

Web MVC sencilla con CRUD

Iris Pérez Aparicio

2º DAM

Campus FP Emprende Humanes

Introducción	4
Añadir un nuevo proyecto a la solución WebMVCsencilla	4
Definir el modelo de datos (Clase C#)	4
CRUD	5
Crear datos	5
Controller	5
Read	5
Controller	5
View	5
Create	7
Controller	7
View	8
Update	9
Controller	9
View	10
Delete	11
Controller	11

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).

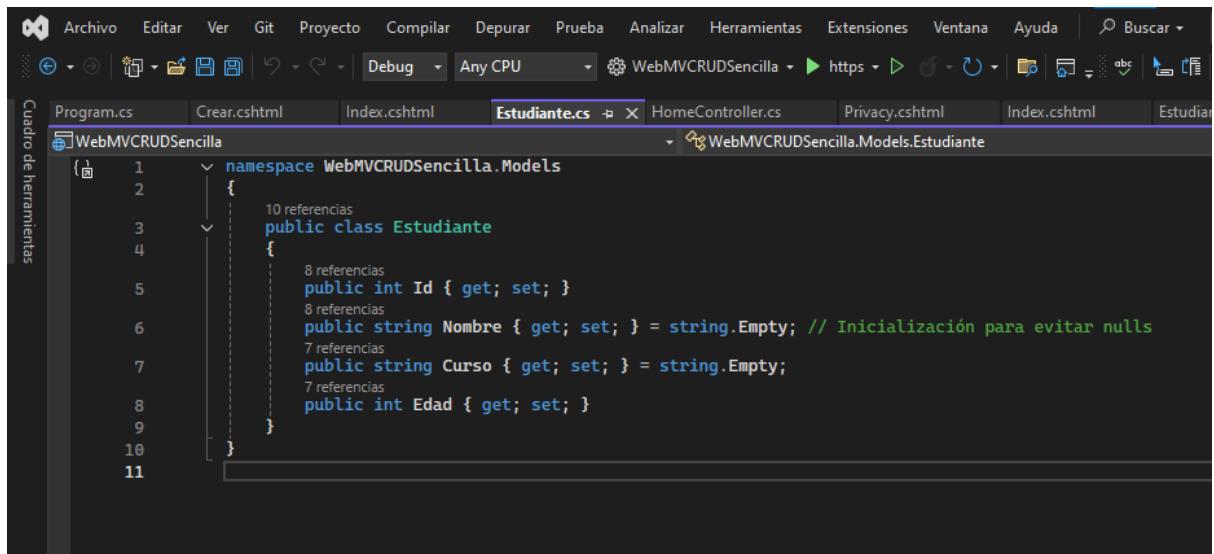
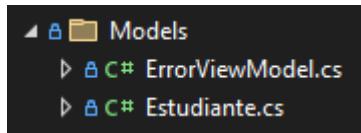
Añadir un nuevo proyecto a la solución WebMVCsencilla

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

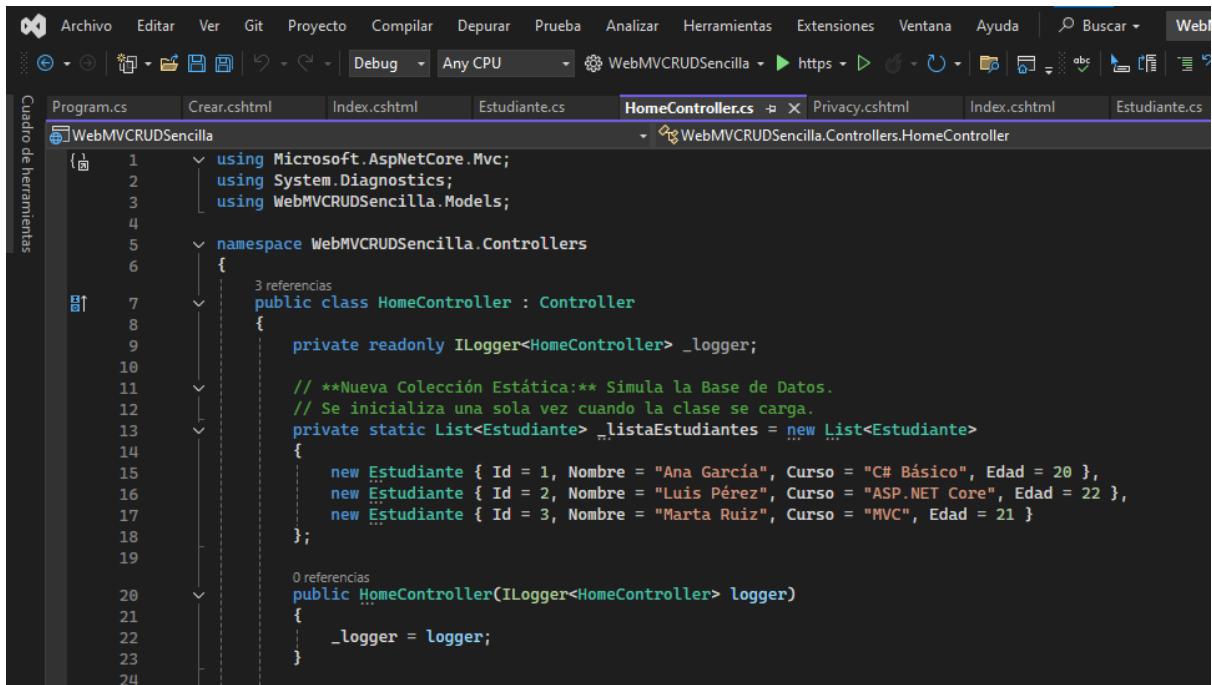
Controller

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 }
    };
}
```



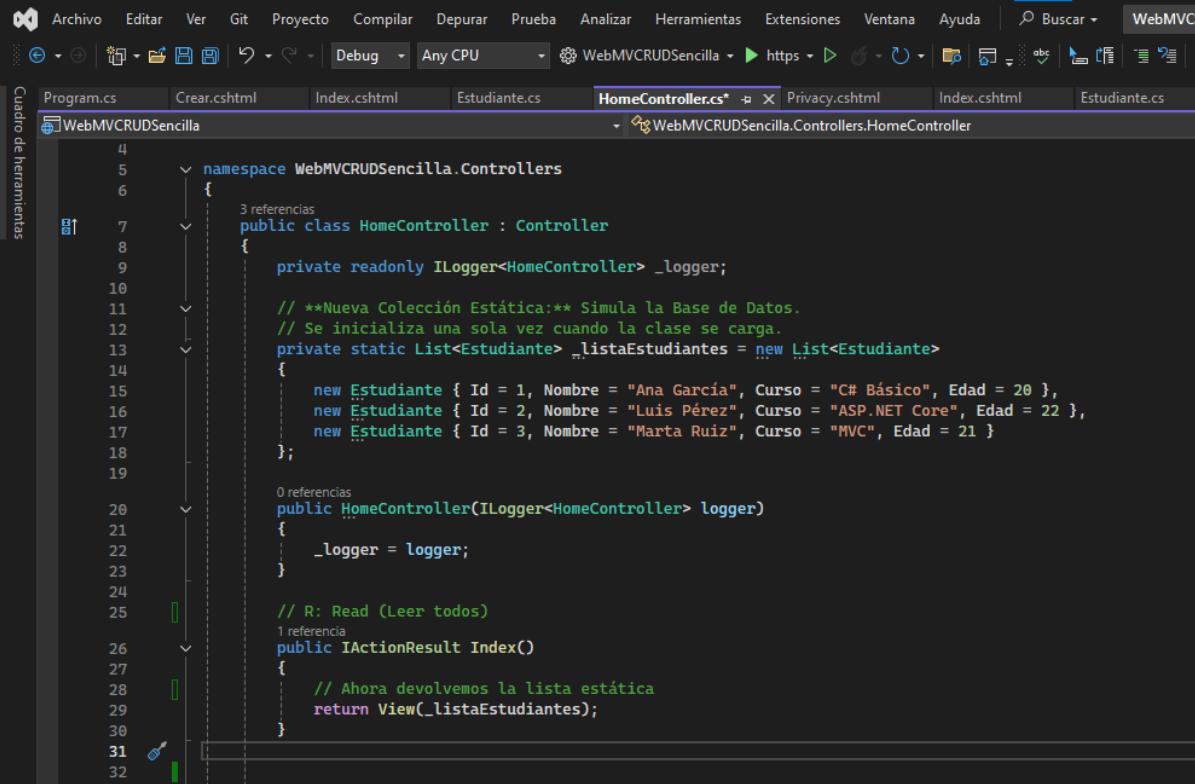
```
1  using Microsoft.AspNetCore.Mvc;
2  using System.Diagnostics;
3  using WebMVCRUDSencilla.Models;
4
5  namespace WebMVCRUDSencilla.Controllers
6  {
7      public class HomeController : Controller
8      {
9          private readonly ILogger<HomeController> _logger;
10
11         // **Nueva Colección Estática:** Simula la Base de Datos.
12         // Se inicializa una sola vez cuando la clase se carga.
13         private static List<Estudiante> _listaEstudiantes = new List<Estudiante>
14         {
15             new Estudiante { Id = 1, Nombre = "Ana García", Curso = "C# Básico", Edad = 20 },
16             new Estudiante { Id = 2, Nombre = "Luis Pérez", Curso = "ASP.NET Core", Edad = 22 },
17             new Estudiante { Id = 3, Nombre = "Marta Ruiz", Curso = "MVC", Edad = 21 }
18         };
19
20         public HomeController(ILogger<HomeController> logger)
21         {
22             _logger = logger;
23         }
24     }
```

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);
}
```



```
4
5     namespace WebMVCRUDSencilla.Controllers
6     {
7         public class HomeController : Controller
8         {
9             private readonly ILogger<HomeController> _logger;
10            // **Nueva Colección Estática:** Simula la Base de Datos.
11            // Se inicializa una sola vez cuando la clase se carga.
12            private static List<Estudiante> _listaEstudiantes = new List<Estudiante>
13            {
14                new Estudiante { Id = 1, Nombre = "Ana García", Curso = "C# Básico", Edad = 20 },
15                new Estudiante { Id = 2, Nombre = "Luis Pérez", Curso = "ASP.NET Core", Edad = 22 },
16                new Estudiante { Id = 3, Nombre = "Marta Ruiz", Curso = "MVC", Edad = 21 }
17            };
18        }
19        public HomeController(ILogger<HomeController> logger)
20        {
21            _logger = logger;
22        }
23        // R: Read (Leer todos)
24        [HttpGet]
25        public IActionResult Index()
26        {
27            // Ahora devolvemos la lista estática
28            return View(_listaEstudiantes);
29        }
30    }
31}
```

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>
}

```

```

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

```

The screenshot shows the Visual Studio IDE interface with the following details:

- Toolbar:** Archivo, Editar, Ver, Git, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda, Buscar.
- Solution Explorer:** Shows files like Program.cs, Crear.cshtml, Index.cshtml, Estudiante.cs, HomeController.cs, Index.cshtml, Estudiante.cs, and Program.cs.
- Code Editor:** The Index.cshtml file contains C# Razor code for generating an HTML table of student data. It includes columns for ID, Nombre, Curso, Edad, and Acciones (with links for Editar and Eliminar).
- Status Bar:** Shows the project name "WebMVCURDSencilla", build configuration "Any CPU", and the URL "https://localhost:44361/".

```

1     @using WebMVCURDSencilla.Models
2     @model List<Estudiante> // La vista espera una lista de objetos Estudiante
3
4     @{
5         ViewData["Title"] = "Listado de Estudiantes CRUD (MVC)";
6     }
7
8     <h1>@ViewData["Title"]</h1>
9     <p>
10        Esta es la lista de estudiantes gestionada por el HomeController.
11    </p>
12
13    <p>
14        <a asp-action="Crear" class="btn btn-success">Crear Nuevo Estudiante</a>
15    </p>
16
17    @if (Model != null && Model.Any())
18    {
19        <table class="table table-striped table-hover">
20            <thead>
21                <tr>
22                    <th>ID</th>
23                    <th>Nombre</th>
24                    <th>Curso</th>
25                    <th>Edad</th>
26                    <th style="width: 180px;">Acciones</th>
27                </tr>
28            </thead>
29            <tbody>
30                @foreach (var estudiante in Model)
31                {
32                    <tr>
33                        <td>@estudiante.Id</td>
34                        <td><strong>@estudiante.Nombre</strong></td>
35                        <td>@estudiante.Curso</td>
36                        <td>@estudiante.Edad</td>
37                        <td>
38                            <a asp-action="Editar"
39                                asp-route-id="@estudiante.Id"
40                                class="btn btn-sm btn-info me-2">
41                                Editar
42                            </a>
43
44                            <form asp-action="Eliminar" method="post" style="display:inline;">
45                                <input type="hidden" name="id" value="@estudiante.Id" />
46                                <button type="submit"
47                                    class="btn btn-sm btn-danger">
48                                    Eliminar
49                            </form>
50                        </td>
51                    </tr>
52                }
53            </tbody>
54        </table>
55    }
56
```

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.

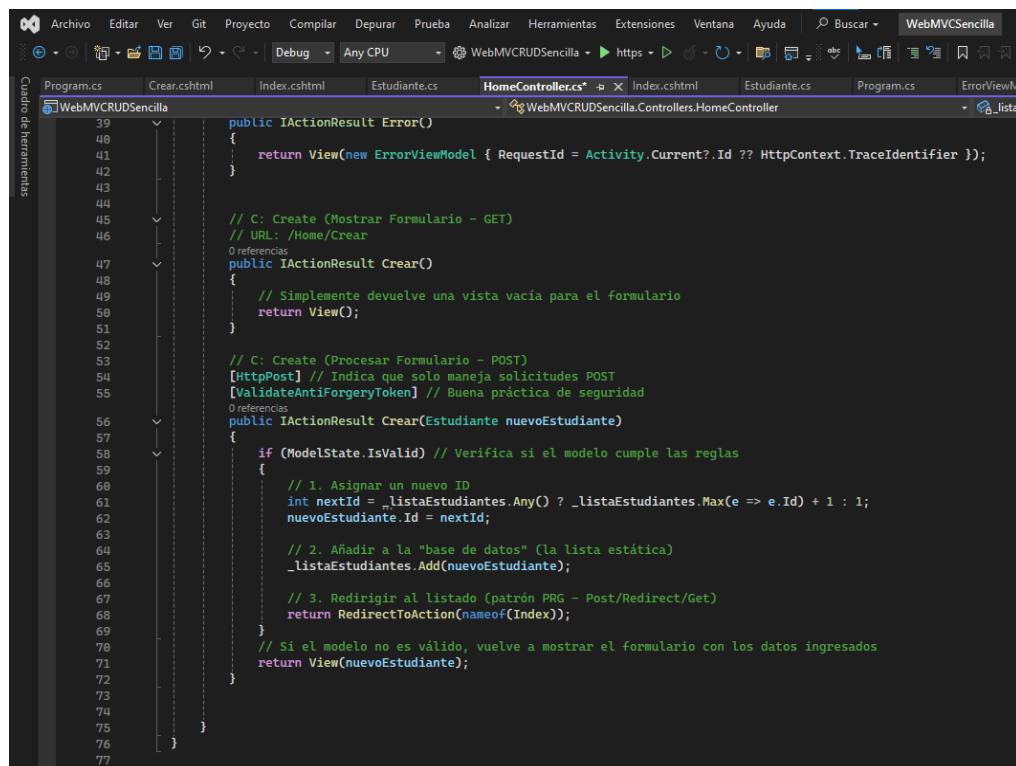
Edita el archivo `HomeController.cs` que está dentro de la carpeta `Controllers` y añade estos métodos para la operación crear:

```
// 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);
}
```



The screenshot shows the Visual Studio IDE interface with the code editor open. The file being edited is `HomeController.cs`. The code implements the `Create` action method for both GET and POST requests. The GET version simply returns an empty view. The POST version checks if the model state is valid. If valid, it assigns a new ID to the new student (either the max ID plus one or 1 if there are none), adds the student to the static list, and then redirects to the index action. If the model state is not valid, it returns the view with the invalid data.

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");}  
}
```

The screenshot shows the Visual Studio IDE interface with the following details:

- Menu Bar:** Archivo, Editar, Ver, Git, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana.
- Toolbar:** Includes icons for Undo, Redo, Save, and various project-related functions.
- Current Project:** WebMVCRUDSencilla
- Current View:** Create.cshtml
- Code Editor Content:**

```
1     @using WebMVCRUDSencilla.Models
2     @model Estudiante // La vista espera un solo objeto Estudiante
3
4     @{
5         ViewData["Title"] = "Crear Nuevo Estudiante";
6     }
7
8     <h1>@ViewData["Title"]</h1>
9     <h4>Estudiante</h4>
10    <hr />
11    <div class="row">
12        <div class="col-md-4">
13            <form asp-action="Crear">
14
15                <div asp-validation-summary="ModelOnly" class="text-danger"></div>
16
17                <div class="form-group">
18                    <label asp-for="Nombre" class="control-label"></label>
19                    <input asp-for="Nombre" class="form-control" />
20                    <span asp-validation-for="Nombre" class="text-danger"></span>
21                </div>
22
23                <div class="form-group">
24                    <label asp-for="Curso" class="control-label"></label>
25                    <input asp-for="Curso" class="form-control" />
26                    <span asp-validation-for="Curso" class="text-danger"></span>
27                </div>
28
29                <div class="form-group">
30                    <label asp-for="Edad" class="control-label"></label>
31                    <input asp-for="Edad" class="form-control" />
32                    <span asp-validation-for="Edad" class="text-danger"></span>
33                </div>
34
35                <div class="form-group mt-3">
36                    <input type="submit" value="Crear" class="btn btn-primary" />
37                </div>
38            </form>
39        </div>
40    <div class="mt-3">
41        <a asp-action="Index">Volver al Listado</a>
42    </div>
43    <@section Scripts {
44        @{
45            await Html.RenderPartialAsync("_ValidationScriptsPartial");
46        }
47    }>
48
49
```

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 estudianteAEellar = _listaEstudiantes.FirstOrDefault(e => e.Id == id);

    if (estudianteAEellar == 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(estudianteAEellar);
}

// 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);
}
```

The screenshot shows the Visual Studio IDE interface with the code editor open to the `HomeController.cs` file. The code implements the `Edit` action for updating student records. It includes logic to find a student by ID, validate the form, and update the database. The code is annotated with comments explaining each step.

```
72 }  
73  
74 // U: Update (Mostrar Formulario de Edición - GET)  
75 // URL: /Home/Editar/5  
76 0 referencias  
77 public IActionResult Editar(int id)  
78 {  
79     // 1. Buscar el estudiante por ID en la lista estática  
80     var estudianteAEellar = _listaEstudiantes.FirstOrDefault(e => e.Id == id);  
81  
82     if (estudianteAEellar == null)  
83     {  
84         // Si no se encuentra, podrías redirigir a un error o al Index  
85         return NotFound(); // Retorna un código de estado 404  
86     }  
87  
88     // 2. Devolver la vista, pasando el objeto encontrado  
89     return View(estudianteAEellar);  
90 }  
91  
92 // U: Update (Procesar Formulario de Edición - POST)  
93 [HttpPost]  
94 [ValidateAntiForgeryToken]  
95 0 referencias  
96 public IActionResult Editar(Estudiante estudianteConCambios)  
97 {  
98     // El 'estudianteConCambios' llega con el Id y los nuevos datos del formulario.  
99     if (ModelState.IsValid)  
100    {  
101        // 1. Buscar el índice del estudiante original en la lista estática  
102        var index = _listaEstudiantes.FindIndex(e => e.Id == estudianteConCambios.Id);  
103  
104        if (index != -1)  
105        {  
106            // 2. Reemplazar el objeto existente con el objeto actualizado del formulario  
107            _listaEstudiantes[index] = estudianteConCambios;  
108  
109            // 3. Redirigir al listado  
110            return RedirectToAction(nameof(Index));  
111        }  
112        // Si no se encuentra, aún así devolvemos la vista con un mensaje de error o 404  
113        return NotFound();  
114    }  
115  
116    // Si el modelo no es válido, vuelve a mostrar el formulario de edición  
117    return View(estudianteConCambios);  
118 }  
119  
120 }  
121 }
```

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 WebMVCRUDSencilla.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");}
}

```

```
1 @using WebMVCRUDSencilla.Models
2 @model Estudiante // La vista espera el objeto Estudiante a editar
3
4 @{
5     ViewData["Title"] = "Editar Estudiante";
6 }
7
8 <h1>@ViewData["Title"]</h1>
9
10 <h4>Estudiante: @Model.Nombre</h4>
11 <hr />
12 <div class="row">
13     <div class="col-md-4">
14         <form asp-action="Editar">
15             <input type="hidden" asp-for="Id" />
16
17             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
18
19             <div class="form-group">
20                 <label asp-for="Nombre" class="control-label"></label>
21                 <input asp-for="Nombre" class="form-control" />
22                 <span asp-validation-for="Nombre" class="text-danger"></span>
23             </div>
24
25             <div class="form-group">
26                 <label asp-for="Curso" class="control-label"></label>
27                 <input asp-for="Curso" class="form-control" />
28                 <span asp-validation-for="Curso" class="text-danger"></span>
29             </div>
30
31             <div class="form-group">
32                 <label asp-for="Edad" class="control-label"></label>
33                 <input asp-for="Edad" class="form-control" />
34                 <span asp-validation-for="Edad" class="text-danger"></span>
35             </div>
36
37             <div class="form-group mt-3">
38                 <input type="submit" value="Guardar Cambios" class="btn btn-primary" />
39             </div>
40         </form>
41     </div>
42 </div>
43
44 <div class="mt-3">
45     <a asp-action="Index">Volver al Listado</a>
46 </div>
47
48 @section Scripts {
49     @{
50         await Html.RenderPartialAsync("_ValidationScriptsPartial");
51     }
52 }
53
54 }
```

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));
}

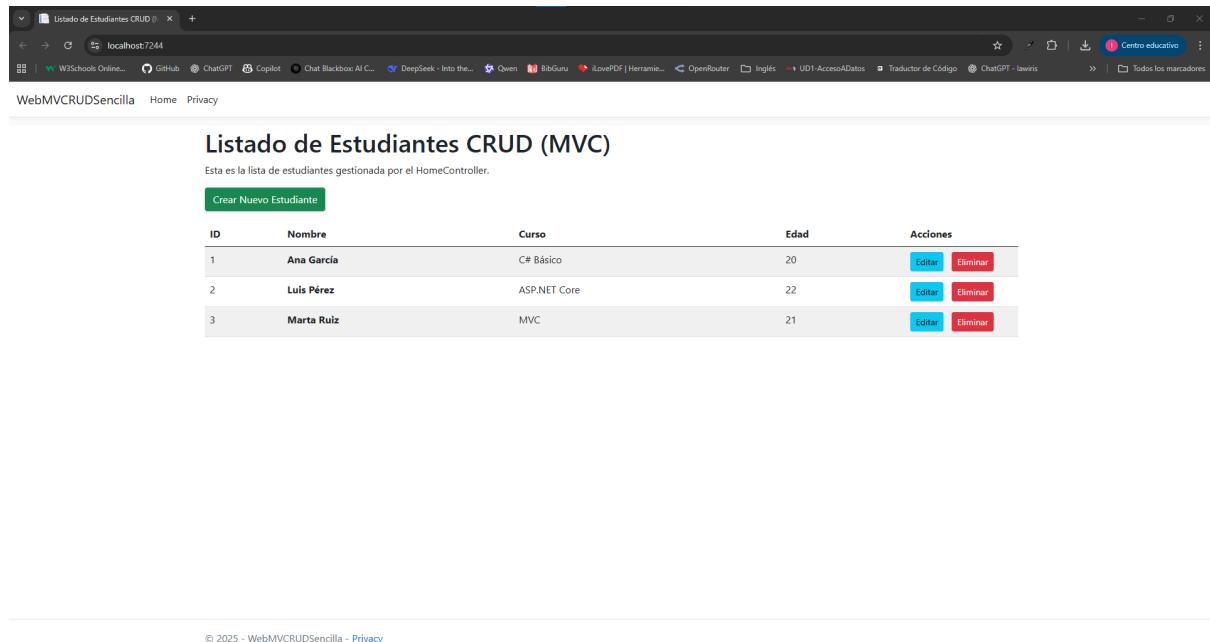
```

The screenshot shows the Visual Studio IDE interface with the following details:

- Toolbar:** Archivo, Editar, Ver, Git, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda.
- Toolbars:** Standard toolbar with icons for file operations, search, and navigation.
- Project Explorer:** Shows the project structure with files like Editar.cshtml, Program.cs, Crear.cshtml, Index.cshtml, Estudiante.cs, HomeController.cs, and Index.cshtml.
- Code Editor:** The main window displays the `HomeController.cs` file content. The code is identical to the one shown in the previous code block, handling the deletion of a student record from a static list.
- Status Bar:** Shows the project name "WebMVCRUDSencilla", the URL "https://", and other build-related information.

Ejecución:

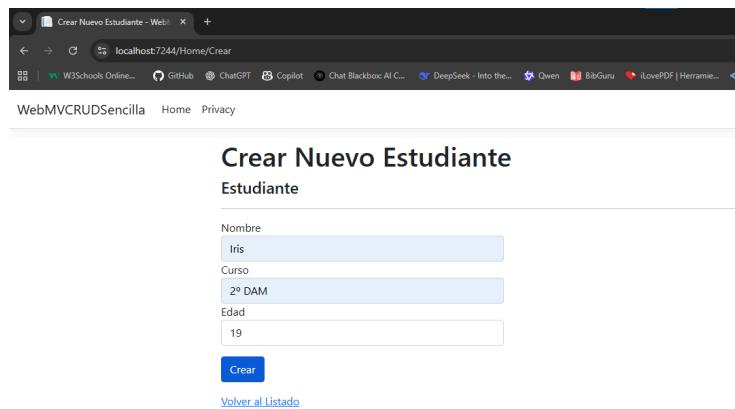
- INICIO



Este es la lista de estudiantes gestionada por el HomeController.

ID	Nombre	Curso	Edad	Acciones
1	Ana García	C# Básico	20	Editar Eliminar
2	Luis Pérez	ASP.NET Core	22	Editar Eliminar
3	Marta Ruiz	MVC	21	Editar Eliminar

- Crear un estudiante



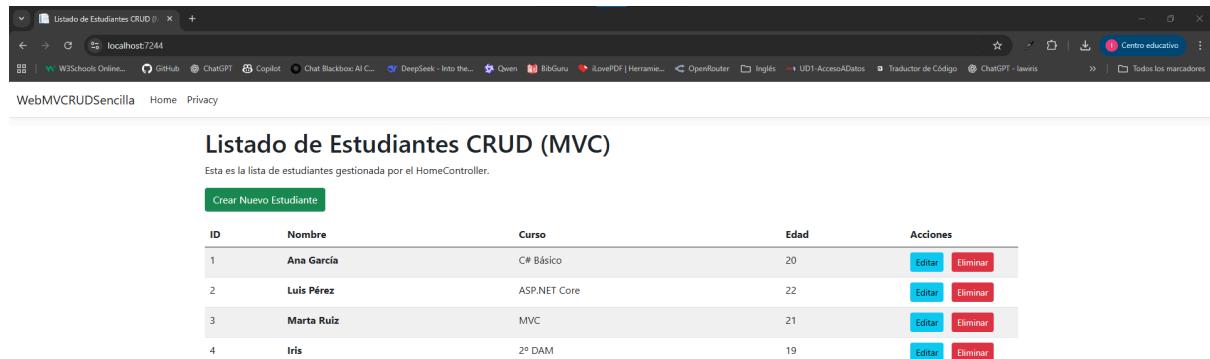
Nombre
Iris

Curso
2º DAM

Edad
19

[Crear](#)

[Volver al Listado](#)



Este es la lista de estudiantes gestionada por el HomeController.

ID	Nombre	Curso	Edad	Acciones
1	Ana García	C# Básico	20	Editar Eliminar
2	Luis Pérez	ASP.NET Core	22	Editar Eliminar
3	Marta Ruiz	MVC	21	Editar Eliminar
4	Iris	2º DAM	19	Editar Eliminar

- Editar (cambio el nombre para añadir el apellido)

Editar Estudiante

Estudiante: Iris

Nombre
Iris Pérez

Curso
2º DAM

Edad
19

[Guardar Cambios](#)

[Volver al Listado](#)

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

Listado de Estudiantes CRUD (MVC)

Esta es la lista de estudiantes gestionada por el HomeController.

ID	Nombre	Curso	Edad	Acciones
1	Ana García	C# Básico	20	Editar Eliminar
2	Luis Pérez	ASP.NET Core	22	Editar Eliminar
3	Marta Ruiz	MVC	21	Editar Eliminar
4	Iris Pérez	2º DAM	19	Editar Eliminar

- Eliminar (elimino al estudiante ‘Luis Pérez’)

Listado de Estudiantes CRUD (MVC)

Esta es la lista de estudiantes gestionada por el HomeController.

[Crear Nuevo Estudiante](#)

ID	Nombre	Curso	Edad	Acciones
1	Ana García	C# Básico	20	Editar Eliminar
2	Luis Pérez	ASP.NET Core	22	Editar Eliminar
3	Marta Ruiz	MVC	21	Editar Eliminar
4	Iris Pérez	2º DAM	19	Editar Eliminar

(Se actualiza la lista de estudiantes)

Listado de Estudiantes CRUD (MVC)

Esta es la lista de estudiantes gestionada por el HomeController.

[Crear Nuevo Estudiante](#)

ID	Nombre	Curso	Edad	Acciones
1	Ana García	C# Básico	20	Editar Eliminar
3	Marta Ruiz	MVC	21	Editar Eliminar
4	Iris Pérez	2º DAM	19	Editar Eliminar