Hands On Lab: Desarrollo ASP.NET MVC

Contenido

[Introducción 3](#_Toc412473532)

[MVC 3](#_Toc412473533)

[El proyecto: TiendaMVC 4](#_Toc412473534)

[Paso 1: Creación de la solución 4](#_Toc412473535)

[Paso 2: Creación de un modelo 6](#_Toc412473536)

[Paso 3: Los “Scaffoldings” de MVC 5 9](#_Toc412473537)

[Paso 4: Los controladores asíncronos de la Web API. 14](#_Toc412473538)

[Paso 5: Crear un carrito de la compra. 17](#_Toc412473539)

# Introducción

Cuando se lanzó ASP.NET se consiguieron dos objetivos de una sola vez, de un lado se dotó a la que en su momento era la nueva plataforma de desarrollo de Microsoft de una tecnología de desarrollo Web y por otro se modernizó al tiempo que se le daba continuidad a la anterior plataforma de desarrollo, ASP.

ASP.NET como el resto de la plataforma .NET ha experimentado numerosos cambios, y si bien se ha ido integrando con las tendencias del desarrollo Web actual siempre se caracterizó por permitir múltiples enfoques y arquitecturas a la hora de crear aplicaciones, no favoreciendo ninguna, lo que al fin y a la postre se convirtió en según qué contextos en un inconveniente.

La razón es clara, en los entornos de desarrollo empresarial la sistematización, la consistencia arquitectural, la previsibilidad en el enfoque o la facilidad en las pruebas se sitúan por encima de la flexibilidad. ASP.NET nos permitía realizar aplicaciones ajustadas a estos mismos estándares pero nos obligaba a ser nosotros como desarrolladores, los guardianes de estas virtudes.

ASP.NET MVC se postuló como la respuesta desde .NET para lograr las virtudes anteriormente descritas reutilizando en el proceso muchas de las tecnologías creadas para ASP.NET.

## MVC

MVC es un patrón de diseño que postula la separación de los artefactos de una aplicación según su funcionalidad en tres grupos o “capas”, cada una de las cuales se encarga de manera aislada de una determinada tarea que puede ser “usada” por las demás, completando así un sistema que tiene sus límites de un lado en el usuario y de otro en un origen de datos.

Dichas “capas” son:

* Modelo: agrupa a las entidades encargadas de extraer e introducir datos de o hacia un origen de datos.
* Vista: agrupa los artefactos que permiten mostrar de manera comprensible los datos al usuario al tiempo que permiten también recoger los datos que se necesiten para su posterior proceso.
* Controlador: las clases situadas en esta capa se encargan de realizar los procesos que permiten mostrar los datos de un modelo en una vista de manera comprensible y almacenar los datos de una vista en un modelo. También es el lugar apropiado para insertar los procesos de negocio pertinentes que se necesiten en la aplicación.

Las “normas” por tanto con las que se relacionan dichas capas son:

* El controlador manipula el modelo
* El modelo actualiza las vistas
* El usuario ve las vistas
* El usuario usa el controlador

# El proyecto: TiendaMVC

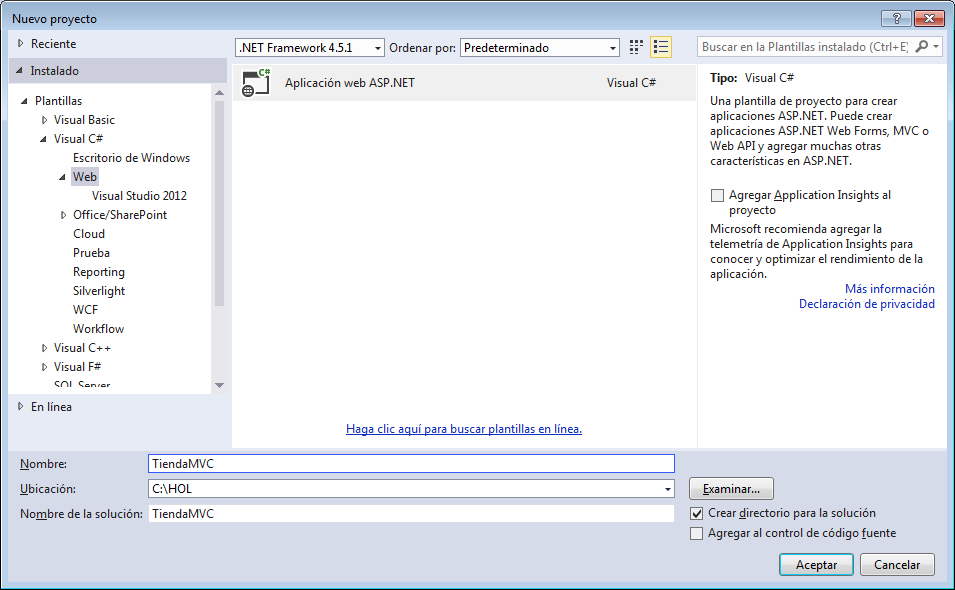
Para la realización de este “Hands on lab” vamos a crear un proyecto ASP.NET MVC que pretende presentar, de manera esquematizada, de una tienda virtual.

Ellos recogeremos las operativas básicas de toda aplicación: presentación y captura de datos, validaciones, seguridad, etc. En ningún caso se pretende asumir los retos reales de una aplicación de tales características y por tanto no se asumirán dichos retos hasta el final, el único objetivo es el de aclarar los procesos básicos del desarrollo en ASP.NET MVC.

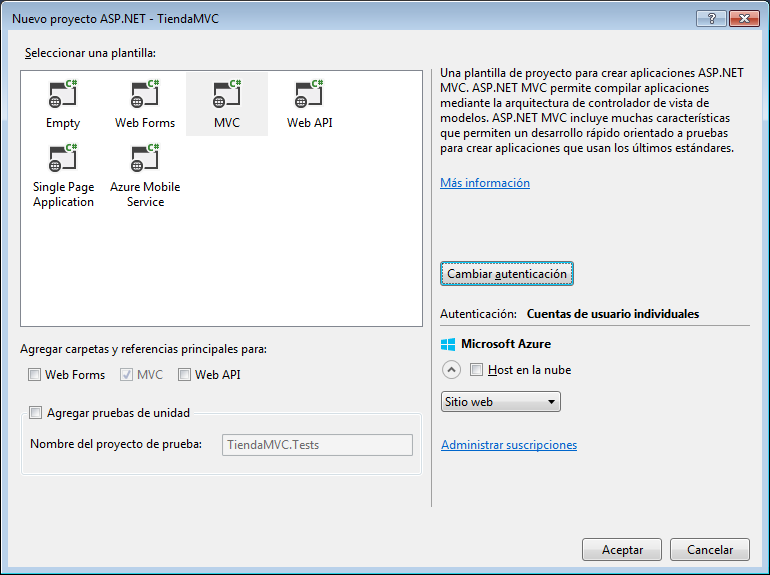
Por otro lado tampoco se asumirán retos que de manera funcional son muy importantes en este tipo de proyectos, como puede ser el proceso de pago, pero que quedan fuera del alcance de este proyecto.

## Paso 1: Creación de la solución

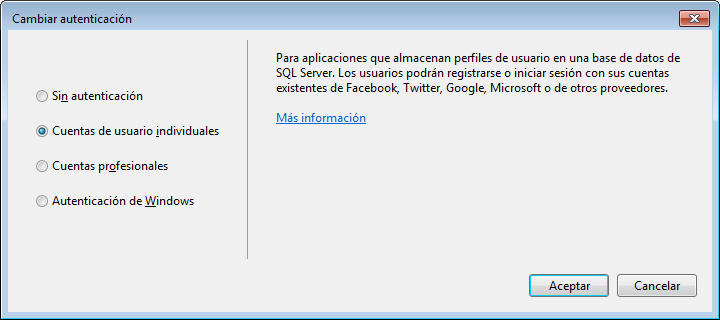
1. Abrir Visual Studio 2013
2. En el menú de *Archivo -> Nuevo* pulsar sobre la opción de *Proyecto…*
3. En la ventana emergente seleccionar dentro del conjunto de plantillas *Instalado* la rama en *Visual C# -> Web*, la plantilla *Aplicación Web ASP.NET* con el nombre “*TiendaMVC*” y situándola en la Ubicación *C:\HOL.*



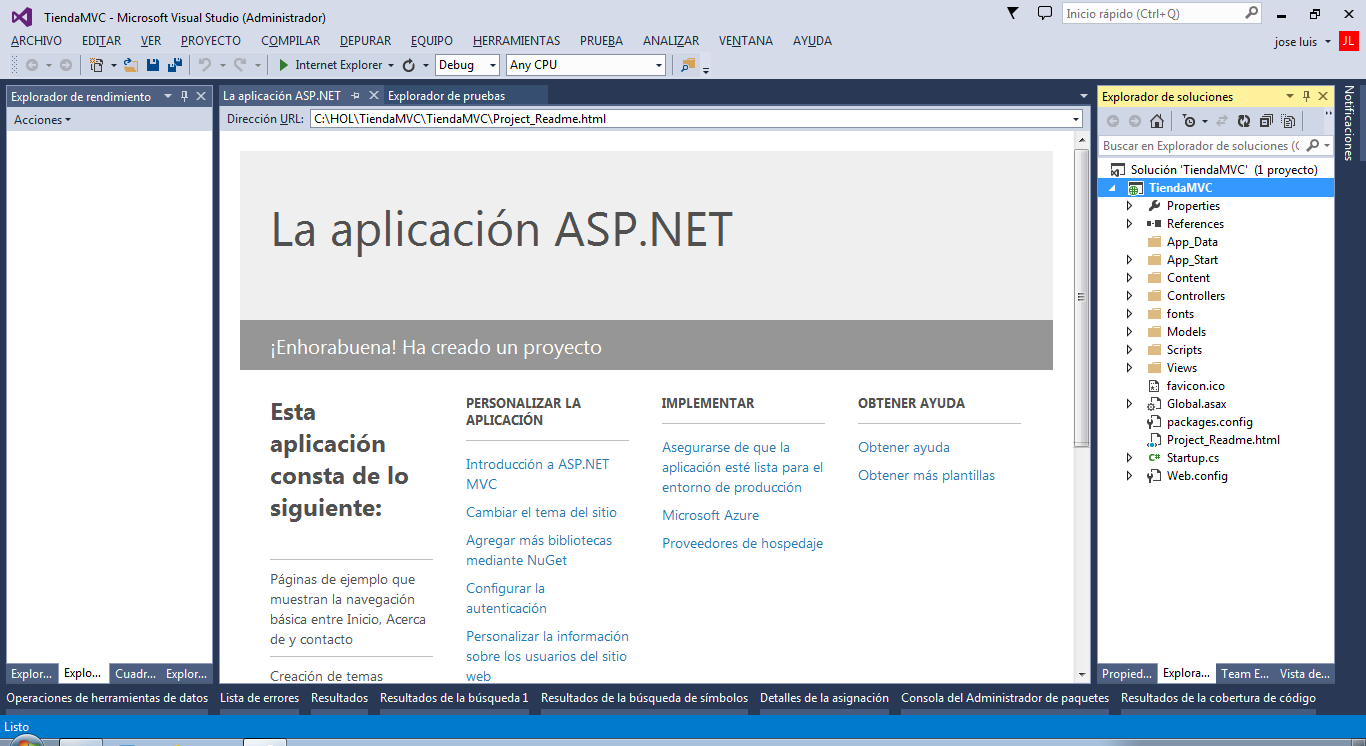
1. Pulsar el botón Aceptar.
2. En la ventana emergente pulsar sobre el botón cambiar autenticación



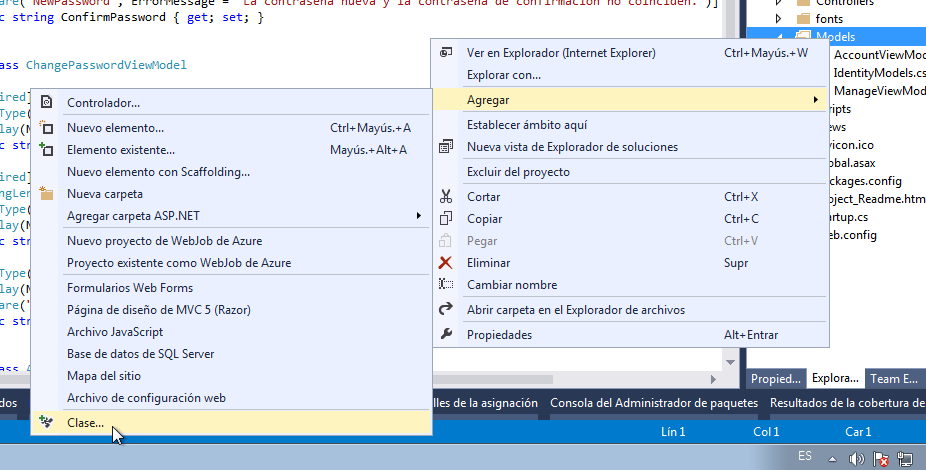
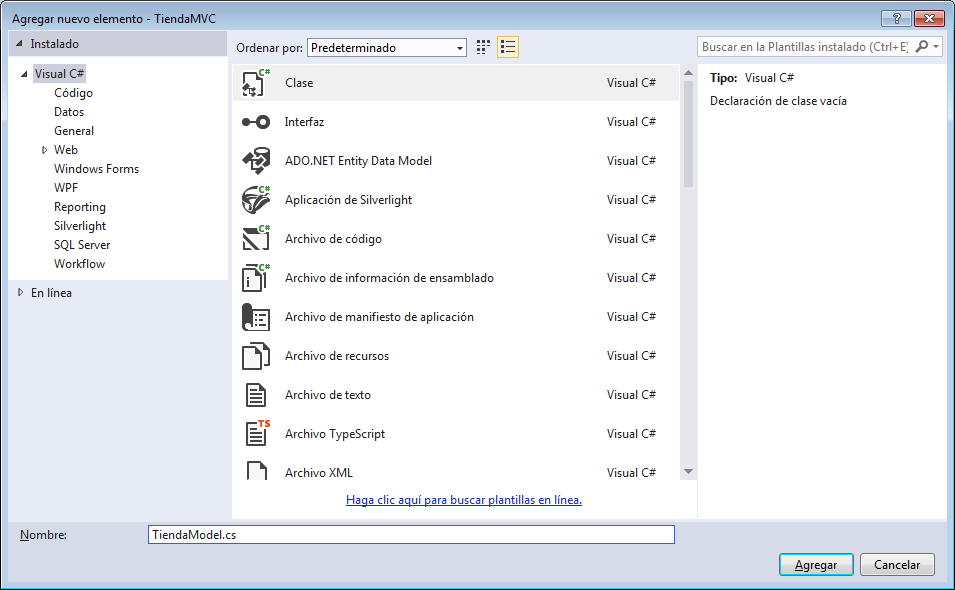
1. En la ventana emergente asegurar que la opción *Cuentas de usuario individuales* está seleccionada.



1. Pulsar el botón Aceptar.
2. Pulsar el botón Aceptar en la ventana de Nuevo Proyecto para crear el proyecto*.*



## Paso 2: Creación de un modelo

1. En el explorador de soluciones localizar el nodo Models
2. Hacer clic derecho sobre él, en el menú emergente seleccionar Agregar->clase…
3. En la ventana de diálogo insertar el nombre Line.cs. 
4. Sustituir el código para la unidad recien creada por el siguiente:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace TiendaMVC.Models

{

public class Line

{

public int Id { get; set; }

public int OrderId { get; set; }

public Order Order { get; set; }

public int ProductId { get; set; }

public Product Product { get; set; }

public int Units { get; set; }

public Decimal Price { get; set; }

}

}

1. Hacer clic derecho sobre él, en el menú emergente seleccionar Agregar->clase…
2. En la ventana de diálogo insertar el nombre Product.cs.
3. Sustituir el código para la unidad recien creada por el siguiente:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace TiendaMVC.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public string Image { get; set; }

public decimal Price { get; set; }

public ICollection<Line> Lines { get; set; }

}

}

1. Hacer clic derecho sobre él, en el menú emergente seleccionar Agregar->clase…
2. En la ventana de diálogo insertar el nombre Order.cs.
3. Sustituir el código para la unidad recien creada por el siguiente:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace TiendaMVC.Models

{

public class Order

{

public int Id { get; set; }

public DateTime Date { get; set; }

public string UserId { get; set; }

public ICollection<Line> Lines { get; set; }

}

}

1. Hacer clic derecho sobre él, en el menú emergente seleccionar Agregar->clase…
2. En la ventana de diálogo insertar el nombre StoreModel.cs.
3. Sustituir el código para la unidad recien creada por el siguiente:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace TiendaMVC.Models

{

public class Order

{

public int Id { get; set; }

public DateTime Date { get; set; }

public string UserId { get; set; }

public ICollection<Line> Lines { get; set; }

}

}

1. Abrir la unidad IdentityModels.cs y agregar a la clase ApplicationUser el siguiente pedazo de código:

C#

[Required, Display(Name="Nombre del usuario")]

public string LongName { 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 reclamaciones de usuario personalizado aquí

return userIdentity;

}

public static ApplicationUser GetUserById(string Id){

Task<ApplicationUser> t = ApplicationDbContext.Create().Users.FirstOrDefaultAsync(x => x.Id == Id);

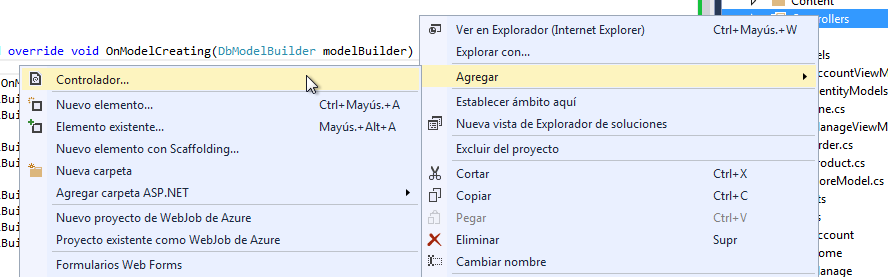
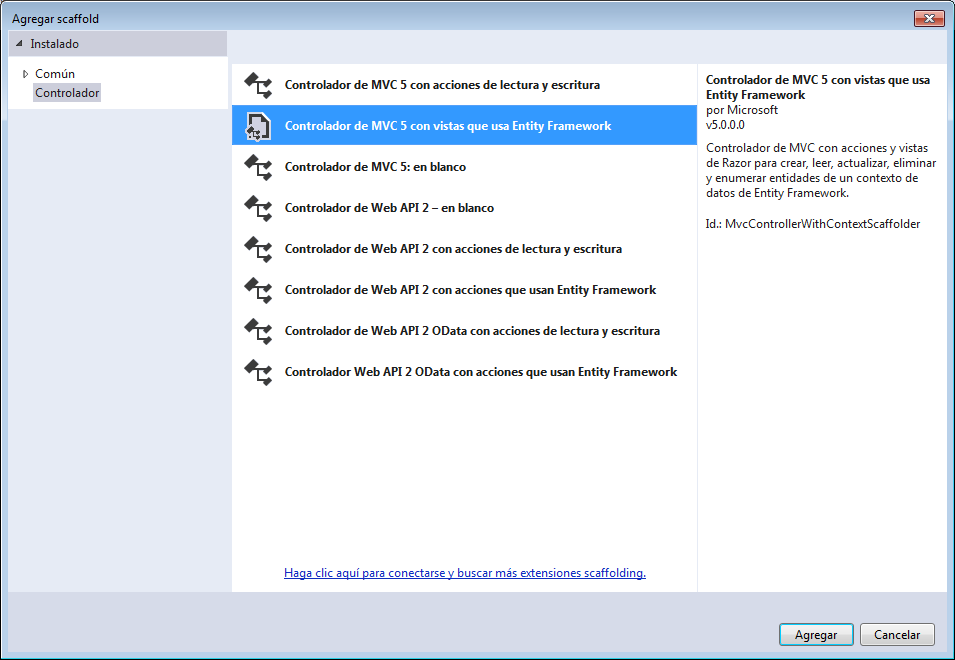
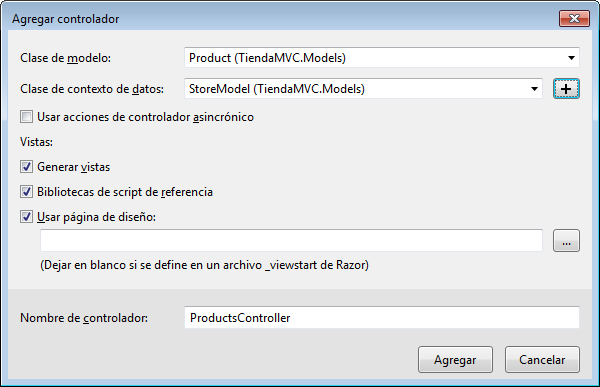
Task.WaitAll(t);

return t.Result;

}

1. Compilar la solución pulsando Crtl + Shift + B o bien haciendo clic derecho sobre la solución y seleccionando el menú Generar.

## Paso 3: Los “Scaffoldings” de MVC 5

1. En el Explorador de soluciones localizar el nodo Controllers y hacer clic derecho sobre él.
2. Seleccionar el menú Agregar->Controlador…
3. En la ventana emergente seleccionar la plantilla “Controlador de MVC r con vistas que usa Entity Framework” y pulsar el botón agregar. 
4. En la ventana agregar controlador seleccionar la clase Product como Clase de Modelo, StoreModel como Clase de contexto de datos y pulsar el botón Agregar. 
5. Localizar la unidad ProductsController dentro de la carpeta “Controllers” y sustituir su código por el siguiente:

C#

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using TiendaMVC.Models;

namespace TiendaMVC.Controllers

{

public class ProductsController : Controller

{

private StoreModel db = new StoreModel();

// GET: Products

public ActionResult Index()

{

return View(db.Products.ToList());

}

// GET: Products/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Product product = db.Products.Find(id);

if (product == null)

{

return HttpNotFound();

}

return View(product);

}

// GET: Products/Create

public ActionResult Create()

{

return View();

}

// POST: Products/Create

// Para protegerse de ataques de publicación excesiva, habilite las propiedades específicas a las que desea enlazarse. Para obtener

// más información vea http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "Id,Name,Image,Price")] Product product)

{

if (ModelState.IsValid)

{

db.Products.Add(product);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(product);

}

// GET: Products/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Product product = db.Products.Find(id);

if (product == null)

{

return HttpNotFound();

}

return View(product);

}

// POST: Products/Edit/5

// Para protegerse de ataques de publicación excesiva, habilite las propiedades específicas a las que desea enlazarse. Para obtener

// más información vea http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "Id,Name,Image,Price")] Product product)

{

if (ModelState.IsValid)

{

db.Entry(product).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(product);

}

// GET: Products/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Product product = db.Products.Find(id);

if (product == null)

{

return HttpNotFound();

}

return View(product);

}

// POST: Products/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Product product = db.Products.Find(id);

db.Products.Remove(product);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

1. A continuación localizar dentro de la carpeta Views\Products la unidad de código Create.cshtml y sustituir su código por el siguiente:

C#

@model TiendaMVC.Models.Product

@{

ViewBag.Title = "Create";

}

<h2>Create</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Product</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Image, htmlAttributes: new { @class = "control-label col-md-2" })

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

<div >

<img alt="Imagen del Producto" id="thumbnail" style="height:100px;width:100px" class="thumbnail" />

<input type="button" value="Seleccionar Imagen" id="selectImage" />

<input type="file" id="imageDialog" style="display:none"/>

</div>

@Html.HiddenFor(model => model.Image, new { htmlAttributes = new { @class = "form-control" } })

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Price, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.Price, new { htmlAttributes = new { @class = "form-control" } })

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

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-default" />

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

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

<script>

$(function () {

$("#imageDialog").change(function (ele) {

alert(ele.file);

var reader = new FileReader();

reader.onload = function (e) {

$("#thumbnail")[0].src = reader.result;

$("#Image").val(reader.result);

}

reader.readAsDataURL(ele.target.files[0]);

});

$("#selectImage").click(function () { $("#imageDialog").click(); });

})

</script>

}

1. En esa misma carpeta editar la unidad Index.cshtml y sustituir la línea:

C#

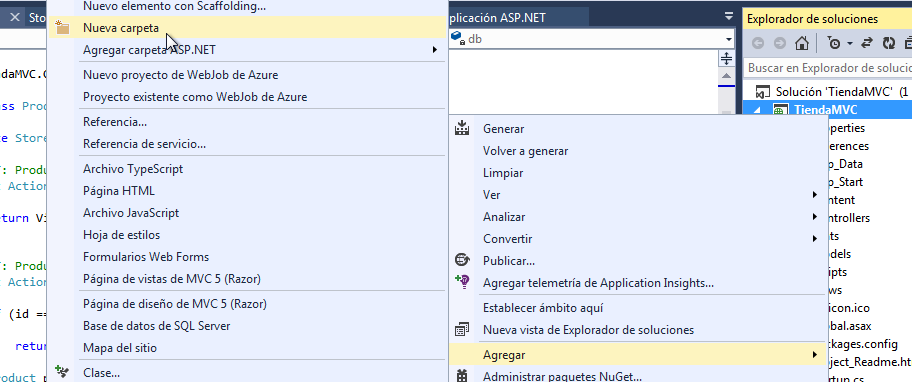
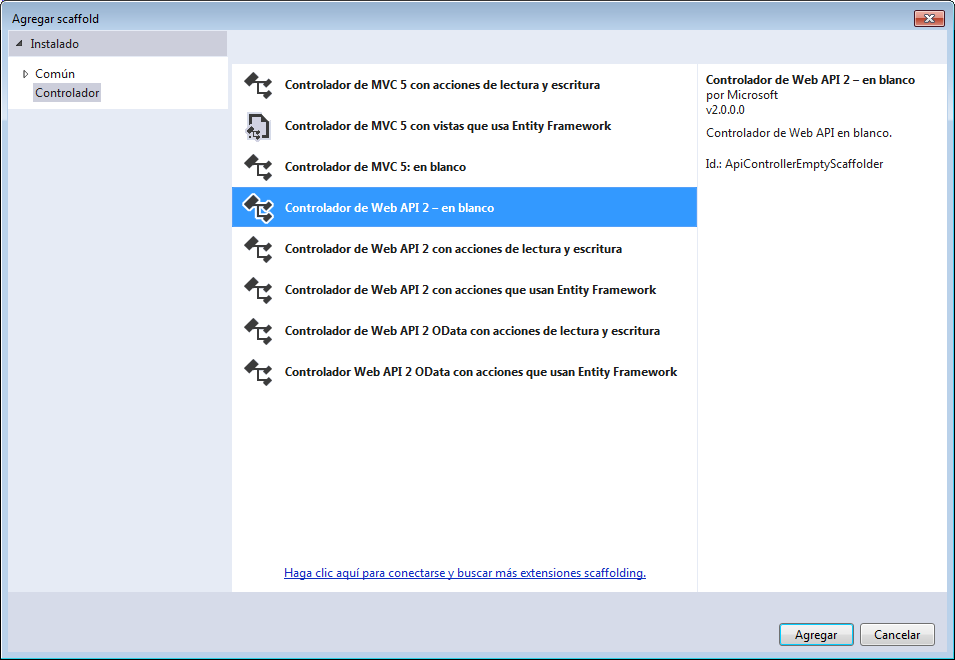
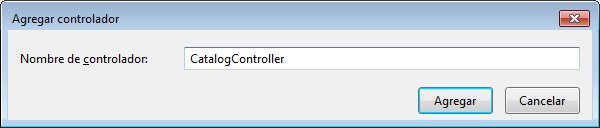
@Html.DisplayFor(modelItem => item.Image)

1. Por:

C#

<img alt="@item.Name" src="@item.Image" />

## Paso 4: Los controladores asíncronos de la Web API.

1. En el explorador de soluciones hacer clic derecho sobre el nombre del proyecto y seleccionar el menú Agregar-> Nueva carpeta
2. Dar el nombre images a la carpeta recién creada y descomprimir el contenido del archivo images.zip adjunto a este documento.
3. Agregar un nuevo controlador al proyecto, en este caso seleccionando la plantilla Controlador de Web API 2 – en blanco
4. Pulsar Agregar
5. En la ventana emergente otorgar el nombre CatalogController al nuevo controlador 
6. Localizar la unidad de código CatalogController.cs y sustituir el código por:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using TiendaMVC.Models;

namespace TiendaMVC.Controllers

{

public class CatalogController : ApiController

{

public IEnumerable<Product> Get(int pageNumber)

{

using (StoreModel model = new StoreModel())

{

return model.Products.Take(6 \* pageNumber).ToList().Skip(6 \* (pageNumber - 1)).Take(6);

}

}

public int GetTotalPages()

{

using (StoreModel model = new StoreModel())

{

int elements = model.Products.Count();

int total = elements % 6 == 0 ? elements / 6 : (elements / 6) + 1;

return total;

}

}

[HttpPost]

public void AddProductToCart(int productId)

{

var session = System.Web.HttpContext.Current.Session;

List<int> products = null;

if (session["cart"] != null)

products = (List<int>)session["cart"];

else

products = new List<int>();

products.Add(productId);

session["cart"] = products;

}

}

}

1. Una vez creada abrir el archivo Global.asax agregar las siguientes referencias de espacios de nombres:

C#

using System.Web.Http;

using System.Web.Routing;

1. Localizar en dicho archivo el método Application\_Start() y situar como primera línea del método lo siguiente:

C#

GlobalConfiguration.Configure(WebApiConfig.Register);

1. Tras dicho método pero dentro de la clase MVCApplication añadir los siguientes miembros:

C#

protected void Application\_PostAuthorizeRequest()

{

if (IsWebApiRequest())

{

HttpContext.Current.SetSessionStateBehavior(SessionStateBehavior.Required);

}

}

private bool IsWebApiRequest()

{

return HttpContext.Current.Request.AppRelativeCurrentExecutionFilePath.StartsWith("~/api");

}

1. Localizar la vista Index.cshtml y sustituir su código por los siguiente:

C#

@{

ViewBag.Title = "Home Page";

}

<div class="jumbotron">

<img src="~/Images/DanysoftStore.png" />

<p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>

</div>

<div class="container">

<div id="catalog" class="row">

</div>

</div>

<div class="container">

<ul id="paginator" class="pagination">

</ul>

<hr />

<footer>

<p>&copy; Danysoft Internacional 2014</p>

</footer>

</div>

@section Scripts {

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

<script>

$(function () {

$.getJSON('/api/catalog/?pageNumber=1',

function(data){

createWindow(data);

});

$.getJSON('/api/catalog/',

function (data) {

createPaginator(data);

});

});

function addTile(destElem, element) {

var html = '<div class="col-md-2">' +

'<h2 style="height:66px">' + element.Name + '</h2>' +

'<img src="' + element.Image + '" />' +

'<p>Precio: ' + element.Price + ' €</p>' +

'<p><a class="btn btn-default" href="#" role="button" productId="' + element.Id + '">Agregar al carrito</a></p>' +

'</div>';

$(destElem).append(html);

}

function createWindow(data) {

$(data).each(function (index, element) {

addTile('#catalog', element);

});

$('a[productId]').click(function (ev) {

var productId = $(ev.target).attr('productId');

$.post('/api/catalog/?productId=' + productId);

});

};

function createPaginator(data) {

var pages = parseInt(data);

$('#paginator').append('<li><a href="#" page="1">&laquo;</a></li>');

for (var i = 1; i <= pages; i++) {

$('#paginator').append('<li><a href="#" page="' + i.toString() + '">' + i.toString() + '</a></li>');

}

$('#paginator').append('<li><a href="#" page="' + pages + '">&raquo;</a></li>');

$('a[page]').click(function (ev) {

var page = $(ev.target).attr('page');

$('#catalog').empty();

$.getJSON('/api/catalog/?pageNumber=' + page,

function(data) {

createWindow(data);

});

$('#paginator').empty();

$.getJSON('/api/catalog/',

function (data) {

createPaginator(data);

});

});

}

</script>

}

## Paso 5: Crear un carrito de la compra.

1. Crear un nuevo Controlador de MVC 5 en blanco con el nombre PaymentController, sustituyendo el código del método Index por el siguiente:

C#

[Authorize()]

public ActionResult Index()

{

var session = System.Web.HttpContext.Current.Session;

using (StoreModel model = new StoreModel())

{

if (session["cart"] != null)

{

var cartProducts = (List<int>)session["cart"];

return View( model.Products.Join(cartProducts, p => p.Id, cp => cp, (p, cp) => p).ToList());

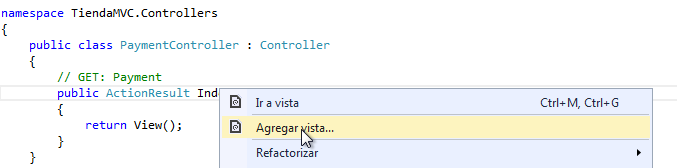
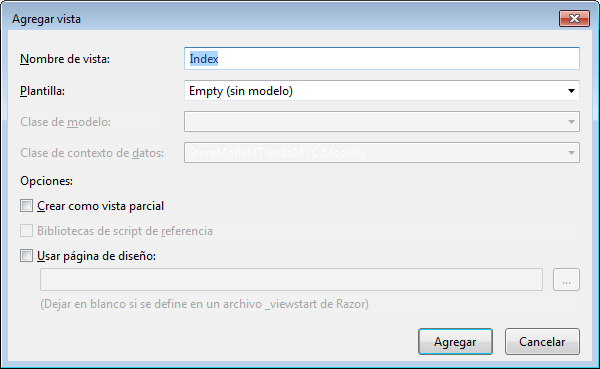
}

else

return View();

}

}

1. Hacer clic derecho sobre el nombre del método Index y del menú emergente seleccionar la opción Agregar Vista 
2. En la ventana de diálogo quitar la marca sobre seleccionar usar página de diseño y pulsar el botón agregar: 
3. Sustituir el código de la vista Index.cshtml en la carpeta Views\Payment por lo siguiente:

C#

@{

ViewBag.Title = "Home Page";

}

<div class="jumbotron">

<img src="~/Images/DanysoftStore.png" />

</div>

<div class="container">

<table id="cart" class="table">

<thead>

<tr>

<th>

Producto

</th>

<th>

Precio

</th>

</tr>

</thead>

<tbody>

@foreach(TiendaMVC.Models.Product ele in Model) {

<tr>

<td>

<img alt="Producto" style="width:5em;" src="@ele.Image" />

@ele.Name

</td>

<td>

@ele.Price<text>€</text>

</td>

</tr>

}

</tbody>

</table>

</div>

<div class="container">

<hr />

<footer>

<p>&copy; Danysoft Internacional 2014</p>

</footer>

</div>

1. Localizar vista \_Layout.cshtml dentro de Views\Shared y sustituir su contenido por:

C#

<!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 aplicación", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Carrito", "Index", "Payment")</li>

</ul>

@Html.Partial("\_LoginPartial")

</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>

1. Localizar vista \_LoginPartial.cshtml dentro de Views\Shared y sustituir su contenido por:

C#

@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 " + TiendaMVC.Models.ApplicationUser.GetUserById(User.Identity.GetUserId()).LongName + "!", "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("Registrarse", "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink" })</li>

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

</ul>

}

1. Localizar la unidad AccountViewModels.cs dentro de la carpeta models y dentro de ella localizar la clase RegisterViewModel y agregar la siguiente propiedad:

C#

[Required]

[Display(Name = "Nombre completo")]

public string LongName { get; set; }

1. Por último localizar la unidad de Register.cshtml en la carpeta Views\Account y tras la línea :

@Html.ValidationSummary("", new { @class = "text-danger" })

1. Agregar lo siguiente

<div class="form-group">

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

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

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

</div>

</div>