles.

- Se pueden tomar 2 caminos para restringir el acceso por roles:

- Tener una vista que se ajuste según el rol asignado.

- Crear una vista para cada tipo de rol.

- Elegir la vista usando el Rol actual (User.IsInRole(RoleName.CanManageMovies).

public ActionResult Index()

{

IEnumerable<Movie> movies = db.Movies.Include(m => m.Genre);

string viewName = "ReadOnlyList";

if (User.IsInRole(RoleName.CanManageMovies))

viewName = "List";

if (movies.Count() != 0)

return View(viewName, movies);

return View(viewName);

}

- Restringir el acceso por tipo de Rol.

[Authorize(Roles = RoleName.CanManageMovies)]

[Route("Movies/Add")]

public ActionResult AddMovie()

{

var viewModel = new MovieFormViewModel

{

Genres = db.Genres.ToList()

};

return View("MovieForm", viewModel);

}

- Declarar una clase estática con una constante para evitar magic strings.

public static class RoleName

{

public const string CanManageMovies = "CanManageMovies";

}

Agregar nuevo campo al usuario

- En el archvio IdentityModels se modifica la clase ApplicationUser:

// Para agregar datos de perfil del usuario, agregue más propiedades a su clase ApplicationUser. Visite https://go.microsoft.com/fwlink/?LinkID=317594 para obtener más información.

public class ApplicationUser : IdentityUser

{

[Required]

[StringLength(255)]

public string DrivingLicense { get; set; }

public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)

{

// Tenga en cuenta que el valor de authenticationType debe coincidir con el definido en CookieAuthenticationOptions.AuthenticationType

var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);

// Agregar aquí notificaciones personalizadas de usuario

return userIdentity;

}

}

- Cuando se agrega un campo como Required Entity Framework agrega el campo con string en blanco o el valor default del dato.

- Una vez agregado el modelo se modifica la vista (Register):

<div class="form-group">

@Html.LabelFor(m => m.DrivingLicense, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.TextBoxFor(m => m.DrivingLicense, new { @class = "form-control" })

</div>

</div>

- Se modifica el modelo de la vista (Vidly.Models.RegisterViewModel):

public class RegisterViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "El número de caracteres de {0} debe ser al menos {2}.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Repeat Password")]

[Compare("Password", ErrorMessage = "La contraseña y la contraseña de confirmación no coinciden.")]

public string ConfirmPassword { get; set; }

[Required]

[Display(Name = "Driving License")]

public string DrivingLicense { get; set; }

}

- Se modifica el controlador:

var user = new ApplicationUser

{

UserName = model.Email,

Email = model.Email,

DrivingLicense = model.DrivingLicense

};

Oauth

Protocolo de autenticación usado por varias plataformas.

Significa Open Authorisation

Se crea una cuenta en la red social o proveedor a usar.

La API de la red social publica una llave secreta para ese usuario.

La aplicación genera una llave para autenticarse con la red social y junto con los datos del usuario se accede a la api, para registralo y pedirle cierta información del usuario.

El usuario concede los permisos en la red social.

La red social regresa un token de autorización.

La aplicación responde con un token a la red social para confirma que la red social en verdad mando la información.

La red social regresa un token de acceso con el cual se pueden consultar ciertas partes del perfil de la red social del usuario.

ASP.NET tiene funciones que se encargan de realizar todo el procedimiento de una manera mas fácil de desarrollar.

Usando redes sociales en ASP.NET

- Habilitar SSL(para comunicaciones seguras).

- Registrar la app en la red social.

- Colocar la url del sitio con ssl.

- En AppStart/Startupu.Auth habilitamos la sección de facebook con la parte que nos pide.

- En el formulario de registro Views/Account/ExternalLoginConfirmation añadimos los campos necesarios definidos en el modelo de usuario.

// Modelo

public class ExternalLoginConfirmationViewModel

{

[Required]

[Display(Name = "Correo electrónico")]

public string Email { get; set; }

[Required]

[StringLength(255)]

[Display(Name = "Driving License")]

public string DrivingLicense { get; set; }

}

// Vista

<div class="form-group">

@Html.LabelFor(m => m.DrivingLicense, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.TextBoxFor(m => m.DrivingLicense, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.DrivingLicense, "", new { @class = "text-danger" })

</div>

</div>

//Controlador

var user = new ApplicationUser

{

UserName = model.Email,

Email = model.Email,

DrivingLicense = model.DrivingLicense

};

- En la tabla AspNetUserLogins se guardan los registros de las cuentas registradas externas.

app.UseFacebookAuthentication(

appId: "374991787270999",

appSecret: "numerosecreto");

Niveles de Optimización

- Una consideración muy importante a recordar es: "La optimización prematura es la raíz de todos los males".

- Para hacer una optimización no se puede generalizar sino que se debe ver el problema en particular y optimizarlo segun sus necesidades.

- En las aplicaciones web se tienen 3 niveles de optimización:

- Cliente: Navegador Web.

- Aplicacción: Servidor de aplicación IIS.

- Datos: Base de datos usada.

- Las principales optimizaciones se hacen en la capa de Datos y por lo general las menos observables en la del cliente.

Optimización en la capa de datos:

- Problemas en la base de datos al no establecer:

- Llaves primarias.

- Relaciones.

- Indices.

- Al usar Entity Framework se evita ese problema pues se definen siempre las relaciones llaves primarias e índices.

- Cuando una consulta generada por EF es muy compleja es mejor crear un procedimiento almacenado y optimizar asi, aunque se salga del patrón.

- Optimizar consultas:

- Usar Execution Plan en SQL Server, que nos muestra como SQL ejecuta nuestra consulta, con esto se puede determinar que parte de la consulta es la mas tardada y si se puede quitar entonces se elimina de la consulta.

- Crear una base de datos de solo lectura para los registros (CQRS), como lecturas por dias o por mes de algún campo, se usan haciendo join de datos para no gastar tiempo haciendo el join despues.

- Usar la cache para guardar los registros en memoria.

- Glimpse para ver tiempos de la aplicación.

- Instalar:

install-package glimpse.ef6

install-package glimpse.mvc5

- Ejecutar:

https://localhost:44335/glimpse.axd

Activar glimpse

- Dentro de Glimpse hay 3 opciones diferentes estas opciones nos permiten inspeccionar distintos aspectos de la aplicación y como se ejecuta esta.

Optimización para la capa de la aplicación

- Una técnica para mejorar el rendimiento del lado del servidor es usar release builds.

- Produce ensamblados ligeramente mejores optimizados para la etapa de producción quitando herramientas de desarrollo para trazabilidad y debugging.

- Deshabilitar la sesión:

- Una sesión es una parte de memoria en el servidor para alojar información temporal de cada usuario.

- Si un sevidor tiene muchos usuarios entonces la memoria se saturará muy rapido.

- Actualmente se usan soluciones mas orientadas a la nube, haciendo una petición al servidor y siendo está respondida sin guardar estados temporales de cada usuario.

En Web.config se añade en dentro de la etiqueta <system.web>:

<sessionState mode="Off"></sessionState>

- Otra técnica para mejorar el rendimiento del lado del servidor es output caching.

- Output Caching consiste en que la aplicación consulta la base de datos una vez para obtener los datos, luego guarda esos datos (las vistas o controladores) en la memoria cache del servidor y empieza a consultar de esa fuente hasta que pase un determinado tiempo o evento.

- Se declara con [OutputCache] y al igual que [Authorize] es un action filter.

[OutputCache]

public ActionResult Index()

{

return View();

}

- OutputCache se puede también aplicar a todo un controlar, ya sea despues de sus ejecución a antes.

- Output cache cuenta con varios parámetros como:

Duration : Establece cada cuando se lee de nuevo desde la DB.

Location : Establece si se va a ejecutar desde el servidor o desde el cliente.

VaryByParam : Estable por cual párametro va actualizarse si cambia.

[OutputCache(Duration = 10, Location = OutputCacheLocation.Server, VaryByParam = "genre")]

public ActionResult Index()

{

return View();

}

- Se debe ser cuidadoso con esta implementación pues no mantiene actualizada la página y solo es recomenbable si la página esta siendo lento al cargar.

- No se debe agregar cuando se muestran cambios como al agregar o editar un registro de la DB.

- Para deshabilitar el uso de cache se debe usar:

[OutputCache(Duration = 0, VaryByParam = "\*", NoStore = true)]

public ActionResult Index()

{

return View();

}

- Data Caching Cache en en los datos consultados.

public ActionResult Index()

{

if(MemoryCache.Default["Genres"] == null)

{

MemoryCache.Default["Genres"] = db.Genres.ToList();

}

var genres = MemoryCache.Default["Genres"] as IEnumerable<Genre>;

return View();

}

Este método solo es recomendable despues de hacer pruebas de rendimiento y encontrar deficits, en otro caso no es conveniente.

Async - Await.

Una de las ideas que se tiene de async y await es que mejoran el rendimiento de la aplicación haciendo las consultas mas rapidas.

Cuando se hace una consulta normal se tiene que esperar un determinado tiempo para que la consulta se ejecute y de esta manera proceder hasta que se tenga la respuesta completa.

Con Async-Await se libera el flujo una vez que se hace la consulta, ya con la consulta terminada se realizan las operaciones que necesitaban de esa consulta, sin bloquear el programa.

Async Await nos permite tener una aplicación mas responsiva pero no reduce el tiempo por consulta.

Al usar async await tenemos una mejor escalabilidad pues podríamos hacer consultas en paralelo dependiendo de la capacidad de hardware disponible, pero no es un mejor rendimiento en las consultas y una vez que se usen todos los recursos ya no se podrá mejorar mas.

La escalabilidad puede convertirse en una leyenda pues si no se tiene el hardware disponible con muchas instancias en verdad solo se estará haciendo una consulta a la vez.

Una manera de tener verdadera escalabilidad es usar un cluster de SQL, MongoDb o SQL Azure.

Optimización para la capa deL cliente

Para optimizar del lado del cliente es recomendable mandar los archivos en su versión mas ligera posible:

Para los DTO se recomienda que solo contengan la información que necesitan y no mas siempre y cuando sea pensada como una API de uso interno.

En los archivos js se sugiere hacer conjuntos de archivos para que estos se envien en un solo paquete y en su versión comprimida de una sola línea.

Cuando se compila el código de c# se hace un proceso para interpretar el código y guardar las instrucciones en el servidor, sin embargo es distinto cuando se corre el proyecto el cual ejecuta lo compilado y luego interpeta el código de JS y CSS en bundles para ser posteriormente enviados.

Para usar los bundles en Web.config se coloca dentro de la etiqueta <system.web>:

<compilation debug="false" targetFramework="4.8" />

Para tener una mejor medida del tiempo que se tarda nuestra aplicación es recomendable quitar el cache que trae por defecto el navegador web, pues en ocasiones no nos refleja los cambios verdaderos o carga mas archivos de los necesarios.

Implementación de una nueva característica

- Para implementar una nueva caracterísitca se puede comenzar desde:

- Frontend

- Backend

- Una mejor solución es comenzar desde el backend teniendo en cuanta lo que se hará en el frontend para asi no complicar el desarrollo.

- Hay que identficar las entradas y salidas.

- Entradas: Clientes y Películas.

- Salidas:

- Caso de uso: New Rental.

- En el frontend necesitamos una acción del backend para llamar, esto se puede consultar desde:

- Controlador MVC (Regresa el código y html).

- Controlador de la API (Genera html conforme a la información).

- Siguiendo el paso de la creación de API es bueno comenzar por crear el método que almacenará en la db con lo que se espera de entradas.

- Creación de API.

public class NewRentalsController : ApiController

{

[HttpPost]

public IHttpActionResult CreateNewRentals(NewRentalDto newRental)

{

throw new NotImplementedException;

}

}

- Creación de DTO como parámetro de entrada de la API.

public class NewRentalDto

{

public int CustomerId { get; set; }

public List<int> MovieIds { get; set; }

}

- Para que la API guarde en la base de datos necesita una tabla que soporte la acción, en este caso un comprador y varias películas, y la tabla Rental es por la tanto es una tabla de asociación.

public class Rental

{

public int Id { get; set; }

[Required]

public int CustomerId { get; set; }

public Customer Customer { get; set; }

[Required]

public int MovieId { get; set; }

public Movie Movie { get; set; }

[Required]

public DateTime DateRented { get; set; }

public DateTime? DateReturned { get; set; }

}

- Añadir en datacontext y añadir la migración.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public DbSet<Customer> Customers { get; set; }

public DbSet<MembershipType> MembershipTypes { get; set; }

public DbSet<Genre> Genres { get; set; }

public DbSet<Movie> Movies { get; set; }

public DbSet<Rental> Rentals { get; set; }

}

add-migration AddRentalModel

update-database

- Un consejo para seguir avanzado en seguir el happy path donde se considera que todo dato ingresado sera valido y no habran problemas, posteriormente se dan las validaciones casos de frontera y demas lógica necesaria, esto para tener el soporte y que no sea todo un caos.

- Posteriormente se prosigue a asignarle la definición a la acción de nuevo sin pensar en casos de frontera o validaciones, lo mas simple posible que pueda agregar registros a la db.

[HttpPost]

public IHttpActionResult CreateNewRentals(NewRentalDto newRental)

{

var todayDate = DateTime.Now; // Obtiene la fecha actual.

var customer = db.Customers.Single( // Leer el comprador.

c => c.Id == newRental.CustomerId);

// Cargar varias películas con contains equivalente a IN(1,2,3)

var movies = db.Movies.Where(

m => newRental.MovieIds.Contains(m.Id));

// Crear todos los registros.

foreach (var movie in movies)

{

var rental = new Rental

{

Customer = customer,

Movie = movie,

DateRented = todayDate

};

// Guardar en memoria.

db.Rentals.Add(rental);

}

// Guardar en la base de datos.

db.SaveChanges();

return Created(new Uri($"{Request.RequestUri}/"), newRental); // Crear recurso con estado 201 y pasar un objeto como respuesta.

}

- Ahora falta agregar la disponiblidad de películas y aunque se puede obtener si las películas estan disponibles consultando el número de peículas en stock(movies) menos el número de películas que tiene los clientes(rentals), conforme la tabla rentals crezca será cada vez más lenta la consulta, por lo tanto es recomendable mejor reflejar los cambios en la misma tabla de películas como una propiedad mas.

- Cuando el código se vuelve mas simple de escribir y a la vez se ejecuta con menor costo computacional no es una optimización prematura sino que se vuelve un código mas mantenible y mejor optimizado.

public class Movie

{

public int Id { get; set; }

...

[Required(ErrorMessage = "The Number in Stock field is required.")]

[Range( 1 , 20, ErrorMessage = "The Number of movies must be between 1 and 20.")]

[Display(Name = "Number in Stock")]

public int NumberInStock { get; set; }

public int NumberAvailable { get; set; }

}

- Una vez que ya se realizo la implementación siguiendo el happy path, se prosigue a ver cuales son los casos de negocio y validaciones necesarias para que la aplicación sea congruente.

Casos de Frontrera.

- El Id del cliente es inválido.

- No hay MovieIds.

- Una o mas MovieIds son inválidas.

- Una o mas películas no estan disponible por su número.

Los casos de frontera se pueden tomar de dos maneras:

Defensiva: Hace todas las comprobaciones sin importar cuales son los casos posibles del frontend. Son importantes si se usa una API pública.

[HttpPost]

public IHttpActionResult CreateNewRentals(NewRentalDto newRental)

{

if (newRental.MovieIds.Count == 0)

return BadRequest("No MovieIds have been given.");

var customer = db.Customers.SingleOrDefault( // Leer el cliente.

c => c.Id == newRental.CustomerId);

if (customer == null)

return BadRequest("CustomerId is not valid.");

// Cargar varias películas con contains equivalente a IN(1,2,3)

var movies = db.Movies.Where(

m => newRental.MovieIds.Contains(m.Id)).ToList();

if (movies.Count != newRental.MovieIds.Count)

return BadRequest("One or more MoviesIds are invalid.");

if (movies.Any(m => m.NumberAvailable <= 0))

return BadRequest("One or more movies are not available.");

var todayDate = DateTime.Now; // Obtiene la fecha actual.

// Crear todos los registros.

foreach (var movie in movies)

{

var rental = new Rental

{

Customer = customer,

Movie = movie,

DateRented = todayDate

};

// Guardar en memoria.

db.Rentals.Add(rental);

movie.NumberAvailable = movie.NumberAvailable - 1;//Decrementar en uno el número de peliculas.

}

// Guardar en la base de datos.

db.SaveChanges();

return Ok(newRental); // Crear recurso con estado 201.

}

Optimista: Se crea cuando solo la API va ser consumida por una aplicación y no son necesarias mas que validaciones muy específicas.

[HttpPost]

public IHttpActionResult CreateNewRentals(NewRentalDto newRental)

{

var customer = db.Customers.SingleOrDefault( // Leer el cliente.

c => c.Id == newRental.CustomerId);

// Cargar varias películas con contains equivalente a IN(1,2,3)

var movies = db.Movies.Where(

m => newRental.MovieIds.Contains(m.Id)).ToList();

if (movies.Any(m => m.NumberAvailable <= 0))

return BadRequest("One or more movies are not available.");

var todayDate = DateTime.Now; // Obtiene la fecha actual.

// Crear todos los registros.

foreach (var movie in movies)

{

var isvalid = ModelState.IsValid;

var rental = new Rental

{

Customer = customer,

Movie = movie,

DateRented = todayDate

};

// Guardar en memoria.

db.Rentals.Add(rental);

movie.NumberAvailable = movie.NumberAvailable -1;//Decrementar en uno el número de peliculas.

}

// Guardar en la base de datos.

db.SaveChanges();

return Ok(newRental); // Crear recurso con estado 201.

}

Implementar frontend

- Para tener una respuesta mas rapida y que la página no se quede congelada se usa AJAX para obtener los datos del servidor y renderizar el html desde el cliente.

- Un elemento usado para obtener los datos asíncronamente es ajax, ya sea nativo o como biblioteca de jQuery que hace su despliegue mas fácil:

$.ajax(url)

.done(function(result){

// Do something with the result

});

- El resultado de ajax puede ser de cualquier tipo, xlml, html o json.

- Para crear la página es prefente usar ajax y por lo tanto desde el controlador no llamar al modelo sino que toda la información se obtendrá por ajax y el formulario será enviado asincronamente por jquery.

- Comandos emmet usados:

div.form-group>label+input[type='text'].form-control

<div class="form-group">

<label></label>

<input type="text" value="" class="form-control" />

</div>

div.form-group>label+input[type='text']

<div class="form-group">

<label></label>

<input type="text" value="" />

</div>

button.btn.btn-primary

<button class="btn btn-primary">Submit</button>

- Para comenzar se debe crear el formulario básico (con su controlador MVC) de como quedará la aplicación y agregar su link correspondiente para acceder a él en este caso la barra de navegación.

// Controlador

public class RentalsController : Controller

{

// Return the form to the client.

public ActionResult New()

{

return View();

}

}

// Vista

@{

ViewBag.Title = "New Rental Form";

}

<h2>@ViewBag.Title</h2>

<form>

<div class="form-group">

<label>Customer</label>

<input type="text" value="" class="form-control" />

</div>

<div class="form-group">

<label>Movie</label>

<input type="text" value="" class="form-control"/>

</div>

<button class="btn btn-primary"></button>

</form>

// Link de navegación en \_NavBar.cshtml

<div class="collapse navbar-collapse" id="navbarColor03">

<ul class="navbar-nav mr-auto">

<li class="nav-item ">

@Html.ActionLink("New Rental", "New", "Rentals", new { area = "" }, new { @class = "nav-link" })

</li>

<li class="nav-item ">

@Html.ActionLink("Customers", "Index", "Customers", new { area = "" }, new { @class = "nav-link" })

</li>

<li class="nav-item">

@Html.ActionLink("Movies", "Index", "Movies", new { area = "" }, new { @class = "nav-link" })

</li>

</ul>

</div>

- Para agregar la característica de autocompletado se puede usar la líberia de tipeahead en jquery desarrollada por twitter.

- Instalar.

install-package Twitter.Typeahead

- Agregar al bundle.

bundles.Add(new ScriptBundle("~/bundles/lib").Include(

"~/Scripts/jquery-{version}.js",

"~/Scripts/bootstrap.js",

"~/Scripts/bootbox.js",

"~/Scripts/dataTables/jquery.dataTables.js",

"~/Scripts/dataTables/dataTables.bootstrap.js",

"~/Scripts/typeahead.bundle.js"

));

- Agregar estilos para Typeahead:

https://twitter.github.io/typeahead.js/examples/

https://twitter.github.io/typeahead.js/css/examples.css

- De la página de estílos se copian los de typeagead y se ñaden en un nuevo archivo de estilos que se añade a content css.

.typeahead {

background-color: #fff;

}

.typeahead:focus {

border: 2px solid #0097cf;

}

.tt-query {

-webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075);

-moz-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075);

box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075);

}

.tt-hint {

color: #999

}

.tt-menu {

width: 422px;

margin: 12px 0;

padding: 8px 0;

background-color: #fff;

border: 1px solid #ccc;

border: 1px solid rgba(0, 0, 0, 0.2);

-webkit-border-radius: 8px;

-moz-border-radius: 8px;

border-radius: 8px;

-webkit-box-shadow: 0 5px 10px rgba(0,0,0,.2);

-moz-box-shadow: 0 5px 10px rgba(0,0,0,.2);

box-shadow: 0 5px 10px rgba(0,0,0,.2);

}

.tt-suggestion {

padding: 3px 20px;

font-size: 18px;

line-height: 24px;

}

.tt-suggestion:hover {

cursor: pointer;

color: #fff;

background-color: #0097cf;

}

.tt-suggestion.tt-cursor {

color: #fff;

background-color: #0097cf;

}

.tt-suggestion p {

margin: 0;

}

- Se agrega el archivo al bundle de css en BundleConfig.cs.

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap-lumen.css",

"~/Content/bootstrap.css",

"~/Content/typeahead.css",

"~/Content/datatables/css/dataTables.bootstrap.css",

"~/Content/site.css"));

- Para consultar el funcionamiento de la libreria tipeahead javascript se debe ver su implementación remota.

https://twitter.github.io/typeahead.js/examples/#remote

// html

<div id="remote">

<input class="typeahead" type="text" placeholder="Oscar winners for Best Picture">

</div>

// JS

var bestPictures = new Bloodhound({

datumTokenizer: Bloodhound.tokenizers.obj.whitespace('value'),

queryTokenizer: Bloodhound.tokenizers.whitespace,

prefetch: '../data/films/post\_1960.json',

remote: {

url: '../data/films/queries/%QUERY.json',

wildcard: '%QUERY'

}

});

$('#remote .typeahead').typeahead(null, {

name: 'best-pictures',

display: 'value',

source: bestPictures

});

- Configurar typeahead

$(document).ready(function () {

// Bloodhound encapsula toda la lógica de negocio, llama a la api y pone en cache los resultados.

// Crear la consulta del objeto.

var customers = new Bloodhound({

// Convierte el dato a una pila de tokens,

datumTokenizer: Bloodhound.tokenizers.obj.whitespace('name'), // Ve la propiedad name de customer y usa espacios en blanco para generar los token.

queryTokenizer: Bloodhound.tokenizers.whitespace, // Transforma la query en una pila de tokens.

// Previene de hacer solicitudes adicionales al servidor.

// prefetch: '../data/films/post\_1960.json', // Proporcionar datos sobre la optimización

remote: {

url: '../api/customers?query=%QUERY', // %QUERY es lo que se pondrá en el text box.

wildcard: '%QUERY'

}

});

// Agregar menú desplegable a customer.

$('#customer').typeahead(

// Primer argumento para establecer el comportamiento de typeahead.

{

minLength: 3, // Hasta tener 3 elementos typeahead realizará la consulta.

highlight: true // Los carácteres que coincidan con la búsqueda serán resaltados en negritas.

},

{

name: 'customers', // Nombre de la propiedad.

display: 'name', // Propiedad a mostrar de la consulta.

source: customers // Fuente de datos (nombre del objeto Bloohound).

}

);

});

- Agregar un modelo para su posterior lectura en el registro.

$(document).ready(function () {

var viewModel = {}; // Inicializa viewmodel en blanco.

// Bloodhound encapsula toda la lógica de negocio, llama a la api y pone en cache los resultados.

var customers = new Bloodhound({

// Convierte el dato a una pila de tokens,

datumTokenizer: Bloodhound.tokenizers.obj.whitespace('name'), // Ve la propiedad name de customer y usa espacios en blanco para generar los token.

queryTokenizer: Bloodhound.tokenizers.whitespace, // Transforma la query en una pila de tokens.

// Previene de hacer solicitudes adicionales al servidor.

// prefetch: '../data/films/post\_1960.json', // Proporcionar datos sobre la optimización

remote: {

url: '../api/customers?query=%QUERY', // %QUERY es lo que se pondrá en el text box.

wildcard: '%QUERY'

}

});

// Elemento que se va a modificar.

$('#customer').typeahead(

// Primer argumento para establecer el comportamiento de typeahead.

{

minLength: 3, // Hasta tener 3 elementos typeahead realizará la consulta.

highlight: true // Los carácteres que coincidan con la búsqueda serán resaltados en negritas.

},

{

name: 'customers', // Nombre de la propiedad.

display: 'name', // Propiedad a mostrar de la consulta.

source: customers // Fuente de datos (nombre del objeto Bloohound).

}

) // Método jquery on.

.on("typeahead:select", function (e, customer) { // Evento y customer

viewModel.customerId = customer.id;

});

// movie

// Bloodhound encapsula toda la lógica de negocio, llama a la api y pone en cache los resultados.

var movies = new Bloodhound({

// Convierte el dato a una pila de tokens,

datumTokenizer: Bloodhound.tokenizers.obj.whitespace('name'), // Ve la propiedad name de movie (objeto regreado de la consulta) y usa espacios en blanco para generar los token.

queryTokenizer: Bloodhound.tokenizers.whitespace, // Transforma la query en una pila de tokens.

// Previene de hacer solicitudes adicionales al servidor.

// prefetch: '../data/films/post\_1960.json', // Proporcionar datos sobre la optimización

remote: {

url: '../api/movies?query=%QUERY', // %QUERY es lo que se pondrá en el text box.

wildcard: '%QUERY'

}

});

// Elemento que se va a modificar.

$('#movie').typeahead(

// Primer argumento para establecer el comportamiento de typeahead.

{

minLength: 3, // Hasta tener 3 elementos typeahead realizará la consulta.

highlight: true // Los carácteres que coincidan con la búsqueda serán resaltados en negritas.

},

{

name: 'myMovies', // Nombre de la propiedad.

display: 'name', // Propiedad a mostrar de la consulta.

source: movies // Fuente de datos (nombre del objeto Bloohound).

}

) // Método jquery on.

.on("typeahead:select", function (e, myMovies) { // Evento y mymovies

viewModel.movieId = mymovies.id;

});

});

- Mejorando los estilos css:

<form>

<div class="form-group">

<label>Customer</label>

<div class="tt-container">

<input id="customer" type="text" value="" class="form-control" />

</div>

</div>

<div class="form-group">

<label>Movie</label>

<div class="tt-container">

<input id="movie" type="text" value="" class="form-control" />

</div>

</div>

<div class="row">

<div class="col-md-4 col-sm-6">

<ul id="movies" class="list-group"></ul>

</div>

</div>

<button class="btn btn-primary">Submit</button>

</form>

- Configuar método para filtrar por query si se pasa el parámetro.

public IHttpActionResult GetCustomers(string query = null)

{

var customersQuery = db.Customers

.Include(c => c.MembershipType);

if (!String.IsNullOrWhiteSpace(query)) // Filtrado de customers por nombre.

{

customersQuery = customersQuery.Where(c => c.Name.Contains(query));

}

var customersDto = customersQuery.ToList()

.Select(Mapper.Map<Customer, CustomerDto>); // Pasa como delegado el método Map que recibe un objeto de tipo Customer como parámetro.

return Ok(customersDto);

}

- Filtrado de películas

public IHttpActionResult GetMovies(string query = null)

{

var moviesQuery = db.Movies.Include(m => m.Genre)

.Where(m => m.NumberAvailable>0);

if (!String.IsNullOrWhiteSpace(query)) // Filtrado de movies por nombre.

{

moviesQuery = moviesQuery.Where(m => m.Name.Contains(query));

}

var moviesDto = moviesQuery.ToList().Select(Mapper.Map<Movie, MovieDto>);

return Ok(moviesDto);

}

- Enviar formulario por ajax.

<form id="newRental"> ... </form>

$("#newRental").submit(function (e) {

e.preventDefault(); // Evita que el formulario se envíe por defecto.

$.ajax({

url: "/api/newRentals",

method: "post",

data: viewModel

})

.done(function () {

console.log("done");

})

.fail(function () {

console.log("error");

});

})

- Añadir notificación que el dato fue registrado.

- Instalar:

install-package toastr

- En bundle config añadir js y css

"~/Scripts/toastr.js"

"~/Content/toastr.css"

- En la función done de ajax modificar el mensaje.

$("#newRental").submit(function (e) {

e.preventDefault(); // Evita que el formulario se envíe por defecto.

$.ajax({

url: "/api/newRentals",

method: "post",

data: viewModel

})

.done(function () {

toastr.success("Rentals successfully recorded.");

})

.fail(function () {

toastr.error("Something unexpected happened.");

});

});

- Implementar validación del lado del cliente.

- Agregar jqueryval

@section scripts

{

@Scripts.Render("~/bundles/jqueryval")

}

- Para tener un campo con validación del lado clente se agrega el atributo required para que sea reconocido como un campo requerido.

<input id="customer" required type="text" value="" class="form-control" />

- Validar con el método validate() de jquery y solo enviar el formulario si es válido.

$("#newRental").validate({

submitHandler: function () { // Función a ejecutar si pasó la validación.

e.preventDefault(); // Evita que el formulario se envíe por defecto.

$.ajax({

url: "/api/newRentals",

method: "post",

data: viewModel

})

.done(function () {

toastr.success("Rentals successfully recorded.");

})

.fail(function () {

toastr.error("Something unexpected happened.");

});

}

});

- Dar estilos a las clases de error generadas al validar.

- Al tener un error jquery asigna la clase "error" al elemento que no pasó la validación.

<input id="customer" required="" type="text" value="" class="form-control tt-input error" autocomplete="off" spellcheck="false" dir="auto" style="position: relative; vertical-align: top; background-color: transparent;" aria-invalid="true">

<label id="customer-error" class="error" for="customer">This field is required.</label>

- Se asignan los estilos de error en el css para que aparezcan remarcados.

.field-validation-error,

label.error {

color: red;

}

.input-validation-error,

input.error {

border: 4px solid red;

}

- Validaciones personalizadas:

- Asignar la etiqueta que tendrá la validación.

<input id="customer" name="customer" data-rule-validCustomer="true" required type="text" value="" class="form-control" />

<input id="movie" name="myMovie" data-rule-atLeastOneMovie="true" type="text" value="" class="form-control" />

- Código de la validación.

$.validator.addMethod("validCustomer", // Nombre de la validación.

function () { // Función para validar.

return viewModel.customerId && viewModel.customerId !== 0;

},

"Please select a valid customer." // Mensaje de error.

);

$.validator.addMethod("atLeastOneMovie", // Nombre de la validación.

function () { // Función para validar.

return viewModel.movieIds.length > 0; // Condición de validación.

},

"Please select at least one movie." // Mensaje de error.

);

- Limpiar la vista y las variables una vez enviado el formulario.

var validator = $("#newRental").validate({

submitHandler: function () { // Función a ejecutar si pasó la validación.

$.ajax({

url: "/api/newRentals",

method: "post",

data: viewModel

})

.done(function () {

toastr.success("Rentals successfully recorded.");

// Limpiar cajas de texto y lista de películas.

$("#customer").typeahead("val", "");

$("#movie").typeahead("val", "");

$("#movies").empty();

// Limpiar variable donde se almacena el modelo.

viewModel = { movieIds: [] };

validator.resetForm(); // Resetear el formulario en términos de su validación.

})

.fail(function () {

toastr.error("Something unexpected happened.");

});

return false // Evita que el formulario se envíe por defecto.

}

});

Construcción de una nueva característica sistemáticamente.

- Entender el problema, sin distracciones.

- Comenzar desde el backend.

- Pensar en las entradas y salidas de la nueva característica.

- Declarar una API sencilla.

- Extender el modelo si es necesario.

- Hacer la implementación de la API conforme al happy path.

- Añadir detalles uno por uno paara validar.

- Desarrollar en el frontend seguiendo los mismos consejos.

Instalar IIS

Ir a panel de control.

Buscar agregar o quitar programas.

Se da clic en arctivar o desactivar caracteristicas.

Se activa IIS y se activan las configuraciones necesarias.

Publicando la aplicación

Ir al proyecto en la opción publish(publicar).

En la ventana se puede elegir entre publicar o integración continua.

- Publicar.

- Azure Websites: Elegir un sitio web creado en azure o crear uno nuevo, se necesita la cuenta de Azure.

- Import: Cuando ya se tiene un perfil creado y se quiere usar ese perfil para republicar el sitio.

- Servidor Web (IIS): Publica localmente usando IIS.

- Web Deploy: Publica usando IIS, para eso el servidor debe estar configurado para soportar IIS.

- Web Deploy Package: Genera solo la configuración de IIS sin guardarlo en IIS local.

- Carpeta: Publica los archivos en una carpeta para posteriormente ser leído por un servidor como IIS.

- Integración continua.

Publicando la base de datos

- Solo es necesario si no se usa web deploy y esto es debido a que no pueden ejecutar las migraciones por default.

- Para ejecutar las migraciones en un archivo se usa:

update-database -script // Ejecuta las migraciones pendientes.

update-database -script -SourceMigration:SeedUsers// Ejecuta desde la migración definida.

- Con las migraciones se garantiza tener el esquema y los datos por defecto configurados, ahorrando tiempo asi en la publicación de un sitio.

- Una vez generado el archivo para crear la base de datos lo podemos ejecutar y asi poder realizar la cadena de conexión.

Configuraciones de Construcción (Build configurations)

Son configuraciones ya sean para produccion, debug o testing, estas configuraciones por lo general tienen diferentes cadenas de conexión para base de datos, diferentes niveles de optimización de código, agregan o quitan configuraciones para detección de errores ,etc.

- Para añadir una nueva configuración se debe seguir los siguientes pasos:

- En debug ir a administrar nueva configuración.

- Seleccionar nueva en la opción para seleccionar una nueva configuración.

- Crear la configuración tomando como referencia la configuración deseada, para testing es recomendable usar release.

- En Web.Config dar clic derecho y seleccionar la opción agregar transformación de configuración.

- Editar las configuraciones deseadas en Web.nombreConfiguracion.config.

- En Web.Testing.config (Nueva configuración generada), se agregan todas las opciones pertinentes a la configuración como usar bundles, debug, configurar las credenciales necesarias para la aplicación.

- Las configuraciones colocadas en la configuración van a sobrescribir las que se tiene en Web.config.

- Sobre esa configuración también se puede crear un perfil y seleccionar la configuración testing para generar los archivos para IIS.

- Los perfiles creados aparecerán en el proyecto en Propierties/PublishProfiles.

Configuraciones de Aplicacción

- Las configurariones de aplicación son referencias que tenemos para sucribirnos a diversos servicios, como un servidor de email, la base datos, credenciales para apliaciones de terceros como un login y que conforme estemos en producción desarrollo o testing cambiaran sus ubicaciones.

- Como buena practica los datos confidenciales no deben estar puestos directamente en el código, si que se guardan en un archivo de configuración.

- En Web.config las podemos encontrar como appSettings.

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

</appSettings>

- Se pueden añadir nuevas configuraciones con una nueva llave y su respectivo valor.

<add key="FacebookAppId" value="374991787270999" />

<add key="FacebookAppSecret" value="numerosecreto" />

- Para acceder a las propiedades se accede mediante la clase ConfigurationManager.

app.UseFacebookAuthentication(

appId: ConfigurationManager.AppSettings["FacebookAppId"],

appSecret: ConfigurationManager.AppSettings["FacebookAppSecret"]);

- Para cambiar una configuración de su versión original (Web Config) a una personalizada se debe declarar en las 2 configuraciones, se debe verificar que esten dentro de <appSettings>.

- En Web.config

- <add key="MailServer" value="develop-smtp.vidly.com" />

- En WebTesting.config se configura (dentro de <appSettings>):

- <add key="MailServer" value="testing-smtp.vidly.com"

xdt:Transform="SetAttributes" xdt:Locator="Match(key)"/>

- Match(key) hace referencia a buscar por nombre de propiedad y SetAttributes indica que sustituirá por los demas valores indicados en los atributos.

- Para ver el preview de como quedará la transformación se usa

- Si no se ha publicado, Clic derecho en WebT.Testing.config y luego publicar.

- Click derecho en en WebT.Testing.config y luego obtener vista previa y transformar (todas las propiedades deben estar bien).

Separar appSettings para dar mayor seguridad

- Añadir un archivo de configuración con el archivo requerido, en este caso appSettings.config

- Llamar el archivo de configuración.

<appSettings configSource="AppSettings.config"></appSettings>

- En el servidor público solo subir un ejemplor de archivo appSettings sin las contraseñas reales establecidas.

Encriptar archivos

Para otorgar mayor seguridad y no dejar expuestas las claves se pueden encriptar los archivos y cuando sea necesario desencriptarlos, para ello .net ofrece una solución.

- La encriptación toma nuestro usuario de windows para encriptar y desencriptar los archivos.

- Ejecutar la consola de visual studio Developer Command Prompt for VS.

- Encriptar archivo appSettings.

aspnet\_regiis -pef "appSettings" "C:\Sitos\_IIS\Vidly\Testing" -prov "DataProtectionConfigurationProvider"

- Desencriptar archivo appSettings.

aspnet\_regiis -pdf "appSettings" "C:\Sitos\_IIS\Vidly\Testing"

Páginas de error personalizadas (Custom error pages)

- Son las páginas que nos aparecen cuando generamos una exepción o no tenemos la ruta configurada.

- Si dejamos nuestra página sin configurar una página de error, por defecto .NET nos mostrará la exepción en pantalla con todo el seguimiento y versiones del framework lo cual puede terminar en un ataque al sitio.

- Para agregar una página de error personalizada se agrega en Web.config dentro de la etiqueta <system.web>.

<system.web>

<customErrors mode="On"></customErrors>

...

</system.web>

- Para configurar solo a usuarios que acceden remotamente.

<customErrors mode="RemoteOnly"></customErrors>

- La vista que se llama al ejecutar la página de error es:

Shared/Error.cshtml.

- El controlador que llama a la vista de error el cual se encarga de atrapar todos los errores esta en AppStartFilterConfig:

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new HandleErrorAttribute()); // Filtro para llamado de error.

filters.Add(new AuthorizeAttribute());

filters.Add(new RequireHttpsAttribute());

}

}

- Otro tipo de error posible es cuando no se encuentra el recurso el cual genera un error 404.

- Para este caso IIS es quien generá la vista default del error.

- Para cambiar la vista de la página se configura a una redirección para cada código de error desplegado.

<customErrors mode="On">

<error statusCode="404" redirect="~/404.html"/>

</customErrors>

- Para cuando se pide un recurso estatico que no concuerda con una ruta de controlador no nos redirecciona pues no está siendo atendida por asp.net.

- Ejemplo de ruta.

https://localhost:44335/image.gif

- Configurar dentro de <system.webServer> y con modo File ya que sino cambia el estado de la respuesta a 200.

<httpErrors errorMode="Custom">

<remove statusCode="404"/>

<error statusCode="404" path="~/404.html" responseMode="File"/>

</httpErrors>

Revisar Exepciones no controladas.

- Para revisar las exepciones no controladas se puede usar una extensión llamada elmah (error logging models and handlers).

- Instalar Elmah

install-package Elmah

- Cuando se ejecuta una exepción.

- Ir a ~/elmah.axd

- Seleccionar detalles.

- Al seleccionar detalles se puede ver todo el seguimiento de la expecion así como los valores de las variables al momento de generarse.

- Elmah solo se puede ejecutar localmente por motivos de seguridad.

- Para acceder remotamente a Elmah se de debe configurar:

- En Web.config encontar elmah.axd.

- Dentro de system.web mas cercano se encuentra el httphandler el cual toma la petición y genera una respuesta, es de las formas de mas bajo nivelL de procesar una solicitud.

<httpHandlers>

<add verb="POST,GET,HEAD" path="elmah.axd" type="Elmah.ErrorLogPageFactory, Elmah" />

</httpHandlers>

- En la etiqueta system.web agregar la autorización, está puede ser para rol o para usuario de asp.net.

<authorization>

<allow roles="admin" />

<deny users="\*" />

</authorization>

<authorization>

<allow users="admin@vidly.com" />

<deny users="\*" />

</authorization>

- Documentación autorizacion http://code.google.com/p/elmah/wiki/SecuringErrorLogPages.

- Elmah también permite enviar correos si se ha detectado alguna excepción esto puede ser útil en producción para saber que ha ocurrido.

Modificar estilos para mejorar la aplicación.

Algunas técnicas para mejorar la visualización de la aplicación son:

- Para cambiar el color de un elemento de la barra de navegación.

- Inspeccionar el elemento.

- Intentar encontrar el background-color dentro de las propiedades.

- Si no se encuentra buscar en el elemento padre, asi hasta encontrarlo.

- No cambiar los colores heredados.

- Poner un color al azar y con la paleta ir buscando el que mas nos convenga.

- En un nuevo archivo css sobrescribir los estilos predefinidos de bootstrap para todas las partes de la aplicación que querramos.

- Para elegir los colores de nuestra aplicación nos podemos guiar con las diversas paletas de colores de color.adobe.com, con esto solo elegimos los colores y le asignamos el tema a nuestra página.

https://color.adobe.com/es/create/color-wheel