

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).

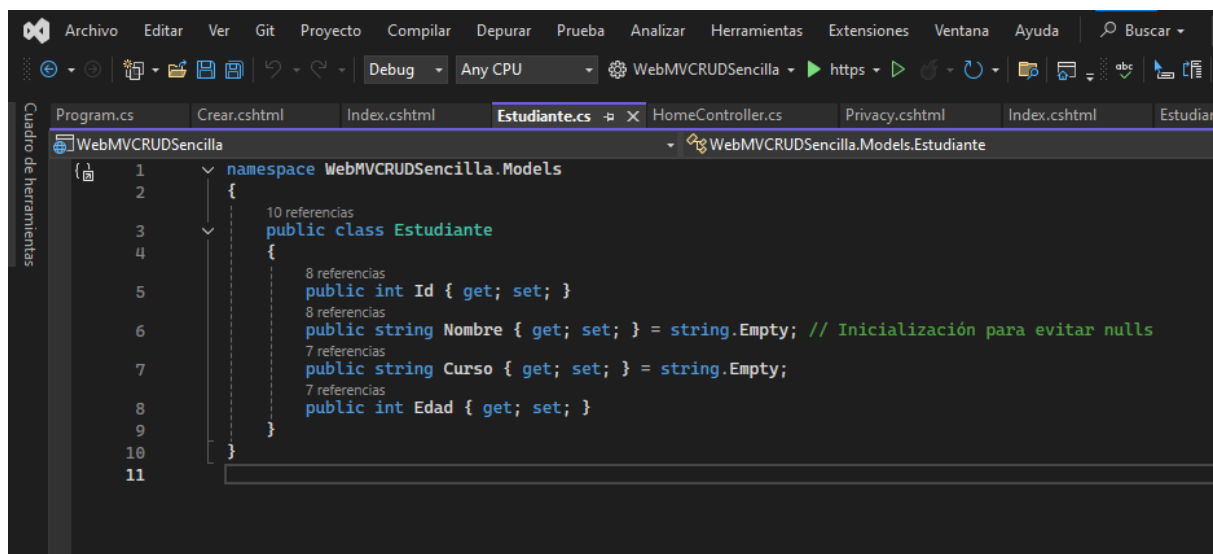
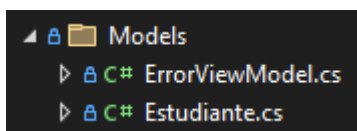
Añadir un nuevo proyecto a la solución WebMVCSencilla

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

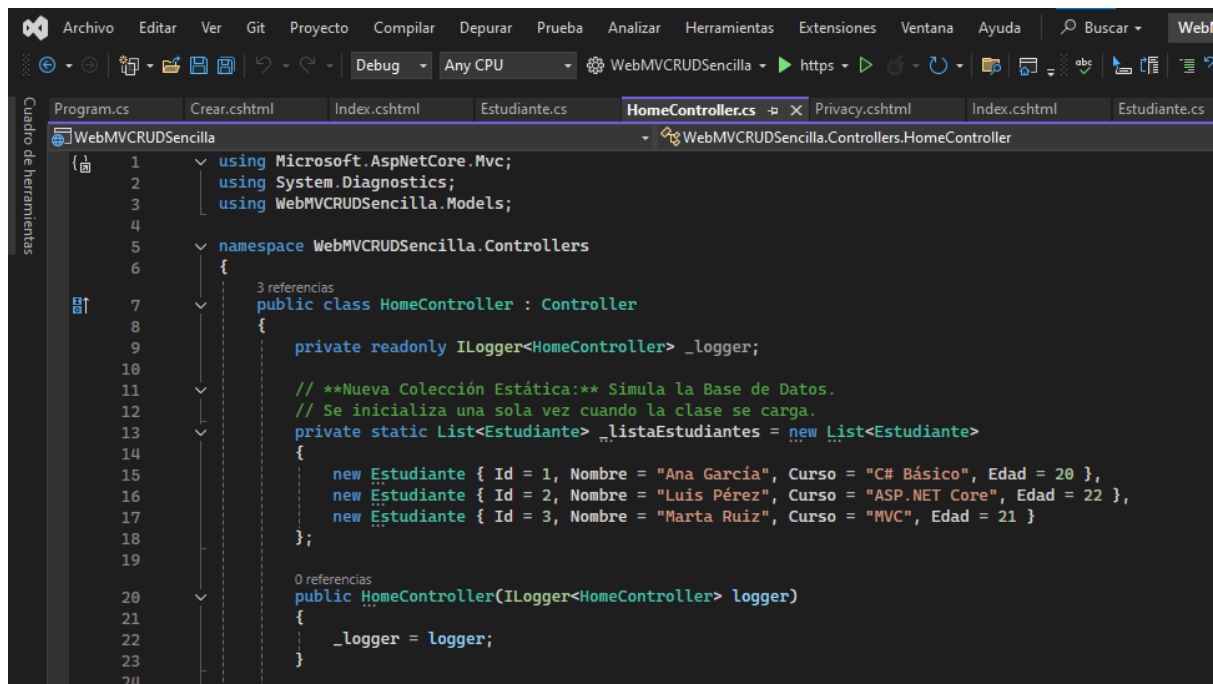
Controller

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 }
    };
}
```

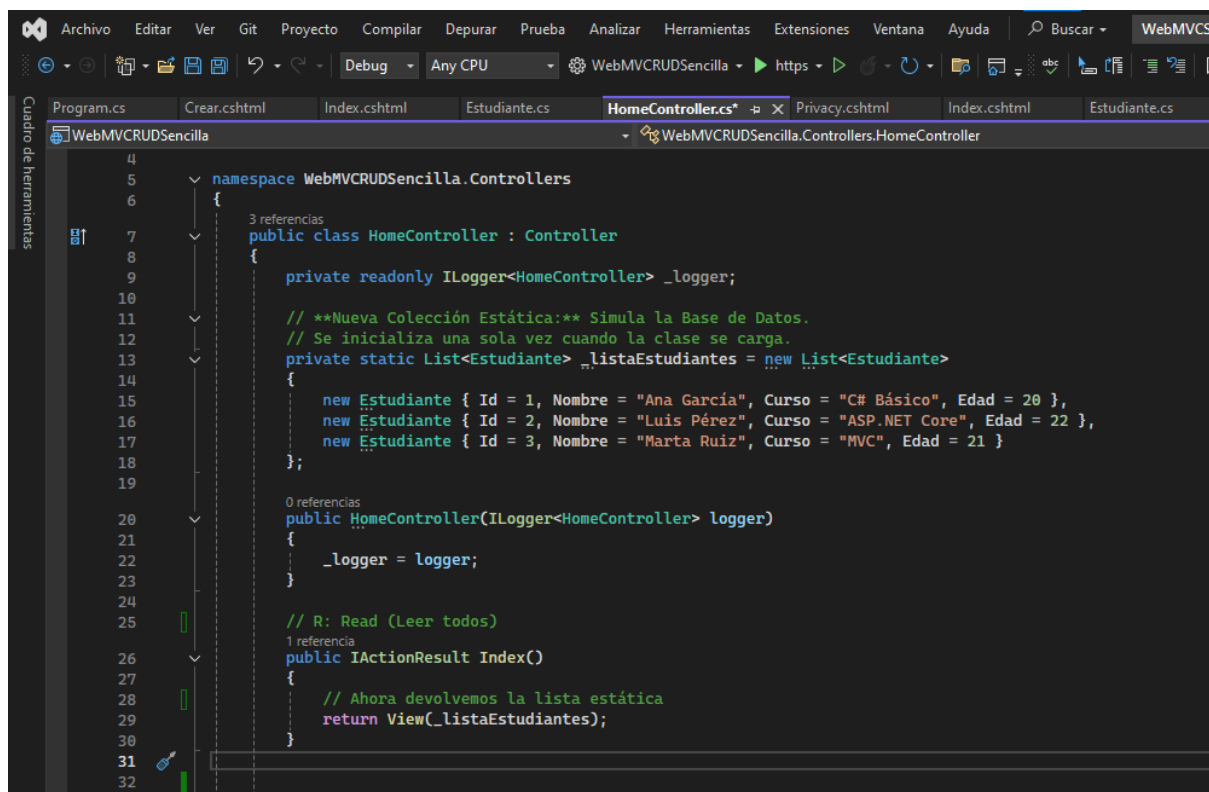


Read

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 WebMVCRUDEncilla.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>
}

```

```

1  @using WebMVCRUDEncilla.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
10 <p>
11     Esta es la lista de estudiantes gestionada por el HomeController.
12 </p>
13
14 <p>
15     <a asp-action="Crear" class="btn btn-success">Crear Nuevo Estudiante</a>
16 </p>
17
18 @if (Model != null && Model.Any())
19 {
20     <table class="table table-striped table-hover">
21     <thead>
22     <tr>
23         <th>ID</th>
24         <th>Nombre</th>
25         <th>Curso</th>
26         <th>Edad</th>
27         <th style="width: 180px;">Acciones</th>
28     </tr>
29     </thead>
30     <tbody>
31     @foreach (var estudiante in Model)
32     {
33         <tr>
34             <td>@estudiante.Id</td>
35             <td><strong>@estudiante.Nombre</strong></td>
36             <td>@estudiante.Curso</td>
37             <td>@estudiante.Edad</td>
38             <td>
39                 <a asp-action="Editar"
40                   asp-route-id="@estudiante.Id"
41                   class="btn btn-sm btn-info me-2">
42                     Editar
43                 </a>
44
45                 <form asp-action="Eliminar" method="post" style="display:inline;">
46                     <input type="hidden" name="id" value="@estudiante.Id" />
47                     <button type="submit"
48                       class="btn btn-sm btn-danger">

```

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.

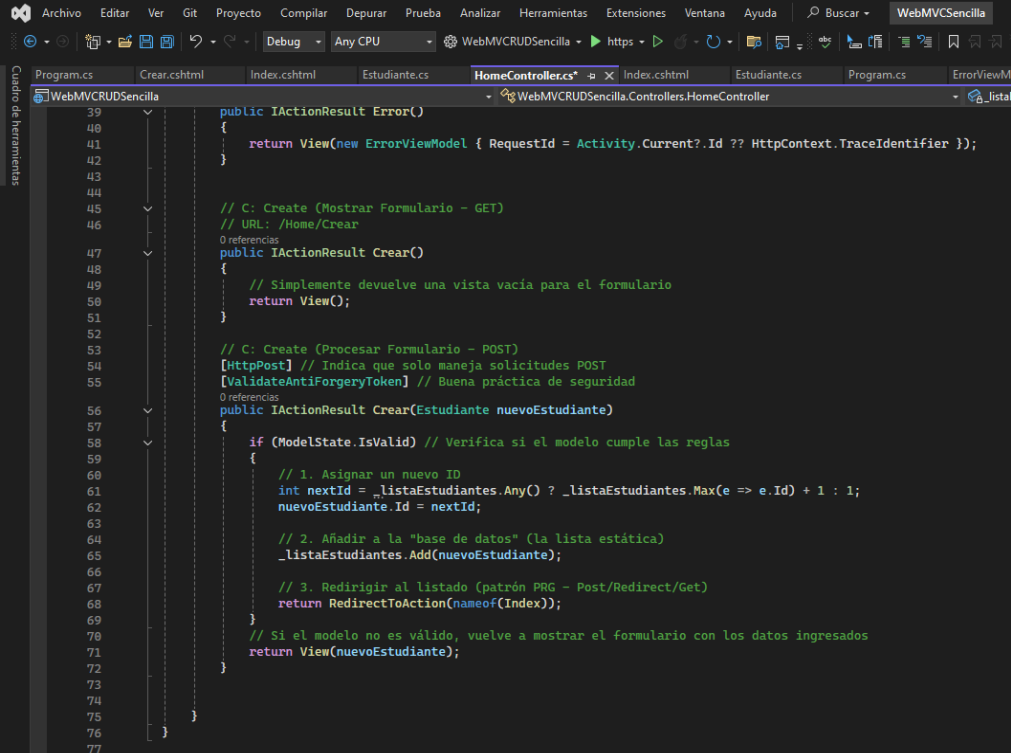
Edita el archivo `HomeController.cs` que está dentro de la carpeta `Controllers` y añade estos métodos para la operación crear:

```
// 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 with the `HomeController.cs` file open. The code is written in C# and includes the following methods:

- `Crear()`: A GET method that returns an empty view for the form.
- `Crear(Estudiante nuevoEstudiante)`: A POST method that handles the form submission. It checks if the model is valid, assigns a new ID, adds the student to the static list, and redirects to the index page. If the model is invalid, it returns the view with the entered data.

The code is formatted with comments in Spanish and uses standard C# syntax for actions, attributes, and collections.

View

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

```
@using WebMVCRUDEncilla.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 class="mt-3">
        <a asp-action="Index">Volver al Listado</a>
    </div>
</div>

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



```
Archivo  Editor  Ver  Git  Proyecto  Compilar  Depurar  Prueba  Analizar  Herramientas  Extensiones  Ventana
Debug  Any CPU  WebMVCRUDSencilla  https

Program.cs  Crear.cshtml  Index.cshtml  Estudiante.cs  HomeController.cs*  Index.cshtml
WebMVCRUDSencilla

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
41 <div class="mt-3">
42     <a asp-action="Index">Volver al Listado</a>
43 </div>
44 @section Scripts {
45     @{
46         await Html.RenderPartialAsync("_ValidationScriptsPartial");
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-llenado 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
    }

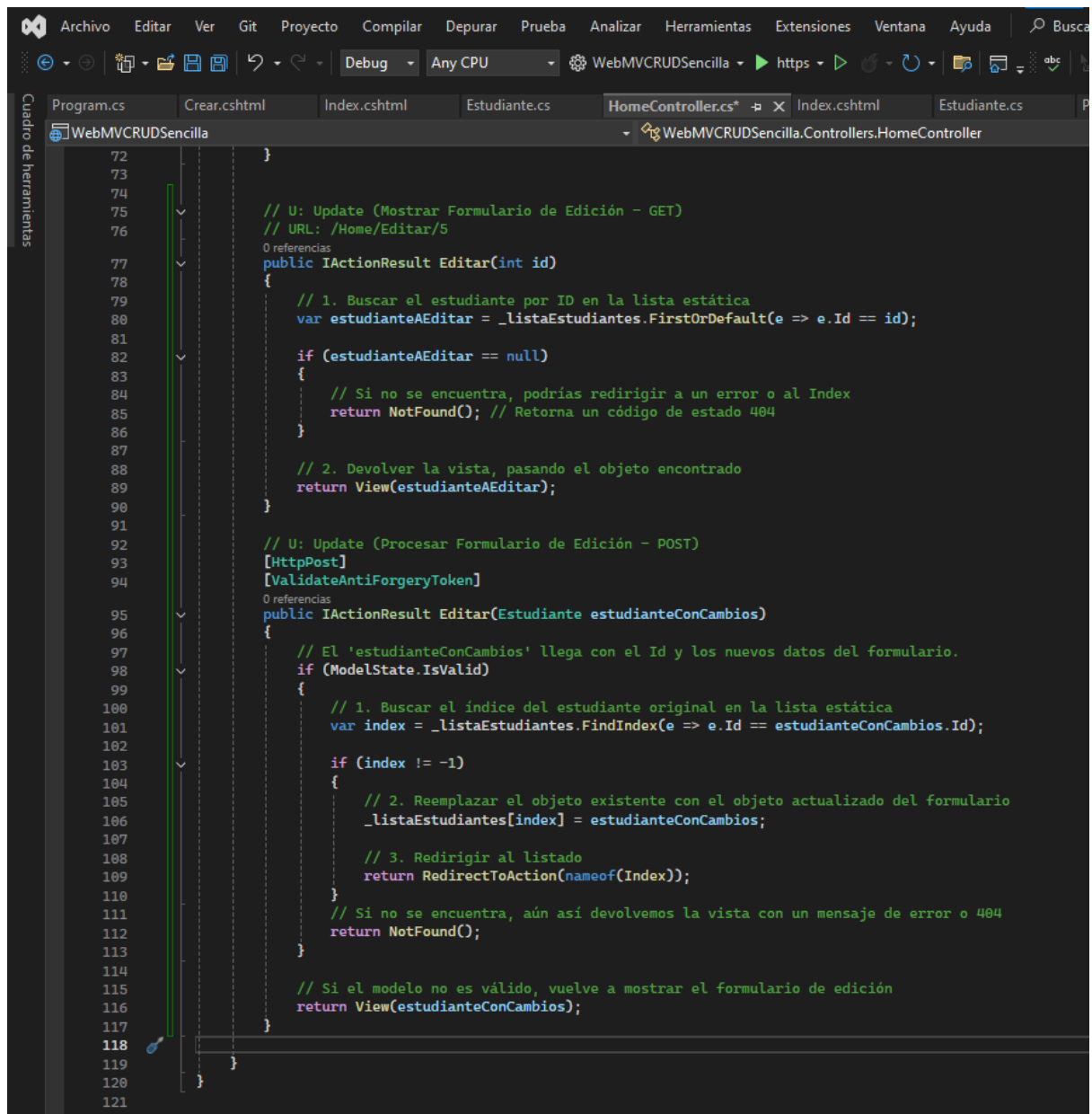
    // 2. Devolver la vista, pasando el objeto encontrado
    return View(estudianteAEditar);
}

// 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);
}
```



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

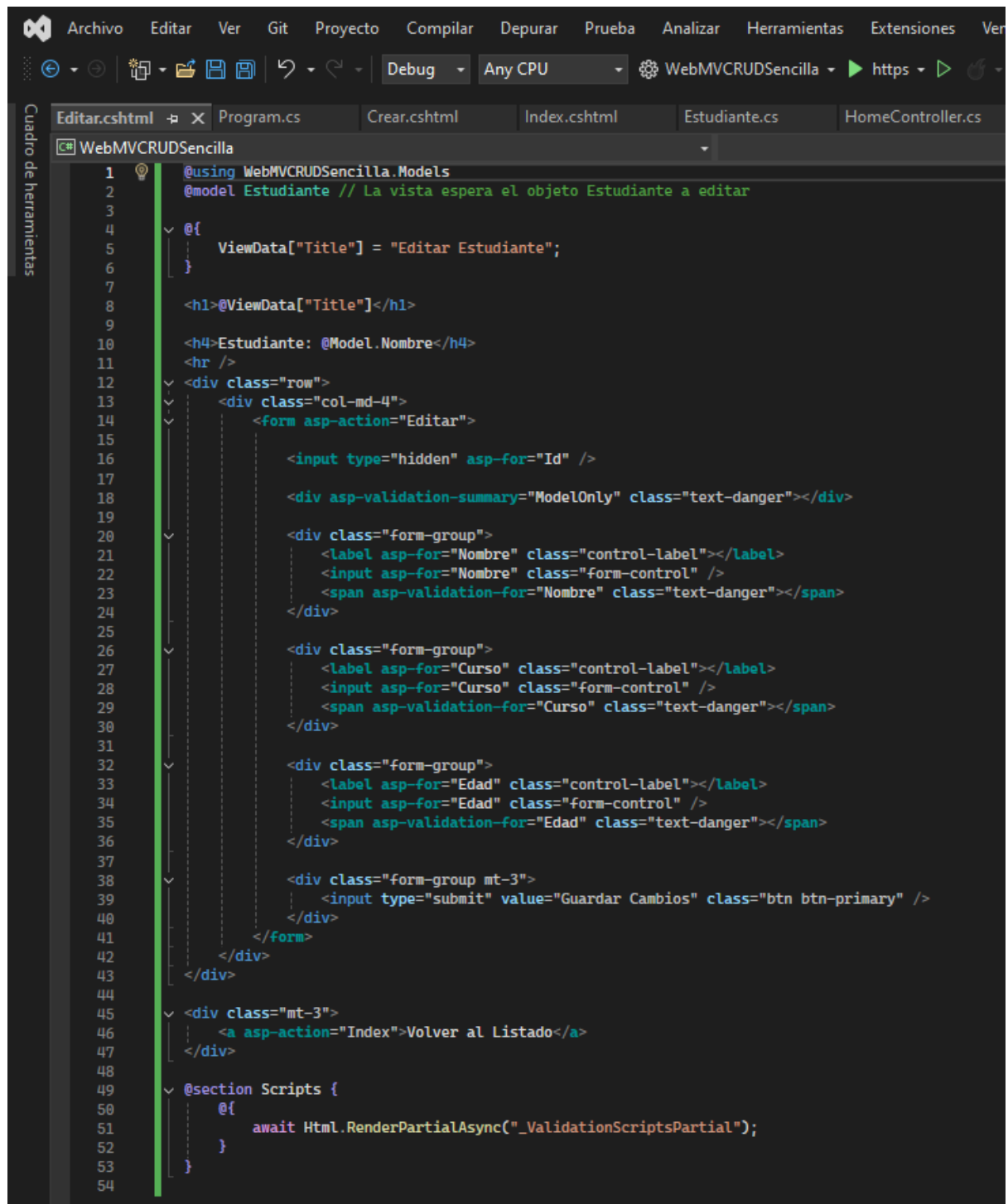
            <div class="form-group">
                <label asp-for="Edad" class="control-label"></label>
                <input asp-for="Edad" class="form-control" />
                <span asp-validation-for="Edad" class="text-danger"></span>
            </div>

            <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");}
}

```



```
1 @using WebMVCRUDEncilla.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             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
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             <div class="form-group">
23                 <label asp-for="Curso" class="control-label"></label>
24                 <input asp-for="Curso" class="form-control" />
25                 <span asp-validation-for="Curso" class="text-danger"></span>
26             </div>
27             <div class="form-group">
28                 <label asp-for="Edad" class="control-label"></label>
29                 <input asp-for="Edad" class="form-control" />
30                 <span asp-validation-for="Edad" class="text-danger"></span>
31             </div>
32             <div class="form-group mt-3">
33                 <input type="submit" value="Guardar Cambios" class="btn btn-primary" />
34             </div>
35         </form>
36     </div>
37 </div>
38
39 <div class="mt-3">
40     <a asp-action="Index">Volver al Listado</a>
41 </div>
42
43 @section Scripts {
44     @{
45         await Html.RenderPartialAsync("_ValidationScriptsPartial");
46     }
47 }
```

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 with the 'HomeController.cs' file open. The code is written in C# and implements the 'Eliminar' method. The method is decorated with '[HttpPost]' and '[ValidateAntiForgeryToken]'. It searches for a student in a static list, removes it if found, and then redirects to the 'Index' action. The code is color-coded, and the line numbers on the left range from 112 to 141.

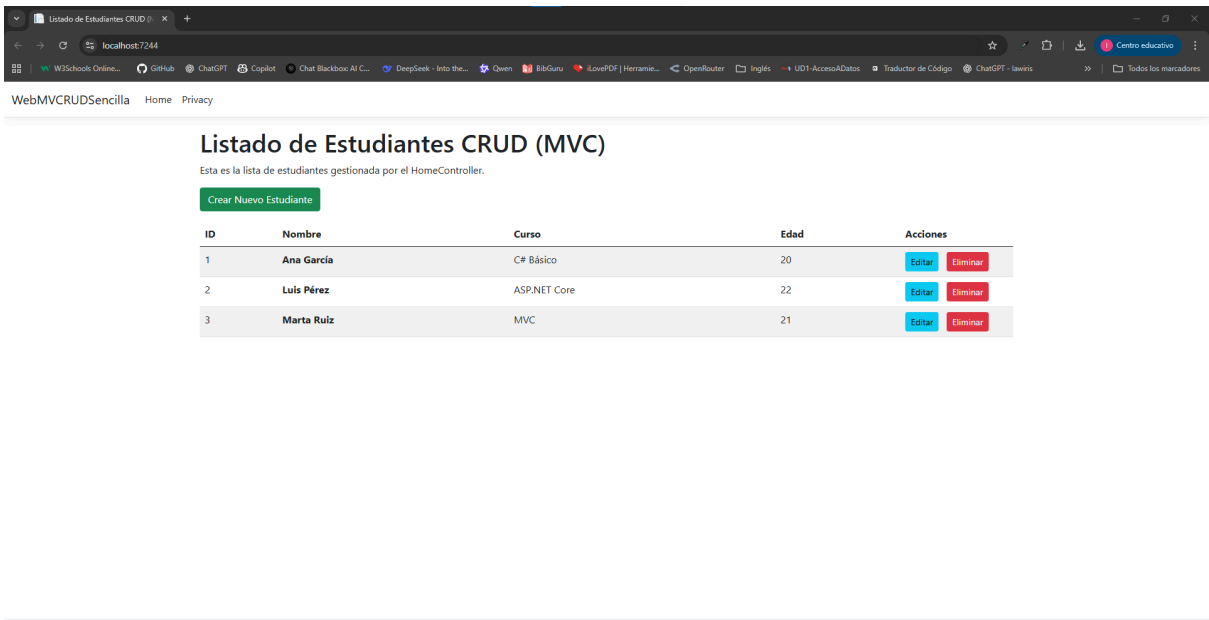
```

112         return NotFound();
113     }
114
115     // Si el modelo no es válido, vuelve a mostrar el formulario de edición
116     return View(estudianteConCambios);
117 }
118
119 // D: Delete (Eliminar por ID - POST)
120 // Usamos [HttpPost] y el parámetro 'id' para evitar que se elimine con una simple URL GET
121 [HttpPost]
122 [ValidateAntiForgeryToken]
123
124 public IActionResult Eliminar(int id)
125 {
126     // 1. Buscar el estudiante en la lista estática
127     var estudianteAEliminar = _listaEstudiantes.FirstOrDefault(e => e.Id == id);
128
129     if (estudianteAEliminar != null)
130     {
131         // 2. Eliminar el estudiante
132         _listaEstudiantes.Remove(estudianteAEliminar);
133     }
134
135     // 3. Redirigir al listado actualizado
136     return RedirectToAction(nameof(Index));
137 }
138
139 }
140
141

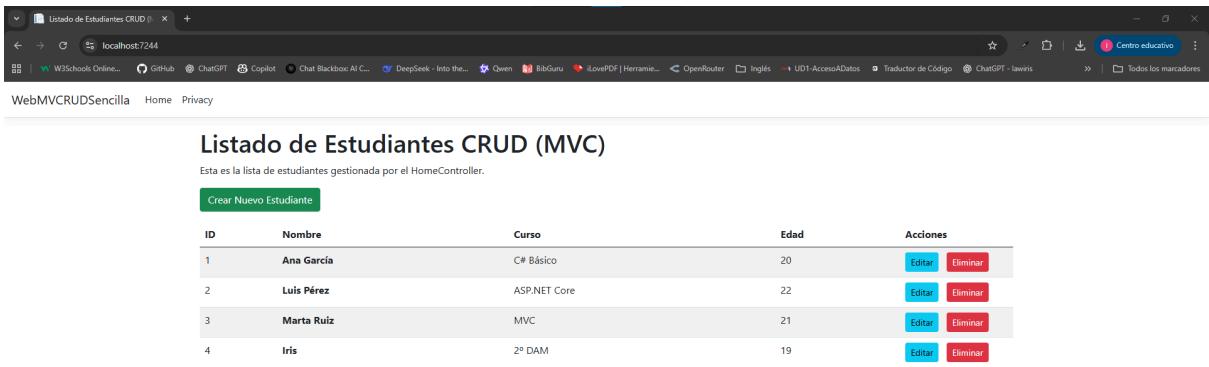
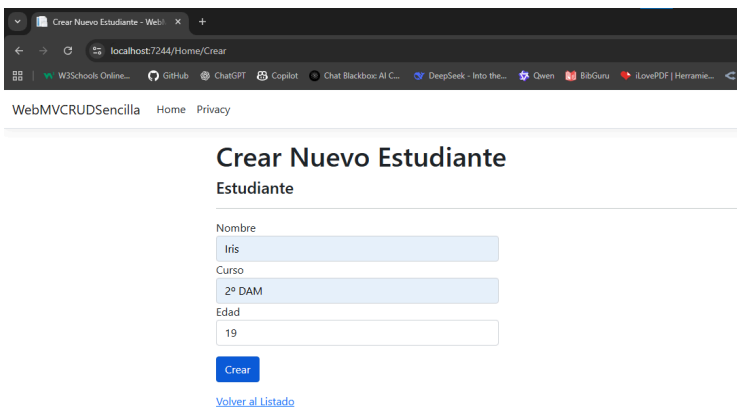
```

Ejecución:

- INICIO



- Crear un estudiante



- Editar (cambio el nombre para añadir el apellido)

WebMVCRUDSencilla Home Privacy

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)

WebMVCRUDSencilla Home Privacy

Listado de Estudiantes CRUD (MVC)

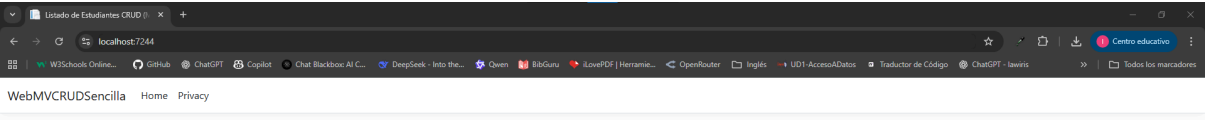
Esta es la lista de estudiantes gestionada por el HomeController.

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

© 2025 - WebMVCRUDSencilla - [Privacy](#)

- Eliminar (elimino al estudiante 'Luis Pérez')



(Se actualiza la lista de estudiantes)

