# **Práctico de integración - .NET MVC**

## Objetivo

El objetivo de este ejercicio es aplicar conocimientos adquiridos, y aprender nuevos, sobre las siguientes tecnologías:



* .NET Framework (C# v7)
* .NET MVC 5
* HTML
* CSS
* Javascript
* Bootstrap (v3, v4)
* SQL

Además, se valorará la capacidad de investigación y resolución de ciertos requerimientos.

## Requisitos

Antes de comenzar, verificar tener instalado lo siguiente:

* .NET Framework 4.7.2
* Visual Studio 2017 o 2019
* SQL Server 2014 (o superior)

Opcionalmente:

* SQL Server Management Studio (o cualquier otro cliente para conectarse a la base de datos, Visual Studio viene con uno incluido).

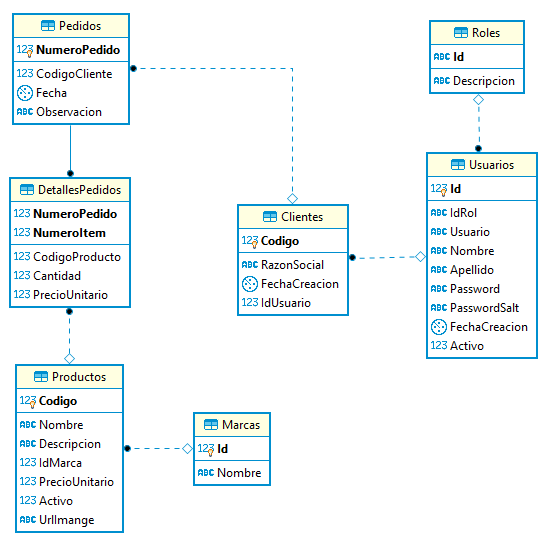
# Comenzando

## Template

Primero, se deberá **buscar un template** HMTL en internet compatible con Bootstrap (3 ó 4) **a elección**, del tipo dashboard o admin. Buscar en Google como “free bootstrap admin template”, hay varios para elegir. Por ejemplo, el “[SB Admin 2](https://startbootstrap.com/themes/sb-admin-2/)” (no se puede utilizar).

## Crear la base de datos

Abrir un cliente de SQL Server y ejecutar el script *db\_practico.sql* que crea la base e inserta datos de prueba iniciales. El modelo de la misma es el siguiente:



## Organización del proyecto

El proyecto consta de una solución con por lo menos tres capas (proyectos).

|  |  |
| --- | --- |
| **Capa** | **Nombre (namespace)** |
| Solución | [Nombre].PracticoMVC |
| Entidades | [Nombre].PracticoMVC.Entidades |
| Acceso datos | [Nombre].PracticoMVC.AccesoDatos |
| Aplicación Web | [Nombre].PracticoMVC.WebApp |

* **Entidades**: datos del dominio (producto, pedido, etc).
* **Acceso datos**: comunicación con la base de datos.

Para ejecutar las operaciones con la base de datos, podrá utilizar ADO.NET, Dapper o EntityFramework a elección. Se valorará más el uso de Dapper.

* **Aplicación web**: controladores, vistas y modelos.

Se podrán agregar todas las capas/proyectos que se crean necesarios.

## Requerimientos

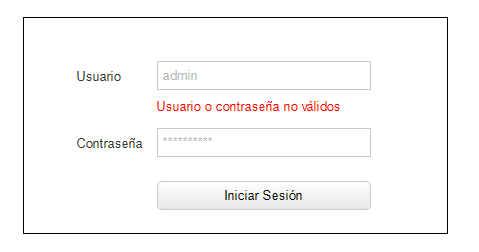
|  |  |
| --- | --- |
| **Nº** | **Descripción** |
| 1 | **Login y logout**  El sistema deberá permitir el ingreso al sistema mediante un login que solicite usuario y contraseña. Está última deberá estar protegida mediante algún algoritmo de **hashing** incluyendo [salt](https://es.wikipedia.org/wiki/Sal_(criptograf%C3%ADa)).  También, deberá permitir al usuario cerrar la sesión, redirigiendo al login. |
| 2 | **Modificación de contraseña**  El sistema deberá permitir el cambio de contraseña del usuario logueado. Solicitará los siguientes datos:   * contraseña actual, * nueva contraseña, y * confirmación contraseña (igual a la nueva contraseña). |
| 3 | **CRUD Productos**  El sistema deberá permitir:   * ver el listado de productos cargados, * dar de alta un nuevo producto, y * modificar un producto seleccionado.   No se hará la eliminación.  El usuario con perfil de administrador solo debe acceder a esta funcionalidad. |
| 4 | **Consulta de productos**  El sistema deberá permitir:   * ver lista de productos disponibles, * agregarlos a un carrito seleccionando la cantidad   El proceso de agregar al carrito deberá ser implementado mediante AJAX.  El administrador puede verlo tabular, pero el cliente debe verlo sí o sí en forma de cards. |
| 5 | **Consulta de pedidos**  El sistema deberá permitir:   * ver el listado de pedidos realizados ordenados por fecha, desde el más reciente.   El usuario administrador puede ver todos los pedidos de todos los cliente, mientras que un cliente, sólo puede ver los propios.  El listado debe mostrarse en forma tabular. |
| 6 | **Creación de pedido (carrito)**  El sistema deberá permitir:   * dar de alta un pedido a partir de los productos agregados al carrito.   Asegurar que el proceso sea atómico (transacción).  Funcionalidad sólo para el cliente. |

## 

## Definición de pantallas

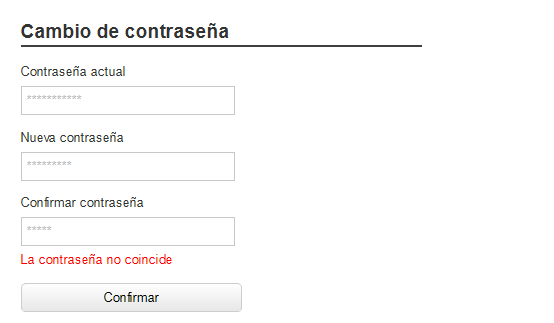
A continuación se detallan el modelo de cómo deberían ser las pantallas a modo orientativo. Hay total libertad para proponer cambios.

**Pantalla de login**



Al arrojar un error de validación, se debe limpiar el campo de contraseña.

**Pantalla de modificación de contraseña**

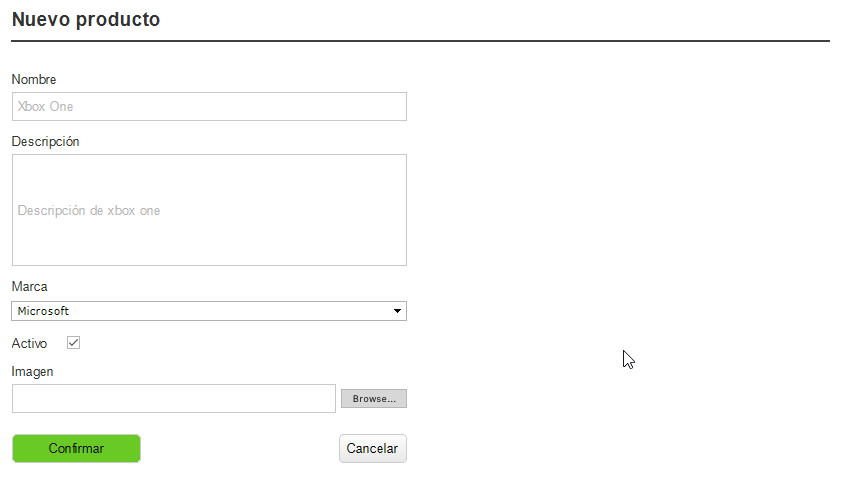


**Pantalla consulta de productos (administrador)**



Considerar que los precios deben estar alineados a la derecha y con su respectivo formato (símbolo de moneda, decimales y separación de miles).

**Pantalla alta de producto**



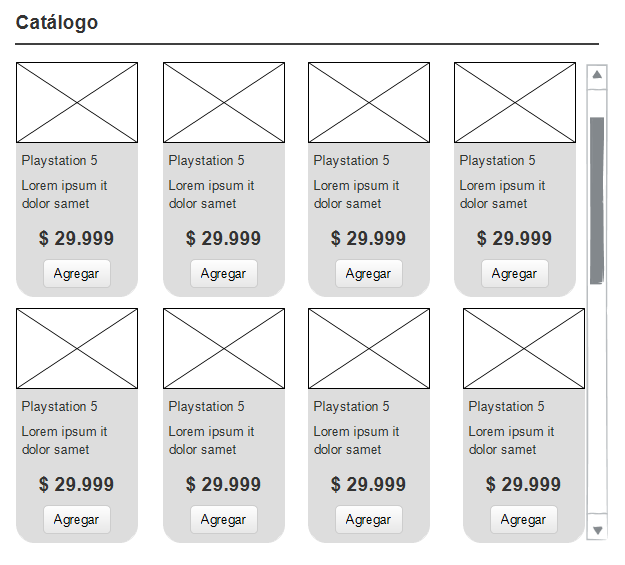
* Por default, debe aparecer activo el campo activo.
* La imagen del producto debe subirse al servidor a una carpeta /images/productos/.
* Al confirmar, debe preguntarme por “Si” o por “No”.

**Pantalla edición de producto**



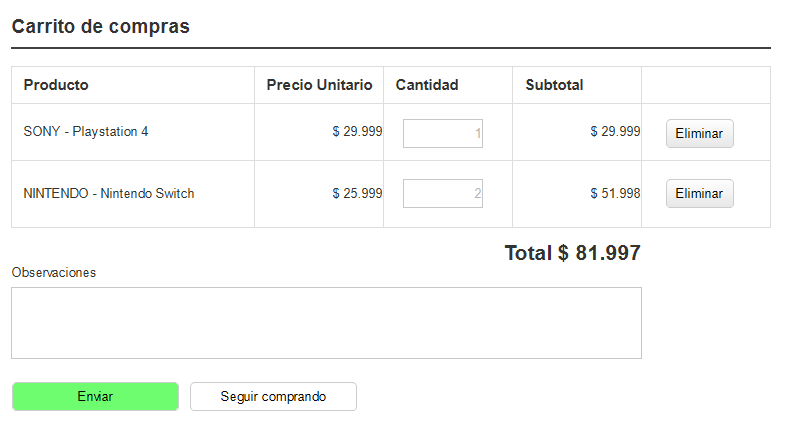
* Debe mostrar en el título (del form, y en el título de la pestaña del navegador el nombre y el código del producto que se está editando).
* El campo de título es de sólo lectura.
* Debe mostrar la imagen que tiene actualmente cargada.

**Pantalla consulta de productos (cliente)**



Los precios deben estar bien formateados.

**Pantalla de carrito (envío de pedido)**



* Los inputs de cantidad deben ser de tipo numérico.
* Al eliminar un ítem me debe preguntar por “Si” o por “No”.
* Al enviar me debe preguntar por “Si” o por “No”.
* Los precios deben estar bien formateados.
* “Seguir comprando” debe llevarme de vuelta al catálogo.

**Pantalla de consulta de pedidos**

Forma tabular similar a la de productos. Debe tener un botón para cada pedido que me permita ver su detalle. Para esto, cuando el usuario haga click sobre el mismo, se debe abrir un modal con los ítems del pedido, mostrando la descripción del producto, su precio unitario, cantidad, subtotal calculado, y al final, el total del pedido y un espacio para la observación enviada si es que hubiere.

## Valoraciones

Se valorarán los siguientes puntos:

* Separación entre capas
* Uso de inyección de dependencias mediante alguna librería (Autofac, Ninject, Identity, etc)
* Prolijidad en los elementos visuales (márgenes, paddings, tamaños de fuente, etc.)
* Uso de librerías y plugins para el frontend (datepickers, select2, checkboxes, etc.)
  + Uso de librería para mensajes de tipos alerts, confirm, prompt.
* Uso de la librería Font Awesome (v4.7) para el uso de iconos.
* Reutilización de formularios (alta y edición por ejemplo).

# Anexo

## Hashing de contraseñas

El hashing de contraseña significa que la clave se modifica en una sola dirección, es decir, no se puede volver a su valor original, y es almacenada de esta forma en la base de datos.

Por ejemplo:

Contraseña: hola123

Hash (md5): 9450476b384b32d8ad8b758e76c98a69

*Nota: no se recomienda utilizar el algoritmo md5, ya que es vulnerable por ser muy rápido. Está a modo de ejemplo.*

En un proceso de inicio de sesión de un usuario, lo que se hace es aplicar el algoritmo en el en el valor de contraseña provisto y compararlo con el almacenado en la base de datos. De esta forma, nunca guardamos el valor original de la contraseña.

**Recomendación (opcional)**

Utilizar los algoritmos **HMACSHA256** o **HMACSHA512** del namespace System.Security.Cryptography.

Creación de hash de contraseña:

Recibe como entrada la cadena de texto con la contraseña, y tiene como salida el hash de la password y el hash del salt codificados como cadenas de texto.

Hay que tener en cuenta que los hashes generados para la contraseña y el salt, son datos binarios (array de bytes). En una base de datos se pueden guardar directamente cómo binarios en un campo BLOB, pero si queremos guardarlos como texto, deberemos codificarlos a Base64.

Verificación de contraseña:

Tiene como entradas el valor de la contraseña que se quiere verificar, el valor del hash y el valor del salt en base64. Cómo salida, simplemente retorna un true si la contraseña se verifica, sino un false.

