
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).

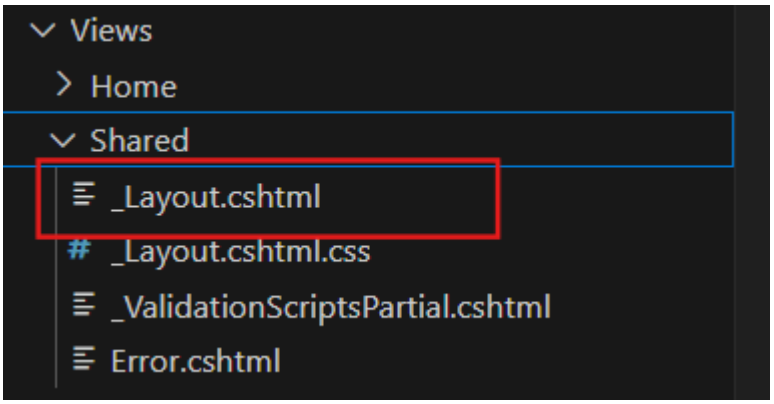
La documentación oficial de Bootstrap, junto con ejemplos y guías de uso, puede consultarse en su sitio web:

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.