Web MVC sencilla con CRUD

Iris Pérez Aparicio

2º DAM

Campus FP Emprende Humanes

Introducción

En esta actividad se va a implementar un CRUD (Create, Read, Update, Delete) básico. Se van a introducir conceptos esenciales de MVC como el manejo de formularios ([HttpPost]), el pase de datos entre acciones, y la gestión de una colección global (simulando una base de datos).

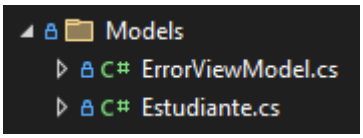
Añadir un nuevo proyecto a la solución WebMVCSencilla

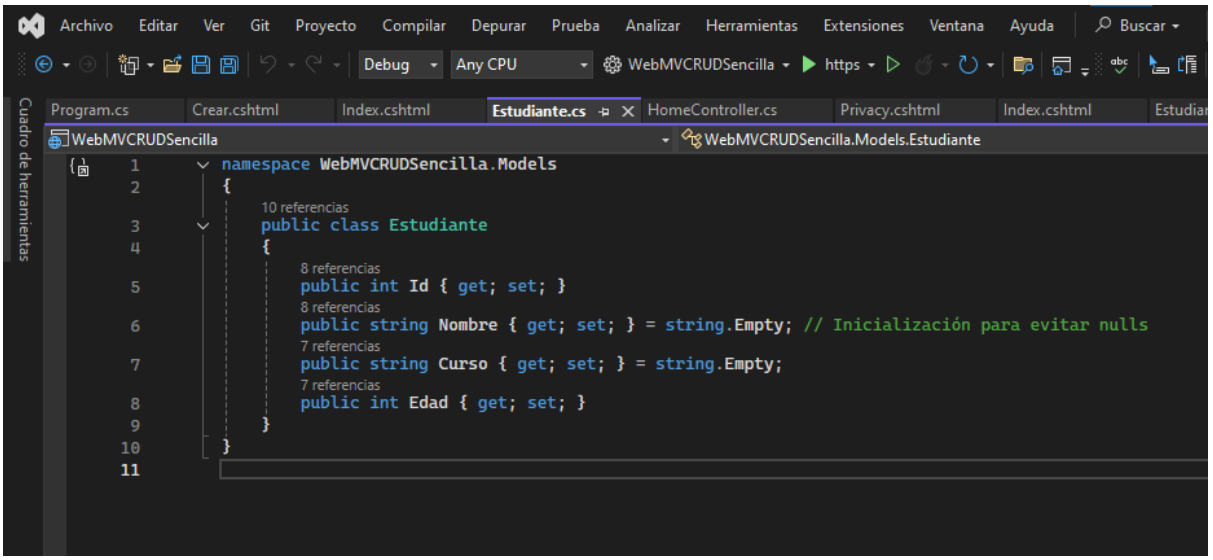
Añade a la solución WebMVCSencilla un nuevo proyecto de nombre WebMVCRUDSencilla y de tipo Aplicación web ASP.NET Core (Modelo-Vista-Controlador).

Definir el modelo de datos (Clase C#)

Crea el archivo Estudiante.cs en la carpeta Models y añade este código:

```
namespace WebMVCRUDSencilla.Models
{
    public class Estudiante
    {
        public int Id { get; set; }
        public string Nombre { get; set; } = string.Empty; // Inicialización para evitar nulls
        public string Curso { get; set; } = string.Empty;
        public int Edad { get; set; }
    }
}
```





CRUD

La lógica de la aplicación se centraliza en la carpeta Controllers. Usaremos el controlador predeterminado, HomeController.

Las vistas de la aplicación se centralizan en varios archivos de la carpeta Views.

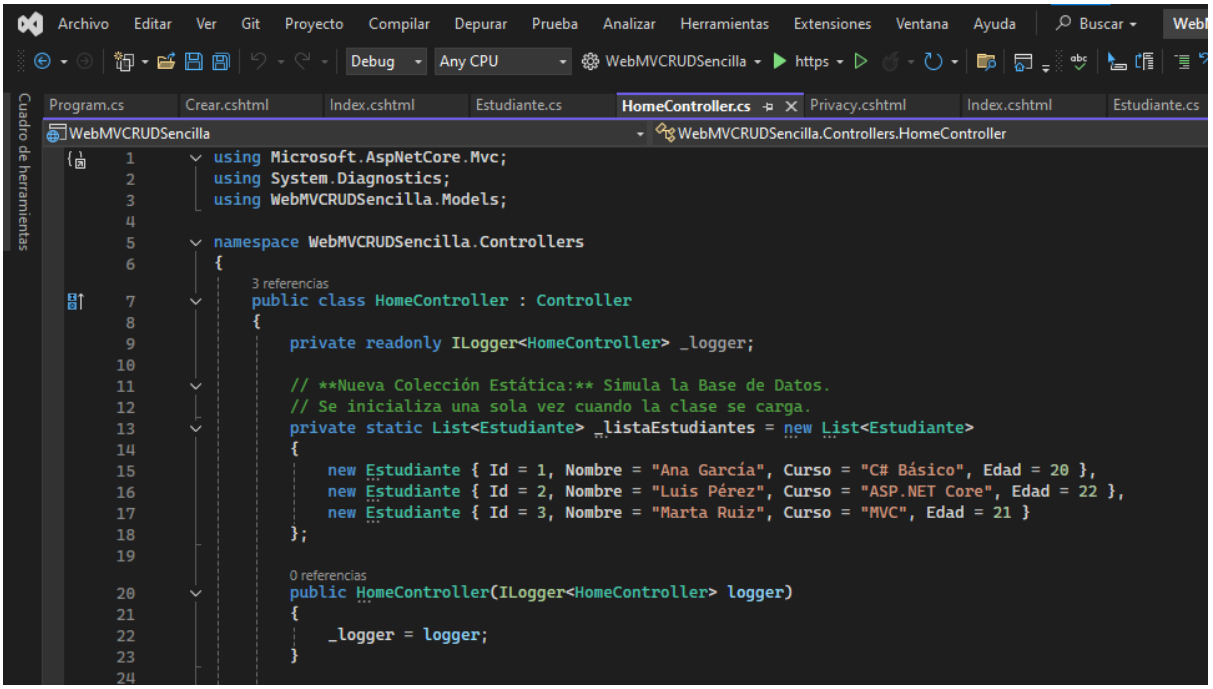
Crear datos

Controller

Edita el archivo `HomeController.cs` que está dentro de la carpeta `Controllers`.

Dado que no estamos usando una base de datos, necesitamos una manera de que la lista de estudiantes persista entre las diferentes solicitudes (acciones) del controlador. Usaremos una propiedad estática privada:

```
...
    public class HomeController : Controller
    {
        // **Nueva Colección Estática:** Simula la Base de Datos.
        // Se inicializa una sola vez cuando la clase se carga.
        private static List<Estudiante> _listaEstudiantes = new List<Estudiante>
        {
            new Estudiante { Id = 1, Nombre = "Ana García", Curso = "C# Básico", Edad = 20 },
            new Estudiante { Id = 2, Nombre = "Luis Pérez", Curso = "ASP.NET Core", Edad = 22 },
            new Estudiante { Id = 3, Nombre = "Marta Ruiz", Curso = "MVC", Edad = 21 }
        };
```

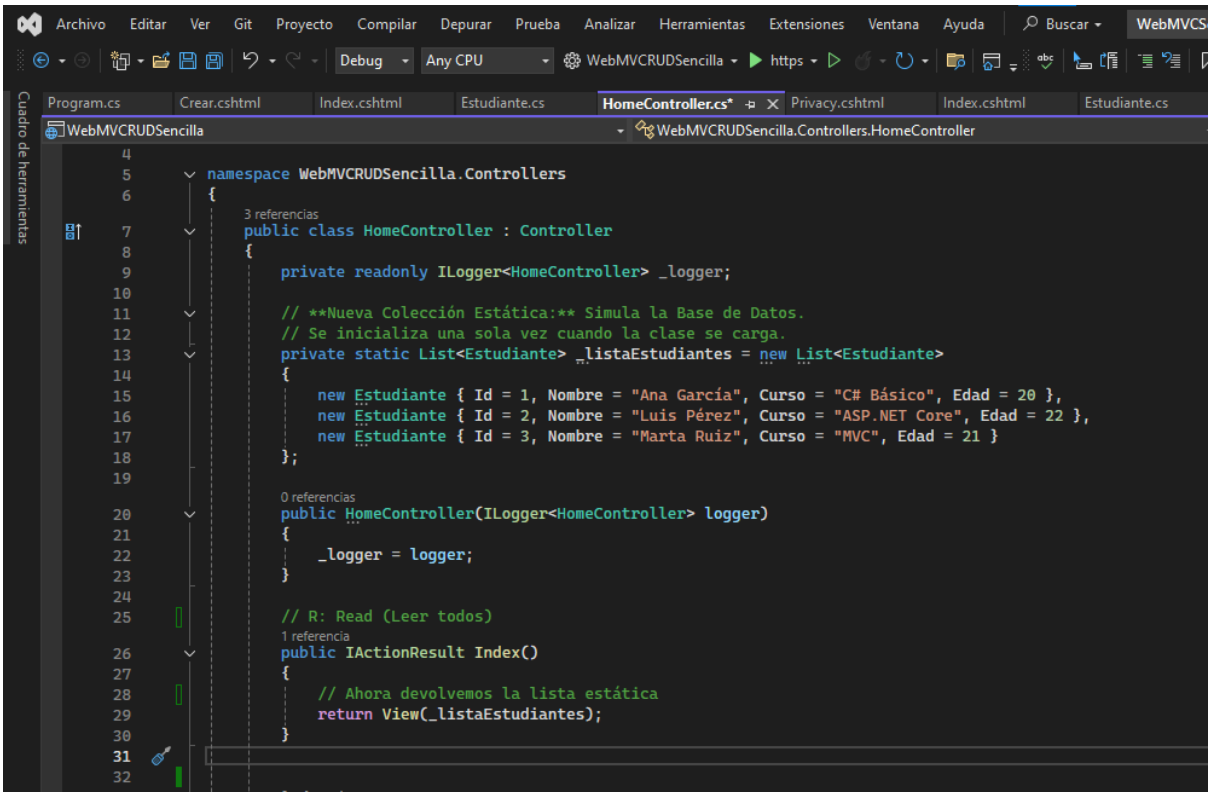


Read

Controller

Edita el archivo `HomeController.cs` que está dentro de la carpeta `Controllers` y modifica el método `index` para que implemente la operación read:

```
// R: Read (Leer todos)
public IActionResult Index()
{
    // Ahora devolvemos la lista estática
    return View(_listaEstudiantes);
}
```



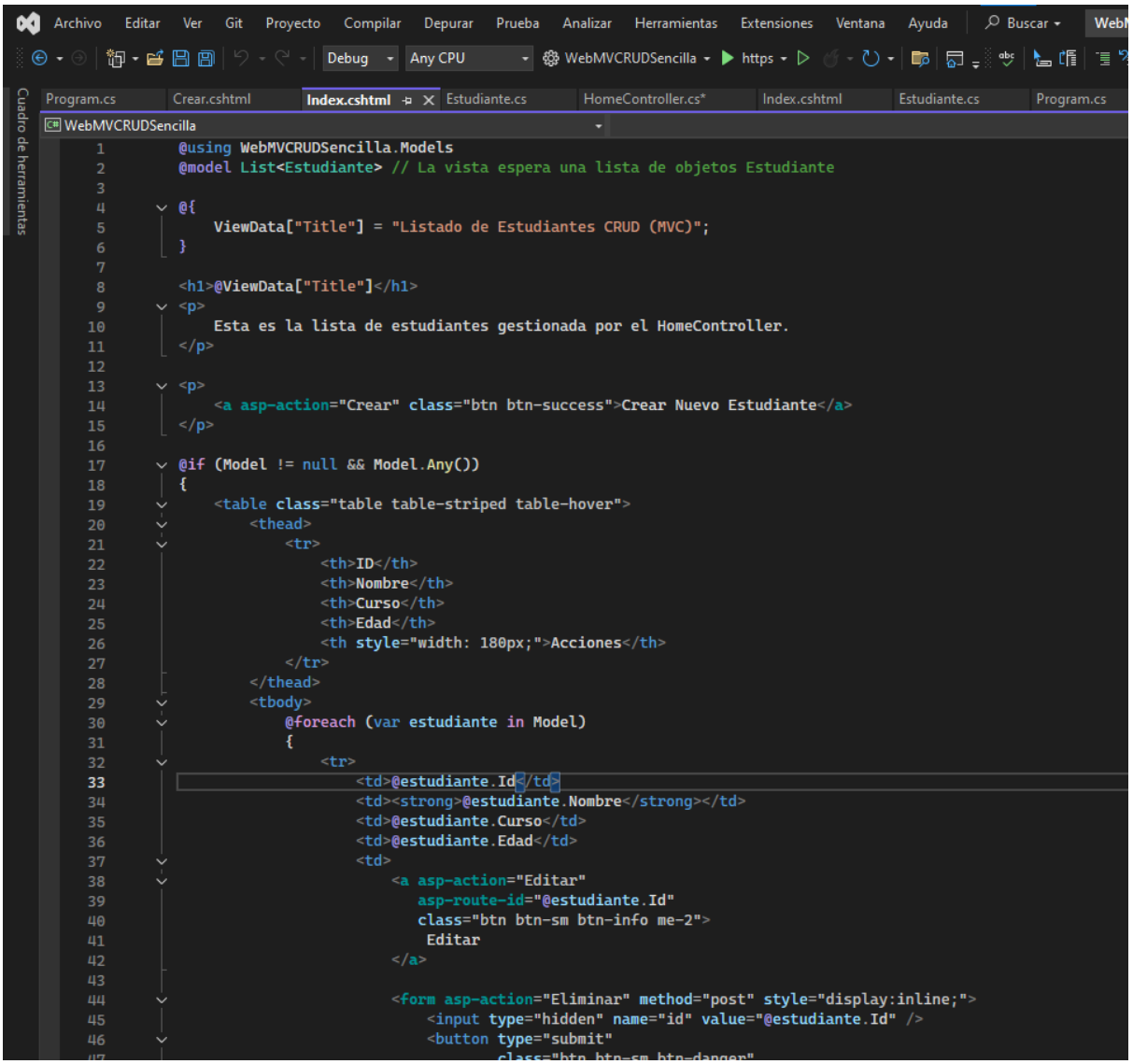
View

Edita el archivo `Index.cshtml` que está dentro de la carpeta `Views/Home` y reemplaza su código por el siguiente:

```
@using WebMVCRUDSencilla.Models
@model List<Estudiante> // La vista espera una lista de objetos Estudiante

@{
    ViewData["Title"] = "Listado de Estudiantes CRUD (MVC)";
}
```

```
<h1>@ViewData["Title"]</h1>
<p>
    Esta es la lista de estudiantes gestionada por el HomeController.
</p>

<p>
    <a asp-action="Crear" class="btn btn-success">Crear Nuevo Estudiante</a>
</p>

@if (Model != null && Model.Any())
{
    <table class="table table-striped table-hover">
        <thead>
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Curso</th>
                <th>Edad</th>
                <th style="width: 180px;">Acciones</th> </tr>
        </thead>
        <tbody>
            @foreach (var estudiante in Model)
            {
                <tr>
                    <td>@estudiante.Id</td>
                    <td><strong>@estudiante.Nombre</strong></td>
                    <td>@estudiante.Curso</td>
                    <td>@estudiante.Edad</td>
                    <td>
                        <a asp-action="Editar"
                           asp-route-id="@estudiante.Id"
                           class="btn btn-sm btn-info me-2">
                            Editar
                        </a>

                        <form asp-action="Eliminar" method="post" style="display:inline;">
                            <input type="hidden" name="id" value="@estudiante.Id" />
                            <button type="submit"
                                    class="btn btn-sm btn-danger"
                                    onclick="return confirm('¿Seguro que quieres eliminar a
@estudiante.Nombre?');">
                                Eliminar
                            </button>
                        </form>
                    </td>
                </tr>
            }
        </tbody>
```

```
    </table>
}
else
{
    <p class="alert alert-warning">No hay estudiantes en la colección para mostrar.</p>
}
```



Create

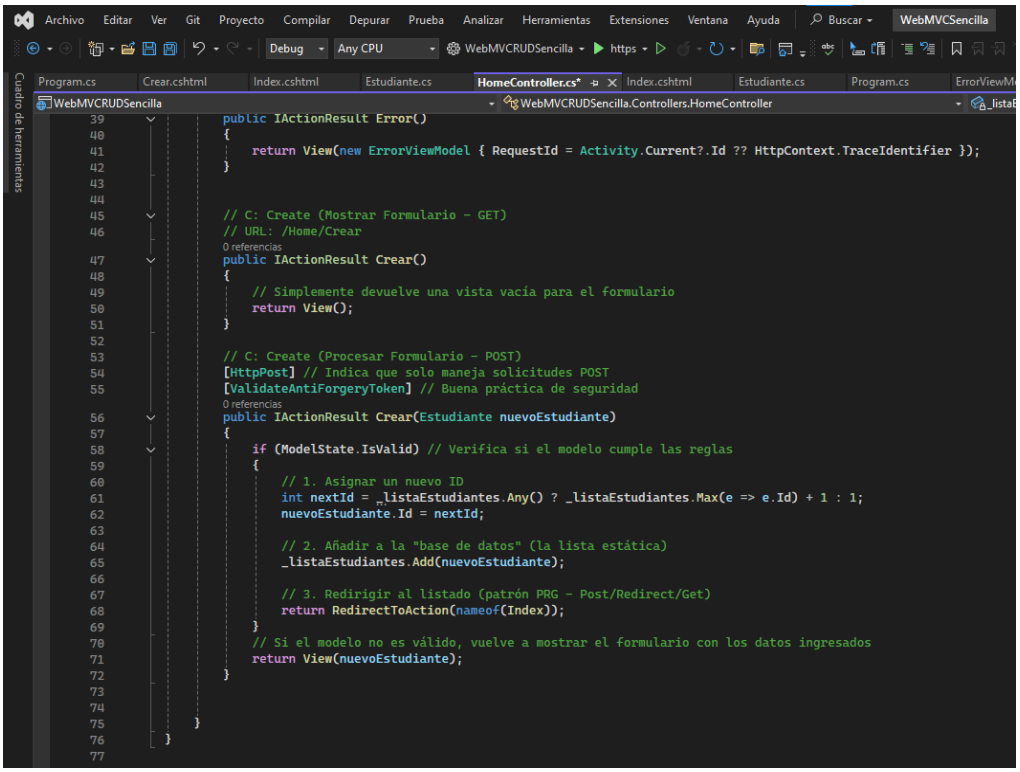
Controller

La creación requiere dos acciones en el controlador:

1. GET: Mostrar el formulario vacío.
2. POST: Recibir los datos del formulario y añadir el estudiante a la lista.

Edita el archivo HomeController.cs que está dentro de la carpeta Controllers y añade estos métodos para la operación crear:

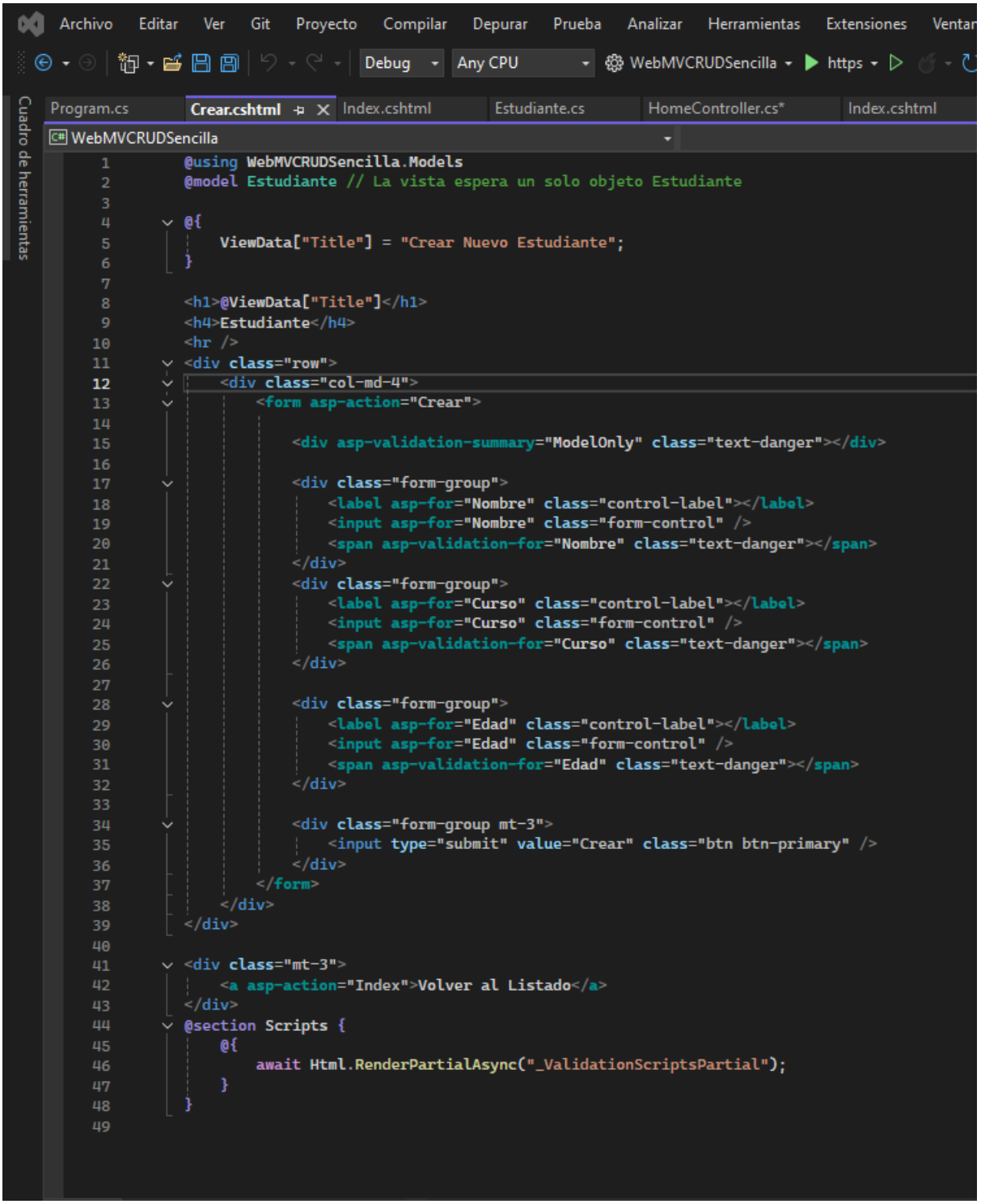
```
// C: Create (Mostrar Formulario - GET)
// URL: /Home/Crear
public IActionResult Crear()
{
    // Simplemente devuelve una vista vacía para el formulario
    return View();
}

// C: Create (Procesar Formulario - POST)
[HttpPost] // Indica que solo maneja solicitudes POST
[ValidateAntiForgeryToken] // Buena práctica de seguridad
public IActionResult Crear(Estudiante nuevoEstudiante)
{
    if (ModelState.IsValid) // Verifica si el modelo cumple las reglas
    {
        // 1. Asignar un nuevo ID
        int nextId = _listaEstudiantes.Any() ? _listaEstudiantes.Max(e => e.Id) + 1 : 1;
        nuevoEstudiante.Id = nextId;

        // 2. Añadir a la "base de datos" (la lista estática)
        _listaEstudiantes.Add(nuevoEstudiante);

        // 3. Redirigir al listado (patrón PRG - Post/Redirect/Get)
        return RedirectToAction(nameof(Index));
    }
    // Si el modelo no es válido, vuelve a mostrar el formulario con los datos ingresados
    return View(nuevoEstudiante);
}
```



View

Crea el archivo `Crear.cshtml` dentro de la carpeta `Views/Home` y reemplaza su código por el siguiente:

```
@using WebMVCRUDSencilla.Models
@model Estudiante // La vista espera un solo objeto Estudiante

@{
    ViewData["Title"] = "Crear Nuevo Estudiante";
}

<h1>@ViewData["Title"]</h1>
<h4>Estudiante</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Crear">

            <div asp-validation-summary="ModelOnly" class="text-danger"></div>

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

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

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

<div class="mt-3">
    <a asp-action="Index">Volver al Listado</a>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```


Archivo Editar Ver Git Proyecto Compilar Depurar Prueba Analizar Herramientas Extensiones Ventar
Debug
Any CPU
WebMVCRUDSencilla
https
Cuadro de herramientas
Program.cs
Crear.cshtml
Index.cshtml
Estudiante.cs
HomeController.cs*
Index.cshtml
WebMVCRUDSencilla
@using WebMVCRUDSencilla.Models
@model Estudiante // La vista espera un solo objeto Estudiante
@{
ViewData["Title"] = "Crear Nuevo Estudiante";
}
<h1>@ViewData["Title"]</h1>
<h4>Estudiante</h4>
<hr />

<form asp-action="Crear">

<label asp-for="Nombre" class="control-label"></label>
<input asp-for="Nombre" class="form-control" />

<label asp-for="Curso" class="control-label"></label>
<input asp-for="Curso" class="form-control" />

<label asp-for="Edad" class="control-label"></label>
<input asp-for="Edad" class="form-control" />

<input type="submit" value="Crear" class="btn btn-primary" />

</form>

<a asp-action="Index">Volver al Listado

@section Scripts {
@{
await Html.RenderPartialAsync("_ValidationScriptsPartial");
}
}

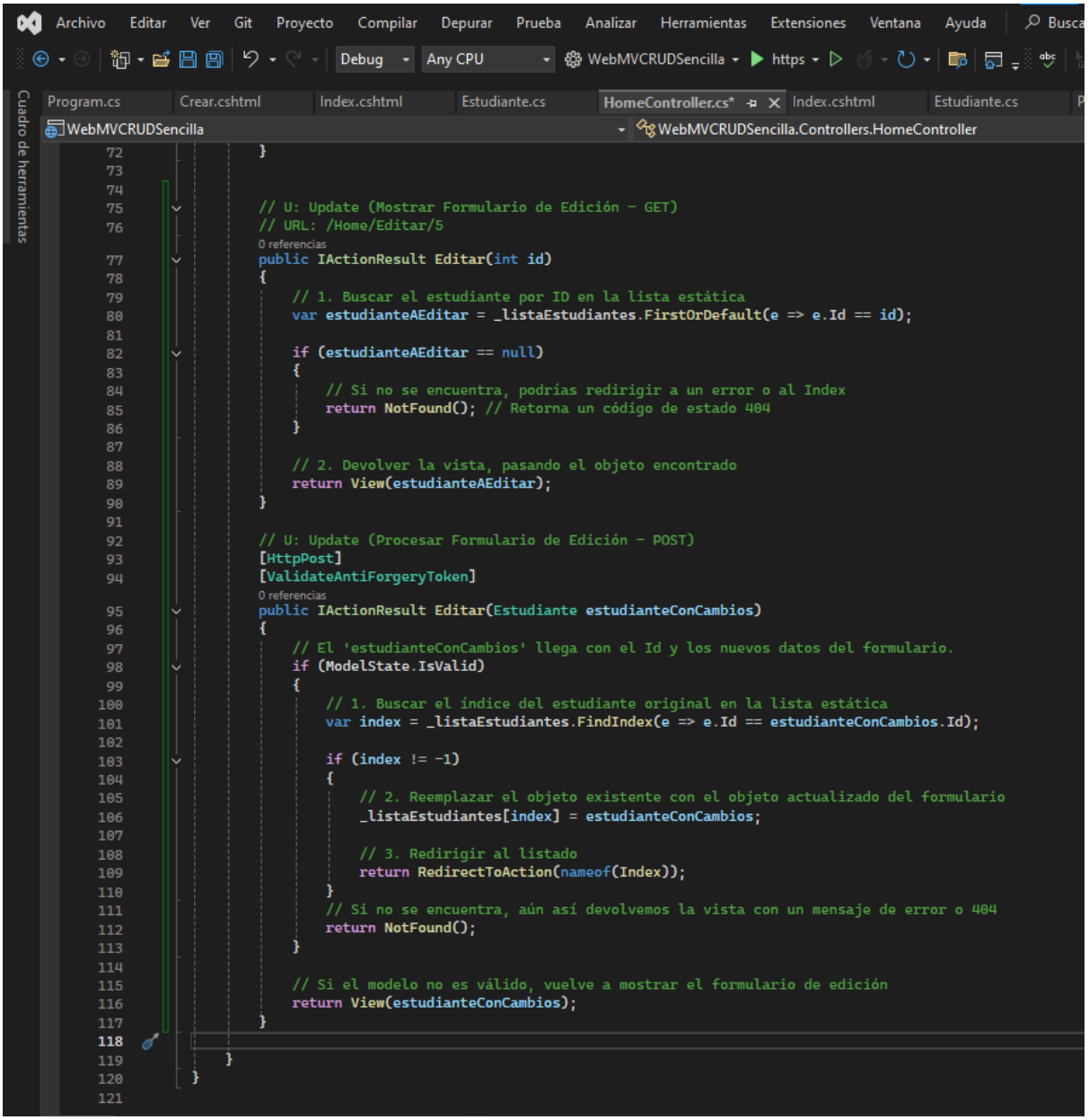
Update

Controller

La operación de edición, al igual que la de creación, requiere dos acciones en el controlador: una GET para mostrar el formulario pre-rellenado y una POST para guardar los cambios.

Edita el archivo `HomeController.cs` que está dentro de la carpeta `Controllers` y añade estos métodos para la operación update:

```
// U: Update (Mostrar Formulario de Edición - GET)
// URL: /Home/Editar/5
public IActionResult Editar(int id)
{
    // 1. Buscar el estudiante por ID en la lista estática
    var estudianteAEditar = _listaEstudiantes.FirstOrDefault(e => e.Id == id);

    if (estudianteAEditar == null)  {
        // Si no se encuentra, podrías redirigir a un error o al Index
        return NotFound(); // Retorna un código de estado 404
    }

    // 2. Devolver la vista, pasando el objeto encontrado
    return View(estudianteAEditar);
}

// U: Update (Procesar Formulario de Edición - POST)
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Editar(Estudiante estudianteConCambios) {
    // El 'estudianteConCambios' llega con el Id y los nuevos datos del formulario.
    if (ModelState.IsValid)  {
        // 1. Buscar el índice del estudiante original en la lista estática
        var index = _listaEstudiantes.FindIndex(e => e.Id == estudianteConCambios.Id);

        if (index != -1)  {
            // 2. Reemplazar el objeto existente con el objeto actualizado del formulario
            _listaEstudiantes[index] = estudianteConCambios;

            // 3. Redirigir al listado
            return RedirectToAction(nameof(Index));
        }
        // Si no se encuentra, aún así devolvemos la vista con un mensaje de error o 404
        return NotFound();
    }

    // Si el modelo no es válido, vuelve a mostrar el formulario de edición
    return View(estudianteConCambios);
}
```



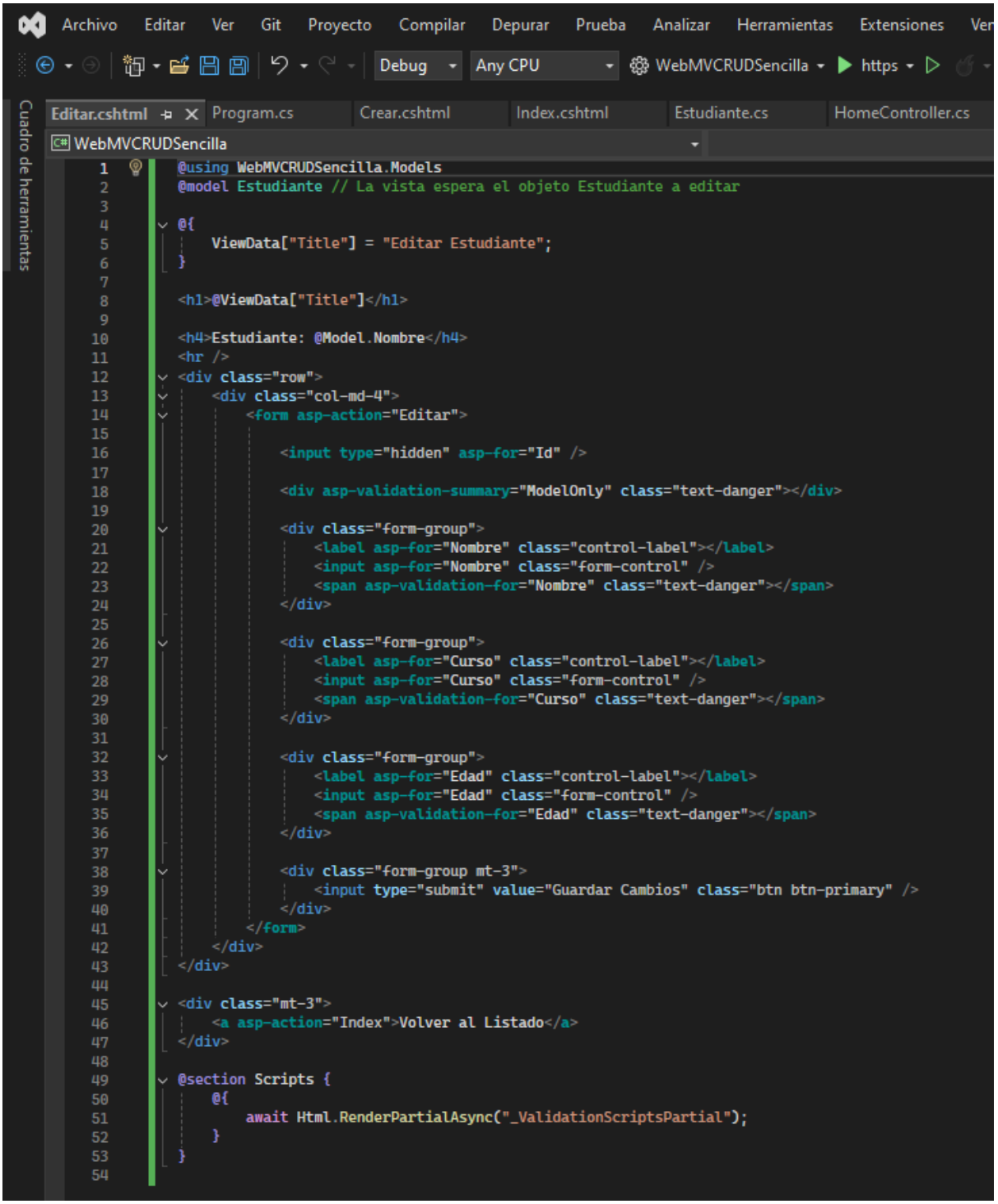
View

Este formulario es muy similar al de Crear.cshtml, pero está diseñado para recibir un objeto existente, mostrar sus valores, y enviar los cambios a la acción POST de Editar.

Crea el archivo Editar.cshtml dentro de la carpeta Views/Home y reemplaza su código por el siguiente:

```
@using WebMVCCRUDSencilla.Models
@model Estudiante // La vista espera el objeto Estudiante a editar

@{
    ViewData["Title"] = "Editar Estudiante";
}
```

```
<h1>@ViewData["Title"]</h1>

<h4>Estudiante: @Model.Nombre</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Editar">

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

            <div asp-validation-summary="ModelOnly" class="text-danger"></div>

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

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

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

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

<div class="mt-3">
    <a asp-action="Index">Volver al Listado</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```



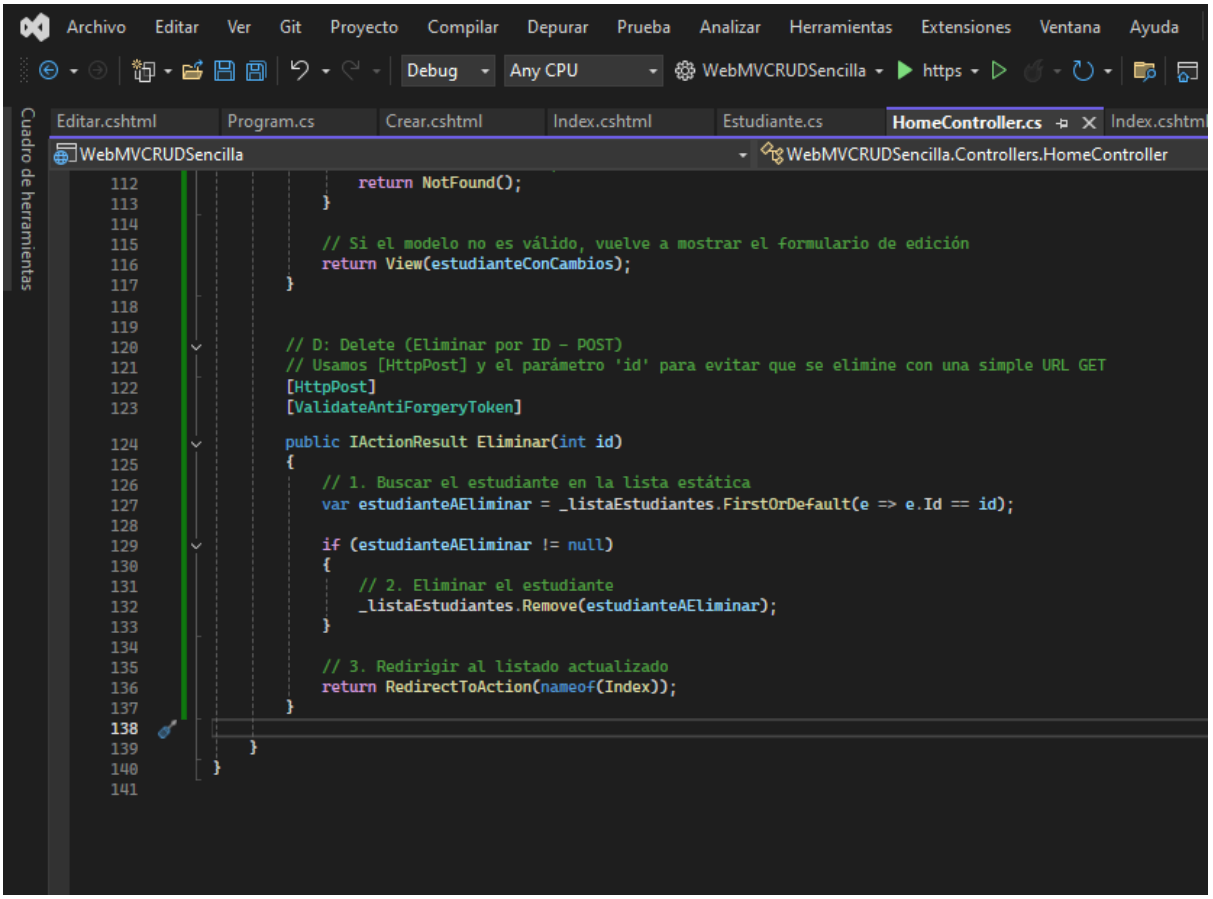
Delete

Controller

El borrado requiere de una acción Post en el controlador. El listado pasa el `Id` al controlador, y este elimina el elemento y redirige.

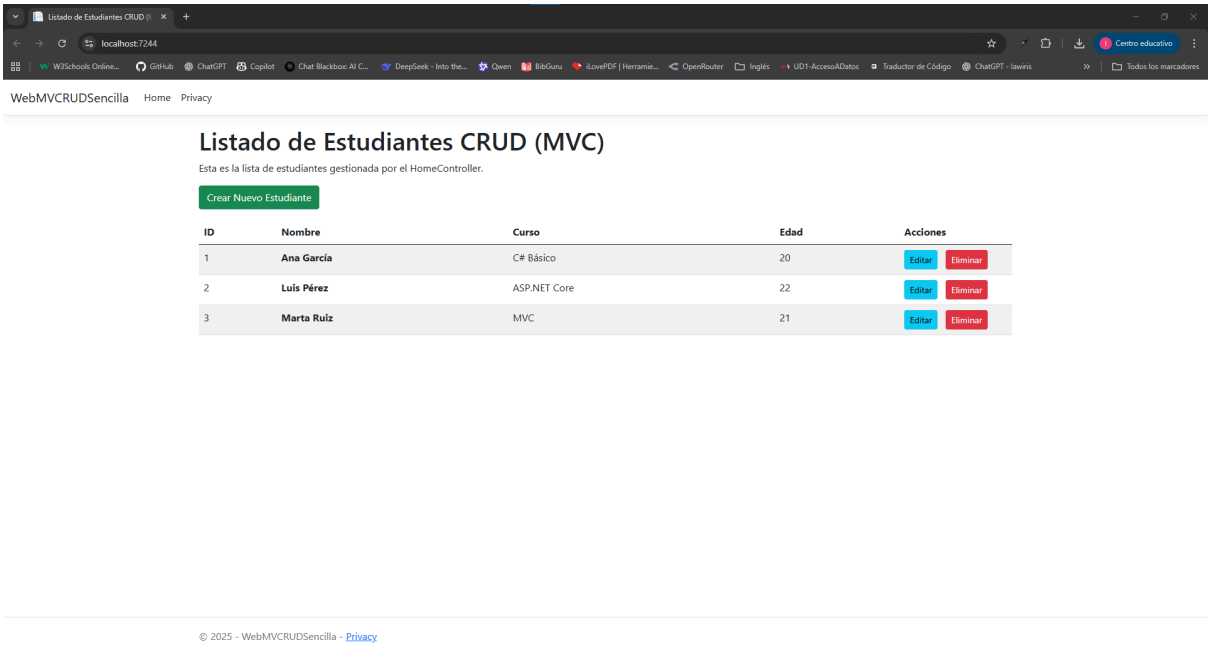
Edita el archivo `HomeController.cs` que está dentro de la carpeta `Controllers` y añade estos métodos para la operación delete:

```
// D: Delete (Eliminar por ID - POST)
// Usamos [HttpPost] y el parámetro 'id' para evitar que se elimine con una simple URL GET
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Eliminar(int id)
{
    // 1. Buscar el estudiante en la lista estática
    var estudianteAEliminar = _listaEstudiantes.FirstOrDefault(e => e.Id == id);

    if (estudianteAEliminar != null)
    {
        // 2. Eliminar el estudiante
        _listaEstudiantes.Remove(estudianteAEliminar);
    }

    // 3. Redirigir al listado actualizado
    return RedirectToAction(nameof(Index));
}
```

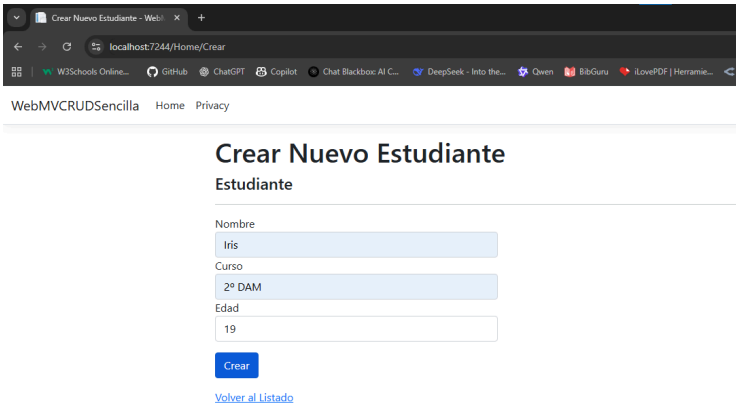


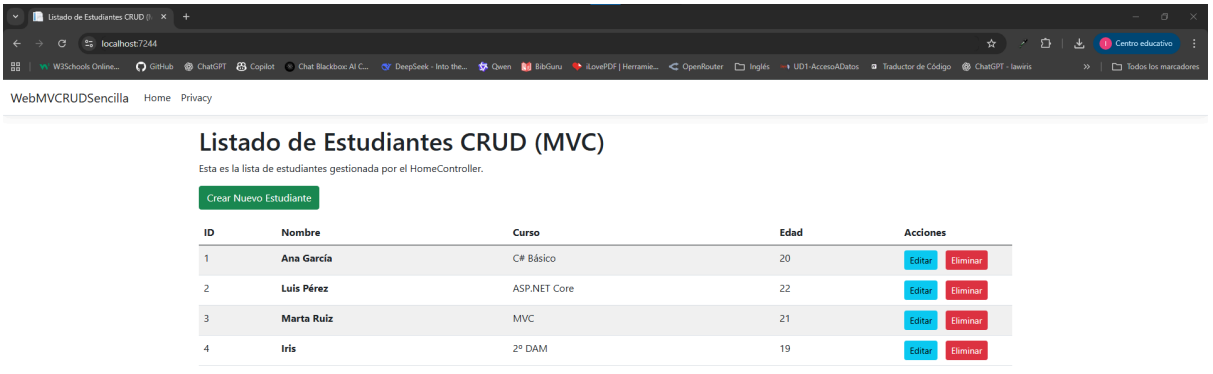
Ejecución:

- INICIO

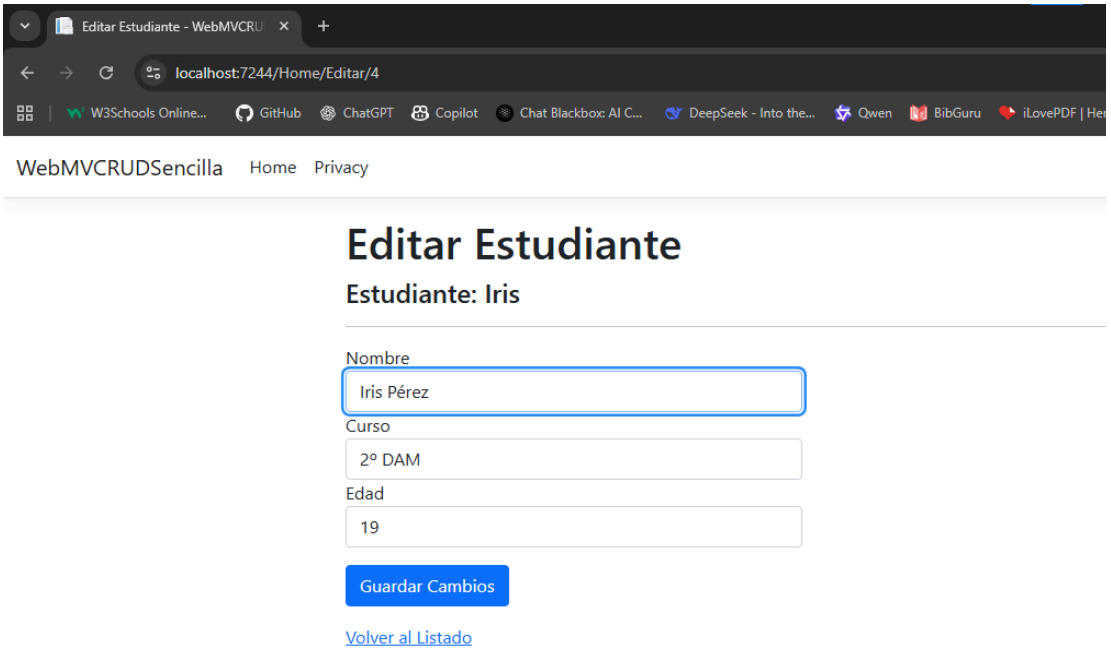


- Crear un estudiante

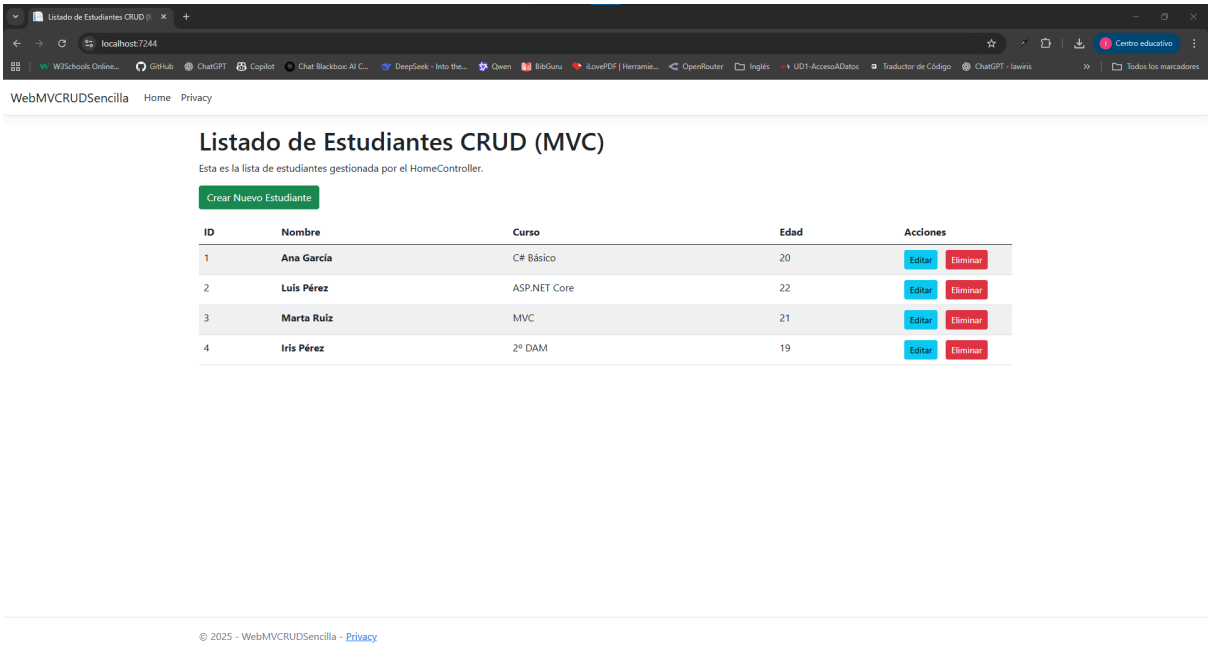




- Editar (cambio el nombre para añadir el apellido)



(Hago clic en 'Guardar Cambios' y se actualizan los datos)



- Eliminar (elimino al estudiante ‘Luis Pérez’)

(Se actualiza la lista de estudiantes)