
CONTENIDOS

- ASP Web MVC
- ASP Layout

INTRODUCCIÓN

Copie el siguiente enlace en su navegador: https://tinyurl.com/TL2-TP8-2025 esto creará el repositorio para poder subir el **Trabajo Práctico Nro. 08**, realice los pasos ya aprendidos para *clonar* el repositorio en su máquina y poder comenzar a trabajar de forma *local*.

Se desea portar la aplicación ASP Web Api construida en el práctico anterior a una aplicación ASP Web Application, para dotar a los operadores una interfaz de usuario para utilizar el sistema.

Debe portar la lógica del proyecto **ASP.NET Web API (TP Nro. 7)** a una nueva aplicación **ASP.NET Core Web Application (MVC)**. El objetivo es crear una **Interfaz de Usuario (UI)** completa para la gestión de Productos y Presupuestos.
Creación y Migración:

- Cree un nuevo proyecto MVC: *> dotnet new mvc*
- Migre los **Modelos** y los **Repositorios** del TP Nro 7. (no los controladores)
- Instale la dependencia para SQLite: *> dotnet add package Microsoft.Data.SQLite*

Dividiremos en 2 etapas la creación del proyecto, avanzaremos desde lo más simple a lo más complejo

Etapa 1: Patrón MVC Básico y Lectura de Datos

Objetivo: Establecer la estructura de la aplicación y el flujo **Modelo** → **Controlador** → **Vista** para la lectura de datos.

1. Estructura y Estilo

Al crear un nuevo proyecto en ASP.NET MVC (versión 9 o superior), el entorno de desarrollo incluye por defecto la biblioteca Bootstrap.

Bootstrap es un framework de CSS y JavaScript que facilita el diseño y la maquetación de interfaces web modernas, adaptables y visualmente atractivas, sin necesidad de escribir todo el código de estilo desde cero.

La inclusión de Bootstrap en ASP.NET MVC permite a los desarrolladores::

- Crear layouts responsivos que se adaptan automáticamente a distintos tamaños de pantalla (computadoras, tablets y celulares).
- Usar componentes predefinidos como botones, formularios, menús de navegación, tarjetas, modales, alertas, entre otros.
- Aplicar tipografía y estilos consistentes en toda la aplicación.

ASP.NET MVC ya incluye las referencias a los archivos CSS y JS de Bootstrap dentro del proyecto, por lo que los estudiantes pueden utilizar sus clases directamente en las vistas (por ejemplo, class="btn btn-primary" para un botón estilizado).

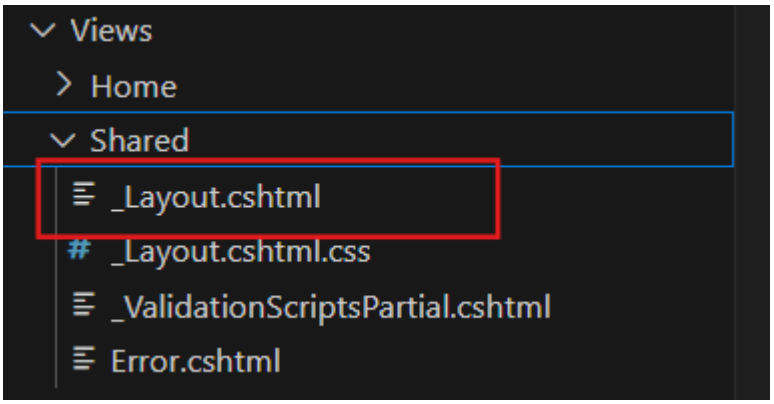
La documentación oficial de Bootstrap, junto con ejemplos y guías de uso, puede consultarse en su sitio web:

https://getbootstrap.com/

En **ASP.NET MVC**, las vistas pueden compartir una misma estructura general mediante un archivo especial llamado **Layout**.
El layout funciona como una **plantilla maestra** que define las secciones comunes de todas las páginas del sitio, como el encabezado, la barra de navegación, los estilos, los scripts y el pie de página.

De esta forma, cada vista individual sólo necesita definir su contenido específico, mientras que la apariencia y el diseño general se mantienen consistentes en toda la aplicación.

Por convención, el layout principal del proyecto se encuentra en la carpeta:
Views/Shared/_Layout.cshtml



Dentro de este archivo se suele incluir:

- **La cabecera (<head>)**: contiene los enlaces a los archivos CSS (por ejemplo, Bootstrap) y los scripts necesarios.
- **La barra de navegación o menú principal**, donde se ubican los enlaces a las distintas secciones de la aplicación.
- **La sección del cuerpo principal**, que se representa con la instrucción `@RenderBody()` y es donde se muestra el contenido de cada vista.
- **El pie de página**, que suele contener información o scripts adicionales.

Para este trabajo práctico, se debe **modificar el archivo `Views/Shared/_Layout.cshtml`** para agregar en la barra de menú las opciones **"Productos"** y **"Presupuestos"**.
Esto puede hacerse añadiendo los elementos correspondientes dentro del bloque de navegación (generalmente dentro de una lista `<ul>` o un contenedor `<nav>`), utilizando las clases de **Bootstrap** para mantener la coherencia visual del sitio.

2. Controladores de Lectura

- Cree el **ProductosController**
- Cree el **PresupuestosController**.
- Implemente la acción **Index (Listar)** para ambas entidades.

Ej. Implementación de controller utilizando el patrón Repository (en ProductosController.cs)

```
public class ProductosController: Controller
{
    private ProductoRepository productoRepository;
    public ProductosController()
    {
        productoRepository= new ProductoRepository();
    }
//A partir de aquí van todos los Action Methods (Get, Post,etc.)
}
```

Ejemplo de cómo "Listar" los producto desde Index

```
    [HttpGet]
    public IActionResult Index()
    {
       List<Producto> productos = productoRepository.GetAll();
       return View(productos);
    }
```

3. Lectura de Datos Complejos (Detalles Relacionales)

- Implemente la acción **Details (Detalle)** en el **PresupuestosController**. Esta vista debe mostrar:
    1. El encabezado del Presupuesto.
    2. El **listado de todos los Productos** asociados, utilizando la lógica relacional ya implementada en su Repositorio de Presupuestos.

Etapa II: Persistencia y Control de Entidades

Objetivo: Dominar el flujo de escritura **Vista (Formulario) → Controlador → Repositorio**

4. CRUD de Escritura para Productos

Implemente el ciclo completo de manipulación de datos para la entidad **Producto** utilizando métodos:

- Acciones **Create (GET y POST)**: Implemente el formulario y la recepción del objeto.
- Acciones **Edit (GET y POST)**.
- Acciones **Delete (GET y POST)**.

5. CRUD de Escritura para Presupuestos

Implemente el ciclo completo de manipulación de datos para la entidad **Presupuesto** utilizando métodos:

- Acciones **Create (GET y POST)**: Implemente el formulario y la recepción del objeto.
- Acciones **Edit (GET y POST)**.
- Acciones **Delete (GET y POST)**

Como guía le dejamos como le tendría que quedar la estructura de su proyecto con los archivos que tiene que crear.

```
[Carpeta Raíz del Proyecto]
|
├── DB/
|   └── tienda.db        <-- Base de datos SQLite (Archivo Migrado)
├── Controllers/
|   ├── HomeController.cs
|   ├── ProductosController.cs    <-- 1. Lógica CRUD Productos
|   └── PresupuestosController.cs <-- 2. Lógica CRUD Presupuestos
|
├── Models/
|   ├── Producto.cs              <-- 3. Modelo de la entidad Producto
|   └── Presupuesto.cs           <-- 4. Modelo de la entidad Presupuesto
|
|
```

```
├── Repositories/
│   ├── ProductoRepository.cs    <-- 5. Acceso a datos de Producto
│   └── PresupuestoRepository.cs <-- 6. Acceso a datos de Presupuesto
│
├── Views/
│   ├── Productos/
│   │   ├── Index.cshtml         <-- Listar
│   │   ├── Details.cshtml       <-- Ver detalle
│   │   ├── Create.cshtml        <-- Formulario Crear (GET/POST)
│   │   ├── Edit.cshtml          <-- Formulario Modificar (GET/POST)
│   │   └── Delete.cshtml        <-- Confirmar Eliminar (GET/POST)
│   │
│   ├── Presupuestos/
│   │   ├── Index.cshtml         <-- Listar
│   │   ├── Details.cshtml       <-- Ver presupuesto con lista de productos
│   │   ├── Create.cshtml        <-- Formulario Crear Presupuesto(GET/POST)
│   │   ├── Edit.cshtml          <-- Formulario Modificar Presupuesto(GET/POST)
│   │   └── Delete.cshtml        <-- Confirmar Eliminar Presupuesto (GET/POST)
│   │
│   │
│   └── Shared/
│       └── _Layout.cshtml       <-- Plantilla maestra de la aplicación
│
├── wwwroot/
│   ├── css/
│   ├── js/
│   └── lib/
│
├── appsettings.json
├── Program.cs
└── SistemaPresupuestos.csproj
```

NOTA : Para correr la aplicación puede usar el comando

- **dotnet** watch

Este comando permite la recarga activa de la aplicación que está desarrollando.

Algunos links útiles

Introducción a bootstrap (layout de vistas):

https://getbootstrap.com/docs/4.5/getting-started/introduction/

Cheatsheet de razor:

---

https://gist.github.com/jwill9999/655533b6652418bd3bc94d864a5e2b49#Index

Más links útiles entre los recursos en la página de la cátedra.

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 9

CONTENIDOS

- ASP Web MVC
- ASP Layout
- Validaciones
- View Models

INTRODUCCIÓN

El objetivo de este TP es migrar los formularios de la aplicación para que utilicen objetos de transferencia dedicados (View Models) y aplicar todas las reglas de negocio de forma segura y eficiente, aprovechando la validación automática de ASP.NET Core (Principio DRY).

Seguiremos trabajando en el mismo repositorio del TP8

1. Creación de View Models (Separación de Responsabilidades)

Cree la carpeta `ViewModels` e implemente los siguientes objetos. Estos objetos **reemplazarán** a las entidades `Producto` y `Presupuesto` en los métodos de `[HttpPost]` y en las vistas de formularios.

1. **`ViewModels/ProductoViewModel.cs`**: Objeto para manejar la creación y edición de Productos.
2. **`ViewModels/PresupuestoViewModel.cs`**: Objeto para manejar la creación y edición de Presupuestos.
3. **`ViewModels/AgregarProductoViewModel.cs`**: Objeto específico para manejar el formulario de la relación **muchos a muchos** (cargar un producto a un presupuesto). Este ViewModel deberá incluir:
    - La propiedad **`ListaProductos`** de tipo `SelectList` para que el Controlador pueda cargar el dropdown de selección.

2. Implementación de Validaciones (Data Annotations)

Aplique los siguientes atributos de validación directamente en las **propiedades de los View Models** creados en el punto 1.

- **En `ProductoViewModel`:**
    - **`Descripcion`**: Es opcional, pero debe tener una longitud máxima de 250 caracteres (`[StringLength]`).
    - **`Precio`**: Debe ser **requerido** (`[Required]`) y debe ser un valor **positivo** (mayor a 0) (`[Range]`).
- **En `PresupuestoViewModel`:**
    - **`NombreDestinatario`**: Debe ser **obligatorio** (`[Required]`).
    - **Opcional/Alternativa (Si se decide guardar Email):** Si se decide guardar un email en lugar del nombre, se debe validar el formato del email (usar el atributo `[EmailAddress]`).
    - **`FechaCreacion`**: Debe ser **requerida** (`[Required]`) y se debe implementar la lógica de validación para asegurar que la fecha sea **menor o igual a la fecha actual** (no puede ser una fecha futura).

- **En `AgregarProductoViewModel`:**
  - **`Cantidad`**: Debe ser **requerida** ([Required]) y un valor **positivo** (mayor a 0) ([Range]).

3. Modificación de Controladores y Vistas (Activación del Flujo Seguro)

Modifique las acciones [HttpPost] y las vistas de formularios para activar el flujo de validación del servidor y del cliente.

- **Modificación de Vistas (`Create.cshtml`, `Edit.cshtml`):**
  - Actualizar la directiva @model para apuntar al respectivo ViewModel.
  - Añadir el Tag Helper **`<span asp-validation-for="...">`** debajo de cada campo que tenga una validación, para que muestre el mensaje de error.
  - Habilitar la validación en el lado del cliente (Front-End) añadiendo la línea **`@section Scripts { <partial name="_ValidationScriptsPartial" /> }`** al final de la vista.
- **Modificación de Controladores (`[HttpPost]` de Create y Edit):**
  - Implementar el chequeo de validez utilizando **`if (!ModelState.IsValid)`** como primera línea de código.
  - **Lógica de Rechazo:** Si la validación falla, se debe usar la sintaxis **`return View(viewModel);`** para devolver el objeto ViewModel con los datos y errores a la vista.
- **Flujo Relacional (Acción `AgregarProducto` POST):**
  - Implementar la lógica de chequeo ModelState.IsValid.
  - **Lógica de Recarga:** Si la validación de la Cantidad falla, es obligatorio **recargar el `SelectList` de Productos** en el ViewModel antes de devolver la vista, para evitar un error de referencia nula en el dropdown del formulario.

Desarrollo Paso a Paso del TP Nro 9

1. Etapa I: Creación y Definición de ViewModels

El primer paso es crear las estructuras de datos que transportarán la información de los formularios, separándolos del Modelo de Dominio.

Tarea 1.1: Crear la Carpeta `ViewModels`

- **Acción:** Cree una nueva carpeta en la raíz del proyecto (junto a Models, Controllers, etc.) llamada ViewModels.

Tarea 1.2: ViewModels/ProductoViewModel.cs (Con Validaciones)

Este VM manejará la creación y edición de productos.

C#

```
using System.ComponentModel.DataAnnotations;

namespace SistemaVentas.Web.ViewModels
{
    public class ProductoViewModel
    {
        // Se incluye Id para la acción de EDICIÓN
        public int IdProducto { get; set; }

        // Validación: Máximo 250 caracteres. Es opcional por defecto si no tiene [Required]
        [Display(Name = "Descripción del Producto")]
        [StringLength(250, ErrorMessage = "La descripción no puede superar los 250 caracteres.")]
        public string Descripcion { get; set; }

        // Validación: Requerido y debe ser positivo
        [Display(Name = "Precio Unitario")]
        [Required(ErrorMessage = "El precio es obligatorio.")]
        [Range(0.01, double.MaxValue, ErrorMessage = "El precio debe ser un valor positivo.")]
        public decimal Precio { get; set; }
    }
}
```

Tarea 1.3: ViewModels/PresupuestoViewModel.cs (Con Validaciones)

Este VM manejará la creación y edición de presupuestos.

C#

```
using System.ComponentModel.DataAnnotations;
using System;

namespace SistemaVentas.Web.ViewModels
{
    public class PresupuestoViewModel
    {
        public int IdPresupuesto { get; set; }

        //  Validación: Requerido
        [Display(Name = "Nombre o Email del Destinatario")]
        [Required(ErrorMessage = "El nombre o email es obligatorio.")]
        // Opcional: Se puede añadir la validación de formato de email si se opta por guardar el
mail.
        // [EmailAddress(ErrorMessage = "El formato del email no es válido.")]
        public string NombreDestinatario { get; set; }
```

```
        // Validación: Requerido y tipo de dato
        [Display(Name = "Fecha de Creación")]
        [Required(ErrorMessage = "La fecha es obligatoria.")]
        [DataType(DataType.Date)]
        public DateTime FechaCreacion { get; set; }

        // La validación de que la fecha no es futura se hará en el Controlador (ver Etapa 3).
    }
}
```

Tarea 1.4: ViewModels/AgregarProductoViewModel.cs (Con Validación y SelectList)

Este VM manejará el formulario para la relación N:M. (Productos en Presupuesto)

C#

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Mvc.Rendering; // Necesario para SelectList

namespace SistemaVentas.Web.ViewModels
{
    public class AgregarProductoViewModel
    {
        // ID del presupuesto al que se va a agregar (viene de la URL o campo oculto)
        public int IdPresupuesto { get; set; }

        // ID del producto seleccionado en el dropdown
        [Display(Name = "Producto a agregar")]
        public int IdProducto { get; set; }

        // Validación: Requerido y debe ser positivo
        [Display(Name = "Cantidad")]
        [Required(ErrorMessage = "La cantidad es obligatoria.")]
        [Range(1, int.MaxValue, ErrorMessage = "La cantidad debe ser mayor a cero.")]
        public int Cantidad { get; set; }

        //  Propiedad adicional para el Dropdown (no se valida, solo se usa en la Vista)
        public SelectList ListaProductos { get; set; }
    }
}
```

---

2. Etapa II: Refactorización y Flujo de Seguridad (Productos)

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 9
Modificaremos el ProductosController.cs y sus vistas para implementar el ViewModel y la lógica ModelState.IsValid.

Tarea 2.1: Modificar Controllers/ProductosController.cs (Acciones POST)

- **Acción:** Reemplace los parámetros del Modelo de Dominio (Producto) por el ViewModel (ProductoViewModel) en los métodos [HttpPost].
- **Lógica Clave:** Implemente el chequeo if (!ModelState.IsValid) y el mapeo manual.

C#

```
// ... using SistemaVentas.Web.Models;
// ... using SistemaVentas.Web.Repositorios;
// ... using SistemaVentas.Web.ViewModels; //  ❗ Nuevo using

public class ProductosController : Controller
{
    private readonly ProductoRepository _repo = new ProductoRepository();

    // ... Acciones GET (Index, Details, Create, Edit) no cambian o se adaptan ...

    // POST: Productos/Create
    [HttpPost]
    public IActionResult Create(ProductoViewModel productoVM)
    {
        //  1. CHEQUEO DE SEGURIDAD DEL SERVIDOR
        if (!ModelState.IsValid)
        {
            //  Si falla: Devolvemos el ViewModel con los datos y errores a la Vista
            return View(productoVM);
        }

        // 2. SI ES VÁLIDO: Mapeo Manual de VM a Modelo de Dominio
        var nuevoProducto = new Producto
        {
            Descripcion = productoVM.Descripcion,
            Precio = productoVM.Precio
        };

        // 3. Llamada al Repositorio
        _repo.Add(nuevoProducto);
        return RedirectToAction(nameof(Index));
    }

    // POST: Productos/Edit
    [HttpPost]
    public IActionResult Edit(int id, ProductoViewModel productoVM)
    {
        if (id != productoVM.IdProducto) return NotFound();
```

```
        //  1. CHEQUEO DE SEGURIDAD DEL SERVIDOR
        if (!ModelState.IsValid)
        {
            return View(productoVM);
        }

        //  2. Mapeo Manual de VM a Modelo de Dominio
        var productoAEditar = new Producto
        {
            IdProducto = productoVM.IdProducto, // Necesario para el UPDATE
            Descripcion = productoVM.Descripcion,
            Precio = productoVM.Precio
        };

        // 3. Llamada al Repositorio
        _repo.Update(productoAEditar);
        return RedirectToAction(nameof(Index));
    }
}
```

Tarea 2.2: Modificar `Views/Productos/Create.cshtml` (Activación Front-End)

- **Acción:** Actualice la directiva `@model` y añada los Tags Helpers de validación.

Razor CSHTML

```
@model SistemaVentas.Web.ViewModels.ProductoViewModel

@{
    ViewData["Title"] = "Crear Producto";
}
<form asp-action="Create">

    <div class="form-group">
        <label asp-for="Descripcion" class="control-label"></label>
        <input asp-for="Descripcion" class="form-control" />
        <span asp-validation-for="Descripcion" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="Precio" class="control-label"></label>
        <input asp-for="Precio" class="form-control" />
        <span asp-validation-for="Precio" class="text-danger"></span>
    </div>

    <div class="form-group mt-3">
        <input type="submit" value="Crear" class="btn btn-primary" />
    </div>
```

```
</form>
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

(Repetir el proceso para `Views/Productos/Edit.cshtml` y las vistas de Presupuestos).

---

3. Etapa III: Lógica Relacional Avanzada (Agregar Producto a Presupuesto)

Aquí implementaremos el flujo de **Carga de Productos** y su validación.

Tarea 3.1: Modificar `Controllers/PresupuestosController.cs` (Acción GET)

Implementamos la acción GET para mostrar el formulario, cargando el *dropdown* con los productos disponibles.

C#

```
//..using SistemaVentas.Web.ViewModels; //Necesario para poder llegar a los ViewModels
//..using Microsoft.AspNetCore.Mvc.Rendering; // Necesario para SelectList

public class PresupuestosController : Controller
{
    private readonly PresupuestoRepository _repo = new PresupuestoRepository();
    // Necesitamos el repositorio de Productos para llenar el dropdown
    private readonly ProductoRepository _productoRepo = new ProductoRepository();

    // GET: Presupuestos/AgregarProducto
    public IActionResult AgregarProducto(int id)
    {
        // 1. Obtener los productos para el SelectList
        List<Producto> productos = _productoRepo.GetAll();

        // 2. Crear el ViewModel
        AgregarProductoViewModel model = new AgregarProductoViewModel
            {
                IdPresupuesto = id, // Pasamos el ID del presupuesto actual
                // 3. Crear el SelectList
                ListaProductos = new SelectList(productos, "IdProducto", "Descripcion")
            };
        return View(model);
    }

    // ❗ El Método CLAVE para la validación de la cantidad
    // POST: Presupuestos/AgregarProducto
    [HttpPost]
    public IActionResult AgregarProducto(AgregarProductoViewModel model)
```

```
    {
        // 1. Chequeo de Seguridad para la Cantidad
        if (!ModelState.IsValid)
        {
            //  LÓGICA CRÍTICA DE RECARGA: Si falla la validación,
            // debemos recargar el SelectList porque se pierde en el POST.
            var productos = _productoRepo.GetAll();
            model.ListaProductos = new SelectList(productos, "IdProducto", "Descripcion");

            // Devolvemos el modelo con los errores y el dropdown recargado
            return View(model);
        }

        //  2. Si es VÁLIDO: Llamamos al repositorio para guardar la relación
        _repo.AddDetalle(model.IdPresupuesto, model.IdProducto, model.Cantidad);

        // 3. Redirigimos al detalle del presupuesto
        return RedirectToAction(nameof(Details), new { id = model.IdPresupuesto });
    }
}
```

Tarea 3.2: Crear Views/Presupuestos/AgregarProducto.cshtml

Creamos la vista de formulario para manejar los tres datos necesarios (ID Presupuesto, ID Producto, Cantidad). nota para el ej. ***SistemaVentas.Web.ViewModels*** es el namespace vinculado a los ViewModels (cambie por el que ud use)

Razor CSHTML

```
@model SistemaVentas.Web.ViewModels.AgregarProductoViewModel

@{
    ViewData["Title"] = "Agregar Producto al Presupuesto";
}

<h2>Agregar Producto al Presupuesto Nro @Model.IdPresupuesto</h2>
<hr />

<div class="row">
    <div class="col-md-6">
        <form asp-action="AgregarProducto">

            <input type="hidden" asp-for="IdPresupuesto" />

            <div class="form-group">
                <label asp-for="IdProducto" class="control-label"></label>
                <select asp-for="IdProducto"
                        asp-items="@Model.ListaProductos"
                        class="form-control">
```

```
                <option value="">-- Seleccionar Producto --</option>
            </select>
            <span asp-validation-for="IdProducto" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="Cantidad" class="control-label"></label>
            <input asp-for="Cantidad" class="form-control" />
            <span asp-validation-for="Cantidad" class="text-danger"></span>
        </div>

        <div class="form-group mt-4">
            <input type="submit" value="Añadir Producto" class="btn btn-primary" />
        </div>
    </form>
  </div>
</div>

<div class="mt-3">
  <a asp-action="Details" asp-route-id="@Model.IdPresupuesto">Volver al Presupuesto</a>
</div>

@section Scripts {
  <partial name="_ValidationScriptsPartial" />
}
```

Recordando validación:

Para poder poder utilizar atributos de validación integrados recuerde qué:

- Se debe agregar las dependencias de validación a los ViewModels, es decir:
  ```
  using System.ComponentModel.DataAnnotations;
  ```
- Utilizar los atributos de validación correspondientes para cada propiedad del ViewModel.
- En las vistas (front-end) donde se quiera utilizar la validación se deben agregar el conjunto de librerías de javascript que validan los datos con las reglas del servidor. Para ello debe agregar la siguiente línea en cada vista
  ```
  @section Scripts{<partial name="_ValidationScriptsPartial"/>}
  ```
- Desde el servidor (back-end) (en los controllers) puede utilizar el objeto **ModelState.IsValid** para asegurar la validez de los campos enviados en los endpoint.

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 9

Ej.

```
[HttpPost]
public IActionResult CrearProducto(AltaProductoViewModel productoVM){
if(!ModelState.IsValid) return RedirectToAction("Index");
//continua la definición del endpoint
}
```

Puede ingresar al siguiente link para ver la documentación sobre validaciones:
https://learn.microsoft.com/es-es/aspnet/core/mvc/models/validation?view=aspnetcore-8.0

NOTA: No se olvide que para correr la aplicación puede usar el comando

- **dotnet** watch

Este comando permite la recarga activa de la aplicación que está desarrollando.

Algunos links útiles

Introducción a bootstrap (layout de vistas):

https://getbootstrap.com/docs/4.5/getting-started/introduction/

Cheatsheet de razor:

https://gist.github.com/jwill9999/655533b6652418bd3bc94d864a5e2b49#Index

Más links útiles entre los recursos en la página de la cátedra.

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 10

CONTENIDOS

- Inyección de dependencias
- Autenticación y Autorización

Introducción

El objetivo es migrar el sistema de Presupuestos a una arquitectura desacoplada mediante **Inyección de Dependencias (DI)** y construir un **Módulo de Autenticación y Autorización por Roles** funcional que use variables de **Sesión** para el control de acceso.

Seguiremos trabajando en el mismo repositorio del TP anterior,

Reglas de Acceso:

- **Administrador:** Acceso total (CRUD) a **Productos** y **Presupuestos**.
- **Cliente:** Acceso de **solo Lectura** (Index, Details) a **Presupuestos**.
- **No Logueado:** Redirigir a /Login/Index.
- **Error UX:** Un usuario *logueado* sin permiso debe ser redirigido a una acción **AccesoDenegado**.

Desarrollo Paso a Paso del TP Nro 10

Fase 1: Infraestructura y Desacoplamiento (DI y Base de Datos)

Esta fase define las abstracciones (interfaces), el modelo de datos para usuarios y configura el contenedor de Inyección de Dependencias.

1. Creación de Interfaces de Abstracción

Cree la carpeta Interfaces para definir las Interfaces necesarias

- **Interfaces de Repositorios Existentes:** Cree las interfaces **IProductoRepository.cs** e **IPresupuestoRepository.cs** con todos los métodos CRUD
  Nota: Modifique sus clases de repositorio (ProductoRepository, PresupuestoRepository) para que implementen explícitamente estas interfaces
- **Interfaz de Usuarios: Defina la interfaz IUserRepository.cs** con el método GetUser(string username, string password) (debe devolver el objeto Usuario completo).

```
C#

namespace MVC.Interfaces;

public interface IUserRepository
{
    // Retorna el objeto Usuario si las credenciales son válidas, sino null.
    Usuario GetUser(string username, string password);
}
```

- **Interfaz de Seguridad: Defina la interfaz IAuthenticationService.cs** con los métodos: Login(user, pass), Logout(), IsAuthenticated() y HasAccessLevel(rol).

```
C#

namespace MVC.Interfaces;

public interface IAuthenticationService
{
    bool Login(string username, string password);
    void Logout();
    bool IsAuthenticated();
    // Verifica si el usuario actual tiene el rol requerido (ej. "Administrador").
    bool HasAccessLevel(string requiredAccessLevel);
}
```

2. Creación de la Tabla de Usuarios en la Base de Datos

- **Acción:** En **SQLiteStudio** o su herramienta de DB, cree la tabla **Usuarios**.
- **Campos:**
  - Id (Numérico)
  - Nombre (Texto)
  - User (Texto)
  - Pass (Texto)
  - Rol (Texto)
- **Poblar:** Inserte registros de prueba, al menos uno con Rol = "Administrador" y uno con Rol = "Cliente".

3. Implementación del Modelo y Repositorio de Usuarios

- **Modelo:** Cree el Modelo de Dominio **Usuario.cs** (incluyendo el campo Rol).
- **Repositorio:** Cree la clase **UsuarioRepository.cs** que implemente **IUserRepository** y contenga la lógica para buscar al usuario en la tabla Usuarios por credenciales.

```
C#

namespace MVC.Repositorios;

public class UsuarioRepository : IUserRepository
{
    // Lógica para conectar con la DB y buscar por user/pass.
    public Usuario GetUser(string usuario, string contrsena)
    {
         Usuario user = null;

        //Consulta SQL que busca por Usuario Y Contrasena
        const string sql = @"
                    SELECT Id, Nombre, User, Pass, Rol
                    FROM Usuarios
                    WHERE User = @Usuario AND Pass = @Contrasena";

        using var conexion = new SqliteConnection(CadenaConexion);
        conexion.Open();

               using var comando = new SqliteCommand(sql, conexion);

        // Se usan parámetros para prevenir inyección SQL
```

```
                comando.Parameters.AddWithValue("@Usuario", usuario);
                comando.Parameters.AddWithValue("@Contrasena",
contrasena);

                using var reader = comando.ExecuteReader();

                if (reader.Read())
                {
                    // Si el lector encuentra una fila, el usuario
existe y las credenciales son correctas
                    user = new Usuario
                    {
                        Id = reader.GetInt32(0),
                        Nombre = reader.GetString(1),
                        User = reader.GetString(2),
                        Pass = reader.GetString(3),
                        Rol = reader.GetString(4)
                    };
                }
                return user;
    }
```

4. Configuración de Servicios en Program.cs

- **Usings:** Asegúrese de incluir los using correctos para sus Interfaces, Repositorios y Servicios.
- **Servicios de Sesión:**
  - Agregue: builder.Services.AddHttpContextAccessor();
  - Agregue: builder.Services.AddSession(...) (con las opciones recomendadas, como IdleTimeout).
- **Registro de DI (AddScoped):** Registre **TODAS** las interfaces y sus implementaciones:
  - builder.Services.AddScoped<IProductoRepository, ProductoRepository>();
  - builder.Services.AddScoped<IPresupuestoRepository, PresupuestoRepository>();
  - builder.Services.AddScoped<IUserRepository, UsuarioRepository>();
  - builder.Services.AddScoped<IAuthenticationService, AuthenticationService>();
- **Pipeline de Sesión:** Asegúrese de que el orden en la configuración de la aplicación sea correcto:
  - app.UseRouting();
  - **app.UseSession();**
  - app.UseAuthorization();

C#

```
// Asegúrese de agregar los using necesarios
using Microsoft.AspNetCore.Http;
using MVC.Interfaces;
using MVC.Repositorios;
using MVC.Services;

var builder = WebApplication.CreateBuilder(args);
```

```
// Servicios de Sesión y Acceso a Contexto (CLAVE para la autenticación)
builder.Services.AddHttpContextAccessor();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

// Registro de la Inyección de Dependencia (TODOS AddScoped)
builder.Services.AddScoped<IProductoRepository, ProductoRepository>();
builder.Services.AddScoped<IPresupuestoRepository, PresupuestoRepository>();
builder.Services.AddScoped<IUserRepository, UsuarioRepository>();
builder.Services.AddScoped<IAuthenticationService, AuthenticationService>();

// ... (Otros servicios y configuración MVC) ...

var app = builder.Build();

// Configuración del Pipeline de Middleware
// El orden es CRUCIAL: UseSession debe ir ANTES de UseRouting/UseAuthorization
app.UseSession(); //  → Habilita el uso de la sesión

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseAuthorization(); // Necesario si usa atributos, aunque aquí lo haremos manual

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

5. Inyección en Controladores Existentes

- **Refactorización:** Modifique los constructores de **TODOS** los controladores (PresupuestosController y ProductosController) para que reciban sus dependencias (todos los Repositorios y el nuevo servicio de autenticación) **exclusivamente por interfaces** como parámetros.

Fase 2: Módulo de Autenticación Central

Esta fase implementa la lógica de sesión y el controlador de acceso (Login).

6. Implementación de AuthenticationService

- **Clase Concreta:** Implemente **AuthenticationService.cs** e inyecte **IUserRepository** y **IHttpContextAccessor** en el constructor.
- **Lógica de Login:** En el método Login(), si el usuario es válido, devuelve true y carga todas las variables de sesion relacionadas al usuario logueado

- **Lógica de Chequeo:** Los métodos `IsAuthenticated()` y `HasAccessLevel(rol)` deben leer la variable de Sesión `"Rol"` a través del `IHttpContextAccessor`.

C#

```
using MVC.Interfaces;
namespace MVC.Services;
public class AuthenticationService : IAuthenticationService
{
    private readonly IUserRepository _userRepository;
    private readonly IHttpContextAccessor _httpContextAccessor;
    //private readonly HttpContext context;

     public AuthenticationService(IUserRepository userRepository,
IHttpContextAccessor httpContextAccessor)
    {
        _userRepository = userRepository;
        _httpContextAccessor = httpContextAccessor;
       // context = _httpContextAccessor.HttpContext;
    }
    public bool Login(string username, string password)
    {
        var context = _httpContextAccessor.HttpContext;
        var user = _userRepository.GetUser(username,password);
        if (user != null)
        {
            if (context == null)
            {
                  throw new InvalidOperationException("HttpContext
no está disponible.");
            }
            context.Session.SetString("IsAuthenticated", "true");
            context.Session.SetString("User", user.User);
            context.Session.SetString("UserNombre", user.Nombre);
            context.Session.SetString("Rol", user.Rol);
            //es el tipo de acceso/rol admin o cliente
            return true;
        }
        return false;
```

```
        }
        public void Logout()
        {
            var context = _httpContextAccessor.HttpContext;

            if (context == null)
            {
                    throw new InvalidOperationException("HttpContext no
    está disponible.");
            }
           /* context.Session.Remove("IsAuthenticated");
            context.Session.Remove("User");
            context.Session.Remove("UserNombre");
            context.Session.Remove("Rol");
            */
            context.Session.Clear();
        }
        public bool IsAuthenticated()
        {
            var context = _httpContextAccessor.HttpContext;
            if (context == null)
            {
                    throw new InvalidOperationException("HttpContext no
    está disponible.");
            }
               return context.Session.GetString("IsAuthenticated") ==
    "true";
        }
        public bool HasAccessLevel(string requiredAccessLevel)
        {
            var context = _httpContextAccessor.HttpContext;
            if (context == null)
            {
                    throw new InvalidOperationException("HttpContext no
    está disponible.");
            }
                        return   context.Session.GetString("Rol")   ==
    requiredAccessLevel;
        }
    }
```

7. Implementación del `LoginController` y Vistas

- **ViewModel:** Cree el **`LoginViewModel`** (con `Username` y `Password`).
- **Controlador:** Cree **`LoginController.cs`** e inyecte **`IAuthenticationService`**.
- **Acciones:**
  - Implemente `[HttpGet] Index()` para mostrar la vista de login.
  - Implemente `[HttpPost] Login(ViewModel)` que llama a `_authService.Login()`. Redirige a `/Home/Index` si es exitoso o muestra error.
  - Implemente `Logout()` que llama a `_authService.Logout()` y redirige a `/Login/Index`.

C#

```
// implementación del LoginController.cs
using Microsoft.AspNetCore.Mvc;
using MVC.Interfaces;
using MVC.ViewModels;

public class LoginController : Controller
{
    private readonly IAuthenticationService _authenticationService;

    public LoginController(IAuthenticationService authenticationService)
    {
        _authenticationService = authenticationService;
    }

    // [HttpGet] Muestra la vista de login
    public IActionResult Index()
    {
        // ... (Crear LoginViewModel)
        return View(new LoginViewModel());
    }

    // [HttpPost] Procesa el login
    [HttpPost]
    public IActionResult Login(LoginViewModel model)
    {
        if (string.IsNullOrEmpty(model.Username) || string.IsNullOrEmpty(model.Password))
        {
            model.ErrorMessage = "Debe ingresar usuario y contraseña.";
            return View("Index", model);
        }

        if (_authenticationService.Login(model.Username, model.Password))
        {
            return RedirectToAction("Index", "Home");
        }
```

```
            model.ErrorMessage = "Credenciales inválidas.";
            return View("Index", model);
        }

        // [HttpGet] Cierra sesión
        public IActionResult Logout()
        {
            _authenticationService.Logout();
            return RedirectToAction("Index");
        }
    }
```

Fase 3: Autorización por Rol y Control de Acceso

Esta fase aplica las reglas de negocio de seguridad en cada acción de los controladores de negocio.

8. Acción de Acceso Denegado (UX de Error)

- **Acción:** Agregue la acción **public IActionResult AccesoDenegado()** en el **PresupuestosController.cs**.
- **Vista:** Cree la vista **AccesoDenegado.cshtml** con un mensaje informativo para usuarios logueados que intentan acceder sin permiso. Haga lo mismo para ProductosControllers.cs (personalice el mje de error que quiera mostrar)

C# AccesoDenegado.cshtml de presupuestos

```
@model dynamic
@{
    ViewData["Title"] = "Acceso Denegado";
}

<div class="row">

    <div class="col-md-10 mx-auto text-center">

        <h1 class="text-danger">❌ Acceso Denegado (Error 403)</h1>

        <hr />
```

```
            <h2>Lo sentimos.</h2>

            <p>            Su rol de **Cliente** no tiene permisos
    para acceder a esta funcionalidad.
                Esta sección está reservada para usuarios con el rol
    de **Administrador**.
            </p>

            <div class="mt-4">

                <a asp-action="Index" class="btn btn-primary">Volver
    a Listado de Presupuestos</a>

                <a asp-controller="Login" asp-action="Logout"
    class="btn btn-secondary">Cerrar Sesión</a>

            </div>

        </div>
    </div>
```

9. Protección de ProductosController (Acceso solo para Administrador)

- Al inicio de **TODAS** las acciones de ProductosController (Index, Details, Create, Edit, Delete), aplique la siguiente lógica:
    1. Si **!_authService.IsAuthenticated()**, redirigir a /Login/Index.
    2. Si **!_authService.HasAccessLevel("Administrador")**, redirigir a **Productos/AccesoDenegado**.

```
C#

public class ProductosController : Controller
{
    private  IProductoRepository _repo; //= new ProductoRepository();
    private  IAuthenticationService _authService;
 public ProductosController(IProductoRepository prodRepo,
IAuthenticationService authService)
    {
        //_repo = new ProductoRepository();
        _repo =prodRepo;

        _authService=authService;
```

```
    }

    public IActionResult Index()
    {

       // Aplicamos el chequeo de seguridad
        var securityCheck = CheckAdminPermissions();
        if (securityCheck != null) return securityCheck;

        List<Producto> productos = _repo.GetAll();
        return View(productos);
    }

    private IActionResult CheckAdminPermissions()
    {
        // 1. No logueado? -> vuelve al login
        if (!_authService.IsAuthenticated())
        {
            return RedirectToAction("Index", "Login");
        }

        // 2. No es Administrador? -> Da Error
        if (!_authService.HasAccessLevel("Administrador"))
        {
                    // Llamamos a AccesoDenegado (llama a la vista
correspondiente de Productos)
            return RedirectToAction(nameof(AccesoDenegado));
        }
        return null; // Permiso concedido
    }
    public IActionResult AccesoDenegado()
{
    // El usuario está logueado, pero no tiene el rol suficiente.
    return View();
}
// resto del código con las correspondientes correcciones
```

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 10

10. Protección de PresupuestosController (Habilitando permisos según rol)

- **Acciones de Modificación:** Al inicio de las acciones que cambian datos (Create, Edit, Delete, AddDetalle), aplique la lógica del punto 9 (solo **Administrador**).
- **Acciones de Lectura:** Al inicio de Index y Details, implemente el chequeo para ambos roles, redirigiendo a AccesoDenegado solo si no es ni Administrador ni Cliente:
    1. Si **!_authService.IsAuthenticated()**, redirigir a /Login/Index.
    2. Si **!(_authService.HasAccessLevel("Administrador") || _authService.HasAccessLevel("Cliente"))**, redirigir a **Presupuestos/AccesoDenegado**.

C#

```
 // ... (El resto de los using)
using MVC.Interfaces;
namespace MVC.Controllers;
public class PresupuestosController : Controller
{
        private  IPresupuestoRepository _repo; //= new
PresupuestoRepository();
        // Se necesita el repositorio de Productos para llenar los
Dropdowns
        private  IProductoRepository _productoRepo; //= new
ProductoRepository();
        private  IAuthenticationService _authService;
    public PresupuestosController(IPresupuestoRepository repo,
IProductoRepository prodRepo, IAuthenticationService authService)
        {
            _repo =repo;
            _productoRepo=prodRepo;
            _authService=authService;
        }

    //
------------------------------------------------------------------------
    // 1. LISTAR (INDEX)
    //
------------------------------------------------------------------------
    public IActionResult Index()
        {
            // Comprobación de si está logueado
        if (!_authService.IsAuthenticated())
        {
            return RedirectToAction("Index", "Login");
        }

        // Verifica Nivel de acceso que necesite validar
        if (_authService.HasAccessLevel("Administrador") ||
_authService.HasAccessLevel("Cliente") )
        {
            //si es es valido entra sino vuelve a login
        var presupuestos = _repo.GetAll();
        return View(presupuestos);
        }
        else
```

```
        {
            return RedirectToAction("Index", "Login");
        }


        }


 public IActionResult Create()
        {
         // Comprobación de si está logueado
           if (!_authService.IsAuthenticated())
        {

            return RedirectToAction("Index", "Login");
        }

         // Verifica Nivel de acceso
        if (!_authService.HasAccessLevel("Administrador"))
        {

            return RedirectToAction(nameof(AccesoDenegado));
        }
           // Se retorna un VM vacío para el formulario
            return View(new PresupuestoViewModel());
        }


   // ... (El resto de las acciones)
}
```