



.Net

Teoría 12

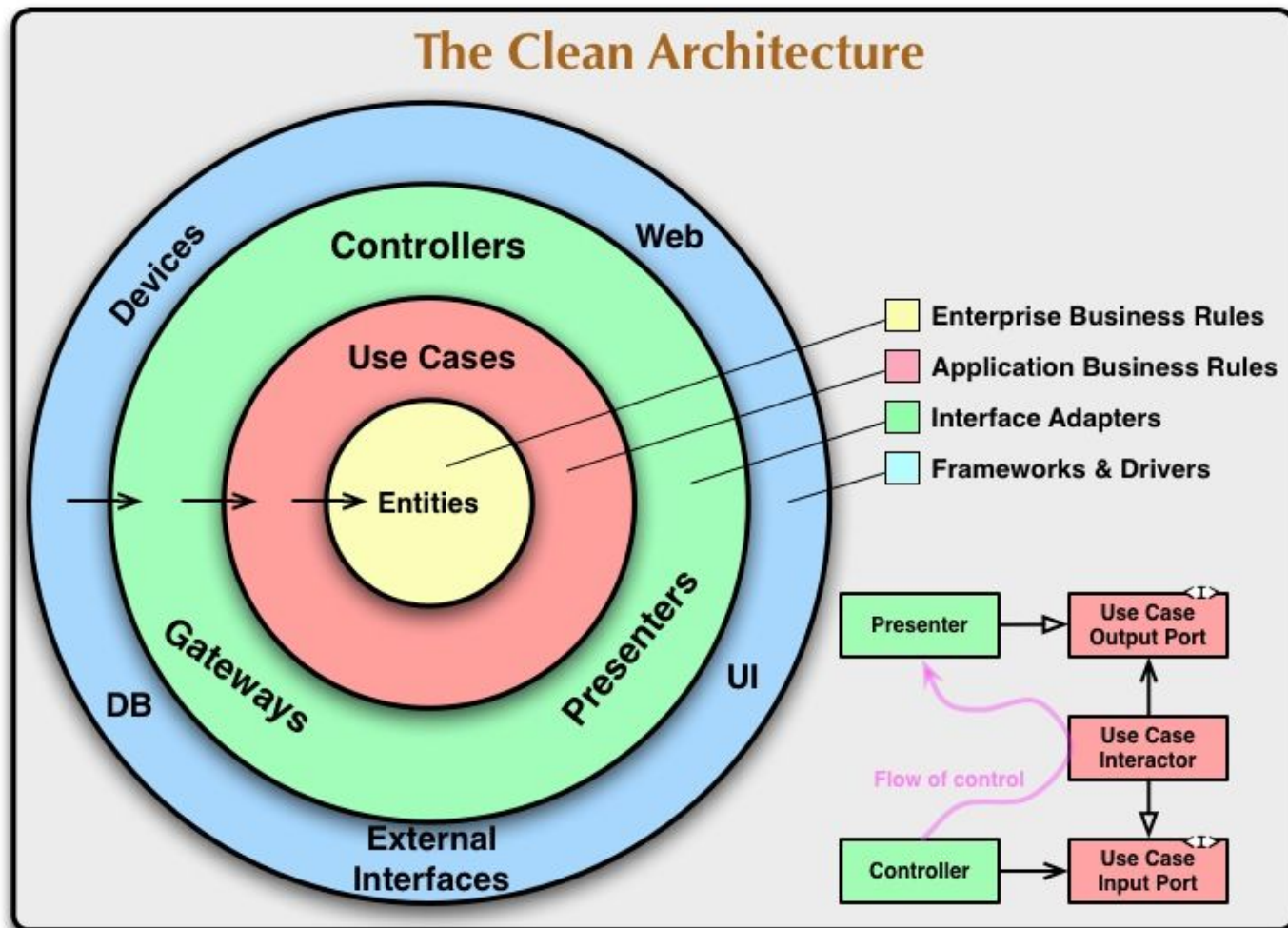
Arquitectura Limpia

Repaso



Arquitectura limpia

- Se refiere a organizar el proyecto para que sea fácil de entender y cambiar a medida que el proyecto crece.
- En los últimos años han aparecido varias ideas con respecto a la arquitectura como **Arquitectura Hexagonal** y **Arquitectura de Cebolla**.
- En su blog, Robert C. Martin (el tío Bob) expone una idea general de una Arquitectura Limpia



<https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

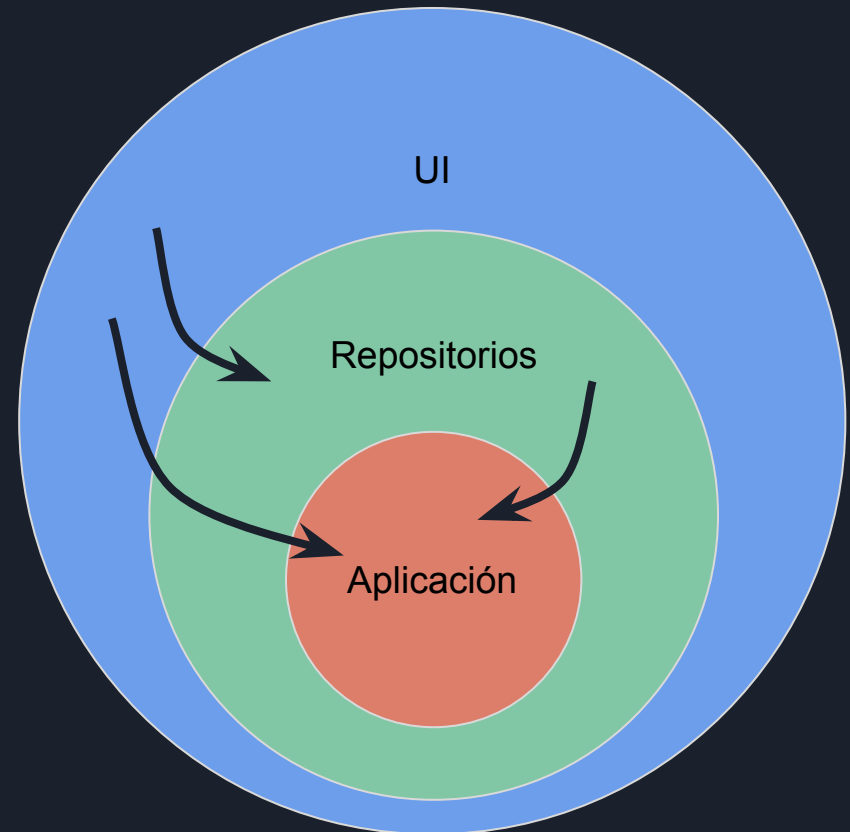
Separa los elementos de un diseño en niveles de anillo. Los anillos externos dependen de los internos y nunca al revés. El código en las capas internas no puede tener conocimiento de las funciones en las capas externas.

Arquitectura limpia

Para trabajar en este curso proponemos la siguiente versión simplificada

Aplicación y Repositorios serán proyectos de biblioteca de clases

UI será un proyecto ejecutable (por ej. una aplicación de consola o Blazor)

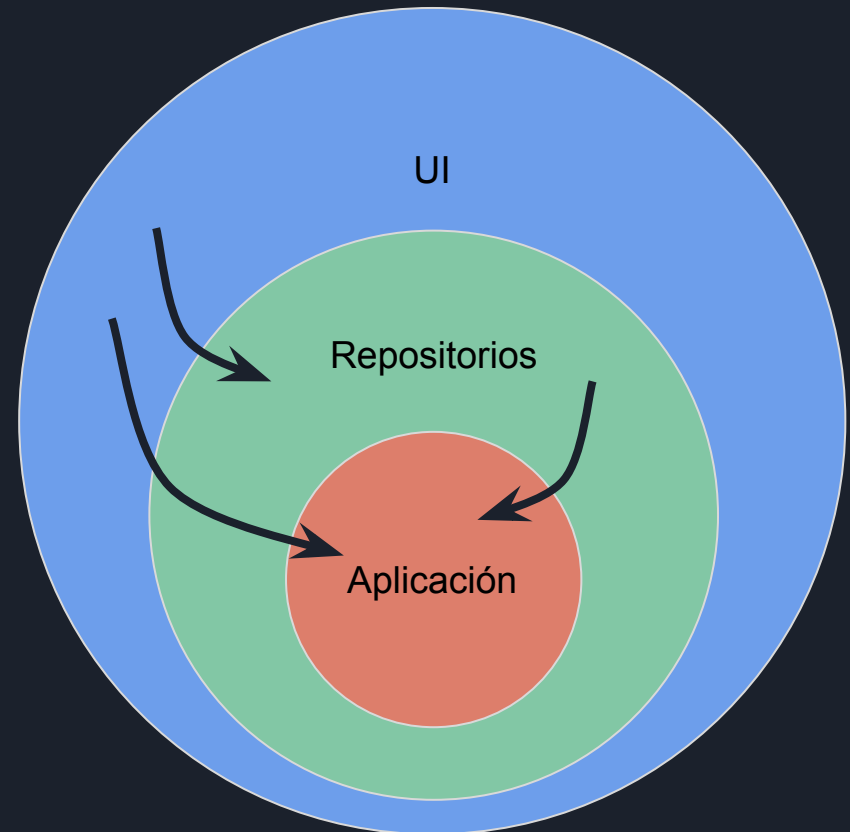


Arquitectura limpia

UI hará referencia a los
proyectos Repositorios y
Aplicación

Repositorios hará
referencia a Aplicación

Aplicación no hará
referencia a ningún otro
proyecto





Codificando una solución con diseño de Arquitectura Limpia



- Abrir una terminal del sistema operativo
- Cambiar a la carpeta `proyectosDotnet`
- Crear la solución `AL` con el siguiente comando:

```
dotnet new sln -o AL
```

- Cambiar a la carpeta `AL`

```
cd AL
```

- Crear los proyectos siguientes proyectos:

```
dotnet new classlib -o AL.Aplicacion
```

```
dotnet new classlib -o AL.Repositorios
```

```
dotnet new blazor --no-https -o AL.UI
```



Luego de ejecutar estos comando abrir Visual Studio Code (en la carpeta de la solución AL)



- Agregar los 3 proyectos a la solución
`dotnet sln add AL.Aplicacion`
`dotnet sln add AL.Repositorios`
`dotnet sln add AL.UI`
- Establecer las referencias entre proyectos para que `AL.UI` conozca a los otros 2 proyectos
`dotnet add AL.UI reference AL.Aplicacion`
`dotnet add AL.UI reference AL.Repositorios`
- Establecer la referencia para que `Al.Repositorios` conozca a `Al.Aplicacion`
`dotnet add AL.Repositorios reference AL.Aplicacion`



En el proyecto AL.Aplicacion crear la carpeta Entidades y dentro la clase Cliente



SOLUTION EXPLORER

- AL
- AL.Aplicacion
 - Dependencias
 - Entidades
 - Cliente.cs
 - AL.Repositorios
 - AL.UI

Cliente.cs

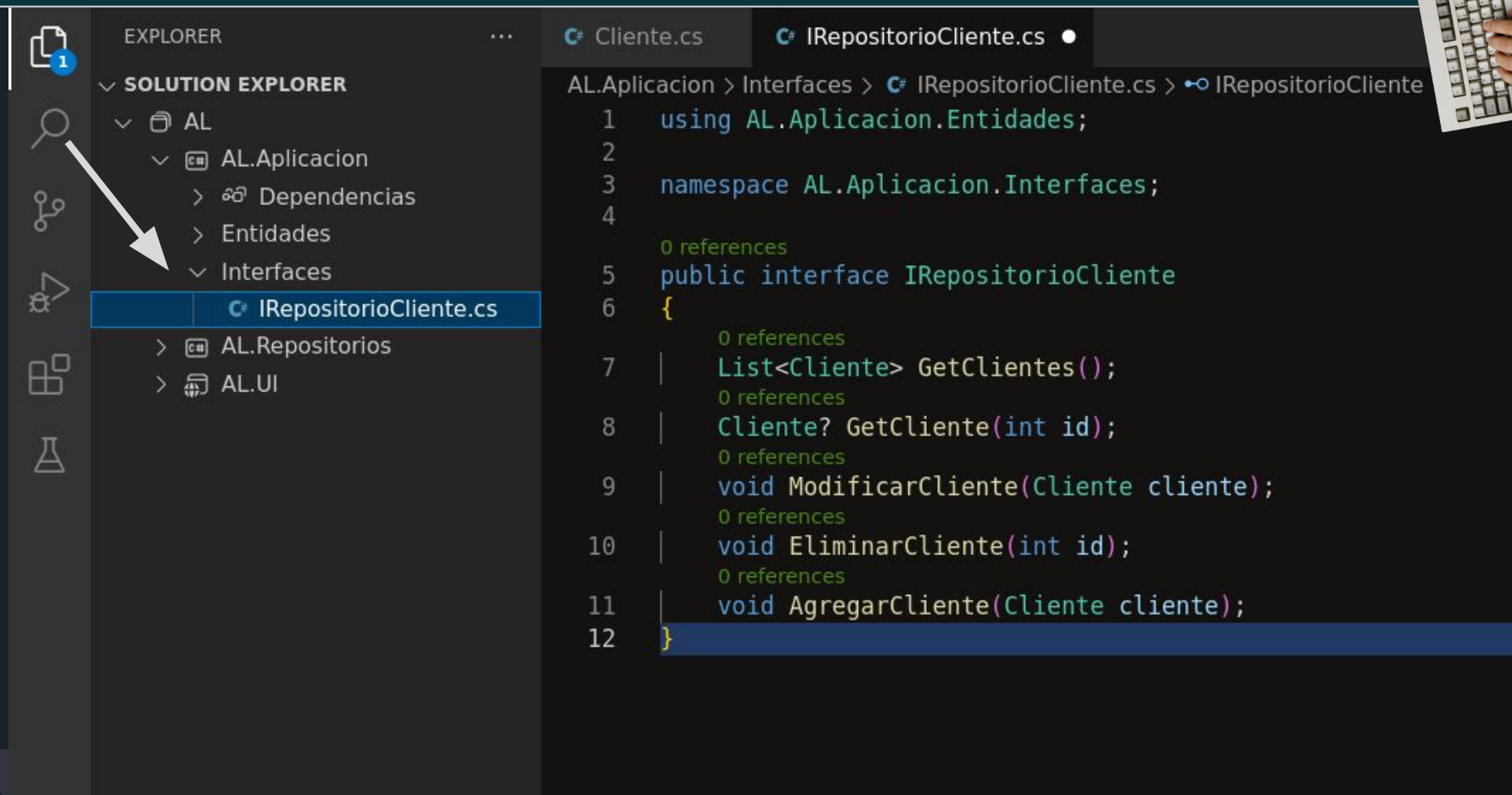
```
1 namespace AL.Aplicacion.Entidades;
2
3 public class Cliente
4 {
5     public int Id { get; set; }
6     public string Nombre { get; set; } = "";
7     public string Apellido { get; set; } = "";
8 }
9
```

Establecer el `namespace` adecuado teniendo en cuenta el proyecto y la carpeta donde se localiza

Copiar el código del archivo
12_RecursosParaLaTeoria



En el proyecto AL.Aplicacion crear la carpeta Interfaces y dentro la interfaz IRepositoryCliente



Copiar el código del archivo
12_RecursosParaLaTeoria



En el proyecto AL.Aplicacion crear la carpeta UseCases y dentro codificar la clase ClienteUseCase



```
using AL.Aplicacion.Interfaces;

namespace AL.Aplicacion.UseCases;

public abstract class ClienteUseCase(IRepositorioCliente repositorio)
{
    protected IRepositorioCliente Repositorio { get; } = repositorio;
}
```

Clase abstracta, base de las clases para los *usecases* relacionados a los clientes



En el proyecto AL.Aplicacion dentro de la carpeta UseCases codificar la clase AgregarClienteUseCase



```
using AL.Aplicacion.Entidades;  
using AL.Aplicacion.Interfaces;
```

```
namespace AL.Aplicacion.UseCases;
```

```
public class AgregarClienteUseCase(IRepositorioCliente repositorio):ClienteUseCase(repositorio)  
{  
    public void Ejecutar(Cliente cliente)  
    {  
        //aquí podríamos insertar código de validación de cliente  
  
        Repositorio.AgregarCliente(cliente);  
    }  
}
```

Inyección de
dependencias!



En el proyecto AL.Aplicacion dentro de la carpeta UseCases codificar la clase EliminarClienteUseCase



```
using AL.Aplicacion.Interfaces;
```

```
namespace AL.Aplicacion.UseCases;
```

```
public class EliminarClienteUseCase(IRepositorioCliente repositorio):ClienteUseCase(repositorio)
{
    public void Ejecutar(int id)
    {
        Repositorio.EliminarCliente(id);
    }
}
```

Inyección de
dependencias!



En el proyecto AL.Aplicacion dentro de la carpeta UseCases codificar la clase ListarClientesUseCase



```
using AL.Aplicacion.Entidades;  
using AL.Aplicacion.Interfaces;
```

```
namespace AL.Aplicacion.UseCases;
```

```
public class ListarClientesUseCase(IRepositorioCliente repositorio):ClienteUseCase(repositorio)  
{  
    public List<Cliente> Ejecutar()  
    {  
        return Repositorio.GetClientes();  
    }  
}
```

Inyección de
dependencias!



En el proyecto AL.Aplicacion dentro de la carpeta UseCases codificar la clase ModificarClienteUseCase



```
using AL.Aplicacion.Entidades;  
using AL.Aplicacion.Interfaces;
```

```
namespace AL.Aplicacion.UseCases;
```

```
public class ModificarClienteUseCase(IRepositorioCliente repositorio):ClienteUseCase(repositorio)  
{  
    public void Ejecutar(Cliente cliente)  
    {  
        Repositorio.ModificarCliente(cliente);  
    }  
}
```

Inyección de
dependencias!



En el proyecto AL.Aplicacion dentro de la carpeta UseCases codificar la clase ObtenerClienteUseCase



```
using AL.Aplicacion.Entidades;  
using AL.Aplicacion.Interfaces;
```

```
namespace AL.Aplicacion.UseCases;
```

```
public class ObtenerClienteUseCase(IRepositorioCliente repositorio):ClienteUseCase(repositorio)  
{  
    public Cliente? Ejecutar(int id)  
    {  
        return Repositorio.GetCliente(id);  
    }  
}
```

Inyección de
dependencias!

Codificaremos un RepositorioClienteMock

Imaginemos que el grupo de programadores encargados de codificar el repositorio concreto **RepositorioCliente** aún no ha terminado.

No hay problema: Codificaremos un **sustituto temporario** para no detener el proyecto. Más adelante será sencillo, gracias a la inversión de dependencias, reemplazarlo por el verdadero





En el proyecto AL.Repositorios crear la clase RepositorioClienteMock



The screenshot shows the Visual Studio IDE with the Solution Explorer on the left and the code editor on the right. The Solution Explorer shows the project structure: AL > AL.Repositorios > RepositorioClienteMock.cs. The code editor shows the implementation of the `RepositorioClienteMock` class, which implements the `IRepositorioCliente` interface. The code includes using statements for `AL.Aplicacion.Entidades` and `AL.Aplicacion.Interfaces`, a namespace declaration for `AL.Repositorios`, and a class definition for `RepositorioClienteMock`. The class has a private readonly list `_listaClientes` initialized with a new list containing a `Cliente` object with `Id=1`, `Nombre="Alberto"`, and `Apellido="García"`. The class also has a public void method `AgregarCliente` that takes a `Cliente` object as a parameter.

```
1 using AL.Aplicacion.Entidades;
2 using AL.Aplicacion.Interfaces;
3
4 namespace AL.Repositorios;
5 public class RepositorioClienteMock : IRepositorioCliente
6 {
7
8     private readonly List<Cliente> _listaClientes = new List<
9         new Cliente(){Id=1, Nombre="Alberto", Apellido="García",
10             Id="Perez"}
11
12     public void AgregarCliente(Cliente cliente)
13     {
14         cliente.Id = s_proximoId++;
15     }
16 }
```

RepositorioClienteMock debe implementar la interfaz **IRepositorioCliente**

Injectaremos en la aplicación este repositorio Mock para hacer algunas pruebas. Lo llamamos **Mock** porque es un sustituto de algún repositorio que va a utilizar la aplicación realmente.

```
using AL.Aplicacion.Entidades;
using AL.Aplicacion.Interfaces;

namespace AL.Repositorios;
public class RepositorioClienteMock : IRepositoryCliente
{
    private readonly List<Cliente> _listaClietes = new List<Cliente>(){
        new Cliente(){Id=1,Nombre="Alberto",Apellido="García"},
        new Cliente(){Id=2,Nombre="Ana",Apellido="Perez"}
    };//hemos hardcodeado dos clientes en la lista

    static int s_proximoId = 3;

    private Cliente Clonar(Cliente c) //se van a devolver copias de los cliente guardados
    {
        return new Cliente()
        {
            Id = c.Id,
            Nombre = c.Nombre,
            Apellido = c.Apellido
        };
    }
    public void AgregarCliente(Cliente cliente)
    {
        cliente.Id = s_proximoId++;
        _listaClietes.Add(Clonar(cliente));
    }
}
```

Copiar el código del archivo
12_RecursosParaLaTeoria
(diapositivas 19 y 20)

```
public void EliminarCliente(int id)
{
    var cliente = _listaClietes.SingleOrDefault(c => c.Id == id);
    if (cliente != null)
    {
        _listaClietes.Remove(cliente);
    }
}
public Cliente? GetCliente(int id)
{
    Cliente? c = _listaClietes.SingleOrDefault(c => c.Id == id);
    if (c != null)
    {
        return Clonar(c);
    }
    return null;
}
public List<Cliente> GetClientes()
{
    return _listaClietes.Select(c => Clonar(c)).ToList();
}
public void ModificarCliente(Cliente cliente)
{
    var cli = _listaClietes.SingleOrDefault(c => c.Id == cliente.Id);
    if (cli != null)
    {
        cli.Apellido = cliente.Apellido;
        cli.Nombre = cliente.Nombre;
    }
}
}
```

Componente cuadro de diálogo

Pasemos a la interfaz de usuario con Blazor.

Primero vamos a crear un componente que nos permita presentar un **cuadro de diálogo** para que el usuario pueda confirmar o rechazar alguna acción





Codificar un componente razor llamado DialogoConfirmacion.razor (en la carpeta Components)



```
@rendermode InteractiveServer
@if(visible)
{
    <h3>@Mensaje</h3>
    <button class="btn btn-primary" @onclick="Cerrar">Aceptar</button>
    <button class="btn btn-secondary" @onclick="Cerrar">Cancelar</button>
}

@code{
    private bool visible = false;
    [Parameter]
    public string Mensaje {get;set;}="";

    public void Mostrar()
    {
        visible=true;
        StateHasChanged();
    }

    void Cerrar()
    {
        visible=false;
    }
}
```

Mostrar será invocado desde código fuera de este componente, por lo tanto nos aseguramos de notificar al motor de renderizado que debe actualizar el componente. Generalmente no es necesario pero hay ocasiones en que debe hacerse

Copiar el código del archivo
12_RecursosParaLaTeoria

Componente cuadro de diálogo

Todavía no terminamos con la funcionalidad requerida pero vamos a ir comprobando su funcionamiento desde el componente **Counter**





Modificar el componente Counter.razor



```
@page "/counter"
@rendermode InteractiveServer

<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

<button class="btn btn-danger" @onclick="ConfirmarReseteo">Resetear contador</button>

<DialogoConfirmacion @ref=dialogo Mensaje="¿Está seguro que desea resetear el
contador?" />
```

Agregamos un botón para resetear el contador. Usaremos el componente `DialogoConfirmacion` para solicitar confirmación al usuario

Asociamos el componente a la variable de instancia `dialogo`

Copiar el código del archivo
12_RecursosParaLaTeoria
(diapositivas 24 y 25)



Modificar el componente Counter.razor



. . .

```
@code {  
    private int currentCount = 0;  
  
    private void IncrementCount()  
    {  
        currentCount++;  
    }  
}
```

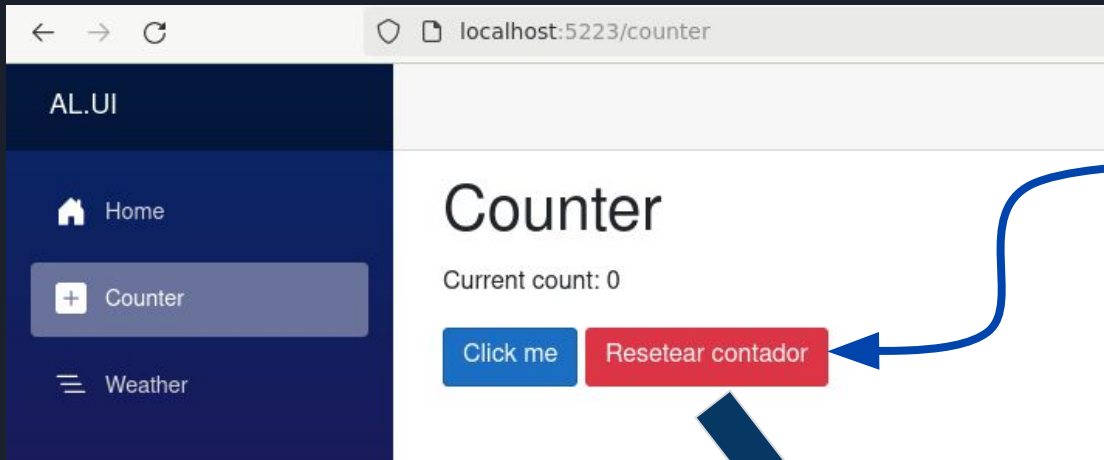
```
    DialogoConfirmacion dialogo = null!;  
    private void ConfirmarReseteo()  
    {  
        dialogo.Mostrar();  
    }  
}
```

Declaramos la variable `dialogo` con la que referenciamos al componente agregado

El signo `!` se utiliza para anular el warning del control de nullables

El método resetear muestra el componente para que el usuario confirme la acción

Arquitectura Limpia - UI con Blazor



Probar
funcionamiento



Counter

Current count: 0

Click me

Resetea contador

¿Está seguro que desea resetear el contador?

Aceptar

Cancelar

Este es el componente
DialogoConfirmacion



Modificar DialogoConfirmacion.razor agregando un evento que se lanzará al hacer click en Aceptar



```
@rendermode InteractiveServer
@if(visible)
{
    <h3>@Mensaje</h3>
    <button class="btn btn-primary" @onclick="CerrarYconfirmar">Aceptar</button>
    <button class="btn btn-secondary" @onclick="Cerrar">Cancelar</button>
}

@code{
    . . .
```

Al hacer clic sobre el botón Aceptar se ejecuta el método `CerrarYconfirmar`

```
[Parameter]
public EventCallback OnConfirmado{get;set;}

void CerrarYconfirmar()
{
    visible = false;
    OnConfirmado.InvokeAsync();
}

}
```

Se declara el evento `OnConfirmado`
El tipo `EventCallback` permite la asignación de un delegado `Action`

Se ejecuta el delegado que se haya asignado a la propiedad `OnConfirmado`

Copiar el código del archivo
12_RecursosParaLaTeoria



Modificar Counter.razor suscribiéndose al evento ConfirmarReseteo del componente DialogoConfirmacion



```
@page "/counter"
```

```
...
```

```
<DialogoConfirmacion @ref=dialogo Mensaje="¿Está seguro que desea resetear el  
contador?" OnConfirmado="Resetear" />
```

```
@code {
```

```
...
```

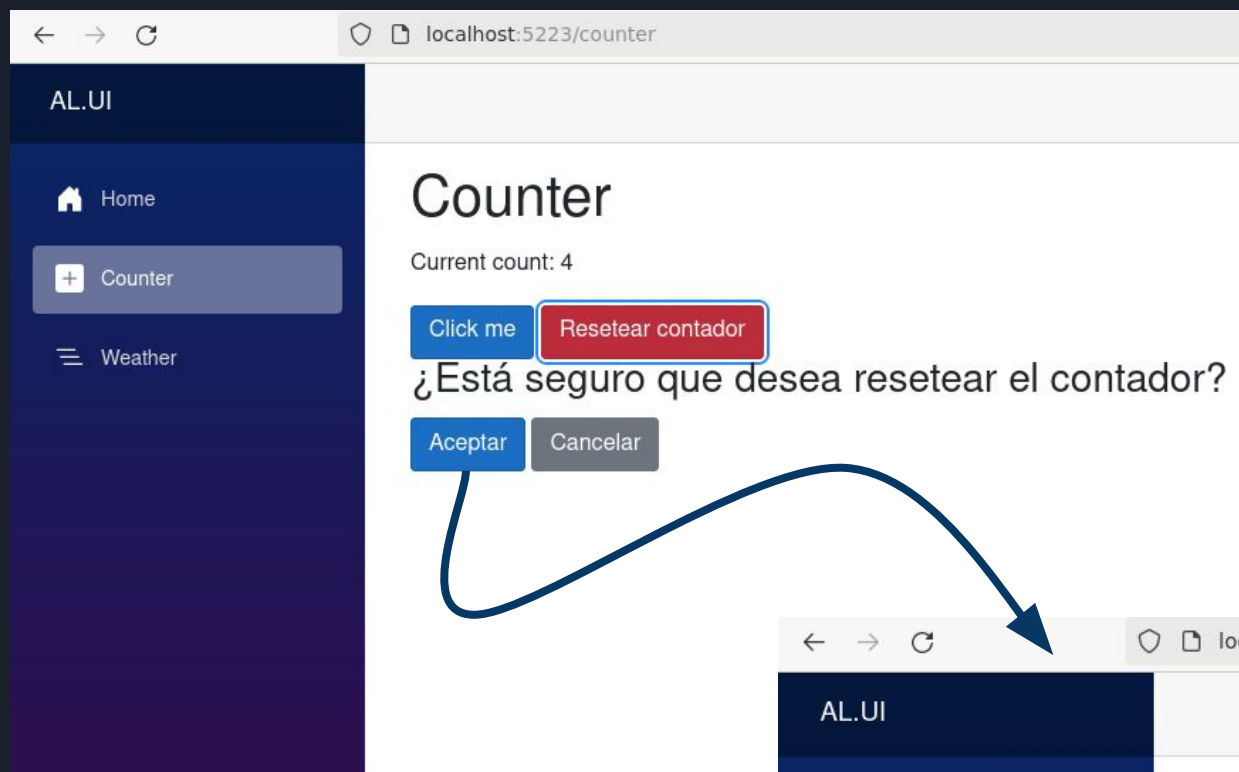
```
private void Resetear()  
{  
    currentCount = 0;  
}
```

```
}
```

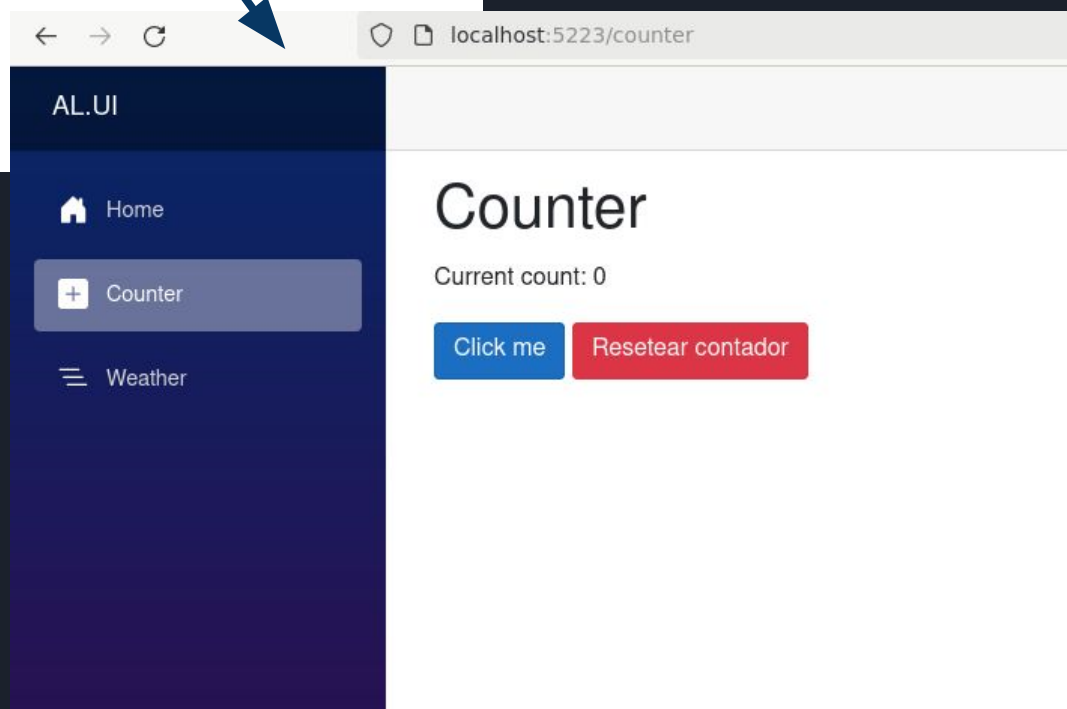
Nos suscribimos al evento `Onconfirmado` con el método `Resetear`

Copiar el código del archivo
12_RecursosParaLaTeoria

Arquitectura Limpia - UI con Blazor



Probar
funcionamiento



Componente cuadro de diálogo

Vamos a darle estilo al
componente
`DialogoConfirmacion` para que
realmente se transforme en un
cuadro de diálogo





Modificar DialogoConfirmacion.razor de la siguiente manera



```
@rendermode InteractiveServer
@if (visible)
{
    <div class="dialogo-contenedor">
        <div class="dialogo-contenido">
            <h3>@Mensaje</h3>
            <button class="btn btn-primary" @onclick="CerrarYconfirmar">Aceptar</button>
            <button class="btn btn-secondary" @onclick="Cerrar">Cancelar</button>
        </div>
    </div>
}
```

```
@code {
```

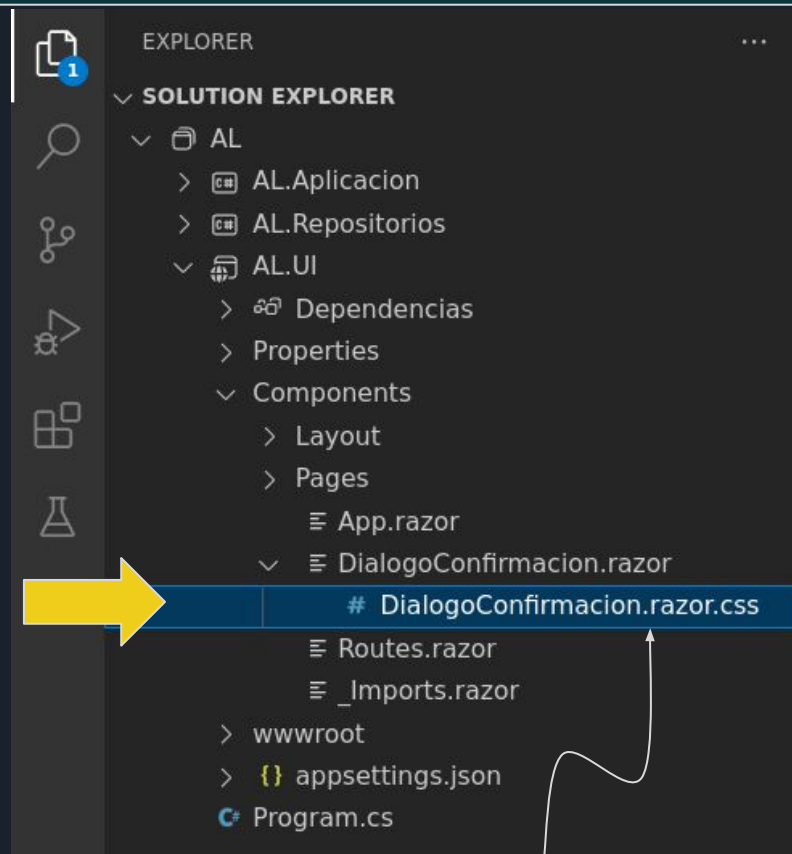
```
...
```

Colocamos el contenido entre dos secciones div para darle estilo utilizando css

Copiar el código del archivo
12_RecursosParaLaTeoria



Agregar el archivo DialogoConfirmacion.razor.css en la carpeta Components



DialogoConfirmacion.razor.css

AL.UI > Components > # DialogoConfirmacion.razor.css > ...

Para definir estilos específicos de un componente, se debe crear un archivo de extensión **.razor.css** cuyo nombre coincida con el del archivo **.razor** del componente en la misma carpeta.

SOLUTION EXPLORER identifica el archivo .css para visualizar que aplica sólo al componente, sin embargo se encuentra en la misma carpeta que el componente



Agregar el archivo DialogoConfirmacion.razor.css en la carpeta Components



```
.dialogo-contenedor {  
    position: fixed;  
    top:0;  
    left: 0;  
    right: 0;  
    bottom:0;  
    background-color: rgba(0, 0, 0, 0.5);  
    z-index: 1;  
    display: flex;  
    align-items: center;  
    justify-content: center;  
}
```

El div más externo ocupará toda la página

Background negro con opacidad del 50%

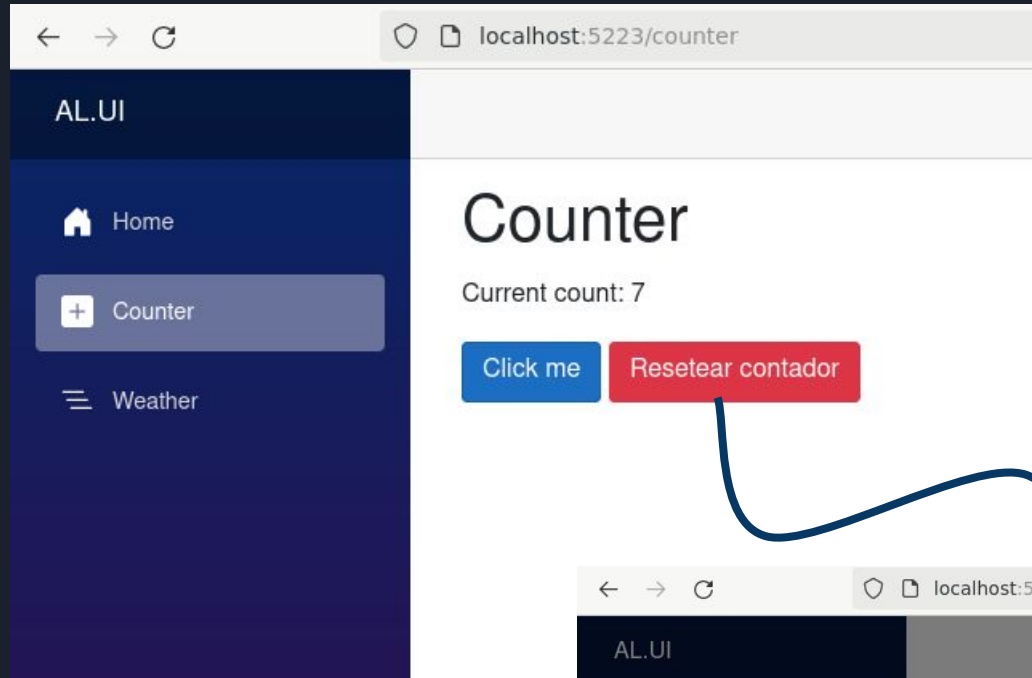
Se colocará por encima de los demás elementos de la página

Su contenido aparecerá centrado vertical y horizontalmente

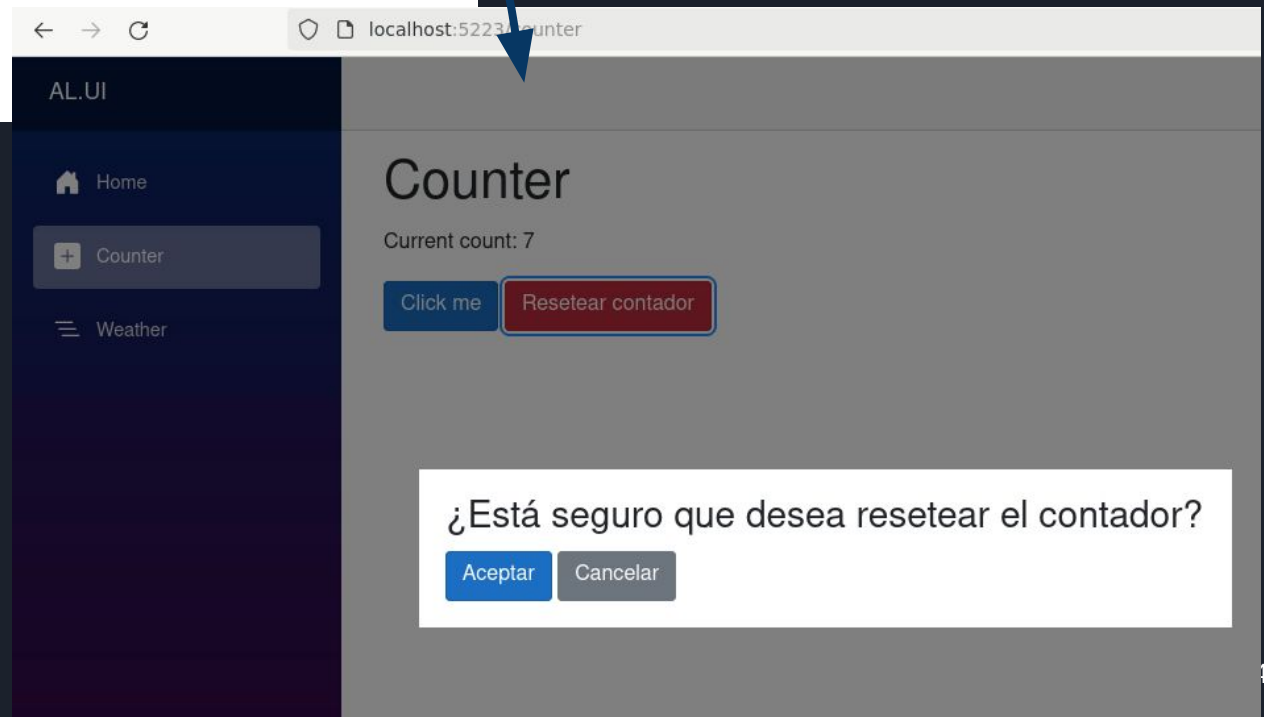
```
.dialogo-contenido{  
    padding: 20px;  
    background-color: white;  
}
```

Background blanco para el div más interno

Copiar el código del archivo
12_RecursosParaLaTeoria



Probar
funcionamiento



Pregunta sobre implementación de DialogoConfirmacion.razor

```
<button class="btn btn-primary" @onclick="CerrarYconfirmar">Aceptar</button>
<button class="btn btn-secondary" @onclick="Cerrar">Cancelar</button>

. . .
```

```
void Cerrar()
{
    visible = false;
}

void CerrarYconfirmar()
{
    visible = false;
    OnConfirmado.InvokeAsync();
}
```

Si reemplazamos
este código

Por este otro

¿Cómo podríamos establecer
el valor de @onclick en
ambos botones?

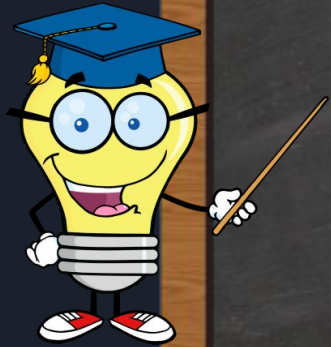


```
<button class="btn btn-primary" @onclick=" " >Aceptar</button>
<button class="btn btn-secondary" @onclick=" " >Cancelar</button>

. . .
```

```
void Cerrar(bool confirmar)
{
    visible = false;
    if (confirmar)
    {
        OnConfirmado.InvokeAsync();
    }
}
```

Componente cuadro de diálogo



```
... @onclick="()=>Cerrar(true)">Aceptar...  
... @onclick="()=>Cerrar(false)">Cancelar ...
```

```
void Cerrar(bool confirmar)  
{  
    visible = false;  
    if (confirmar)  
    {  
        OnConfirmado.InvokeAsync();  
    }  
}
```

Podemos escribir una
expresión lambda que
invoque
adecuadamente al
método Cerrar

Contenedor de Inyección de Dependencias en Blazor

La aplicación Blazor que usaremos como interfaz de usuario ya viene con un **DI container** integrado.

Registraremos nuestros servicios en la clase program por medio de `builder.Services` que devuelve un `IServiceCollection`





Modificar Program.cs de la aplicación AL.UI



```
using AL.UI.Components;
```

```
//agregamos estas directivas using  
using AL.Repositorios;  
using AL.Aplicacion.UseCases;  
using AL.Aplicacion.Interfaces;
```

```
var builder = WebApplication.CreateBuilder(args);
```

```
// Add services to the container.  
builder.Services.AddRazorComponents()  
    .AddInteractiveServerComponents();
```

```
//agregamos estos servicios al contenedor DI  
builder.Services.AddTransient<AgregarClienteUseCase>();  
builder.Services.AddTransient<ListarClientesUseCase>();  
builder.Services.AddTransient<EliminarClienteUseCase>();  
builder.Services.AddTransient<ModificarClienteUseCase>();  
builder.Services.AddTransient<ObtenerClienteUseCase>();  
builder.Services.AddScoped<IRepositorioCliente, RepositorioClienteMock>();
```

```
var app = builder.Build();
```

```
. . .
```

Copiar el código del archivo
12_RecursosParaLaTeoria



Agregar las directivas @using en el archivo _imports.razor de la aplicación AL.UI



The screenshot shows the Visual Studio IDE. On the left, the **SOLUTION EXPLORER** pane displays the project structure. The **AL.UI** project is expanded, showing the **Components** folder. The file **_Imports.razor** is selected and highlighted in blue. On the right, the code editor shows the content of **_Imports.razor**. The file contains a list of `@using` directives. A green bracket on the right side of the editor groups the last four lines (lines 12-15), and a green arrow points to this group, indicating the focus of the instruction.

```
1 @using System.Net.Http
2 @using System.Net.Http.Json
3 @using Microsoft.AspNetCore.Components.Forms
4 @using Microsoft.AspNetCore.Components.Routing
5 @using Microsoft.AspNetCore.Components.Web
6 @using static Microsoft.AspNetCore.Components.Web.RenderMode
7 @using Microsoft.AspNetCore.Components.Web.Virtualization
8 @using Microsoft.JSInterop
9 @using AL.UI
10 @using AL.UI.Components
11
12 @using AL.Aplicacion.Entidades
13 @using AL.Aplicacion.Interfaces
14 @using AL.Aplicacion.UseCases
15 @using AL.Repositorios
16
```



Codificar un componente razor llamado ListadoClientes.razor (en la carpeta Pages)



```
@page "/listadoclientes"
@rendermode InteractiveServer
@inject ListarClientesUseCase ListarClientesUseCase

@code {
    List<Cliente> _lista = new List<Cliente>();
    protected override void OnInitialized()
    {
        _lista = ListarClientesUseCase.Ejecutar();
    }
}
```

Este método se invoca una vez creada la instancia de la clase que representa el componente

Con la directiva `@inject` directiva inyectamos en la propiedad `ListarClientesUseCase` un objeto de tipo `ListarClientesUseCase`

Copiar el código del archivo
12_RecursosParaLaTeoria



Agregar el siguiente código a ListadoClientes.razor



```
@page "/listadoclientes"
@rendermode InteractiveServer
@inject ListarClientesUseCase ListarClientesUseCase
<table class="table">
  <thead>
    <tr>
      <th>ID</th>
      <th>NOMBRE</th>
      <th>APELLIDO</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var cli in _lista)
    {
      <tr>
        <td>@cli.Id</td>
        <td>@cli.Nombre</td>
        <td>@cli.Apellido</td>
      </tr>
    }
  </tbody>
</table>
@code {
  ...
}
```

Copiar el código del archivo
12_RecursosParaLaTeoria



Modificar el componente **NavMenu.razor** que está en la carpeta **Components/Layout** y ejecutar



...

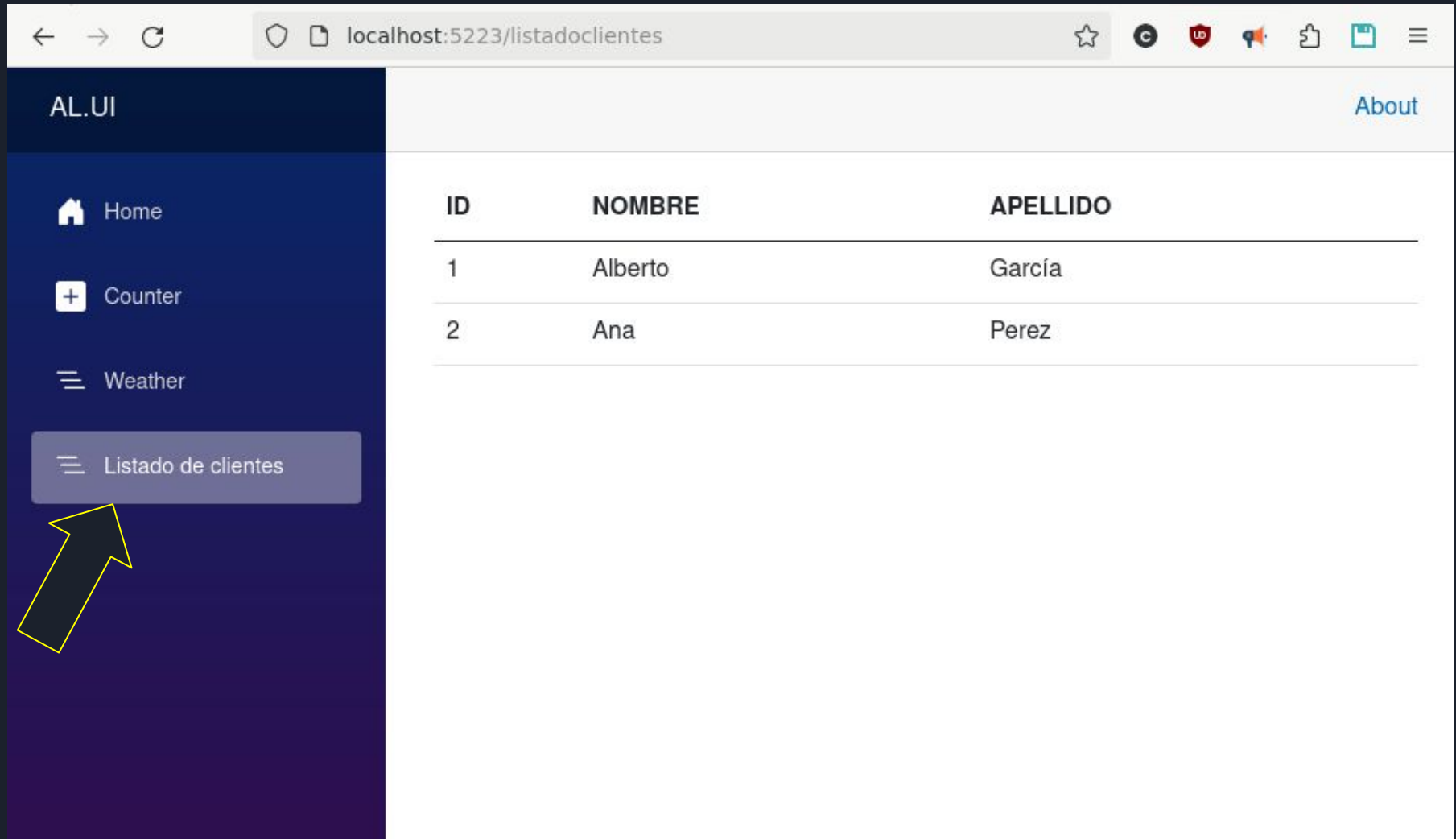
```
<div class="nav-item px-3">
  <NavLink class="nav-link" href="weather">
    <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Weather
  </NavLink>
</div>
```



```
<div class="nav-item px-3">
  <NavLink class="nav-link" href="listadoclientes">
    <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Listado de clientes
  </NavLink>
</div>
```

...

Copiar y pegar las 5 líneas que corresponden al menú "Weather" y luego cambiar el texto y el valor del atributo **href**





Agregar un nuevo item en el componente NavMenu.razor



. . .

```
<div class="nav-item px-3">  
    <NavLink class="nav-link" href="listadoclientes">  
        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>  
        Listado de clientes  
    </NavLink>  
</div>
```

```
<div class="nav-item px-3">  
    <NavLink class="nav-link" href="cliente">  
        <span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span>  
        Agregar Cliente  
    </NavLink>  
</div>
```

. . .



Codificar un componente razor llamado EditarCliente.razor. Ejecutar



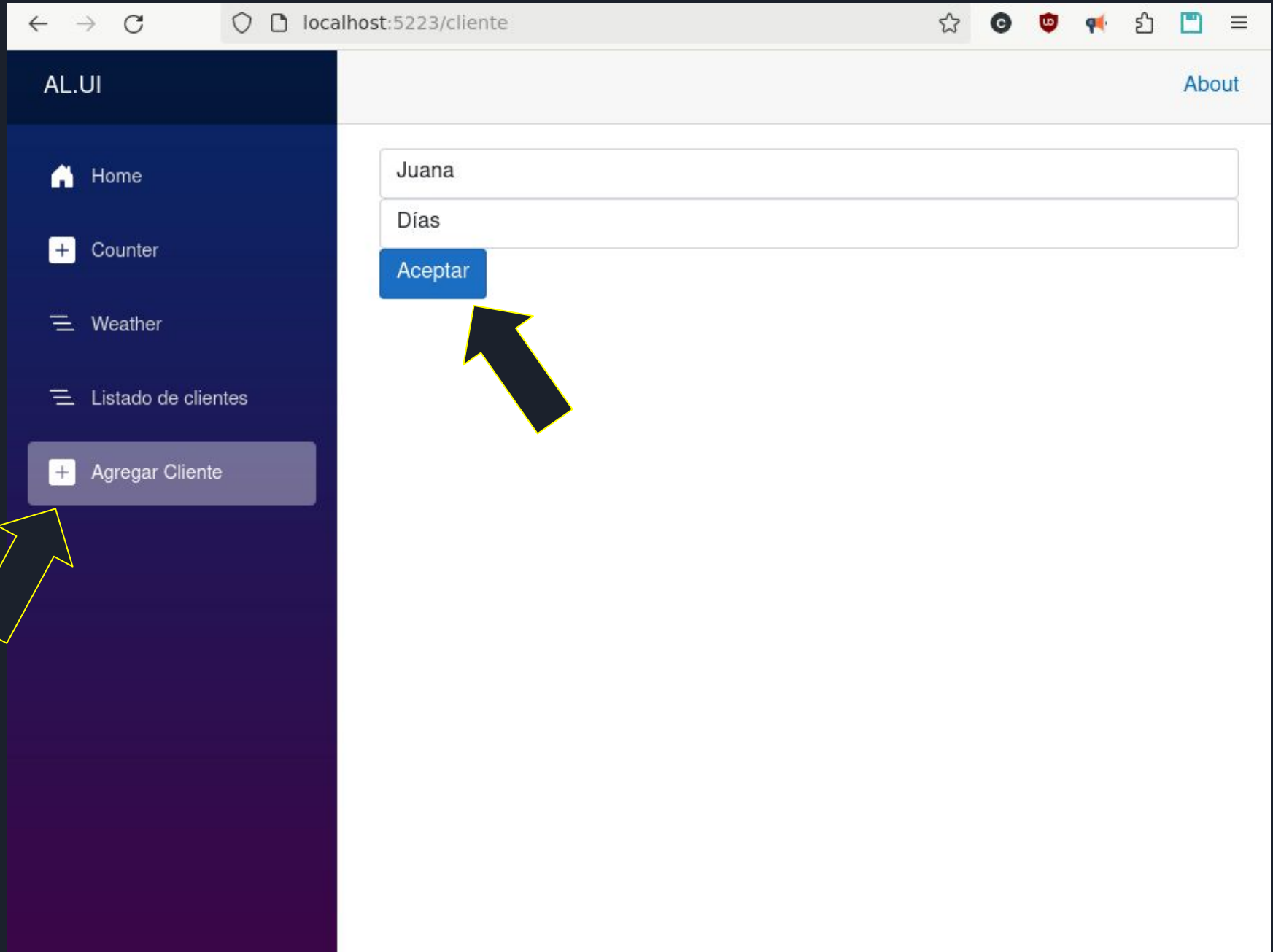
```
@page "/cliente"
@rendermode InteractiveServer
@inject NavigationManager Navegador;
@inject AgregarClienteUseCase AgregarClienteUseCase

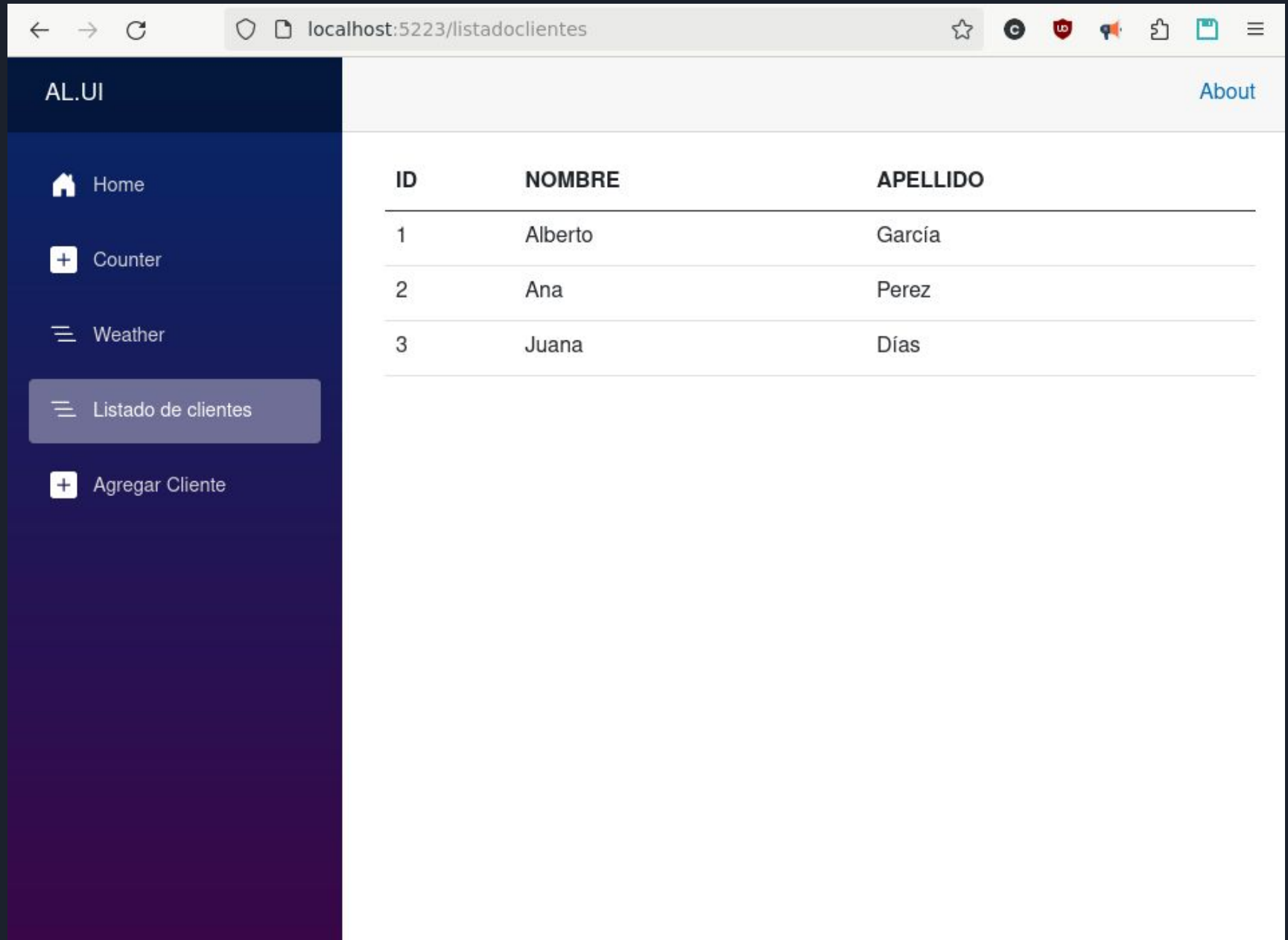
<input placeholder="Nombre" @bind="_cliente.Nombre" class="form-control">
<input placeholder="Apellido" @bind="_cliente.Apellido" class="form-control">
<button class="btn btn-primary" @onclick="Aceptar">Aceptar</button>

@code {
    Cliente _cliente = new Cliente();
    void Aceptar()
    {
        AgregarClienteUseCase.Ejecutar(_cliente);
        _cliente = new Cliente();
        Navegador.NavigateTo("listadoclientes");
    }
}
```

Injectamos un objeto `NavigationManager` que nos permite navegar entre las páginas

Copiar el código del archivo
12_RecursosParaLaTeoria





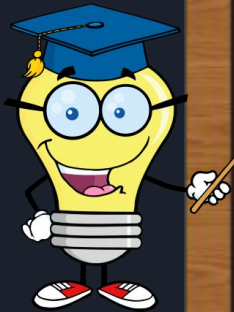
The screenshot shows a web browser window with the address bar displaying `localhost:5223/listadoclientes`. The application has a dark blue sidebar on the left with the following navigation items:

- Home
- Counter
- Weather
- Listado de clientes (highlighted)
- Agregar Cliente

The main content area displays a table with the following data:

ID	NOMBRE	APELLIDO
1	Alberto	García
2	Ana	Perez
3	Juana	Días

Comprobar funcionamiento en otra pestaña del navegador



1. Sin bajar el servidor, comprobar el funcionamiento en otra pestaña del navegador

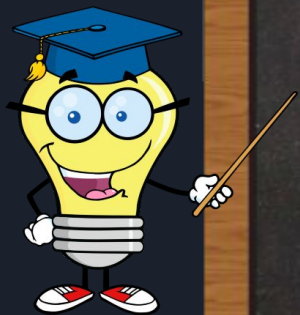
Comprobar funcionamiento en otra pestaña del navegador

1. Sin bajar el servidor, comprobar el funcionamiento en otra pestaña del navegador
2. Bajar el servidor, registrar el servicio IRepositoryCliente como Singleton



Comprobar funcionamiento en otra pestaña del navegador

1. Sin bajar el servidor, comprobar el funcionamiento en otra pestaña del navegador
2. Bajar el servidor, registrar el servicio IRepositoryCliente como Singleton
3. Volver a comprobar el funcionamiento en dos pestañas distintas del navegador



Esto ocurre porque se está utilizando un repositorio que guarda los datos en su memoria interna, no ocurriría si estuviésemos usando una base de datos en disco



Agregar el archivo EditarCliente.razor.css



Para definir estilos específicos de un componente, se debe crear un archivo `.razor.css` cuyo nombre coincida con el del archivo `.razor` del componente en la misma carpeta.


```
-----EditarCliente.razor.css-----
```


```
.form-control {  
    margin-bottom: 10px;  
}
```


Agregamos margen debajo de los elementos que tengan el atributo `class="form-control"`
Esto afecta sólo al componente `EditarCliente.razor`

AL.UI

 Home

 Counter

 Fetch data

 Listado de clientes

Nombre

Apellido

Aceptar



Modificar el componente ListadoClientes y ejecutar




```
<table class="table">
  <thead>
    <tr>
      <th>ID</th>
      <th>NOMBRE</th>
      <th>APELLIDO</th>
      <th>ACCIÓN</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var cli in _lista)
    {
      <tr>
        <td>@cli.Id</td>
        <td>@cli.Nombre</td>
        <td>@cli.Apellido</td>
        <td>
          <button class="btn btn-primary">Editar</button>
          <button class="btn btn-danger">Eliminar</button>
        </td>
      </tr>
    }
  </tbody>
</table>
```


Copiar el código del archivo
12_RecursosParaLaTeoria


AL.UI


[About](#)

 Home

 Counter

 Weather

 Listado de clientes

 Agregar Cliente

ID	NOMBRE	APELLIDO	ACCIÓN	
1	Alberto	García	Editar	Eliminar
2	Ana	Perez	Editar	Eliminar



Falta agregarle
comportamiento a
estos botones



Inyectar el servicio EliminarClienteUseCase y colocar el componente DialogoConfirmacion



```
@page "/listadoclientes"
@rendermode InteractiveServer
@inject ListarClientesUseCase ListarClientesUseCase
@inject EliminarClienteUseCase EliminarClienteUseCase

<DialogoConfirmacion @ref="dialogo" OnConfirmado="Eliminar"/>
<table class="table">
  <thead>
    <tr>
      <th>ID</th>
      <th>NOMBRE</th>
      <th>APELLIDO</th>
      <th>ACCIÓN</th>
    </tr>
  </thead>
  <tbody>
    . . .
```

Copiar el código del archivo
12_RecursosParaLaTeoria
(diapositivas 55, 56 y 57)



Usar una expresión lambda para que se invoque ConfirmarEliminacion con el parámetro adecuado



```
. . .
@foreach (var cli in _lista)
{
    <tr>
        <td>@cli.Id</td>
        <td>@cli.Nombre</td>
        <td>@cli.Apellido</td>
        <td>
            <button class="btn btn-primary">
                Editar
            </button>
            <button class="btn btn-danger" @onclick="()=>ConfirmarEliminacion(cli)">
                Eliminar
            </button>
        </td>
    </tr>
}
. . .
```




Agregar el código resaltado



```
. . .  
@code {  
    List<Cliente> _lista = new List<Cliente>();  
    protected override void OnInitialized()  
    {  
        _lista = ListarClientesUseCase.Ejecutar();  
    }  
}
```

```
DialogoConfirmacion dialogo = null!;  
Cliente? _clienteParaEliminar = null;  
private void ConfirmarEliminacion(Cliente cli)  
{  
    _clienteParaEliminar = cli;  
    dialogo.Mensaje = $"¿Desea eliminar al cliente {cli.Nombre} {cli.Apellido}?";  
    dialogo.Mostrar();  
}  
private void Eliminar()  
{  
    if (_clienteParaEliminar != null)  
    {  
        EliminarClienteUseCase.Ejecutar(_clienteParaEliminar.Id);  
        _lista = ListarClientesUseCase.Ejecutar(); //se actualiza la lista de clientes  
    }  
}  
}
```


AL.UI


[About](#)

 Home

 Counter

 Weather

 Listado de clientes

 Agregar Cliente

ID	NOMBRE	APELLIDO	ACCIÓN	
1	Alberto	García	Editar	Eliminar
2	Ana	Perez	Editar	Eliminar

¿Desea eliminar al cliente Ana Perez?

[Aceptar](#)

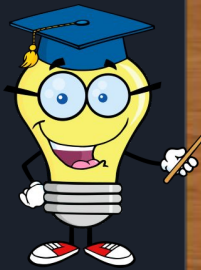
[Cancelar](#)

Otra forma de resolverlo sin utilizar la variable de instancia `_clienteParaEliminar`

```
. . .  
<DialogoConfirmacion @ref="dialogo" />  
. . .  
  
@code {  
. . .  
    private void ConfirmarEliminacion(Cliente cli)  
    {  
        dialogo.Mensaje = $"¿Desea eliminar . . .  
        dialogo.OnConfirmado = EventCallback.Factory.Create(this,  
                                                                () => Eliminar(cli));  
        dialogo.Mostrar();  
    }  
    private void Eliminar(Cliente cli)  
    {  
        EliminarClienteUseCase.Ejecutar(cli.Id);  
        _lista = ListarClientesUseCase.Ejecutar();  
    }  
}
```



Otra forma de resolverlo sin utilizar la variable de instancia `_clienteParaEliminar`



La sentencia

```
EventCallback.Factory  
    .Create(this, () => Eliminar(cli));
```

convierte la expresión lambda `() => Eliminar(cli)`
en un tipo `EventCallback`

¿Cómo modificar un cliente?

El componente `EditorCliente.razor` lo utilizamos para agregar nuevos clientes. Vamos a reutilizarlo también para modificar un cliente existente.

Agregaremos un `parámetro opcional` a la ruta `/cliente` para identificar al cliente que queremos modificar. En caso de que ese parámetro sea `null`, estaremos agregando un nuevo cliente tal cual lo venimos haciendo hasta ahora



Modificar el componente EditarCliente.razor



```
@page "/cliente/{Id:int?}"
@rendermode InteractiveServer
@Inject ObtenerClienteUseCase ObtenerClienteUseCase
@Inject ModificarClienteUseCase ModificarClienteUseCase
@Inject NavigationManager Navegador
@Inject AgregarClienteUseCase AgregarClienteUseCase
...
```

Injectar los casos de uso
ObtenerClienteUseCase y
ModificarClienteUseCase

Agregar en la directiva **@page** un
parámetro de ruta.
Debe coincidir con una propiedad
pública del componente calificada con
[Parameter]

Copiar el código del archivo
12_RecursosParaLaTeoria
(diapositivas 62, 63, 64 y 65)



Modificar el componente EditarCliente.razor



```
...  
@code {  
    Cliente _cliente = new Cliente();  
    [Parameter] public int? Id { get; set; }  
    bool _esNuevoCliente=true;  
    protected override void OnParametersSet()  
    {  
        if (Id != null)  
        {  
            var cli_hallado = ObtenerClienteUseCase.Ejecutar(Id.Value);  
            if (cli_hallado != null)  
            {  
                _cliente = cli_hallado;  
                _esNuevoCliente=false;  
            }  
        }  
    }  
}
```

Debe coincidir con el parámetro de ruta de la directiva `@page`

Este método se invoca cuando el componente ha recibido parámetros y los valores entrantes se han asignado a las propiedades

...



Modificar el componente EditarCliente.razor



```
. . .  
@if (_esNuevoCliente)  
{  
    <h3>Agregando un nuevo Cliente</h3>  
}  
else  
{  
    <h3>Modificando al Cliente "@_cliente.Nombre"</h3>  
}  
  
<input placeholder="Nombre" @bind="_cliente.Nombre" class="form-control">  
<input placeholder="Apellido" @bind="_cliente.Apellido" class="form-control">  
<button class="btn btn-primary" @onclick="Aceptar">Aceptar</button>  
. . .
```

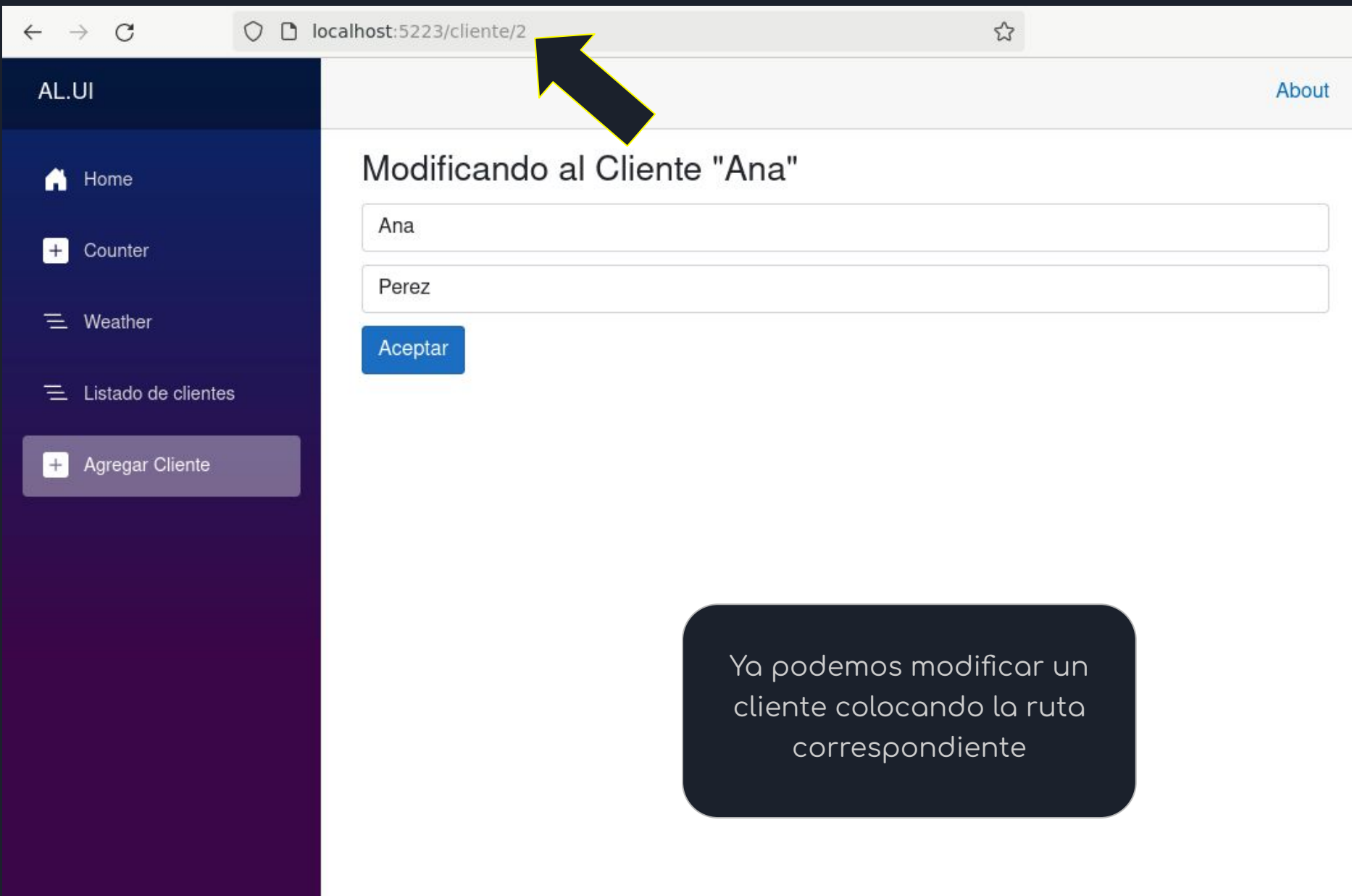
Agregamos un título
distinto en función de lo
que se esté realizando



Modificar el método Aceptar del componente EditarCliente.razor



```
. . .  
    void Aceptar()  
    {  
        if (_esNuevoCliente)  
        {  
            AgregarClienteUseCase.Ejecutar(_cliente);  
        }  
        else  
        {  
            ModificarClienteUseCase.Ejecutar(_cliente);  
        }  
        _cliente = new Cliente();  
        Navegador.NavigateTo("listadoclientes");  
    }  
. . .
```



← → ↻ localhost:5223/listadoclientes ☆

AL.UI [About](#)

- Home
- Counter
- Weather
- Listado de clientes
- Agregar Cliente

ID	NOMBRE	APELLIDO	ACCIÓN
1	Alberto	García	<button>Editar</button> <button>Eliminar</button>
2	Ana	Perez	<button>Editar</button> <button>Eliminar</button>

Falta incorporar la navegación correspondiente a los botones de edición en el componente ListadoClientes.razor



Modificar el componente ListadoClientes.razor



```
@page "/listadoclientes"  
@rendermode InteractiveServer  
@inject ListarClientesUseCase ListarClientesUseCase  
@inject IJSRuntime JsRuntime;  
@inject EliminarClienteUseCase EliminarClienteUseCase  
@inject NavigationManager Navegador
```



```
<table class="table">  
  <thead>  
    <tr>  
      <th>ID</th>  
      <th>NOMBRE</th>  
      <th>APELLIDO</th>  
      <th>ACCIÓN</th>  
    </tr>  
  </thead>  
  . . .
```



Modificar el componente ListadoClientes.razor



```
. . .  
@code {
```

```
. . .
```

```
    void ModificarCliente(Cliente cli)  
    {  
        Navegador.NavigateTo($"cliente/{cli.Id}");  
    }  
  
}
```



Modificar el componente ListadoClientes.razor



```
. . .
@foreach (var cli in _lista)
{
    <tr>
        <td>@cli.Id</td>
        <td>@cli.Nombre</td>
        <td>@cli.Apellido</td>
        <td>
            <button class="btn btn-primary" @onclick="()=>ModificarCliente(cli)">
                Editar
            </button>
            <button class="btn btn-danger" @onclick="()=>ConfirmarEliminacion(cli)">
                Eliminar
            </button>
        </td>
    </tr>
}
. . .
```



The screenshot shows a web application running on localhost:5223/listadoclientes. The left sidebar contains navigation links: Home, Counter, Weather, Listado de clientes (highlighted), and Agregar Cliente. The main content area displays a table of clients with columns for ID, NOMBRE, APELLIDO, and ACCIÓN. The table lists two clients: Alberto García and Ana Perez. Each client has two buttons: 'Editar' (blue) and 'Eliminar' (red). A yellow arrow points from a callout box to the 'Editar' button of the second client.

ID	NOMBRE	APELLIDO	ACCIÓN
1	Alberto	García	<button>Editar</button> <button>Eliminar</button>
2	Ana	Perez	<button>Editar</button> <button>Eliminar</button>

Ya podemos acceder al componente para modificar un cliente desde los botones correspondientes

Fin teoría 12

No se proporcionan ejercicios prácticos sobre esta teoría.

El contenido de esta teoría será aplicado en la resolución del trabajo final del curso.