

Supuesto práctico integrador

Desarrollo de una aplicación Web interactiva de tienda online

Objetivos

- Integrar y aplicar los conocimientos adquiridos en el módulo de Desarrollo de Aplicaciones Web en el Entorno del Servidor del ciclo formativo de Desarrollo de Aplicaciones Web.
- Abordar y solucionar los problemas tecnológicos que comporta completar el desarrollo de una aplicación Web interactiva con acceso a datos, empleando la tecnología ASP.NET Core MVC.
- Comprender el proceso del desarrollo de una aplicación Web con un alto nivel de acabado.
- Experimentar, indagar y profundizar sobre diversos aspectos técnicos de detalle relacionados con el desarrollo de aplicaciones Web interactivas y su integración con bases de datos.

Enunciado

Se desea construir una aplicación Web interactiva para la venta online de productos. Esta aplicación Web se construirá utilizando la tecnología de desarrollo en entorno del servidor ASP.NET Core MVC y el servidor de base de datos SQL Server para resolver el almacenamiento de datos. La construcción de la aplicación Web se realizará de modo que se obtenga el mayor nivel de acabado posible.

La aplicación web interactiva de tienda online que se va a construir, deberá quedar adaptada a la venta de un tipo de productos concreto, por ejemplo: artículos de ferretería, prendas de vestir, videojuegos, vinos, bisutería, material deportivo, etc. Por este motivo, cada alumno deberá elegir el tipo de productos que se venderán en la tienda online que vaya a desarrollar. Dado que el tipo de productos que se venderán es de **contenido libre**, convendrá previamente consultar la idea a desarrollar con el profesor para verificar la idoneidad de los objetivos a alcanzar.

La aplicación Web resultante deberá permitir el acceso a los procesamientos de gestión a realizar por los usuarios comunes de la aplicación Web, así como a los procesamientos de administración o gestión avanzada. De modo que se definirán, al menos, dos perfiles o casos de uso: el perfil de Usuario y el perfil de Administrador. Según el perfil del usuario autenticado, se accederá a una parte u otra de la aplicación Web. Los usuarios comunes accederán al menú de usuario, dónde estarán disponibles las opciones de acceso a los procesamientos de compra y gestión de pedidos del usuario. Los administradores de la aplicación Web accederán al menú de administración que incluirá las opciones para el mantenimiento de la información y los procesamientos de gestión avanzada (*Back-office*).

Desarrollo

Para abordar la construcción de la aplicación Web de tienda online, que deberá completarse con un alto nivel de acabado, se propone la realización de las siguientes tareas:

- Seleccionar el tipo de productos que se venderán a través de la aplicación Web de tienda online y consultarlo con el profesor. Además, elegir un nombre para la aplicación Web.
- Crear la aplicación Web basada en ASP.NET Core MVC con autenticación de Cuentas de usuario individuales.

- Crear el Modelo, teniendo en cuenta los requerimientos de la tienda online concreta a desarrollar en cada caso. Para ello, se puede considerar el ejemplo genérico para tienda online de las clases de datos y de la clase del contexto de datos que se incluye en el anexo.
- Registrar el contexto de datos en el archivo *Startup.cs* para asignarle la cadena de conexión **DefaultConnection**. Además, comprobar que la cadena de conexión **DefaultConnection** está especificada en el archivo *appsettings.json*.
- Realizar una migración inicial para crear la base de datos y comprobar que se ha creado.
- Crear los controladores, y sus vistas asociadas, para generar los procesamientos de tipo CRUD de cada una de las tablas de la base de datos que maneja la aplicación Web.
- Crear las opciones de menú correspondientes para cada uno de los controladores creados.
- Introducir y almacenar información suficiente en cada una de las tablas para poder realizar las pruebas correspondientes. Se recomienda introducir 10 filas en cada tabla como mínimo.


Construcción del escaparate

- Crear un nuevo controlador denominado **Escaparate**. A continuación, añadir la acción *Index()*, y la vista correspondiente, para implementar la funcionalidad de mostrar la lista de productos del escaparate de la tienda, de modo que se obtenga un resultado similar al siguiente.

MvcTienda Home Privacy Pedidos Productos Categorías Clientes Estados Escaparate Register Login

Escaparate


[Climatización](#)
[Electrodomésticos](#)
[Herramientas](#)
[Herramientas eléctricas](#)
[Iluminación](#)
[Instrumentos de medida](#)
[Maquinaria](#)
[Material de redes informáticas](#)
[Material eléctrico](#)



Rack pequeño 50 cm. con alimentación

83,54 €


Añadir al Carrito



Pack de 4 bombillas led de 5 W. E27

15,83 €


Añadir al Carrito



Lámpara de noche SUEÑOS

20,65 €


Añadir al Carrito



Rack mediano 100 cm

268,35 €

Añadir al Carrito



Lámpara de techo LUNA. Diseño actual

125,50 €

Añadir al Carrito

Vicente Aracil Miralles <varacil@iesmarenostrum.com>

Página 2

La acción *Index()* del controlador *Escaparate* recibe como parámetro el valor del Id de la categoría de productos que se desea mostrar. Si el valor del Id recibido como parámetro es *null*, entonces se mostrarán los productos del escaparate, es decir, aquellos que tienen el valor *true* en la propiedad *Producto.Escararate*. Para pasar la lista de las categorías a la vista desde el modelo se empleará el objeto *ViewData*, una vez obtenida la lista de categorías ordenada en el modelo. En la vista, se presentarán los enlaces correspondientes a cada categoría en una tabla, de modo que se pueda seleccionar una categoría concreta para ver los productos que correspondan a esa categoría. Añadir la opción “Escaparate” en el menú de la aplicación Web.

- Incorporar el sistema de autenticación y autorización de usuarios basado en *ASP.NET Core Identity* y enlazar la información de los usuarios registrados con la información almacenada en la base de datos que maneja la aplicación Web, a través del correo electrónico del usuario y del cliente. Para ello, desarrollar el controlador *MisDatos* y las acciones *GET Create()* y *POST Create()* para almacenar los datos del cliente correspondiente al usuario actual, en caso de que estos datos no estén ya almacenados. A continuación, desarrollar el procesamiento para modificar los datos del cliente correspondiente al usuario actual, añadiendo para ello los métodos *GET Edit()* y *POST Edit()* correspondientes en el controlador *MisDatos*.
- Establecer el menú de la aplicación Web según el rol del usuario que accede a la aplicación Web, de acuerdo con la siguiente tabla.

Opción de menú	Rol del usuario	Controlador	Método de acción
Inicio	Cualquiera	<i>Home</i>	<i>Index()</i>
Privacidad	Cualquiera	<i>Home</i>	<i>Privacy()</i>
Estados	Administrador	<i>Estados</i>	<i>Index()</i>
Categorías	Administrador	<i>Categorias</i>	<i>Index()</i>
Productos	Administrador	<i>Productos</i>	<i>Index()</i>
Clientes	Administrador	<i>Clientes</i>	<i>Index()</i>
Pedidos	Administrador	<i>Pedidos</i>	<i>Index()</i>
Escaparate	Usuario	<i>Escaparate</i>	<i>Index()</i>
Mis Datos	Usuario	<i>MisDatos</i>	<i>Edit()</i>

- Teniendo en cuenta la tabla anterior, especificar el filtro *Autorize* en los controladores y/o en las acciones que corresponda, de manera que se puedan evitar los accesos indebidos.

Construcción del carrito de la compra

- Implementar la acción *GET AñadirCarrito()* y *POST AñadirCarrito()* en el controlador *Escaparate* para añadir un producto al carrito de la compra desde el escaparate, a través del botón “Añadir al Carrito” de cada producto. Se recibe como parámetro el valor del Id del producto a añadir. La acción *GET* mostrará los datos del producto a añadir y un botón de confirmación que iniciará la acción *POST*. Si se trata del primer producto que se añade al carrito, entonces debe crearse un nuevo pedido. Al crear un nuevo pedido, se asignará el valor “Pendiente” al estado del pedido y se guardará el número del pedido en una variable de sesión, denominada *NumPedido*.
- Crear un nuevo controlador denominado **Carrito**. A continuación, añadir la acción *Index()*, y la vista correspondiente, para implementar la funcionalidad que permita ver el carrito de la compra del pedido actual, a través de un procesamiento de tipo maestro-detalle, tal como se muestra en la siguiente ilustración.

Carrito de la compra

Pedido

Núm. Pedido 11
Fecha 04/01/2021
Confirmado
Preparado
Enviado
Cobrado
Devuelto
Anulado
Cliente Maria Terol Lloisà
Estado Pendiente

	Id Producto	Descripción del producto	Cantidad	Precio	Descuento	Total	
	6	Pack de 4 bombillas led de 5 W. E27	1 	15,83	0,00	15,83	Eliminar
	7	Lámpara de techo LUNA. Diseño actual	 2 	125,50	0,00	251,00	Eliminar
						Total	266,83

[Confirmar Pedido >](#)

[Seguir comprando](#)

En la ilustración anterior, puede observarse que el carrito de la compra muestra los datos del pedido actual en la cabecera y los datos del detalle del pedido en las líneas que corresponden con los productos que se están comprando en el pedido actual. El enlace "Seguir comprando" redirige a la acción *Index()* del controlador *Escaparate*. Para facilitar el desarrollo, se utiliza una variable de sesión, denominada *NumPedido*, para guardar el valor del pedido que se encuentra actualmente en el carrito de la compra. Si el valor de esta variable de sesión es *null*, entonces el carrito está vacío. Añadir la opción "Carrito" en el menú de usuario de la aplicación Web.

- En el controlador *Carrito*, implementar las siguientes acciones para resolver las diversas opciones disponibles en el carrito y que pueden observarse sobre la ilustración anterior.
 - Acción *ConfirmarPedido()*. Implementa la funcionalidad para confirmar la compra en firme del pedido actual que se encuentra en el carrito de la compra. Se recibe como parámetro el valor del Id del Pedido a confirmar. Si el carrito está vacío, entonces no puede confirmarse el pedido. Al confirmar el pedido actual, se modifica el valor del estado del pedido al valor "Confirmado" y se almacena la fecha en la que el pedido fue confirmado. Una vez confirmado el pedido, entonces el valor de la variable de sesión *NumPedido* se pone a *null*. Al finalizar, se redirige la ejecución hacia ver el escaparate.
 - Acción *CarritoVacio()*. Implementa la funcionalidad para mostrar una página con un texto que indica que el carrito está vacío, lo que ocurre cuando el valor de la variable de sesión *NumPedido* es *null* y, por tanto, no existe ningún producto añadido al carrito.

- Acción *EliminarLinea()*. Implementa la funcionalidad para eliminar una línea de detalle del carrito. Se recibe como parámetro el valor del Id del detalle de la línea que se desea eliminar. Al finalizar, se redirige la ejecución hacia ver el carrito.
- Acción *MasCantidad()*. Implementa la funcionalidad de modificar el detalle del producto actual para incrementar la cantidad en una unidad de compra. Se recibe como parámetro el valor del Id del Detalle de la línea que se desea modificar para aumentar la cantidad a comprar. Al finalizar, redirige la ejecución hacia ver el carrito.
- Acción *MenosCantidad()*. Implementa la funcionalidad de modificar el detalle del producto actual para decrementar la cantidad en una unidad, si el valor de la cantidad es mayor que 1. Se recibe como parámetro el valor del Id del Detalle de la línea que se desea modificar. Al finalizar, redirige la ejecución hacia ver el carrito.
- Establecer adecuadamente el filtro *Autorize* en los controladores *Escaparte* y *Carrito*.

Desarrollo opcional

Opcionalmente, se propone la construcción de aspectos avanzados de la aplicación Web, que deberán completarse, igualmente, con un alto nivel de acabado.

- Construir el controlador **MisPedidos**, y las vistas asociadas, para poder gestionar los pedidos correspondientes al usuario actual.
- Implementar la opción “Detalles” en el controlador **MisPedidos** para mostrar los datos del pedido, así como las líneas del pedido a través de un procesamiento de tipo maestro-detalle.
- Añadir la opción “Mis Pedidos” en el menú de usuario.
- Establecer el filtro *Autorize* sobre el controlador **MisPedidos** de la manera más adecuada.

Entrega

La entrega del supuesto práctico integrador se realizará en la sesión de clase asignada mediante presentación y demostración pública de la aplicación Web desarrollada antes del **25/02/2022**.

Calificación

Como síntesis del aprovechamiento docente, la superación del supuesto práctico integrador se basa en alcanzar un alto nivel de acabado de la funcionalidad de la aplicación Web planteada. Los criterios de calificación a aplicar se basan en la valoración de los siguientes aspectos:

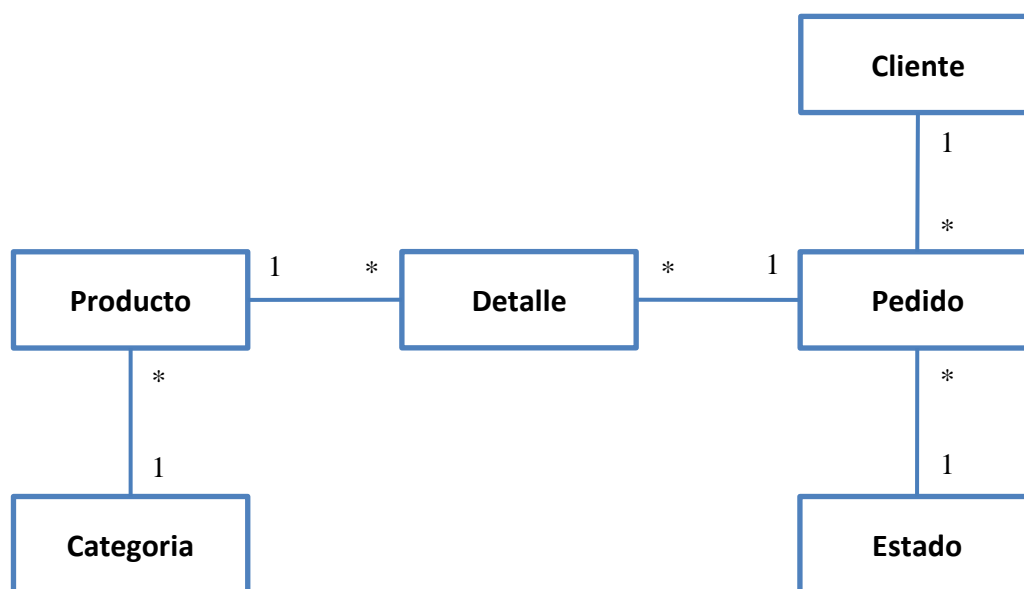
Criterio de calificación	Valoración
Nivel de acabado y perfección alcanzado de la implementación y presentación visual	35 %
Grado de dificultad de la aplicación Web en relación con los objetivos planteados	35 %
Adecuación y coherencia de la exposición oral en la presentación y demostración	30 %

Las máximas calificaciones se obtendrán si se completan, con un alto nivel de acabado, también las tareas de desarrollo opcional. El supuesto práctico integrador es una actividad de enseñanza y aprendizaje de carácter individual. El trabajo desarrollado deberá ser original por parte del alumno/a que lo realiza. En caso de que no fuera original, la calificación obtenida será la menor posible.

Anexo. Ejemplo de Modelo de datos

A continuación, se incluye un ejemplo de modelado de datos de una tienda online genérica que puede ayudar a definir de manera adecuada las clases de datos y la clase del contexto de datos en cada caso concreto. Será necesario ajustar la definición de las clases que forman el modelo, así como las especificaciones de validación de las propiedades, considerando las funcionalidades concretas que se requieran y las características del tipo de productos concreto que vaya a ser vendido a través de la aplicación de tienda online a desarrollar en cada caso.

- Modelado de datos



- Clases de datos

```
public class Cliente
{
    public int Id { get; set; }
    [Required(ErrorMessage = "El nombre del cliente es un campo requerido.")]
    public string Nombre { get; set; }
    public string Email { get; set; }
    public string Telefono { get; set; }
    public string Direccion { get; set; }
    public string Poblacion { get; set; }
    public stringCodigoPostal { get; set; }
    public string Nif { get; set; }

    public ICollection<Pedido> Pedidos { get; set; }
}
```

```
public class Estado
{
    public int Id { get; set; }
    [Display(Name = "Descripción")]
    [Required(ErrorMessage = "La descripción es un campo requerido.")]
    public string Descripcion { get; set; }

    public ICollection<Pedido> Pedidos { get; set; }
}
```

```
public class Pedido
{
    [Display(Name = "Núm. Pedido")]
    public int Id { get; set; }
    [Required(ErrorMessage = "La fecha del pedido es un campo requerido.")]
    [DataType(DataType.Date)]
    public DateTime Fecha { get; set; }
    [DataType(DataType.Date)]
    public DateTime? Confirmado { get; set; }
    [DataType(DataType.Date)]
    public DateTime? Preparado { get; set; }
    [DataType(DataType.Date)]
    public DateTime? Enviado { get; set; }
    [DataType(DataType.Date)]
    public DateTime? Cobrado { get; set; }
    [DataType(DataType.Date)]
    public DateTime? Devuelto { get; set; }
    [DataType(DataType.Date)]
    public DateTime? Anulado { get; set; }
    public int ClienteId { get; set; }
    public int EstadoId { get; set; }

    public Cliente Cliente { get; set; }
    public Estado Estado { get; set; }
    public ICollection<Detalle> Detalles { get; set; }
}
```

```
public class Detalle
{
    public int Id { get; set; }
    [Display(Name = "Núm. Pedido")]
    public int PedidoId { get; set; }
    [Display(Name = "Id. Producto")]
    public int ProductoId { get; set; }
    public int Cantidad { get; set; }
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Precio { get; set; }
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Descuento { get; set; }

    public virtual Pedido Pedido { get; set; }
    public virtual Producto Producto { get; set; }
}
```



```
public class Producto
{
    public int Id { get; set; }
    [Display(Name = "Descripción")]
    [Required(ErrorMessage = "La descripción es un campo requerido.")]
    public string Descripcion { get; set; }
    public string Texto { get; set; }
    [DisplayFormat(DataFormatString = "{0:n2}")]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Precio { get; set; }
    [Display(Name = "Precio")]
    [RegularExpression(@"^-0123456789]+[0-9.]*$",
        ErrorMessage = "El valor introducido debe ser de tipo monetario.")]
    [Required(ErrorMessage = "El precio es un campo requerido")]
    public string PrecioCadena
    {
        get
        {
            return Convert.ToString(Precio).Replace(',', '.');
        }
        set
        {
            Precio = Convert.ToDecimal(value.Replace('.', ','));
        }
    }
    public int? Stock { get; set; }
    public bool? Escaparate { get; set; }
    public string Imagen { get; set; }
    public int CategoriaId { get; set; }

    public Categoria Categoria { get; set; }
    public ICollection<Detalle> Detalles { get; set; }
}
```

```
public class Categoria
{
    public int Id { get; set; }
    [Display(Name = "Descripción")]
    [Required(ErrorMessage = "La descripción es un campo requerido.")]
    public string Descripcion { get; set; }

    public ICollection<Producto> Productos { get; set; }
}
```

Las anotaciones de validación añadidas en algunas propiedades también deberán adaptarse a los requerimientos concretos en cada caso. Las clases de datos anteriores se han definido considerando el comportamiento predeterminado de las convenciones de *Entity Framework Core*: los nombres de propiedad de una clase de datos se usan como nombres de columna, las propiedades de una entidad que se denominan ID o Id se reconocen como clave principal y una propiedad se interpreta como clave ajena si se denomina <nombre de propiedad de navegación><nombre de propiedad clave principal>.

- Clase del contexto de datos

```
public class MvcTiendaContexto : DbContext
{
    public MvcTiendaContexto(DbContextOptions<MvcTiendaContexto> options)
        : base(options)
    {
    }

    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Producto> Productos { get; set; }
    public DbSet<Cliente> Clientes { get; set; }
    public DbSet<Estado> Estados { get; set; }
    public DbSet<Pedido> Pedidos { get; set; }
    public DbSet<Detalle> Detalles { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Deshabilitar la eliminación en cascada en todas las relaciones
        base.OnModelCreating(modelBuilder);
        foreach (var relationship in
            modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys()))
        {
            relationship.DeleteBehavior = DeleteBehavior.Restrict;
        }
    }
}
```

Considerando el comportamiento predeterminado de las convenciones de *Entity Framework Core*, los nombres de las propiedades *DbSet* se usarán como nombres de las tablas.