DAT División de Alta Tecnología

5

Aplicando Técnicas en una Aplicación ASP.NET MVC

Visual Studio 2017 Web Developer

Objetivos

Al finalizar el capítulo, el alumno:

- Diseñar el URL Routing de una aplicación MVC.
- Aplicar Action Filters y crea unos personalizados.
- Crear Html Helpers personalizados.
- Comprender el model binding y lo personaliza.
- Emplear y hace uso de View Components en MVC.
- Utilizar ViewModels cuando es necesario.
- Emplear y hace uso de Inyección de dependencia en vista.

Agenda

- Revisión de Razor
- Configuración de URL routing de MVC
- Creación de action filters
- Creación de HTML helpers
- Model binders y value providers MVC
- Creación viewmodels
- View components

Revisión de Razor

- Razor es sintaxis de marcado de código de lado del servidor que se agrega directamente a html. Fácil de aprender y usar. Es similar a C#.

- El uso de la @ funciona de dos maneras básicas:

  - **@expresión:** Renderiza la expresión en el navegador. Así @item.Nombre muestra el valor de ítem.Nombre. Es decir @expresión equivale a <%: expresión %>
  - **@{ código }:** Permite ejecutar un código que no genera salida HTML. Es decir @{código} equivale a <% Código %>

Revisión de Razor

```
@*
    Este es un bloque de comentario
*@


@{
    var greeting = "Welcome to our site!";
    var weekDay = DateTime.Now.DayOfWeek;
    var greetingMessage = greeting + " Today is: " + weekDay;
}
```

Definición de Variables

```
// Using the var keyword:
    var greeting = "Welcome to W3Schools";
    var counter = 103;
    var today = DateTime.Today;

    // Using data types:
    string greeting = "Welcome to W3Schools";
    int counter = 103;
    DateTime today = DateTime.Today;
```

Revisión de Razor

- Operadores

Tipos de operadores	Descripción
Operadores de asignación	=
Operadores matemáticos	+, -,* y /
Operadores de comparación	==, !=,<,>,<=,>=
Operadores lógicos	&&, ||
Operadores de concatenación	+

Revisión de Razor

- Estructuras repetitivas:

```
<ul>
   @for (int i = 0; i < 10; i++)
   {
      <li>@i</li>
   }
</ul>
```

```
<ul>
   @foreach (var x in Request.ServerVariables)
   {
   <li>@x</li>}
</ul>
```

```
@{
   var i = 0;
   while (i < 5)
   {
      i += 1;
      <p>Line @i</p>
   }
}
```

Revisión de Razor

- Estructuras condicionales:

```
@{var price = 50;}
<html>
<body>
   @if (price > 30)
   {
      <p>The price is too high.</p>
   }
</body>
</html>
```

```
@{var price = 20;}
<html>
<body>
   @if (price > 30)
   {
      <p>The price is too high.</p>
   }
   else
   {
      <p>The price is OK.</p>
   }
</body>
</html>
```

```
@{var price = 25;}
<html>
<body>
   @if (price >= 30)
   {
      <p>The price is high.</p>
   }
   else if (price > 20 && price < 30)
   {
      <p>The price is OK.</p>
   }
   else
   {
      <p>The price is low.</p>
   }
</body>
</html>
```

Revisión de Razor

- Consideraciones de sintaxis:

"Romper" el código de servidor

Mal

```
@for (int i = 0; i < 10; i++)
{
    El valor de i es: @i <br />
}
```

Bien

```
@for (int i = 0; i < 10; i++)
{
    <span>El valor de i es:</span> @i
<br />
}
```

Bien

```
@for (int i = 0; i < 10; i++)
{
    @:El valor de i es: @i <br />
}
```

Revisión de Razor

- Consideraciones de sintaxis:

Correo Electrónico

Envíame un mail: a usuario@servidor.com; en este caso Razor trata el texto como correo electrónico.

Escapar la arroba

Hacer uso del doble arroba si se necesita imprimir una arroba(@) en la pantalla.

```
<style>
@@media screen
{
    body { font-size: 13px;}
}
</style>
```

Configuración de URL Routing de MVC

Cuando se crea una aplicación ASP.NET MVC, se define una tabla de enrutamiento que se encarga de decidir qué controlador gestiona cada petición web, basándose en la URL de dicha petición

En cada petición URL no se asigna un archivo físico del disco, tal como una página .aspx, sino que se asigna la acción de un controlador

```
app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
```

Configuración de URL Routing de MVC

Cuando se crea una aplicación ASP.NET MVC, se define una tabla de enrutamiento que se encarga de decidir qué controlador gestiona cada petición web, basándose en la URL de dicha petición

En cada petición URL no se asigna un archivo físico del disco, tal como una página .aspx, sino que se asigna la acción de un controlador

```
app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
```

La ruta Default responde a los requests de la siguiente forma:

http://webserver - Controller: Home, Action: Index
http://webserver/Productos - Controller: Productos, Action: Index
http://webserver/Clientes/Details/5 - Controller: Clientes, Action: Details, Id: 5

Configuración de URL Routing de MVC

- Tipos de Rutas

Ruta Estática

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}")
        .MapRoute(
        name: "Catalogo",
        template: "Catalogo",
        defaults: new { controller="Product", action="Index"});
});
```

http://webserver/Catalogo - ***Controller: Product***, Action: Index

Configuración de URL Routing de MVC

- Tipos de Rutas

  Ruta Dinámica

```
.MapRoute(
name: "CatalogoDinamica",
template: "Catalogo/{id}/{action}",
defaults: new { controller = "Product", action = "details" },
constraints: new { id=@"\d+" });
```

http://webserver/Catalogo/5/Details

- Controller: Product, Action: Details, Id: 5

Configuración de URL Routing de MVC

- Tipos de Rutas

  Ruta SEO Friendly

```
.MapRoute(
name: "CatalogoSEO",
template: "CatalogoSEO/{nombre}/{id}",
defaults: new { controller = "Product", action = "details"},
constraints: new { id = @"\d+" })
```

http://webserver/CatalogoSEO/5/libro-mvc-dot-net - Controller: Product, Action: Details, Id: 5

Usando el atributo Route

No se debe indicar el atributo Route en el controlador. Si las demás acciones no tienen el atrributo Route, entonces toma el definido en el MapRoute de la clase Startup.

```
        [Route("Productos/Editar/{id}")]
            //GET: Product/Create
            public IActionResult Edit(int id)
            {
                var bus = new ProductBus();
                var model = bus.GetProducto(new Product() { ProductID = id });
                return View(model);
            }
    }
```

Uso y creación de Action Filters

Los Action Filters agregan lógica antes y después de la ejecución de los Action Methods. Se pueden definir para cada action method o a nivel de todo el controller, agregando el nombre del action filter entre [ ].

- **Action Filter por defecto:**

ResponseCache

```
[ResponseCache(Duration = 20)]
3 references
public IActionResult Index()
{
    return View();
}
```

Authorize

```
[Authorize(Roles="Administrator")]
public ActionResult Create()
{
```

Uso y creación de Action Filters

- **Filtros Personalizados**

ActionFilterAttribute

Se utiliza como clase base para crear filtros personalizados.
Namespace: Microsoft.AspNet.Mvc.Filter

```
public class LoggingFilterAttribute : ActionFilterAttribute
{
    protected static readonly log4net.ILog log =
       log4net.LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

    //Antes de que el action method se ejecute.
    0 references
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        var message = string.Format("Inicia ejecucion de: Controller {0}, Action {1}, Hora Inicio {2}",
            filterContext.Controller.ToString(),
            filterContext.ActionDescriptor.Name,
            DateTime.Now.ToLongTimeString());
        log.Debug(message);
    }

    //Despues de que el action method se ejecutó.
    0 references
    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        var message = string.Format("Termina ejecucion de: Controller {0}, Action {1}, Hora Fin {2}",
             filterContext.Controller.ToString(),
            filterContext.ActionDescriptor.Name,
            DateTime.Now.ToLongTimeString());
        log.Debug(message);
    }
}
```

Uso y creación de Action Filters

- **Filtros Personalizados**

ExceptionFilterAttribute: Se utiliza como clase base para crear filtros personalizados que permitan atrapar errores.

Namespace: Microsoft.AspNet.Mvc.Filter

```
1 reference
public class HandleCustomError: ExceptionFilterAttribute
{
    protected static readonly ILog log =
       LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

    1 reference
    public override void OnException(ExceptionContext filterContext)
    {
        var message = string.Format("Controller: {0}, Action: {1}, Exception: {2}, Hora: {3}",
            filterContext.RouteData.Values["controller"].ToString(),
            filterContext.RouteData.Values["action"].ToString(),
            filterContext.Exception.ToString(),
            DateTime.Now.ToString());

        log.Error(message);

        base.OnException(filterContext);
    }
}
```

Revisión de Model Binders y Value Providers

- **Model Binders**

Supongamos que no sabemos nada acerca de las características que tiene ASP.NET MVC respecto al Model Binding, que como vamos a ver hace el trabajo más fácil.

```
public ActionResult Edit()
{
    var album = new Album();
    album.Titulo = Request.Form["Titulo"];
    album.Precio = Decimal.Parse(Request.Form["Precio"]);
    //Continua con las demás propiedades...
    return View(album);
}
```

Con el Binding de MVC es mucho más fácil.

```
public ActionResult Edit(Album album)
{
    //Lógica del action method...
    return View(album);
}
```

Revisión de Model Binders y Value Providers

- **Model Binders**

  Usando el atributo Bind

```
0 references
public IActionResult CreateProd([Bind("ProductID, Name")] Product product)
```

- **Binding Manual**

  Se debe utilizar la función ***TryUpdateModelAsync***

```
[ActionName("Create")]
[HttpPost]
0 references
public IActionResult CreateProd()
{
    Product product = new Product();

    bool x = TryUpdateModelAsync(product).Result;

    var bus = new ProductBus();

    if (bus.Insert(product) > 0)
    {
        return RedirectToAction("Index");
    }

    return View(product);
}
```

Revisión de Model Binders y Value Providers

- **Custom Binder**

IModelBinder: Permite implementar binders para leer y enlazar los valores que los ValueProvider han leído de la petición Http.

```
namespace Microsoft.AspNet.Mvc.ModelBinding
{
    ...public interface IModelBinder
    {
        ...Task<ModelBindingResult> BindModelAsync(ModelBindingContext bindingContext);
    }
}
```

BindModelAsync: Este método se invoca al momento de realizarse el enlace a los datos que llegan en la petición Http.

¿Cómo usar?

```
[ActionName("Create")]
[HttpPost]
0 references
public IActionResult CreateProd([ModelBinder(BinderType = typeof(ProductBinder))] Product product)
```

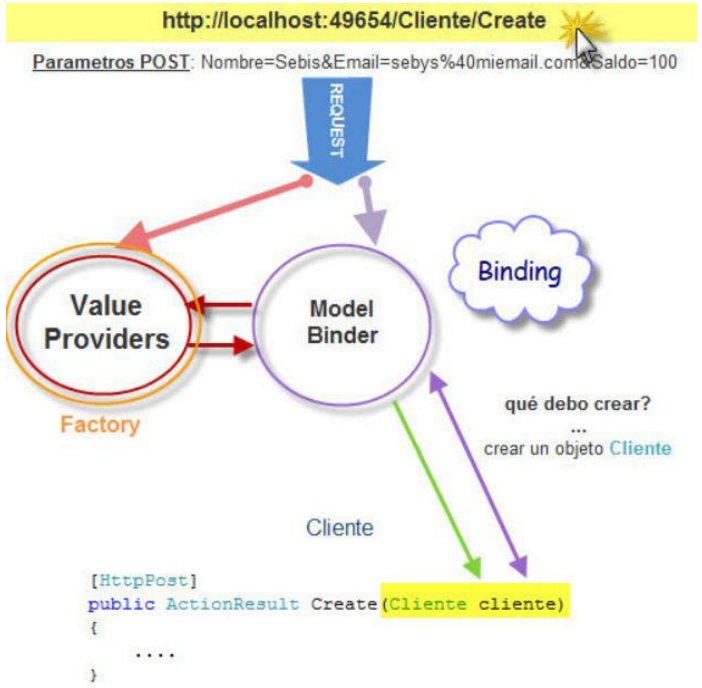
Revisión de Model Binders y Value Providers

- **Value Providers**

  Provider por defecto:

  FormValueProvider

  - RouteDataValueProvider
  - QueryStringValueProvider
  - HttpFileCollectionValueProvider



Revisión de Model Binders y Value Providers

- **Value Providers (Custom)**

IValueProvider: Permite implementar el comportamiento de lectura de los valores que se envían desde el browser.

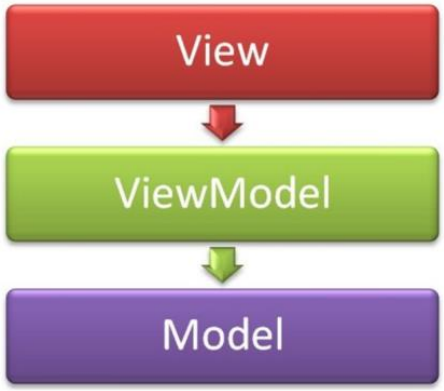
```
namespace Microsoft.AspNet.Mvc.ModelBinding
{
    public interface IValueProvider
    {
        bool ContainsPrefix(string prefix);
        ValueProviderResult GetValue(string key);
    }
}
```

IValueProviderFactory

```
namespace Microsoft.AspNet.Mvc.ModelBinding
{
    public interface IValueProviderFactory
    {
        Task<IValueProvider> GetValueProviderAsync(ValueProviderFactoryContext context);
    }
}
```

Creación y uso de ViewModels

- Con frecuencia, una Vista tiene que mostrar una variedad de datos que no corresponde directamente a un modelo de dominio.
- Uno de los enfoques que se puede tomar, y el que ofrece ventajas como intellisense y permite la creación de vistas strongly-typed, es el de escribir una clase personalizada para la Vista, es decir, un ViewModel.



Introducción a los View Components

Los View Components son muy similares a lo que en versiones anteriores se conocían como Vistas parciales, las cuales buscan renderizar pequeños bloques de HTML, sirve en casos en los que se requiera manipular pequeñas porciones de una página.

1. Implementar una clase que herede de ViewComponent.

```
public class PriorityListViewComponent : ViewComponent
{
```

2 . Crear la vista asociada a este componente

Ejercicio Nº 5.1: Implementa inyección de dependencia con SimpleInjector

Implementa inyección de dependencias con SimpleInjector.

Al finalizar el laboratorio, el alumno logrará:

- Implementar y configurar SimpleInjector como contenedor para inyección de dependencias.

Ejercicio Nº 5.2: Crear mantenimiento para Customer

Crea el mantenimiento para Customer

Al finalizar el laboratorio, el alumno logrará:

- Implementar un mantenimiento del tipo CRUD con ASP NET MVC.

Ejercicio Nº 5.3: Crear el Action Filters

Crear dos filtros personalizados, uno para atrapar error y guardarlo en un log de error tipo Log4Net, y el otro para registrar cada evento en las acciones de los controladores.

Al finalizar el laboratorio, el alumno logrará:

- Identificar la sintaxis correcta para usar y crear filtros personalizados.

Ejercicio Nº 5.4: Crear el HTML Helper

Utilizar el código creado en el laboratorio N° 3.2, para crear un Tag Helper que permita redireccionar al correo del cliente.

Al finalizar el laboratorio, el alumno logrará:

- Identificar la sintaxis correcta para usar y crear Tag Helper.

Ejercicio Nº 5.5: Crear el ViewModel

Crear un ViewModel que muestre la las ordenes de un Customer.

Al finalizar el laboratorio, el alumno logrará:

- Identificar el concepto de ViewModel.

Tarea Nº 3.1: Modificar el mantenimiento de productos realizado en los laboratorios

Modificar el mantenimiento de Productos realizado en los laboratorios, para que en vez de pasar por el ViewBag los datos que se necesitan cargar en combos (tales como ProductModels y ProductSubcategories y UnitMeasures), estos formen parte de un ProductViewModel. Por lo tanto, las vistas tendrán que ser tipadas con este ProductViewModel en vez de Product.

Al finalizar el laboratorio, el alumno logrará:

- Identificar el concepto de ViewComponent.

Tarea Nº 3.2 Crear el listado de Productos haciendo uso de ViewComponent

Crear el listado de Productos haciendo uso de ViewComponent.

Al finalizar el laboratorio, el alumno logrará:

- Identificar el concepto de ViewComponent.