

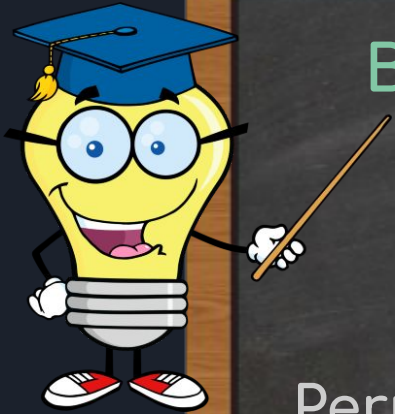


.Net

Teoría 12

Aplicaciones Web con ASP.NET Core Blazor

¿Qué es Blazor?



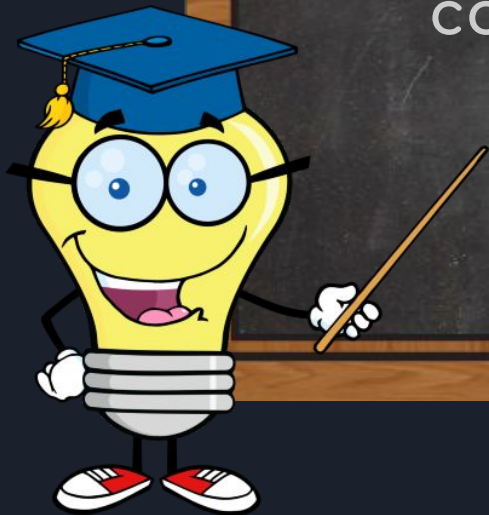
Blazor es un framework de interfaz de usuario para .NET

Es parte de ASP NET Core

Permite crear SPAs (Single-page application) usando como lenguajes de programación C# y Razor Pages, haciendo nula la necesidad de programar en Javascript o frameworks derivados

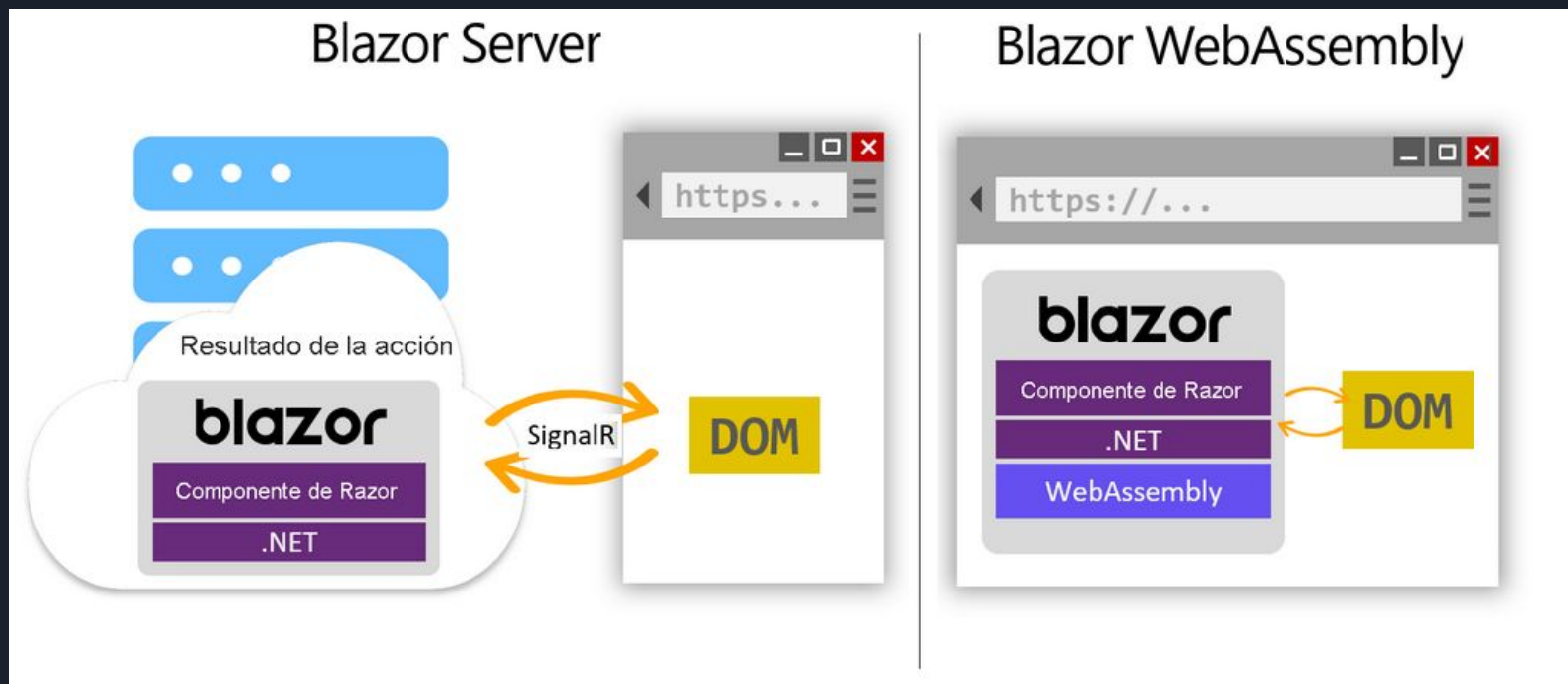
¿Qué es Razor?

Razor es un formato para generar contenido basado en texto como **HTML**. Los archivos **Razor** tienen una extensión de archivo **cshtml** o **razor** y contienen una combinación de código **C#** junto con **HTML**



¿Aplicación del lado del cliente o del servidor?

Las **aplicaciones Blazor** se pueden ejecutar en un servidor como parte de una aplicación ASP.NET o en el explorador del usuario.





Aplicación Blazor Server

- Una **aplicación Blazor Server** se implementa en un servidor web.
- El servidor mantiene con el navegador del usuario un canal de **comunicación bidireccional SignalR**.
- Las acciones de los usuarios sobre la **aplicación** se transmiten por esta conexión **SignalR** al servidor y, si es necesario actualizar la interfaz de usuario, el **framework de Blazor Server** envía en tiempo real al navegador los cambios para que se apliquen a la interfaz de usuario



Aplicación Blazor WebAssembly

- En una **aplicación Blazor WebAssembly**, las **DLL** de la aplicación se transmiten al navegador del usuario y se ejecutan sobre una versión de .NET optimizada para el entorno de ejecución **WebAssembly** del navegador.
- Se desplaza todo el procesamiento de la aplicación a la máquina del usuario. Para obtener datos o interactuar con otros servicios, la aplicación puede usar tecnologías web estándar para comunicarse con servicios HTTP.



Componentes

- Las aplicaciones **Blazor** se basan en componentes.
- Un componente es un elemento de la interfaz de usuario, como una página, un cuadro de diálogo o un formulario de entrada de datos.
- Utilizan sintaxis **Razor** (**C#** y **HTML**) y se escriben en archivos con extensión **.razor**
- Los componentes se compilan en clases **.NET**
- Se pueden anidar y reutilizar.
- Pueden ser “ruteables” (directiva **@page**)



Crear un proyecto Blazor Server



1. Abrir una terminal del sistema operativo
2. Cambiar a la carpeta `proyectosDotnet`
3. Crear la aplicación de Blazor Server con el comando:

Indicamos que no vamos a utilizar una conexión segura https para este proyecto

```
dotnet new blazorserver --no-https -o HolaBlazor
```

4. Abrir `Visual Studio Code` sobre este proyecto y ejecutar

EXPLORER

- HOLABLAZOR
 - .vscode
 - bin
 - Data
 - obj
 - Pages
 - Properties
 - Shared
 - wwwroot
 - @_Imports.razor
 - @App.razor
 - appsettings.Development.json
 - appsettings.json
 - HolaBlazor.csproj
 - Program.cs**

Program.cs

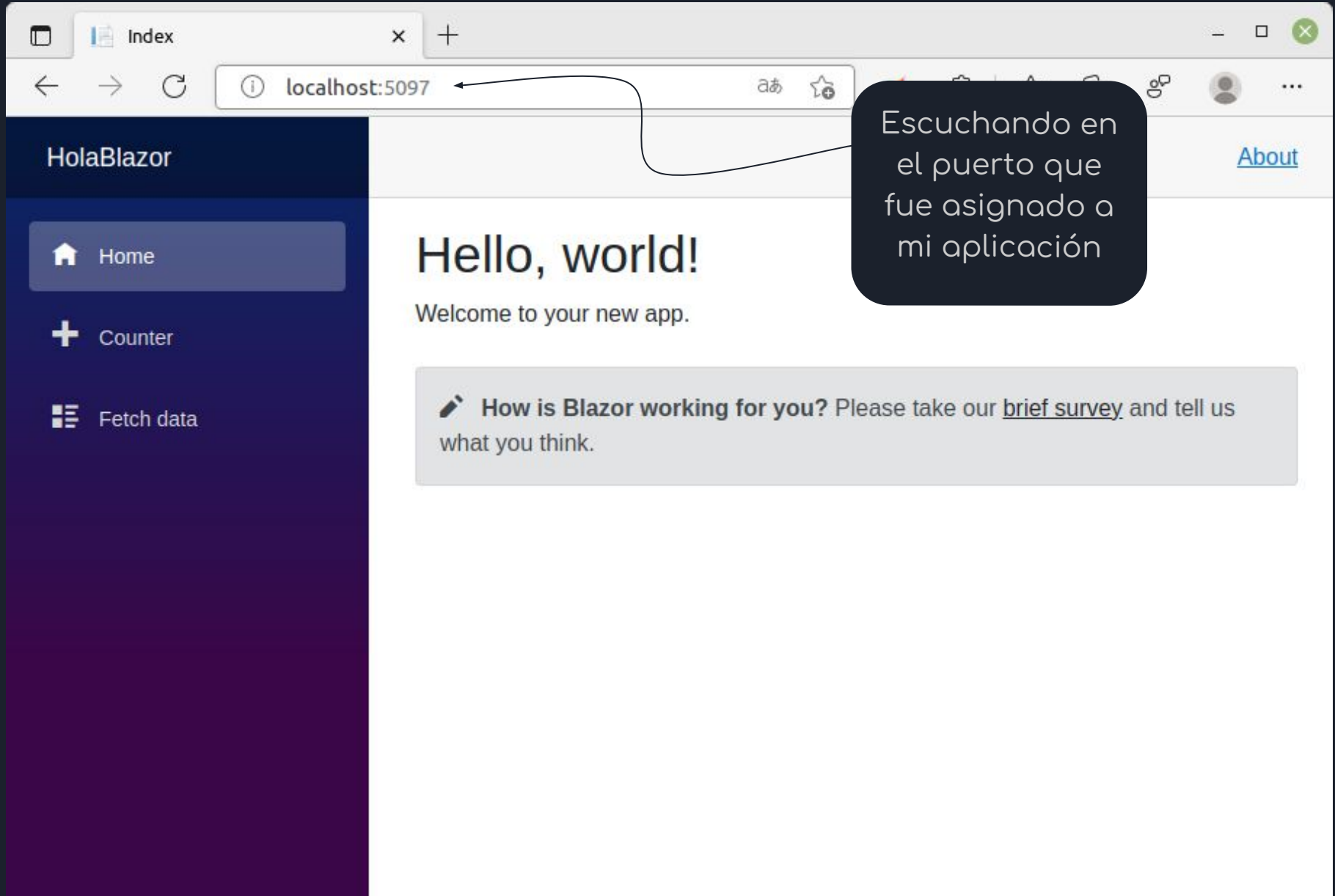
```
1 using Microsoft.AspNetCore.Components;  
2 using Microsoft.AspNetCore.Hosting;  
3 using HolaBlazor.Pages;  
4  
5 var builder = WebApplication.CreateBuilder(args);  
6  
7 // Add services to the container.  
8 builder.Services.AddRazorPages();  
9 builder.Services.AddHttpClient();  
10  
11 var app = builder.Build();  
12  
13 // Configure the HTTP request pipeline.  
14 if (!app.Environment.IsDevelopment())  
15 {  
16     app.UseExceptionHandler("/Error");  
17 }  
18 app.Run();
```

DEBUG CONSOLE

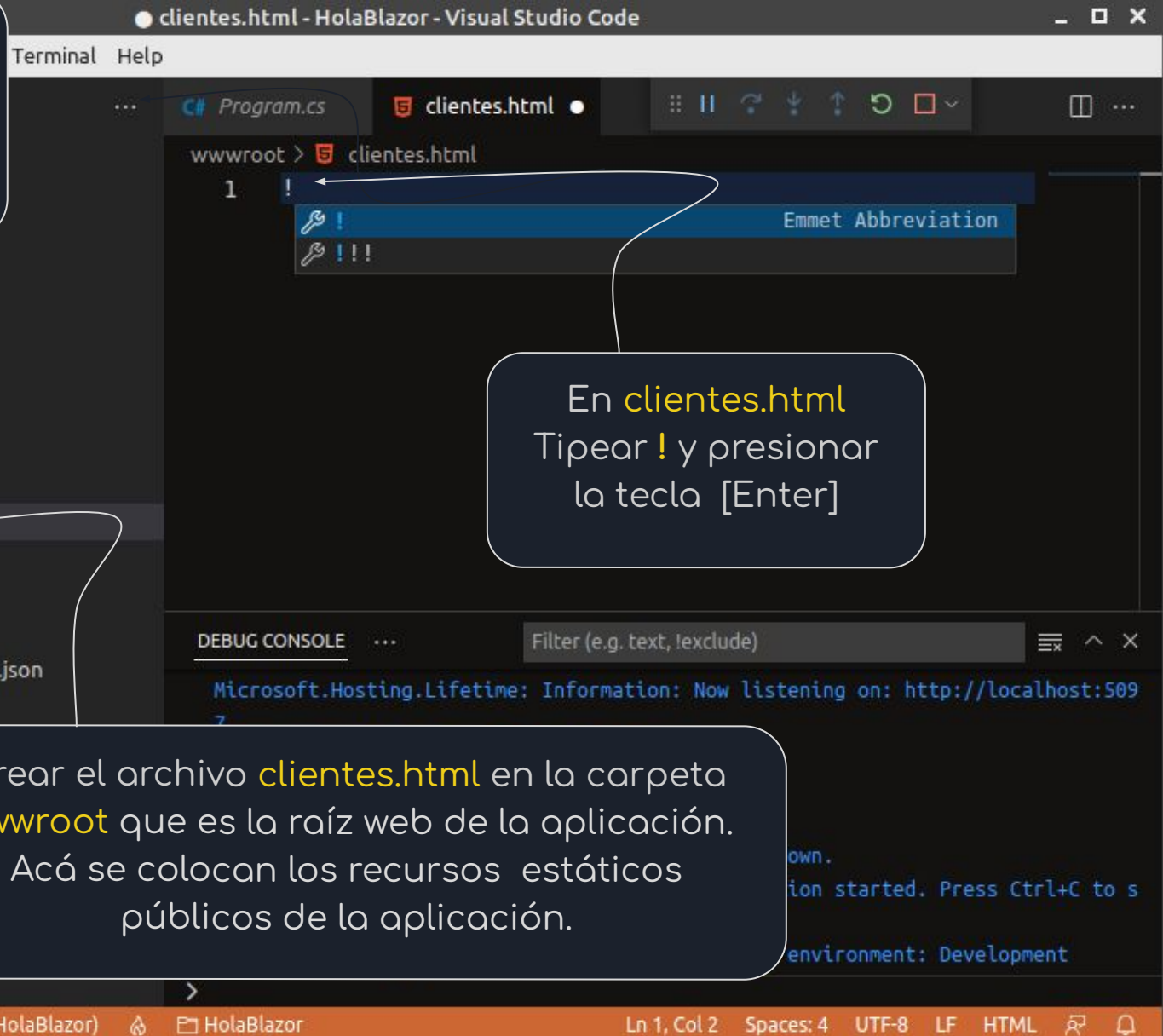
```
do y la opción del depurador 'Sólo mi código' está habilitada. cargado  
Microsoft.Hosting.Lifetime: Information: Now listening on: http://localhost:5097  
info: Microsoft.Hosting.Lifetime[14]  
    Now listening on: http://localhost:5097  
info: Microsoft.Hosting.Lifetime[0]  
    Application started. Press Ctrl+C to shut down.  
Microsoft.Hosting.Lifetime: Information: Application started. Press Ctrl+C to shut down.  
info: Microsoft.Hosting.Lifetime[0]  
    Hosting environment: Development  
Microsoft.Hosting.Lifetime: Information: Hosting environment: Development
```

Annotations:

- En entorno de desarrollo (points to the **Program.cs** file in the Explorer)
- Escuchando en el puerto 5097. Un número que oscila entre 5000 y 5300 y se asigna automáticamente en la creación del proyecto. Se guarda en el archivo **/Properties/launchSettings.json** junto a otras configuraciones (points to the **appsettings.json** file in the Explorer)



Primero
veamos algo
de HTML



Crear el archivo `clientes.html` en la carpeta `wwwroot` que es la raíz web de la aplicación. Acá se colocan los recursos estáticos públicos de la aplicación.

Visual Studio Code nos crea el esquema básico de un documento **html**

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <title>Document</title>
8 </head>
9 <body>
10
11 </body>
12 </html>
```

Entre las etiquetas **<body>** y **</body>** colocaremos el contenido visible de la página web

DEBUG CONSOLE

```
Microsoft.Hosting.Lifetime: Information: Now listening on: http://localhost:5097
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5097
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
Microsoft.Hosting.Lifetime: Information: Application is shutting down.
```


No cerrar la aplicación, que siga corriendo

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <title>Document</title>
8 </head>
9 <body>
10  <h1> Listado de clientes </h1>
11 </body>
12 </html>
```

Agregar, salvar el archivo y con el navegador acceder a </clientes.html>

Entre las etiquetas `<h1>` y `</h1>` se coloca un encabezado de nivel 1

Document
localhost:5097/clientes.html

Listado de clientes

```
Microsoft.Hosting.Lifetime: Information: Now listening on: http://localhost:5097
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5097
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

Elementos HTML

- La mayoría de los elementos se definen con un tag de apertura y uno de cierre. Ejemplo:

```
<p></p>
```

- Hay algunas excepciones a esta regla. Ejemplo:

```
<br>, <img>
```

- Los tags pueden tener atributos. Ejemplo:

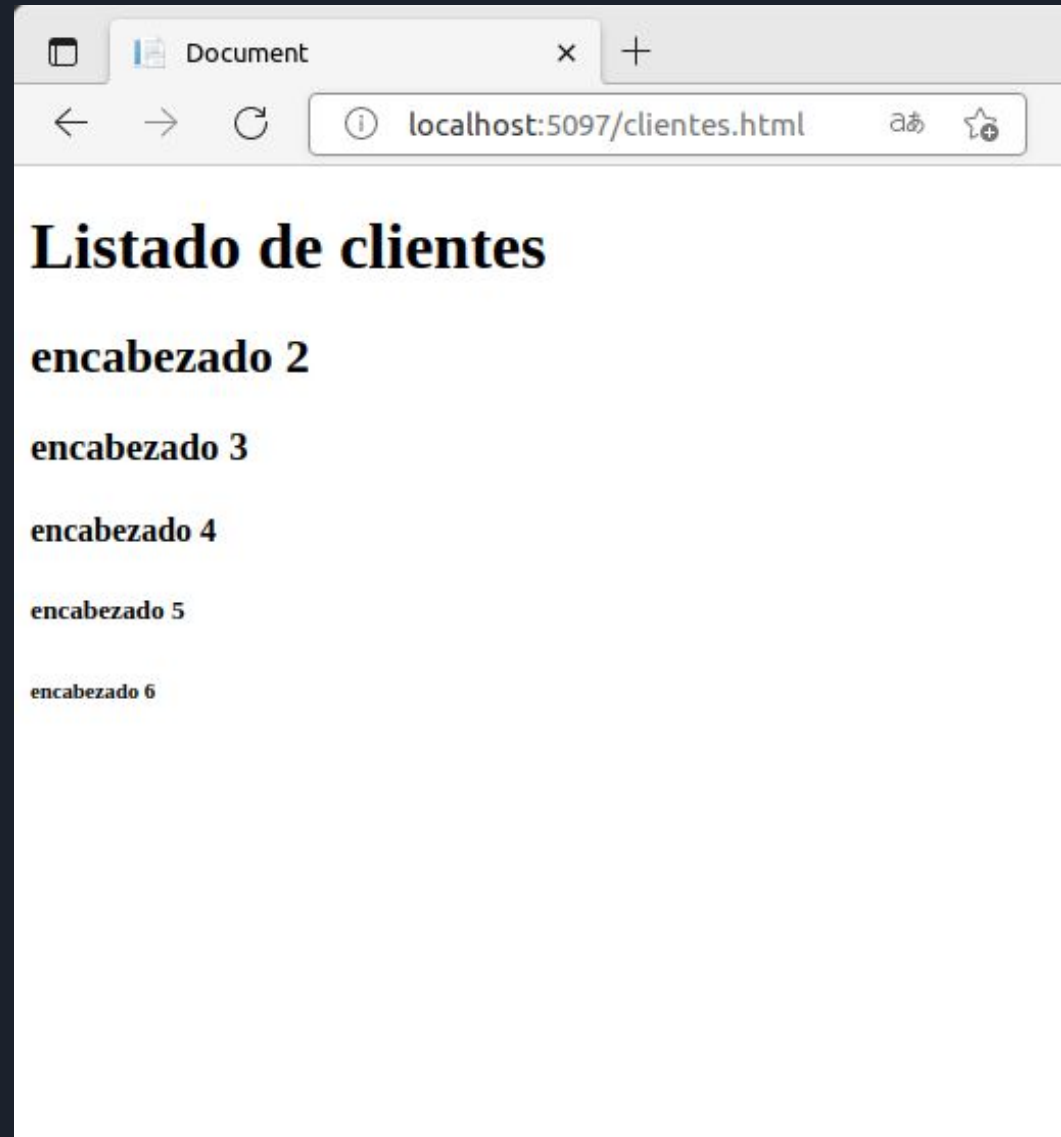
```
<a href="http://www.google.com">Google</a>
```

- Puede haber comentarios en el código HTML, el cuál no será procesado por el navegador. Ejemplo:

```
<!-- Esto es un comentario -->
```

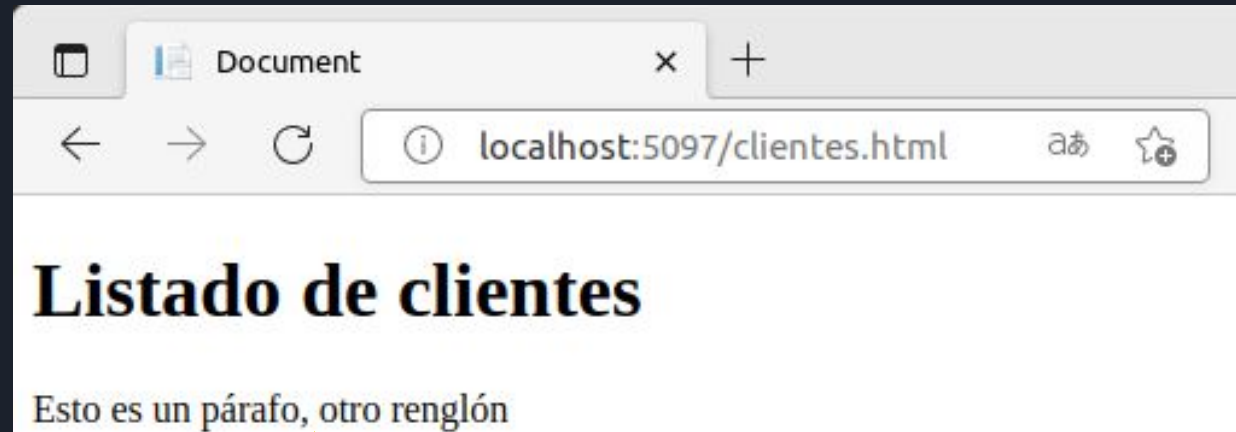
Realizar las siguientes pruebas

```
<body>  
  <h1> Listado de clientes </h1>  
  <h2>encabezado 2</h2>  
  <h3>encabezado 3</h3>  
  <h4>encabezado 4</h4>  
  <h5>encabezado 5</h5>  
  <h6>encabezado 6</h6>  
</body>
```



Probar el siguiente código

```
<body>  
  <h1> Listado de clientes </h1>  
  <p> Esto es un párafo,  
    otro renglón </p>  
</body>
```



Se ignoran los fines de línea

Probar el siguiente código

```
<body>  
  <h1> Listado de clientes </h1>  
  <p> Esto es un párafo, <br>  
    otro renglón </p>  
</body>
```

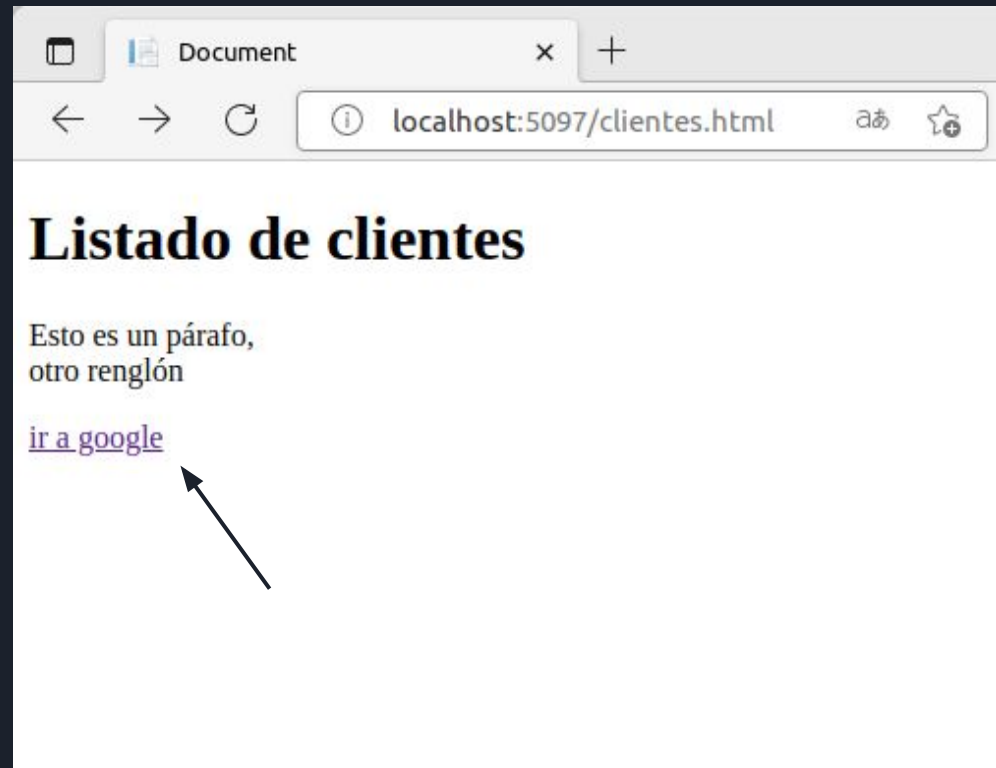
Agregar la etiqueta
de fin de línea



Probar el siguiente código

```
<body>  
  <h1> Listado de clientes </h1>  
  <p> Esto es un párafo,<br>  
    otro renglón </p>  
  <a href="http://www.google.com">ir a google</a>  
</body>
```

La etiqueta `<a>` se utiliza para colocar un link





Atributos HTML

- Los atributos HTML proporcionan información adicional sobre los elementos HTML.
- Todos los elementos HTML pueden tener atributos
- Los atributos siempre se especifican en la etiqueta de inicio
- Los atributos generalmente vienen en pares de la forma: `nombre="valor"`

Atributos HTML

- La etiqueta `` se utiliza para incrustar una imagen en una página HTML. El atributo `src` especifica la ruta a la imagen que se mostrará. También puede contener los atributos `width` y `height`, que especifican el ancho y el alto de la imagen (en píxeles). Ejemplo:

```

```

- Siempre se debe incluir el atributo `lang` en la etiqueta `<html>`, para declarar el idioma de la página web. Esto está destinado a ayudar a los motores de búsqueda y navegadores. Ejemplo:

```
<html lang="es">
```

Probar el siguiente código

```
<body>
  <h1> Listado de clientes </h1>
  <ul>
    <li title="El primero">Juan</li>
    <li>María</li>
    <li>Laura</li>
  </ul>
  <ol>
    <li>Juan</li>
    <li>María</li>
    <li>Laura</li>
  </ol>
</body>
```

`` se usa para especificar un ítem dentro de una lista
¿Cual es la diferencia entre `` y ``?
¿Para qué sirve el atributo `title`?



Etiquetas <div> y

- <div>

Elemento utilizado para agrupar otros elementos HTML. Es un elemento a nivel bloque, por lo tanto, el navegador por defecto mostrará un salto de línea antes y después de él

-

Elemento utilizado para agrupar elementos de texto. Es un elemento a nivel línea, por lo tanto, el navegador por defecto NO mostrará un salto de línea antes y después de él

Probar el siguiente código

```
<head>
  . . .
  <style>
    #encabezado {
      background-color: gray;
      font-size: xx-large;
    }
    #rojo {
      color: red;
    }
  </style>
</head>
<body>
  <div id="encabezado">
    <p>primer párrafo</p>
    <p>segundo párrafo</p>
    <span id="rojo">este texto es rojo</span> pero este no
  </div>
  <p>esto está fuera del encabezado</p>
</body>

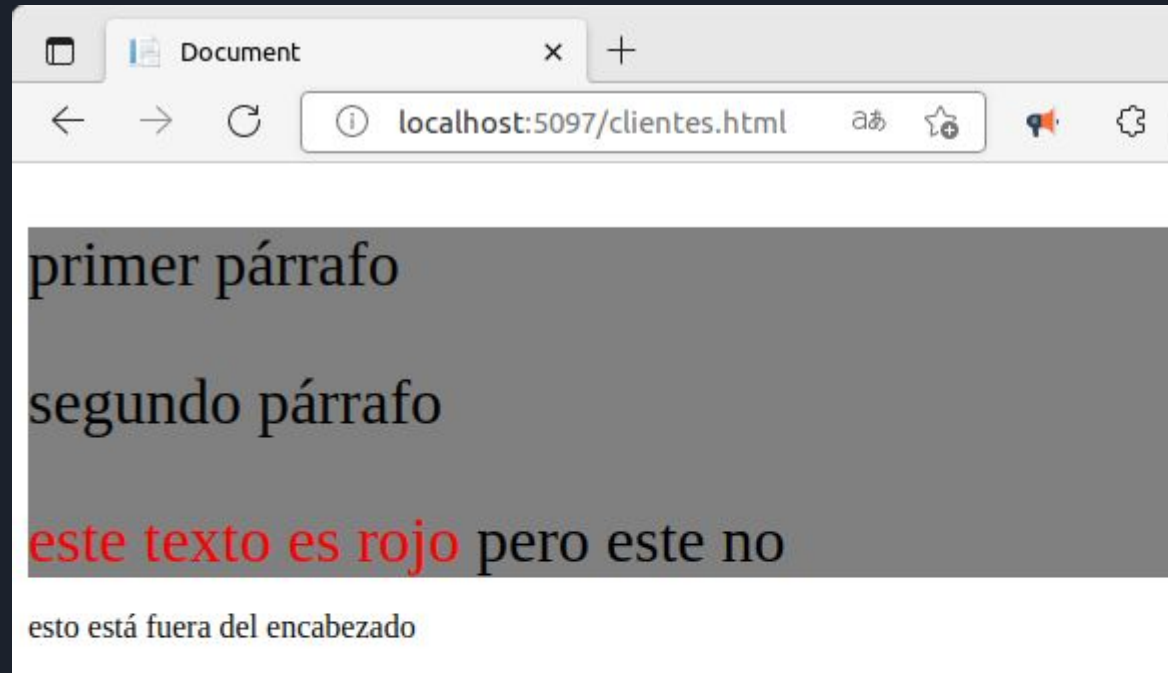
</html>
```

Estilos CSS que se aplican a los
elementos con atributo
`id="encabezado"` y `id="rojo"`

Probar el siguiente código

```
<head>
  . . .
  <style>
    #encabezado {
      background-color: gray;
      font-size: xx-large;
    }
    #rojo {
      color: red;
    }
  </style>
</head>
<body>
  <div id="encabezado">
    <p>primer párrafo</p>
    <p>segundo párrafo</p>
    <span id="rojo">este texto es rojo</span> pero este no
  </div>
  <p>esto está fuera del encabezado</p>
</body>

</html>
```

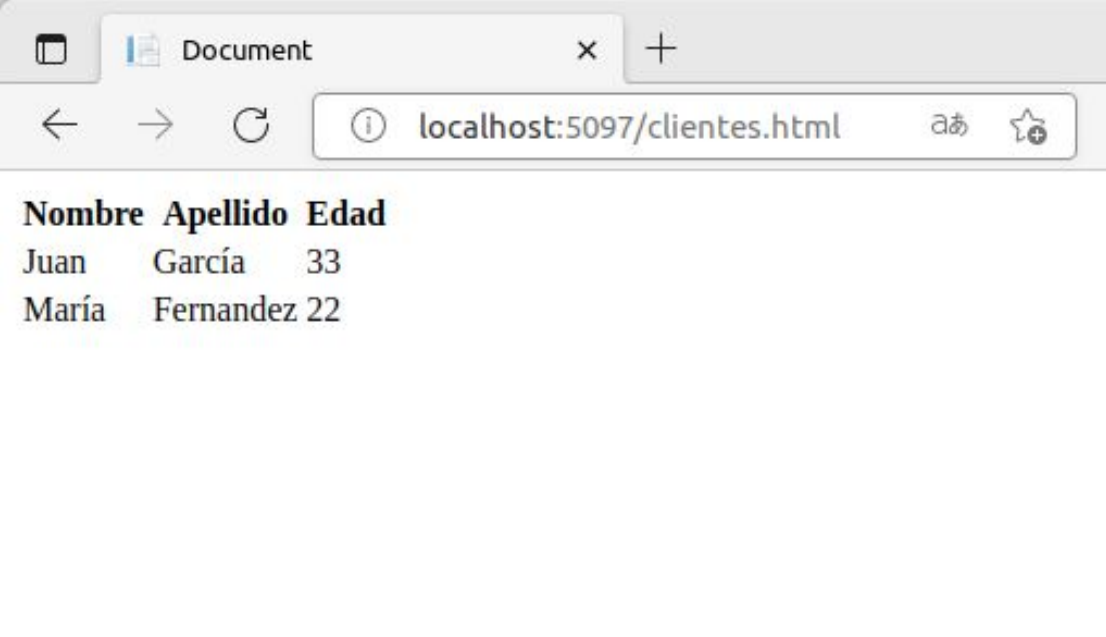


Tablas

- La etiqueta `<table>` se utiliza colocar una tabla
- Las tablas están conformadas por filas demarcadas con etiquetas `<tr>` (table row)
- Cada fila tendrá una cierta cantidad de celdas demarcadas con etiquetas `<td>` (table data)
- Usualmente las celdas de la primera fila se indican con etiquetas `<th>` (table header)

Probar el siguiente código

```
<body>
  <h1> Listado de clientes </h1>
  <table>
    <tr>
      <th>Nombre</th>
      <th>Apellido</th>
      <th>Edad</th>
    </tr>
    <tr>
      <td>Juan</td>
      <td>García</td>
      <td>33</td>
    </tr>
    <tr>
      <td>María</td>
      <td>Fernandez</td>
      <td>22</td>
    </tr>
  </table>
</body>
```



A screenshot of a web browser window. The title bar shows 'Document' and a single tab. The address bar displays 'localhost:5097/clientes.html'. The page content shows a table with three columns: 'Nombre', 'Apellido', and 'Edad'. The first row contains 'Juan', 'García', and '33'. The second row contains 'María', 'Fernandez', and '22'.

Nombre	Apellido	Edad
Juan	García	33
María	Fernandez	22

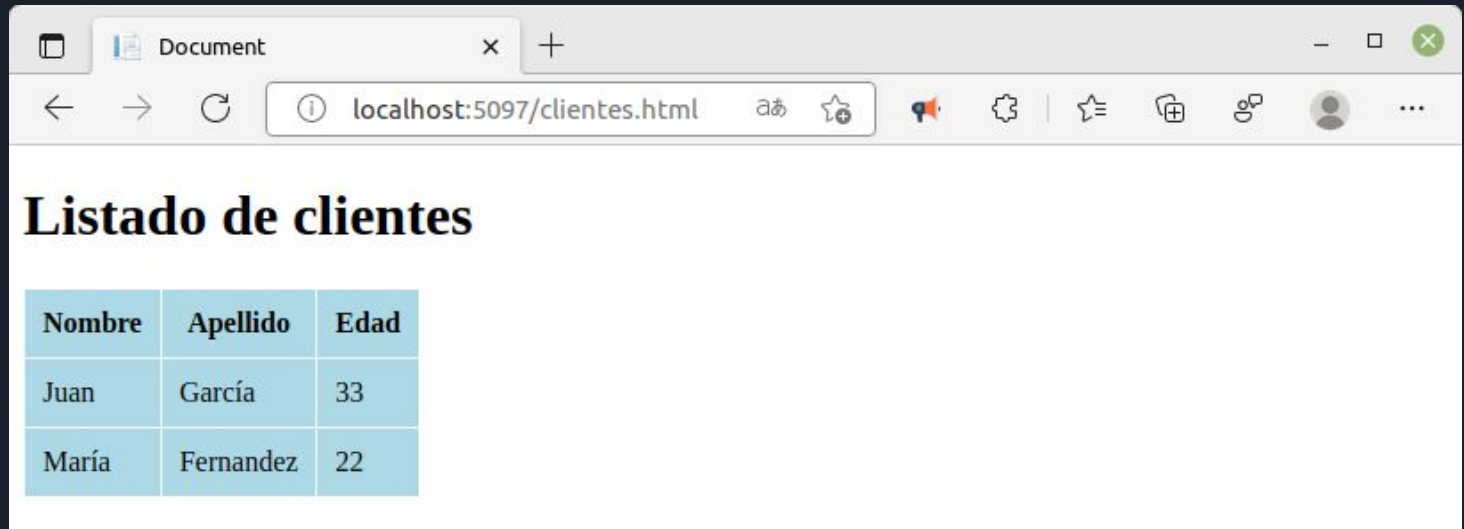
Copiar el código del archivo
12_RecursosParaLaTeoria.txt

Probar el siguiente código

```
<style>
  table, th, td {
    border: 1px solid white;
    border-collapse: collapse;
    padding: 10px;
  }
  th, td {
    background-color: #96D4D4;
  }
</style>
```

Estilos CSS que se aplican a todos los elementos `table`, `th` y `td`

Estilos CSS que se aplican a todos los elementos `th` y `td`



Componentes Blazor

The screenshot shows the Visual Studio Code interface with the file explorer on the left and the code editor on the right. The file explorer shows a project named 'HOLABLAZOR' with a 'Pages' folder. The 'Pages' folder is expanded, showing several files including '@_Host.cshtml', '@_Layout.cshtml', '@Counter.razor', '@Error.cshtml', '@Error.cshtml.cs', '@FetchData.razor', '@Hola.razor', and '@Index.razor'. The '@Hola.razor' file is selected. The code editor shows the content of '@Hola.razor' with the following code:

```
1 @page "/hola"
2
3 <h1>Hola mundo</h1>
4
5
6
7
```

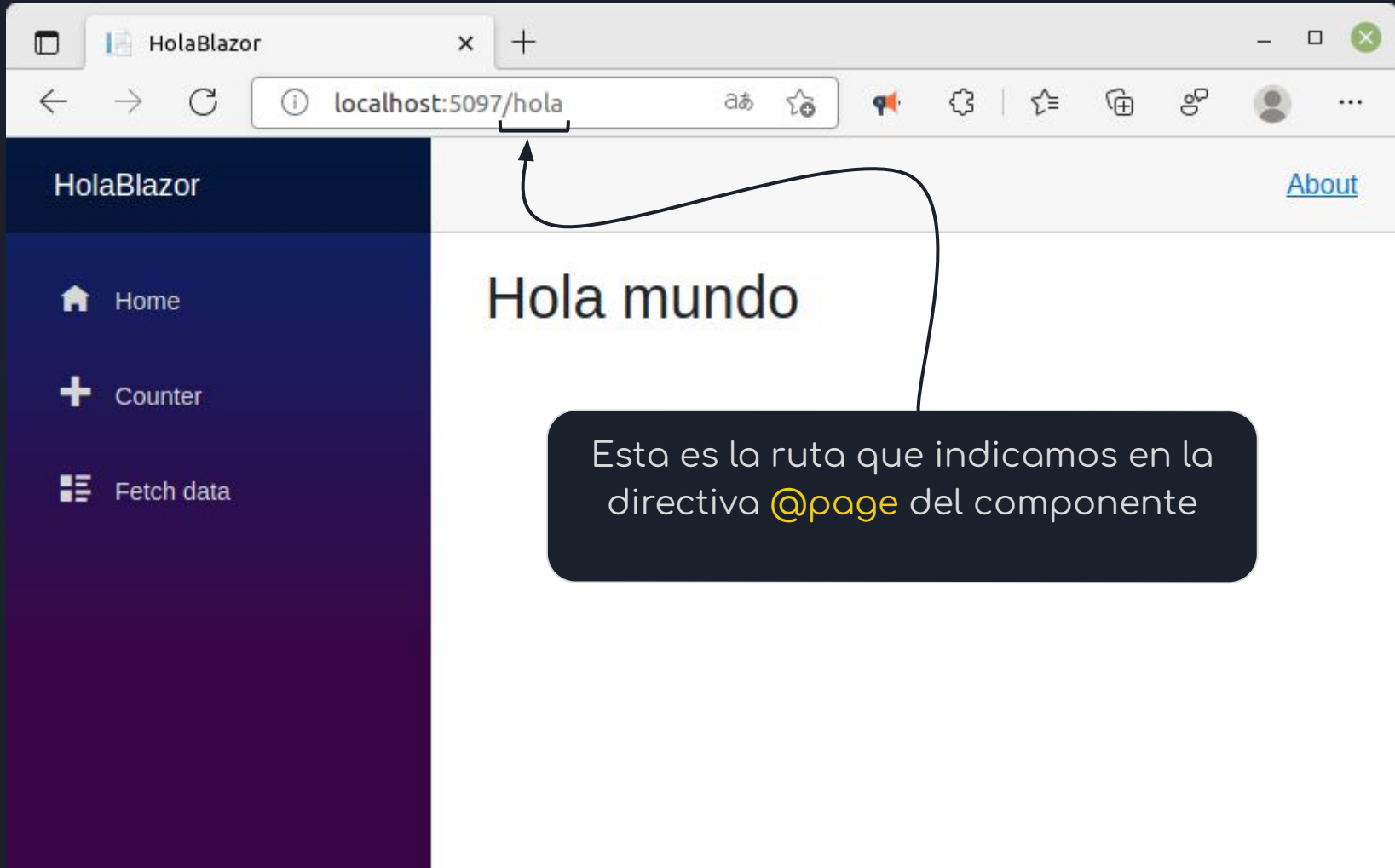
A yellow box highlights the code in the editor. A callout box with a white border and a dark blue background contains the following text:

Crear el archivo **Hola.razor** en la carpeta **Pages** con este contenido
Por convención en esta carpeta se colocan los componentes "ruteables"

The status bar at the bottom shows the following information: 0 errors, 0 warnings, .NET Core Launch (web) (HolaBlazor), HolaBlazor, Ln 7, Col 1, Spaces: 4, UTF-8, LF, ASP.NET Razor.



Detener la aplicación y ejecutarla nuevamente.
Acceder desde el navegador a /hola



Index.razor - HolaBlazor - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

- HOLA BLAZOR
 - @ _Host.cshtml
 - @ _Layout.cshtml
 - @ Counter.razor
 - @ Error.cshtml
 - # Error.cshtml.cs
 - @ FetchData.razor
 - @ Hola.razor
 - @ Index.razor
- Properties
- Shared
 - @ MainLayout.razor
 - MainLayout.razor.css
 - @ NavMenu.razor
 - NavMenu.razor.css
 - @ SurveyPrompt.razor
- wwwroot
- @ _Imports.razor
- @ App.razor

Pages > @ Index.razor

```
5 <h1>Hello, world!</h1>
6
7 Welcome to your new app.
8
9 <SurveyPrompt Title="How is Blazor working for you?"
10
11 <Hola/>
12 <Hola/>
13
```

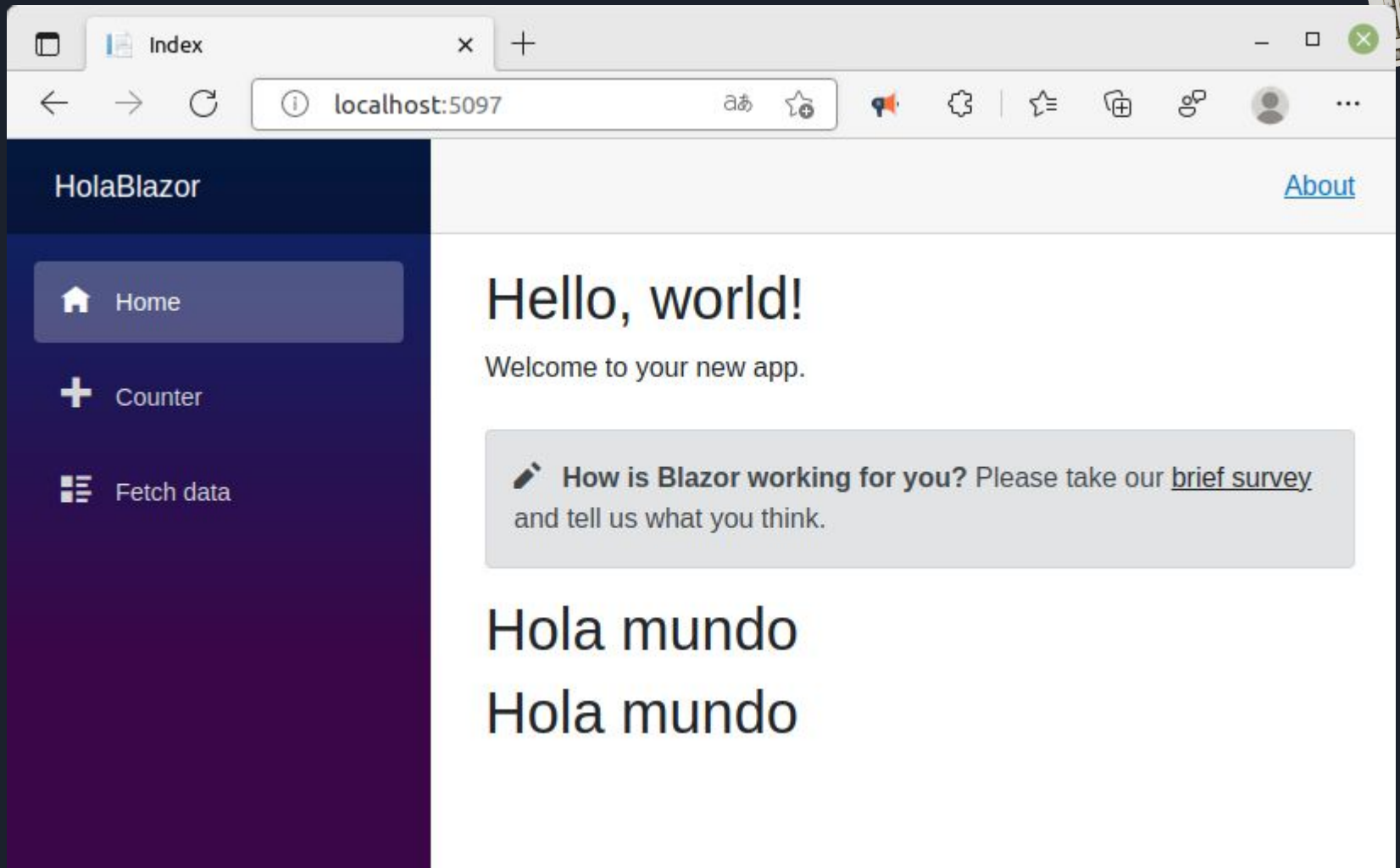
Agregar en **Index.razor** estas dos etiquetas

Hola es un componente, por lo tanto se puede insertar como una etiqueta en páginas razor y otros componentes razor

0 0 .NET Core Launch (web) (HolaBlazor) HolaBlazor Ln 4, Col 1 Spaces: 4 UTF-8 with BOM CRLF ASP.NET Razor



Detener la aplicación y ejecutarla nuevamente





Modificar Hola.razor de la siguiente manera y volver a ejecutar



```
@page "/hola"
```

```
<h1>Hola @nombre</h1>
```

```
@code{
```

```
    string nombre="Juan";
```

```
}
```

Esta es la sección de la vista.
Utilizamos la @ para cambiar
entre código HTML y C#, en este
caso para acceder a la variable
nombre

Sección de código C#, en este caso
sólo estamos definiendo la variable
nombre

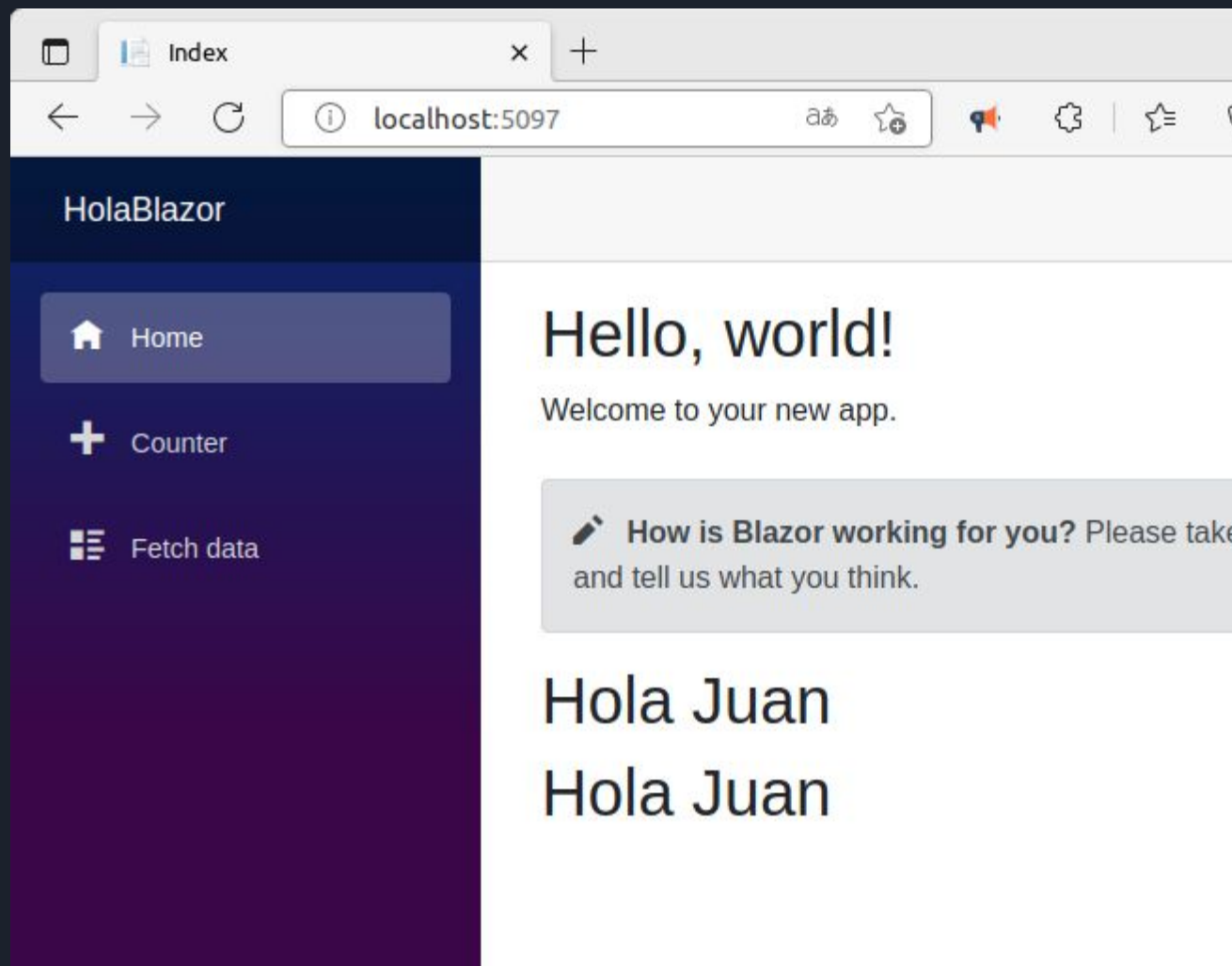
La sección se identifica con

@code{...}

```
@page "/hola"
```

```
<h1>Hola @nombre</h1>
```

```
@code{  
    string nombre="Juan";  
}
```



The screenshot shows the Visual Studio Code interface with the file `Hola.razor` open. The editor displays the following code:

```
1 @page "/hola"
2
3 <h1>Hola @nombre</h1>
4
5 @code{
6     string nombre="Juan";
7 }
8
9
```

A breakpoint is set on line 6, which is highlighted in yellow. The left sidebar shows the **WATCH** pane with the variable `this: {HolaBlazor.Pages.Hola}`. The **CALL STACK** pane shows the current execution context: `.NET ThreadPool Worker` (PAUSED) and `.NET ThreadPool ...` (PAUSED ON BREAKPOINT). The bottom status bar indicates the file is `HolaBlazor` at `Ln 6, Col 25 (21 selected)` with `Spaces: 4`, `UTF-8`, `LF`, and `ASP.NET Razor` syntax.

Si colocamos un breakpoint, podemos ver en ejecución que la clase C# que se crea a partir del archivo .razor coincide con el nombre del archivo y su namespace se determina por la carpeta en la que está incluido

Secuencia de escape para @

Si fuese necesario escribir la @ en un componente razor se debe utilizar @@ por ejemplo:

@@Username

Se visualizará @Username

De lo contrario se esperaría que Username fuese un campo o una propiedad definida en el componente





Modificar Hola.razor de la siguiente manera y volver a ejecutar



```
@page "/hola"
```

```
<h1>Hola @nombre.ToUpper()</h1>
```

```
<p>El doble de 5 es @(5*2) y la Sumatoria  
de 1 a 10 es @Sumatoria(10) </p>
```

```
@code {  
    string nombre = "Juan";  
    int Sumatoria(int n) =>  
        Enumerable.Range(1, n).Sum();  
}
```

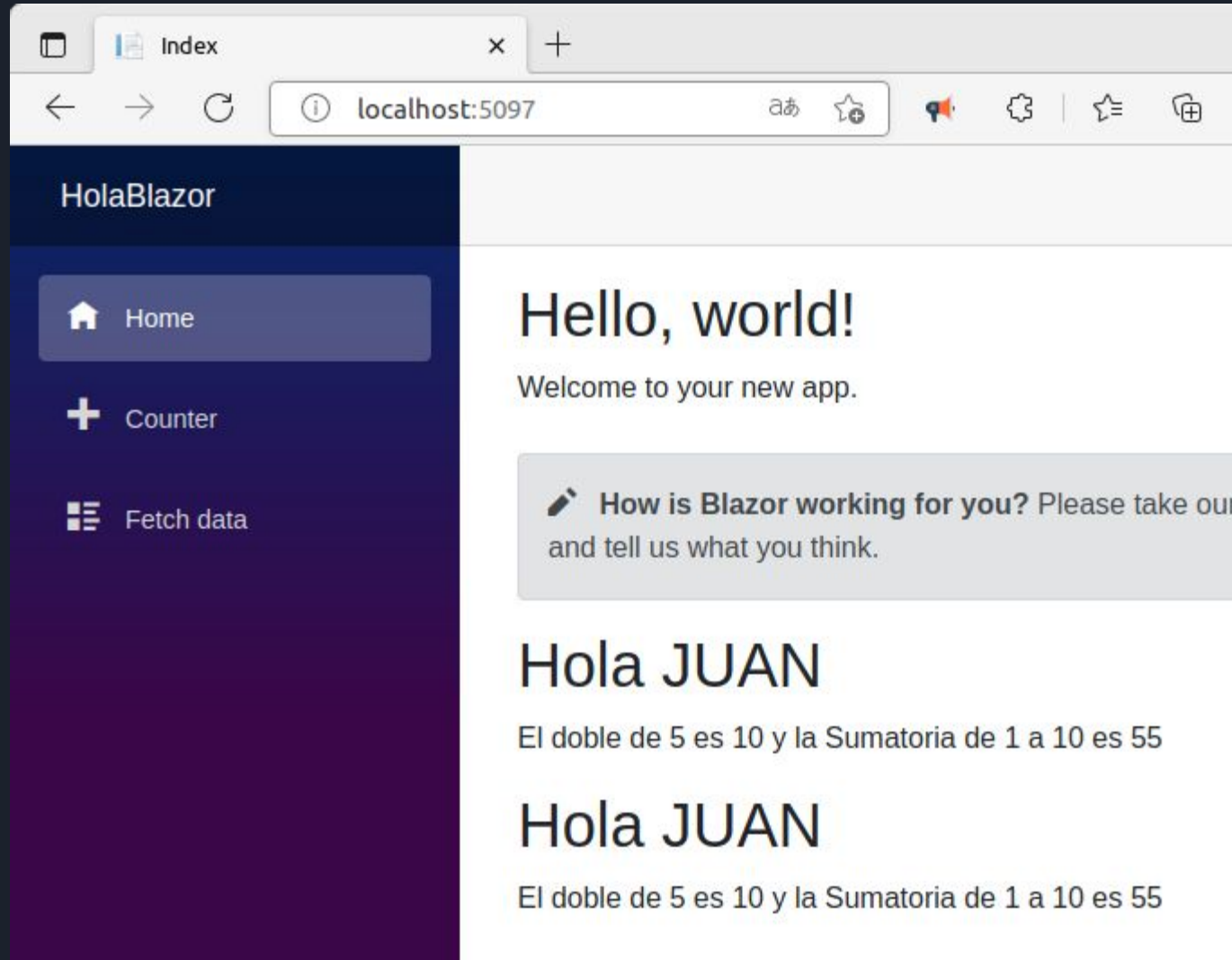
HTML

```
@page "/hola"
```

```
<h1>Hola @nombre.ToUpper()</h1>
```

```
<p>El doble de 5 es @(5*2) y la Sumatoria  
de 1 a 10 es @Sumatoria(10) </p>
```

```
@code {  
    string nombre = "Juan";  
    int Sumatoria(int n) =>  
        Enumerable.Range(1, n).Sum();  
}
```



Expresiones implícitas y explícitas en Razor

Las expresiones implícitas Razor comienzan por @ seguida de código de C#. Generalmente no admiten espacios y no se indica dónde terminan, se trata de expresiones simples, por ejemplo

@nombre.ToUpper()

En algunas expresiones es necesario indicar cuál es el fin de la misma, se llaman expresiones explícitas y se denotan entre paréntesis, por ejemplo:

@(5*2)

De lo contrario, @5*2 provocaría error de compilación



Modificar los componentes Index.razor y Hola.razor y volver a ejecutar



```
----- Index.razor -----
```

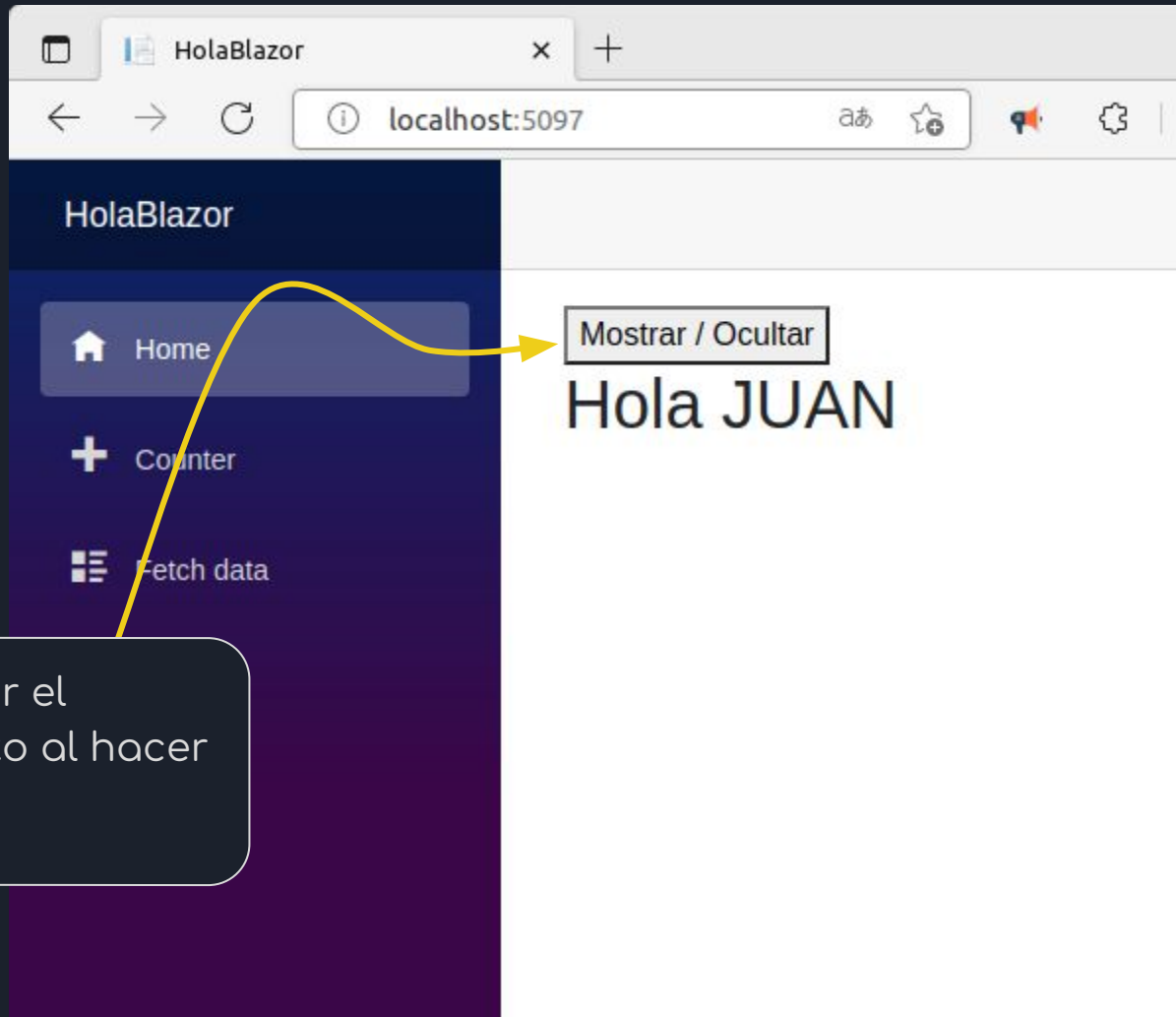
```
@page "/"  
<Hola/>
```

```
----- Hola.razor -----
```

```
@page "/hola"  
<button @onclick="Cambiar">Mostrar / Ocultar</button>  
@if (EsVisible)  
{  
    <h1>Hola @nombre.ToUpper()</h1>  
}  
@code {  
    string nombre = "Juan";  
    bool EsVisible = true;  
    void Cambiar() {  
        EsVisible = !EsVisible;  
    }  
}
```

HTML

```
@page "/hola"
<button @onclick="Cambiar">Mostrar / Ocultar</button>
@if (EsVisible)
{
    <h1>Hola @nombre.ToUpper()</h1>
}
@code {
    string nombre = "Juan";
    bool EsVisible = true;
    void Cambiar() {
        EsVisible = !EsVisible;
    }
}
```



Persona.cs - HolaBlazor - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

HOLABLAZOR

- .vscode
- bin
- Data
- Entidades
- C# Persona.cs
- obj
- Pages
 - @_Host.cshtml
 - @_Layout.cshtml
 - @Counter.razor
 - @Error.cshtml
 - C# Error.cshtml.cs
 - @FetchData.razor
 - @Hola.razor
 - @Index.razor
- Properties
- Shared
- wwwroot
 - css
 - clientes.html
 - favicon.ico

OUTLINE

TIMELINE

Entidades > C# Persona.cs > ...

```
1 namespace HolaBlazor.Entidades;
2 class Persona
3 {
4     public string Nombre { get; set; } = "";
5     public string Apellido { get; set; } = "";
6     public int? Edad { get; set; } //podría faltar el dato de la edad
7
8     // vamos a hardcodear una lista de personas
9     // que usaremos en los siguientes ejemplos
10    // para ello definimos el siguiente método estático
11    public static List<Persona> GetLista()
12    {
13        return new List<Persona>() {
14            new Persona() {Nombre="Pablo",Apellido="Perez", Edad=34},
15            new Persona() {Nombre="Laura",Apellido="García", Edad=30},
16            new Persona() {Nombre="José",Apellido="Lopez", Edad=45},
17            new Persona() {Nombre="Ana",Apellido="Colombo", Edad=21},
18            new Persona() {Nombre="María",Apellido="Suarez", Edad=15},
19        };
20    }
21 }
22
```

Crear la carpeta Entidades y dentro de ella la clase Persona

Definirla en el namespace HolaBlazor.Entidades

Copiar el código del archivo 12_RecursosParaLaTeoria.txt



Modificar el componente Hola.razor y volver a ejecutar



```
@page "/hola"
```

```
@using HolaBlazor.Entidades
```

```
<h1>Listado de personas</h1>
```

```
<ul>
```

```
@foreach (var p in lista)
```

```
{
```

```
    <li>@p.Apellido, @p.Nombre (@p.Edad)</li>
```

```
}
```

```
</ul>
```

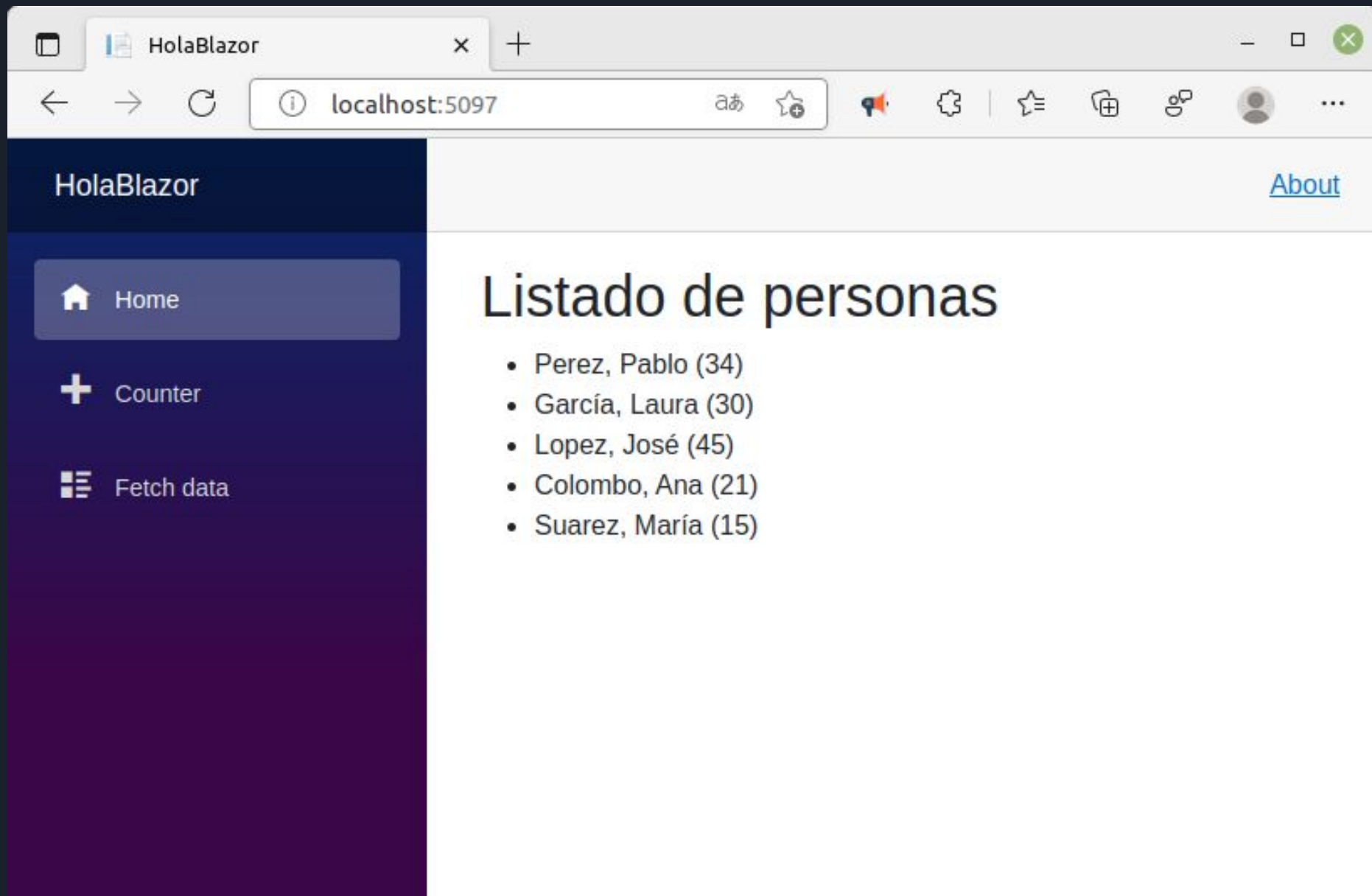
```
@code {
```

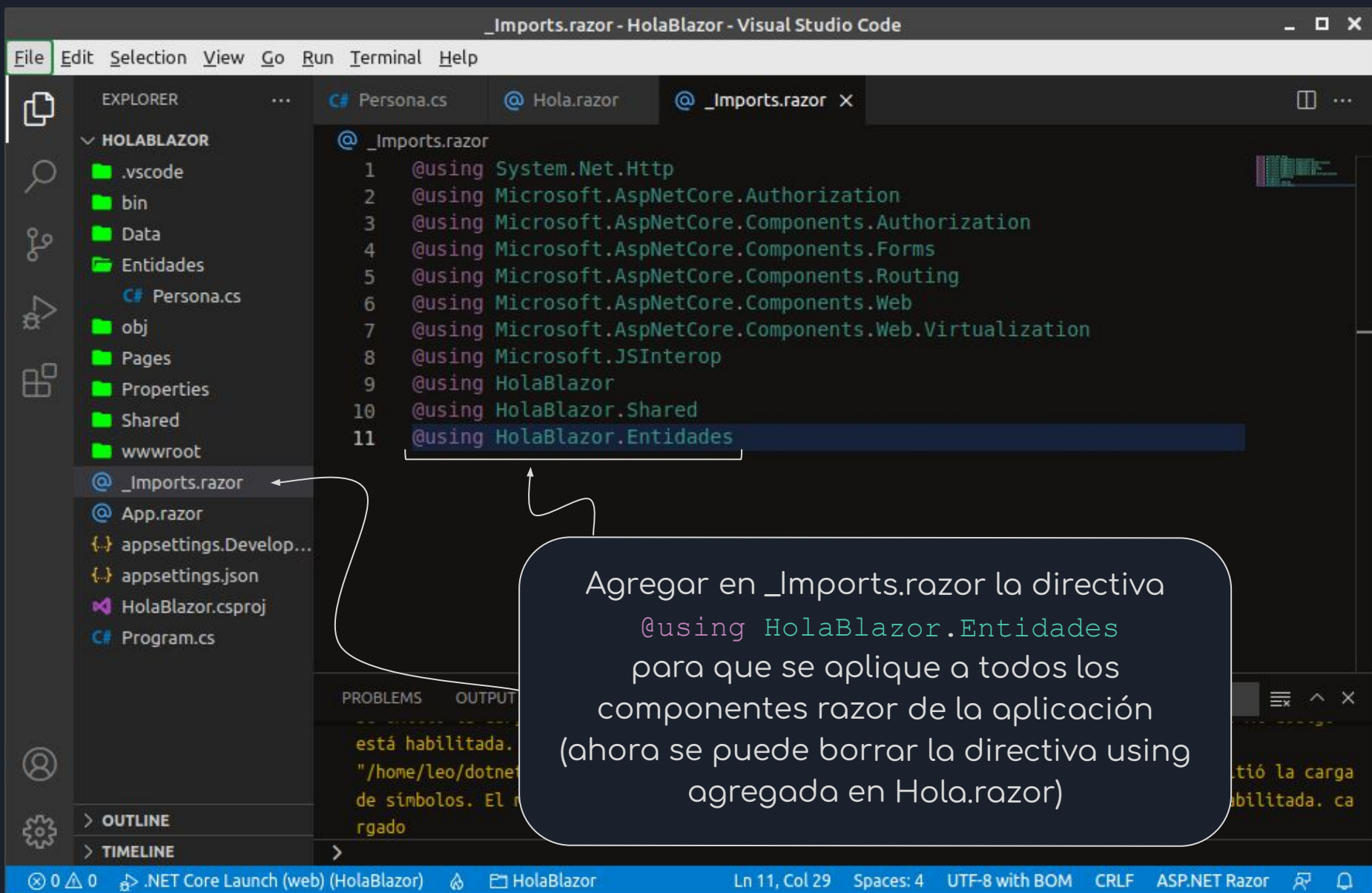
```
    List<Persona> lista = Persona.GetLista();
```

```
}
```

Directiva `@using`
Observar que no es necesario
el punto y coma (;) final

Copiar el código del archivo
12_RecursosParaLaTeoria.txt





Visual Studio Code interface showing the file explorer on the left with the project structure of **HOLABLAZOR**. The file **_Imports.razor** is selected in the explorer and open in the editor. The editor shows the following code in **@_Imports.razor**:

```
1 @using System.Net.Http
2 @using Microsoft.AspNetCore.Authorization
3 @using Microsoft.AspNetCore.Components.Authorization
4 @using Microsoft.AspNetCore.Components.Forms
5 @using Microsoft.AspNetCore.Components.Routing
6 @using Microsoft.AspNetCore.Components.Web
7 @using Microsoft.AspNetCore.Components.Web.Virtualization
8 @using Microsoft.JSInterop
9 @using HolaBlazor
10 @using HolaBlazor.Shared
11 @using HolaBlazor.Entidades
```

The last line, `@using HolaBlazor.Entidades`, is highlighted. A callout box explains the purpose of this directive:

Agregar en `_Imports.razor` la directiva `@using HolaBlazor.Entidades` para que se aplique a todos los componentes razor de la aplicación (ahora se puede borrar la directiva using agregada en `Hola.razor`)

The status bar at the bottom indicates the file is `Ln 11, Col 29` with `Spaces: 4`, `UTF-8 with BOM`, `CRLF`, and `ASP.NET Razor`.



Modificar el componente Hola.razor y volver a ejecutar



```
@page "/hola"

<h1>Listado de personas</h1>
<ul>
    @foreach (var p in lista)
    {
        <li>@p.Apellido, @p.Nombre (@p.Edad)</li>
    }
</ul>
<button @onclick="Agregar">Agregar a Carlos</button>

@code {
    List<Persona> lista = Persona.GetLista();
    void Agregar() => lista.Add(new Persona() {
        Nombre = "Carlos",
        Apellido = "Maldini",
        Edad = 66
    });
}
```

Copiar el código del archivo
12_RecursosParaLaTeoria.txt

HolaBlazor

[About](#)

Listado de personas

- Perez, Pablo (34)
- García, Laura (30)
- Lopez, José (45)
- Colombo, Ana (21)
- Suarez, María (15)

Agregar a Carlos

Cada vez que se hace click se agrega Carlos a la lista



Modificar el componente Hola.razor y volver a ejecutar



```
@page "/hola"

<h1>Listado de personas</h1>
<ul>
    @foreach (var p in lista)
    {
        <li>@p.Apellido, @p.Nombre (@p.Edad)</li>
    }
</ul>
<input placeholder="Nombre" @bind="p.Nombre" /><br>
<input placeholder="Apellido" @bind="p.Apellido" /><br>
<input type="number" placeholder="Edad" @bind="p.Edad" /><br>
<button @onclick="Agregar">Agregar</button>

@code {
    List<Persona> lista = Persona.GetLista();
    Persona p = new Persona();
    void Agregar()
    {
        lista.Add(p);
        p = new Persona();
    }
}
```

HolaBlazor

Home

Counter

Fetch data

About

Listado de personas

- Perez, Pablo (34)
- García, Laura (30)
- Lopez, José (45)
- Colombo, Ana (21)
- Suarez, María (15)

Nombre

Apellido

Edad

Agregar

Ahora se puede editar el nombre, apellido y edad de la nueva persona

```
. . .  
</ul>  
<input @ref="input_01" placeholder="Nombre" @bind="p.Nombre" /><br>  
<input placeholder="Apellido" @bind="p.Apellido" /><br>  
<input type="number" placeholder="Edad" @bind="p.Edad" /><br>  
<button @onclick="Agregar">Agregar</button>
```

```
@code {  
    List<Persona> lista = Persona.GetLista();  
    ElementReference input_01;  
    Persona p = new Persona();  
    void Agregar()  
    {  
        lista.Add(p);  
        p = new Persona();  
        input_01.FocusAsync();  
    }  
}
```

Con estos cambios, luego de agregar una nueva persona a la lista se establece el foco en el primer input para poder ingresar el nombre de la próxima persona a agregar



Modificar el componente Hola.razor



```

...
@if (!SoloLectura)
{
    <input @ref="input_01" placeholder="Nombre" @bind="p.Nombre" /><br>
    <input placeholder="Apellido" @bind="p.Apellido" /><br>
    <input type="number" placeholder="Edad" @bind="p.Edad" /><br>
    <button @onclick="Agregar">Agregar</button>
}

```

Mostramos estos
elementos sólo si la
propiedad
SoloLectura es false

```

@code {
    [Parameter]
    public bool SoloLectura { get; set; } = false;
    List<Persona> lista = Persona.GetLista();
    ElementReference input_01;
    Persona p = new Persona();
    void Agregar()
    {
        lista.Add(p);
        p = new Persona();
        input_01.FocusAsync();
    }
}

```

El atributo **[Parameter]**
aplicado a la propiedad
SoloLectura, hace posible
establecer su valor en el
momento de utilizar la
etiqueta **<Hola>**

Copiar el código del archivo
12_RecursosParaLaTeoria.txt



Modificar el componente Index.razor y ejecutar



```
@page "/"  
<Hola SoloLectura="true"/>  
<Hola />
```

HolaBlazor

Home

Counter

Fetch data

About

Listado de personas

- Perez, Pablo (34)
- Garcia, Laura (30)
- Lopez, José (45)
- Colombo, Ana (21)
- Suarez, María (15)

Listado de personas

- Perez, Pablo (34)
- García, Laura (30)
- Lopez, José (45)
- Colombo, Ana (21)
- Suarez, María (15)

Nombre

Apellido

Edad

Agregar

<Hola SoloLectura="true"/>

<Hola />

Fin de Teoría 12