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.