# Routing

1. Orden Rutas es Importante

routes.MapRoute(  
"Producto-Detalles",  
"Productos/{id}",  
new { controller = "Productos", action = "Detalle" });

RouteDebug.RouteDebugger.RewriteRoutesForTesting(RouteTable.Routes);

1. Ejercicio Rutas

    [TestClass]  
    public class RouteRegisterTests  
    {  
        [ClassInitialize]  
        public static void Initialize(TestContext context)  
        {  
            MvcApplication.RegisterRoutes(RouteTable.Routes);  
        }  
   
   
        [TestMethod]  
        public void LasUrlsConcuerdanConLasRutasRegistradas()  
        {  
            "~/".ShouldMapTo<ProductosController>(action => action.Index(null, 1));  
            "~/libros".ShouldMapTo<ProductosController>(action => action.Index("libros", 1));  
            "~/pagina2".ShouldMapTo<ProductosController>(action => action.Index(null, 2));  
            "~/libros/pagina2".ShouldMapTo<ProductosController>(action => action.Index("libros", 2));  
        }  
    }

            routes.MapRoute(null,  
                "",  
                new { controller = "Productos", action = "Index", pagina = 1, categoria = (string)null }  
            );  
   
            routes.MapRoute(null,  
                "pagina{pagina}",  
                new { controller = "Productos", action = "Index", categoria = (string)null },  
                new { pagina = @"\d+" }  
            );  
   
            routes.MapRoute(null,  
                "{categoria}",  
                new { controller = "Productos", action = "Index", pagina = 1 }  
            );  
   
            routes.MapRoute(null,  
                "{categoria}/pagina{pagina}",  
                new { controller = "Productos", action = "Index" },  
                new { pagina = @"\d+" }  
            );

public IEnumerable<Producto> Buscar(string categoria, int pagina, int tamañoPagina)  
        {  
            var productos = categoria == null  
                ? context.Productos  
                : context.Productos.Where(x => x.Categoria.Nombre == categoria);  
   
            return productos.OrderBy(x => x.Nombre)  
                .Skip((pagina - 1) \* tamañoPagina).Take(tamañoPagina).ToList();  
        }

1. Ejercicio Links

<div class="left\_content">  
 <div class="title\_box">Categorias</div>  
 <ul class="left\_menu"> <li class="odd">@Html.ActionLink("Libros","index",new {categoria="libros"})</li>  
 <li class="even">@Html.ActionLink("Deportes","index",new {categoria="deportes"})</li>  
 </ul>   
</div><!-- end of left content -->

1. Link Area

<li>@Html.ActionLink("Home","Index","Productos", null, new { @class="nav"})</li>  
 <li class="divider"></li>  
 <li>@Html.ActionLink("Admin", "Index", "Home", new { area = "administracion" }, new { @class="nav"})</li>  
 <li class="divider"></li>

# Controllers and Actions

1. ViewBag en el controller

public ActionResult Crear()

{

ViewBag.Categorias = categoriasRepository.Todos();

return View(new Producto());

}

        [HttpPost]  
        public ActionResult Crear(Producto producto)  
        {  
            productosRepository.Save(producto);  
            productosRepository.Commit();  
            TempData["Mensaje"] =   
                "Se ha creado el producto: " + producto.Nombre;  
            return RedirectToAction("index");  
        }

@model TiendaVirtual.Domain.Producto  
   
<!DOCTYPE html>  
<html>  
<head>  
    <title>Crear</title>  
    <link href="@Url.Content("~/Content/themes/ecommerce/admin.css")" rel="stylesheet" type="text/css" />  
</head>  
<body>  
    <h1>  
        Crear un Producto  
    </h1>  
   
    @using (@Html.BeginForm())  
    {   
        <div class="editor-label">  
            @Html.LabelFor(x=>x.Nombre)  
        </div>  
        <div class="editor-field">  
            @Html.TextBoxFor(x => x.Nombre)  
        </div>  
   
        <div class="editor-label">  
            @Html.LabelFor(x=>x.Precio)  
        </div>  
        <div class="editor-field">  
            @Html.TextBoxFor(x => x.Precio)  
        </div>  
      
        <div class="editor-label">  
            @Html.LabelFor(x=>x.Categoria)  
        </div>  
        <div class="editor-field">  
            @Html.DropDownListFor(x => x.CategoriaId, new SelectList(ViewBag.Categorias, "Id", "Nombre"))  
        </div>  
          
        <input type="submit" value="Grabar" />  
        @Html.ActionLink("Cancelar y regresar al listado", "Index")  
    }  
   
</body>  
</html>

1. Strongly Typed

    public class EditarProductoViewModel  
    {  
        public EditarProductoViewModel(Producto producto  
            , IEnumerable<Categoria> categorias)  
        {

this.Id = producto.Id;

this.Nombre = producto.Nombre;  
            this.Precio = producto.Precio.ToString();  
            this.Categorias = categorias.Select(x =>  
                new SelectListItem  
                {  
                    Text = x.Nombre,  
                    Value = x.Id.ToString(),  
                    Selected = producto.CategoriaId == x.Id  
                });  
        }

public String Id { get; set; }

        public String Nombre { get; set; }  
   
        public String Precio { get; set; }  
   
        public IEnumerable<SelectListItem> Categorias { get; set; }  
    }

        public ActionResult Editar(int id)  
        {  
            var producto = productosRepository.ById(id);  
            var categorias = categoriasRepository.Todos();  
            return View(new EditarProductoViewModel(producto, categorias));  
        }

        [HttpPost]  
        public ActionResult Editar(int id, FormCollection values)  
        {  
            var producto = productosRepository.ById(id);  
            UpdateModel(producto);  
            productosRepository.Commit();  
            return RedirectToAction("index");  
        }

@model TiendaVirtual.Web.Areas.Administracion.Models.EditarProductoViewModel  
   
<!DOCTYPE html>  
<html>  
<head>  
    <title>Editar</title>  
    <link href="@Url.Content("~/Content/themes/ecommerce/admin.css")" rel="stylesheet" type="text/css" />  
</head>  
<body>  
    <h1>  
        Editar @Model.Nombre  
    </h1>  
   
    @using (@Html.BeginForm())  
    {   
        <div class="editor-label">  
            @Html.LabelFor(x => x.Nombre)  
        </div>  
        <div class="editor-field">  
            @Html.TextBoxFor(x => x.Nombre)  
        </div>  
   
        <div class="editor-label">  
            @Html.LabelFor(x => x.Precio)  
        </div>  
        <div class="editor-field">  
            @Html.TextBoxFor(x => x.Precio)  
        </div>  
      
        <div class="editor-label">  
            @Html.LabelFor("CategoriaId","Categoria")  
        </div>  
        <div class="editor-field">  
            @Html.DropDownList("CategoriaId", Model.Categorias)  
        </div>  
          
        <input type="submit" value="Grabar" />  
        @Html.ActionLink("Cancelar y regresar al listado", "Index")  
    }  
</body>  
</html>

1. Listado de Productos

        public ActionResult Index()  
        {  
            var productos = productosRepository.Todos();  
            return View(productos);  
        }

@model IEnumerable<TiendaVirtual.Domain.Producto>  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Index</title>  
    <link href="@Url.Content("~/Content/themes/ecommerce/admin.css")" rel="stylesheet" type="text/css" />  
</head>  
<body>  
   
    <h1>  
        Listado de Productos  
    </h1>  
   
    @if (@TempData["Mensaje"]!=null)  
    {  
        <div class="mensaje">@TempData["Mensaje"]</div>  
    }  
   
    <table class="Grid">  
        <tr>  
            <th>  
                Nombre  
            </th>  
            <th>  
                Precio  
            </th>  
            <th>  
                Categoria  
            </th>  
            <th>  
            </th>  
        </tr>  
        @foreach (var item in Model)  
        {  
            <tr>  
                <td>  
                    @item.Nombre  
                </td>  
                <td>  
                    @String.Format("{0:F}", item.Precio)  
                </td>  
                <td>  
                    @item.Categoria.Nombre  
                </td>  
                <td>  
                    @Html.ActionLink("Editar", "Editar", new { id = item.Id })  
                </td>  
            </tr>  
        }  
    </table>  
    <p>  
        @Html.ActionLink("Crear Nuevo Producto", "Crear")  
    </p>  
</body>  
</html>

1. Agregando una imagen al producto

* En el EDMX, agregar un objeto complejo imagen, con 2 propiedades : nombre, tipo
* Generar script migración
* Actualizar Model Poco Template

<add key="DirectorioImagenes" value="~/content/productos"/>

        public ActionResult Imagen(int id)  
        {  
            var productos = this.productosRepository.ById(id);  
            var path = Path.Combine(ConfigurationManager.AppSettings["DirectorioImagenes"],   
                       productos.Imagen.Nombre);  
            return File(path, productos.Imagen.Tipo);  
        }

* Actualizar ViewModel

this.TieneImagen = producto.Imagen != null;

        <div class="editor-field">  
            @if (Model.TieneImagen)  
            {  
                <img src="@Url.Action("imagen", "productos", new { area = "", id = @Model.Id })" alt="@Model.Nombre" />  
            }  
            <div>  
                Cargar nueva imagen:<input name="archivo" type="file" value="Imagen" /></div>  
        </div>

        [HttpPost]  
        public ActionResult Editar(int id, HttpPostedFileBase archivo)  
        {  
            var producto = productosRepository.ById(id);  
            UpdateModel(producto);  
            if (archivo != null)  
            {  
                producto.Imagen = new Imagen{Nombre = archivo.FileName,  
                                             Tipo = archivo.ContentType};  
                var path=Path.Combine(ConfigurationManager.AppSettings["DirectorioImagenes"], archivo.FileName);  
                archivo.SaveAs(path);  
            }  
            productosRepository.Commit();  
            return RedirectToAction("index");  
        }

        public ActionResult Imagen(int id)  
        {  
            var productos = this.productosRepository.ById(id);  
            var path = Path.Combine(ConfigurationManager.AppSettings["DirectorioImagenes"],   
                       productos.Imagen.Nombre);  
            return File(path, productos.Imagen.Tipo);  
        }

@if (producto.Imagen != null)  
            {  
                <div class="product\_img">  
                    <img src="@Url.Action("imagen", "productos", new { id = @producto.Id })"   
                         alt="@producto.Nombre" title="@producto.Nombre" border="0" />  
                </div>  
            }

1. Creando un Filtro de Logging

* Agregar el xml de log4net

private static void ConfigureLogging()  
  {  
   log4net.Config.XmlConfigurator.Configure(  
                new FileInfo(HttpContext.Current.Server.MapPath("~/log4net.config")));  
  }

//[AttributeUsageAttribute(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = false)]  
    public class LoggingAttribute : FilterAttribute, IActionFilter, IExceptionFilter  
    {  
        private ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);  
   
        public void OnActionExecuting(ActionExecutingContext filterContext)  
        {  
            string parameters = filterContext.ActionParameters  
                            .Select(x => x.Value).Aggregate((a, b) => a + "," + b).ToString();  
            log.Info(this.GetMessage("OnActionExecuting", filterContext.RouteData, parameters));  
        }  
   
        public void OnActionExecuted(ActionExecutedContext filterContext)  
        {  
            string resultType = filterContext.Result.GetType().Name;  
            log.Info(this.GetMessage("OnActionExecuted", filterContext.RouteData, resultType: resultType));  
        }  
   
        public void OnException(ExceptionContext filterContext)  
        {  
            log.Error(this.GetMessage("OnException", filterContext.RouteData)  
                , filterContext.Exception);  
            filterContext.Result = new ViewResult { ViewName = "Error" };  
            filterContext.ExceptionHandled = true;  
        }  
   
        public string GetMessage(string methodName, RouteData routeData, string parameters = "", string resultType = "")  
        {  
            return string.Format("{0} controller:{1} action:{2} parameters:[{3}] result:{4}", methodName,  
                   routeData.Values["controller"], routeData.Values["action"], parameters, resultType);  
        }  
    }

* Agregar el atributo primero a nivel de método y luego a nivel de clase

[Logging]  
    public class ProductosController : Controller  
    {  
        [Logging]  
        public ActionResult Index(String categoria, int pagina)

* Para probar el OnException quitar la constraint de la ruta /pagina{pagina}.
* No incluir en el código de los alumnos la parte de ViewResult en el OnException y mostrar únicamente que pasa en vivo.

1. Filtro Authorize

    public class LoginviewModel  
    {  
        public String Usuario { get; set; }  
        public String Password { get; set; }  
    }

public class UsuarioController : Controller  
    {  
        public ActionResult LogOn()  
        {  
            return View();  
        }  
   
        [HttpPost]  
        public ActionResult LogOn(LoginviewModel model, string returnUrl)  
        {  
            if (!Autenticar(model.Usuario, model.Password))  
            {  
                return View();  
            }  
   
            FormsAuthentication.SetAuthCookie(model.Usuario, false);  
            return Redirect(returnUrl ?? Url.Action("Index", "Home"));  
        }  
   
        private bool Autenticar(string usuario, string password)  
        {  
            return usuario == "admin" && password == "password";  
        }  
    }

@model TiendaVirtual.Web.Areas.Administracion.Models.LoginviewModel  
@{  
    Layout = null;  
}  
<!DOCTYPE html>  
<html>  
<head>  
    <title>LogOn</title>  
    <link href="@Url.Content("~/Content/themes/ecommerce/admin.css")" rel="stylesheet" type="text/css" />  
</head>  
<body>  
    <h1>Log In</h1>  
    <p>Escriba sus datos para acceder a la parte administrativa</p>  
   
    @using (Html.BeginForm())  
    {   
        <div class="editor-label">  
            @Html.LabelFor(x => x.Usuario)  
        </div>  
        <div class="editor-field">  
            @Html.TextBoxFor(x => x.Usuario)  
        </div>  
          
        <div class="editor-label">  
            @Html.LabelFor(x => x.Password)  
        </div>  
        <div class="editor-field">  
            @Html.TextBoxFor(x => x.Password)  
        </div>  
        <p>  
            <input type="submit" value="Log In" />  
        </p>  
    }  
</body>  
</html>

    [Authorize]  
    public class HomeController : Controller

    <authentication mode="Forms">  
      <forms loginUrl="~/administracion/usuario/logon" timeout="2880" />  
    </authentication>

# Views

1. Helper Method de paginación.

Crear la clase en Domain/Pagination/PageList.cs

    public interface IPagedList  
    {  
        int TotalItems { get; set; }  
   
        int ItemsPerPage { get; set; }  
   
        int CurrentPage { get; set; }  
   
        int TotalPages { get; }  
   
        bool HasPreviousPage { get; }  
   
        bool HasNextPage { get; }  
    }

Crear la clase en Web/HtmlHelpers

public static class PagingHelpers  
    {  
        public static MvcHtmlString PageLinks(this HtmlHelper html,  
                                              IPagedList pagedList,  
                                              Func<int, string> pageUrl)  
        {  
            StringBuilder liHtml = new StringBuilder();  
              
            if (pagedList.HasPreviousPage)  
            {  
                TagBuilder liTag = CreatePageLinkTag(pageUrl, pagedList.CurrentPage - 1, "<<Anterior");  
                liHtml.AppendLine(liTag.ToString());  
            }  
   
   
            for (int i = 1; i <= pagedList.TotalPages; i++)  
            {  
                TagBuilder liTag = CreatePageLinkTag(pageUrl, i, i.ToString());  
                if (pagedList.CurrentPage==i)  
                {  
                    liTag=new TagBuilder("li");  
                    liTag.AddCssClass("active");  
                    liTag.InnerHtml = i.ToString();  
                }  
   
                liHtml.AppendLine(liTag.ToString());  
            }  
   
            if (pagedList.HasNextPage)  
            {  
                TagBuilder liTag = CreatePageLinkTag(pageUrl, pagedList.CurrentPage + 1, "Siguiente>>");  
                liHtml.AppendLine(liTag.ToString());  
            }  
   
            TagBuilder ul = new TagBuilder("ul");  
            ul.InnerHtml = liHtml.ToString();  
   
            return MvcHtmlString.Create(ul.ToString());  
        }  
   
        private static TagBuilder CreatePageLinkTag(Func<int, string> pageUrl, int pageNumber, string tagText)  
        {  
            TagBuilder aTag=new TagBuilder("a");  
            aTag.MergeAttribute("href", pageUrl(pageNumber));  
            aTag.InnerHtml = tagText;  
            TagBuilder liTag=new TagBuilder("li");  
            liTag.InnerHtml = aTag.ToString();  
            return liTag;  
        }

public class PagedList<T> : List<T>, IPagedList  
    {  
        public PagedList(IEnumerable<T> source, int totalItems, int currentPage, int pageSize)  
        {  
            this.TotalItems = totalItems;  
            this.CurrentPage = currentPage;  
            this.ItemsPerPage = pageSize;  
            this.AddRange(source);  
        }  
   
        public int TotalItems { get; set; }  
        public int ItemsPerPage { get; set; }  
        public int CurrentPage { get; set; }  
   
        public int TotalPages  
        {  
            get { return (int)Math.Ceiling((decimal)TotalItems / ItemsPerPage); }  
        }  
   
        public bool HasPreviousPage  
        {  
            get { return (CurrentPage > 1); }  
        }  
   
        public bool HasNextPage  
        {  
            get { return CurrentPage < TotalPages; }  
        }  
    }

Crear la clase en Persistence/Extensions

    public static class Pagination  
    {  
        public static PagedList<T> ToPagedList<T>(this IQueryable<T> source, int currentPage, int pageSize)  
        {  
            var items = source.Skip((currentPage - 1) \* pageSize).Take(pageSize).ToList();  
            var totalItems=source.Count();  
            return new PagedList<T>(items, totalItems, currentPage, pageSize);  
        }  
    }

        <add namespace="TiendaVirtual.Web.HtmlHelpers" />  
        <add namespace="TiendaVirtual.Domain.Pagination"/>

<div id="pagination">  
    @Html.PageLinks(Model, i => Url.Action("index", new { pagina = i, categoria = ViewContext.RouteData.Values["categoria"] }))  
</div>

1. Partial view \_product

@model TiendaVirtual.Domain.Producto  
<div class="prod\_box">  
    <div class="center\_prod\_box">  
        <h4 class="product\_title">@Model.Nombre</h4>  
        <div class="product\_img">  
            @if (Model.Imagen.Nombre != null)  
            {  
                <img src="@Url.Action("imagen", "productos", new { id = @Model.Id })"   
                         alt="@Model.Nombre" title="@Model.Nombre" border="0" />  
            }  
        </div>  
        <span class="price">Precio: @Model.Precio$</span>  
    </div>  
    <div class="prod\_details\_tab">  
        <a href="#" class="prod\_buy">Añadir a Carro</a> <a href="#" class="prod\_details">Detalles</a>  
    </div>  
</div>

Realizar un cambio al view \_product: poner el if dentro del div product\_img y hacer la condicional utilizando Model.Imagen.Nombre.

@foreach (var producto in Model)  
{   
    @Html.Partial("\_producto",producto)  
}

1. Child Action Menu

    public class NavegacionController : Controller  
    {  
        readonly CategoriasRepository categoriasRepository=new CategoriasRepository();  
        [ChildActionOnly]  
        public ActionResult Menu()  
        {  
            var categorias = categoriasRepository.Todos();  
            return View("\_Menu",categorias);  
        }  
   
    }

<div class="title\_box">  
    Categorias</div>  
<ul class="left\_menu">  
    @for (int i = 0; i < Model.Count; i++)  
    {  
        <li class="@(i % 2 == 1 ? "even" : "odd")">@Html.ActionLink(Model[i].Nombre, "index",  
                            "productos", new { categoria = Model[i].Nombre }, null)</li>  
    }  
</ul>

<div class="left\_content">  
@Html.Action("menu","navegacion")  
</div><!-- end of left content -->

1. Layout área de administración

<!DOCTYPE html>  
<html>  
<head>  
    <title>@ViewBag.Title</title>  
    <link href="@Url.Content("~/Content/themes/ecommerce/admin.css")" rel="stylesheet" type="text/css" />  
</head>  
<body>  
    @RenderBody()  
</body>  
</html>

Mover el contenido de las vistas

Especificar la ruta del layout en la vista LogOn.

Crear la vista \_ViewStart y especificar el layout para todas las vistas.

# Models

1. Explorando los Default Templates

@using (Html.BeginForm())  
{  
    @Html.ValidationSummary(true)  
    <div class="editor-label">  
        @Html.LabelFor(x => x.Usuario)  
    </div>  
    <div class="editor-field">  
        @Html.EditorFor(x => x.Usuario)  
    </div>  
          
    <div class="editor-label">  
        @Html.LabelFor(x => x.Password)  
    </div>  
    <div class="editor-field">  
        @Html.EditorFor(x => x.Password)  
    </div>  
    <p>  
        <input type="submit" value="Log In" />  
    </p>  
}

Agregando el template password explícitamente

@Html.EditorFor(x => x.Password,"password")

Descargar MvcFutures y explorar String,Password y Object

Sobreescribir el template Shared/EditorTemplates/String.cshtml

@Html.TextBox("", ViewData.TemplateInfo.FormattedModelValue, new { @class = "text-box" })

1. DateTime custom template

Modificar el LoginViewModel para que utilice metadata

@Html.EditorFor(x => x.Password)

[DataType(DataType.Password)]  
        public String Password { get; set; }

Crear /shared/editortemplates/datetime.cshtml

@model DateTime?  
@Html.TextBox("", Model.HasValue ? Model.Value.ToShortDateString() : DateTime.Now.ToShortDateString(),  
                new { @class = "datePicker" })  
<script type="text/javascript">  
    $(function () {  
        $(".datePicker").datepicker();  
    });  
</script>

Agregar propiedad al EditarViewModel

public DateTime? InicioVentas { get; set; }

Modificar la Vista

<div class="editor-label">  
     @Html.LabelFor(x => x.InicioVentas)  
</div>  
<div class="editor-field">  
     @Html.EditorFor(x => x.InicioVentas)  
</div>

1. Agregar Validación al Login

Atributos LoginViewModel

Explorar los parámetros de los atributos

    public class LoginViewModel  
    {  
        [Required]  
        public String Usuario { get; set; }  
   
        [Required]  
        [DataType(DataType.Password)]  
        public String Password { get; set; }  
    }

Modificando el controller

        [HttpPost]  
        public ActionResult LogOn(LoginViewModel model, string returnUrl)  
        {  
            if (ModelState.IsValid)  
                if (!Autenticar(model.Usuario, model.Password))  
                    ModelState.AddModelError("", "El usuario o password son incorrectos");  
   
            if (!ModelState.IsValid)  
                return View();  
   
            FormsAuthentication.SetAuthCookie(model.Usuario, false);  
            return Redirect(returnUrl ?? Url.Action("Index", "Home"));  
        }

Mostrando los errors en la vista

@Html.ValidationSummary(true)  
    <div class="editor-label">  
        @Html.LabelFor(x => x.Usuario)  
    </div>  
    <div class="editor-field">  
        @Html.EditorFor(x => x.Usuario)  
        @Html.ValidationMessageFor(x=>x.Usuario)  
    </div>  
          
    <div class="editor-label">  
        @Html.LabelFor(x => x.Password)  
    </div>  
    <div class="editor-field">  
        @Html.EditorFor(x => x.Password)  
        @Html.ValidationMessageFor(x=>x.Password)  
    </div>

1. Agregar Validación Editar Producto

Crear la clase ProductoValidation y agregar la referencia System.Component.DataAnnotations

    [MetadataType(typeof(ProductoValidation))]  
    public partial class Producto  
    {  
        public class ProductoValidation  
        {  
            [Required(ErrorMessage = "El nombre es obligatorio")]  
            public string Nombre { get; set; }  
   
            [Required]  
            public decimal Precio { get; set; }  
        }  
    }

Modificando el Crear Action en el HomeController

[HttpPost]  
        public ActionResult Crear(Producto producto)  
        {  
            if (!ModelState.IsValid)  
            {  
                ViewBag.Categorias = categoriasRepository.Todos();  
                return View();  
            }  
            productosRepository.Save(producto);  
            productosRepository.Commit();  
            TempData["Mensaje"] =  
                "Se ha creado el producto: " + producto.Nombre;  
            return RedirectToAction("index");  
        }

Modificar la vista para mostrar errores

    <div class="editor-field">  
        @Html.TextBoxFor(x => x.Nombre)  
        @Html.ValidationMessageFor(x => x.Nombre)  
    </div>  
   
  
    <div class="editor-field">  
        @Html.TextBoxFor(x => x.Precio)  
        @Html.ValidationMessageFor(x => x.Precio)  
    </div>

Creando el atributo Min

    public class MinAttribute : ValidationAttribute  
    {  
        private readonly double minValue;  
   
        public MinAttribute(double minValue)  
        {  
            this.minValue = minValue;  
        }  
   
        public override bool IsValid(object value)  
        {  
            var doubleValue = Convert.ToDouble(value);  
            return doubleValue >= minValue;  
        }  
    }

1. Agreando Validación de Cliente

Explorar el Web.Config para ver la validación a nivel cliente y

Agregar los scripts necesarios en el layout.

Probarel login o el crear producto.

Activar unobtrusive validación y realizar nuevamente la prueba.

1. Extendiendo la Validación en el Cliente

       public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata, ControllerContext context)  
        {  
            var rule = new ModelClientValidationRule();  
            rule.ErrorMessage = this.FormatErrorMessage(metadata.DisplayName);  
            rule.ValidationType = "min";  
            rule.ValidationParameters.Add("value", minValue);  
            yield return rule;  
        }

/// <reference path="jquery-1.4.4-vsdoc.js" />  
/// <reference path="jquery.validate-vsdoc.js" />  
/// <reference path="jquery.validate.unobtrusive.js" />  
   
jQuery.validator.addMethod("min", function (value, element, param) {  
    return value >= param;  
});  
   
jQuery.validator.unobtrusive.adapters.addSingleVal("min", "value");

1. Remote Validation

[Remote("ExisteNombre", "Home", ErrorMessage = "El nombre ya existe")]

        public ActionResult ExisteNombre(string nombre)  
        {  
            var producto = productosRepository.PorNombre(nombre);  
            var existe = producto == null;  
            return Json(existe, JsonRequestBehavior.AllowGet);  
        }

# AJAX

Eliminando un producto

Request Normal

[HttpPost]  
        public ActionResult Eliminar(int id)  
        {  
            productosRepository.Delete(id);  
            productosRepository.Commit();   
            return RedirectToAction("Index");  
        }

@using (Html.BeginForm("eliminar","home",FormMethod.Post, new { @class="form-eliminar"}))  
{  
 <input type="hidden" name="id" value="@item.Id"/>  
 <a href="#" class="delete-link">Eliminar</a>  
}

.form-eliminar  
{  
 display:inline;  
}

<script type="text/javascript">

$(function () {

$(".delete-link").click(function (e) {

e.preventDefault();

$(this).parent().submit();

});

});

</script>

Ajax

@Html.ActionLink("Eliminar", "Eliminar", new { id = item.Id }, new { @class = "delete-link" })

$(function () {  
        $(".delete-link").click(function (e) {  
            e.preventDefault();  
            var link = $(this);  
            var url = link.attr("href");  
            $.ajax({  
                type: "POST",  
                url: url,  
                success: function (data) {  
                    link.parent().parent().fadeTo(500, 0, function () {  
                        $(this).remove();  
                    });  
                }  
            });  
        });  
    });

if (Request.IsAjaxRequest())   
                return Content("OK");