

Unidad 5 – Capítulo 1 – Laboratorio 1

Creando una aplicación Web con Blazor

Objetivos

Crear una aplicación web simple con Blazor.
Familiarizarse con la tecnología Web Assembly en .NET.

Duración Aproximada

60 minutos

Consigna Principal

Realizar el ABMC / CRUD de una entidad o concepto del dominio y exponer cada una de estas operaciones a través de una aplicación Web con Blazor. La entidad sobre la que se trabaje deberá persistir en una base de datos relacional, para lo cual podemos apoyarnos en EF Core.

En los pasos que se describen a continuación, se adopta como ejemplo la entidad Alumno de un hipotético caso de negocio en el contexto de una Universidad, pero el ejemplo es fácilmente trasladable a cualquier otro escenario de negocio.

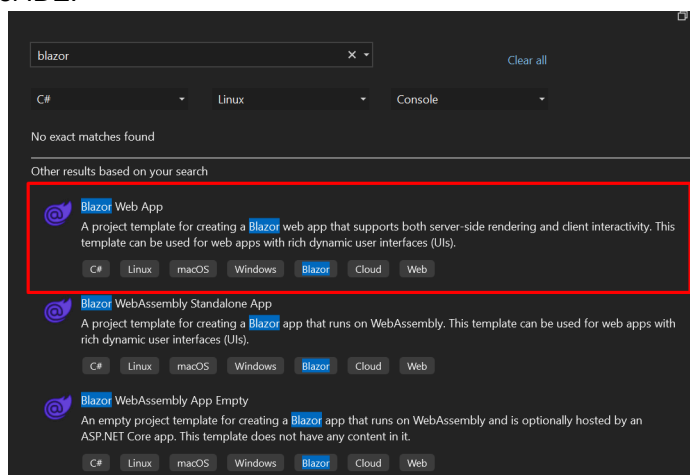
Comentarios Previos

Para la creación de la capa de datos, tener presente los conceptos adquiridos al realizar la práctica: Unidad 4 - Capítulo 2 - Lab 1 Entity Framework. En los siguientes pasos nos enfocaremos puntualmente a la parte de configuración de Blazor.

1) Crear nuestra primera aplicación Blazor

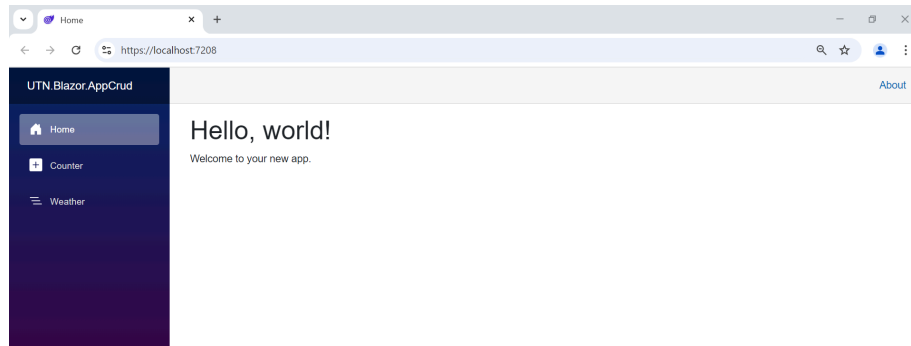
Pasos sugeridos

- 1) Utilizando Visual Studio, crear un proyecto de tipo Blazor, como se muestra en la imagen a continuación. En los siguientes pasos, le damos un nombre y mantenemos los valores por defecto que nos propone el IDE.

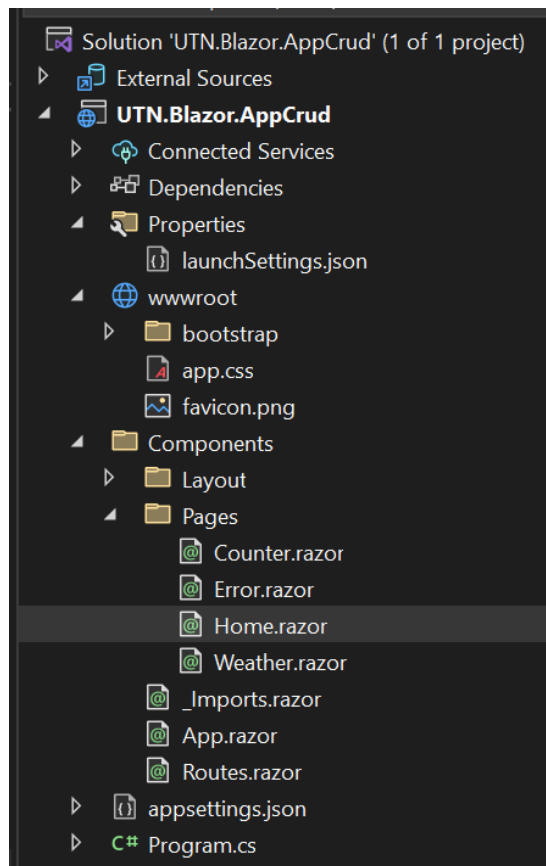


NOTA: En caso de no tener el template Blazor Web App disponible utilizar el Blazor Server App. Ver anexo para más detalle.

- 2) Una vez creada la solución, el template básico que nos propone el IDE ya nos debería permitir correr nuestra primera aplicación Blazor ejecutando con F5:



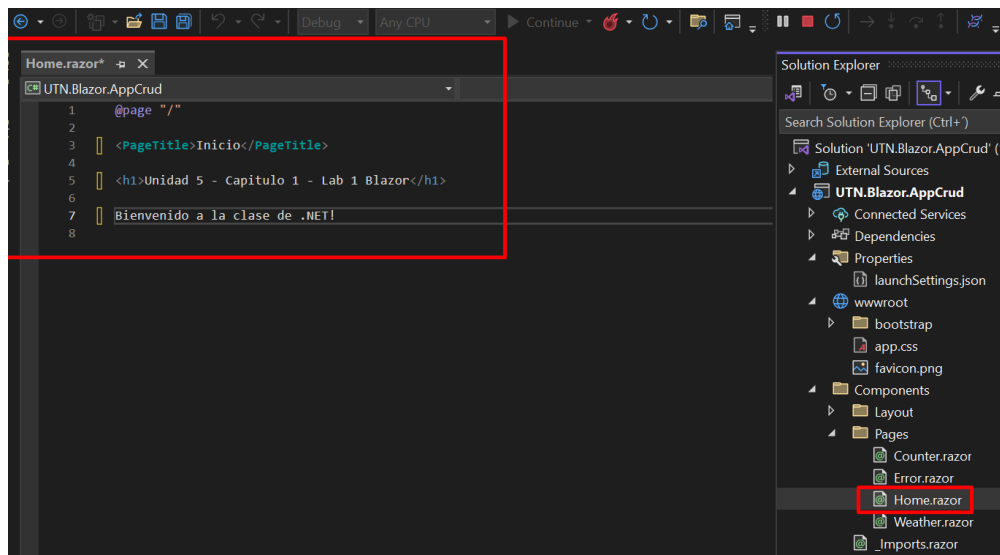
- 3) En base a los contenidos datos en teoría, analizar cada uno de los archivos creados por el template base del IDE:



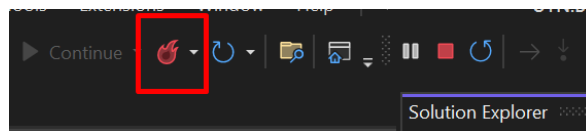
- 4) Analizar el comportamiento de las 3 opciones de menú que tiene nuestra primera aplicación “Counter”; “Home”; “Weather” y relacionar su comportamiento con las clases que vemos en la solución.

2) Crear nuestro primer contenido estático web

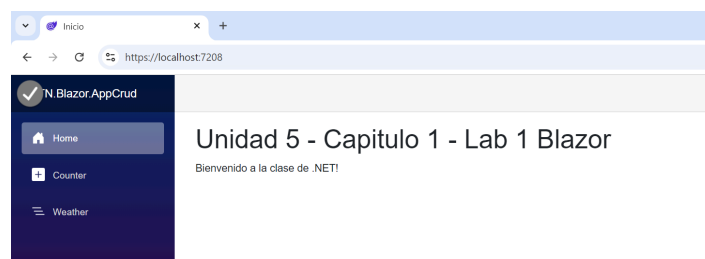
- 1) Vamos a ubicarnos en la página Home y modificamos el contenido del mismo:



Notar que si hacemos modificaciones “en caliente” mientras nuestra aplicación está ejecutándose no vamos a ver los cambios salvo que nos apoyemos en la funcionalidad del “Hot Reload” del IDE:

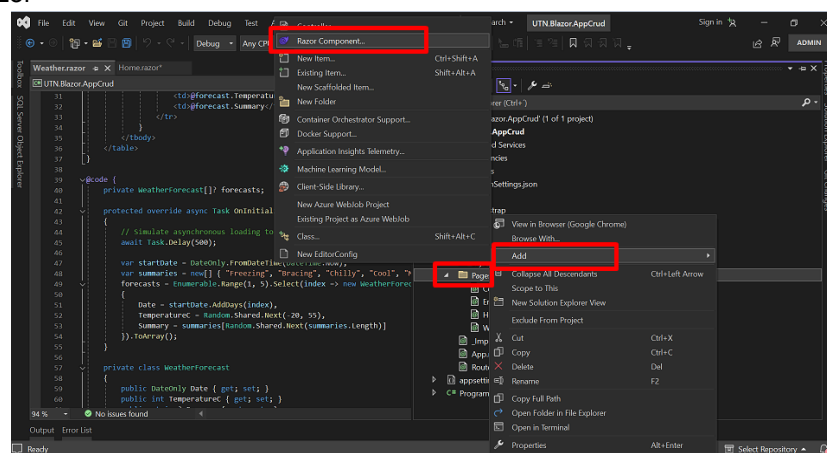


El resultado observado debería ser:

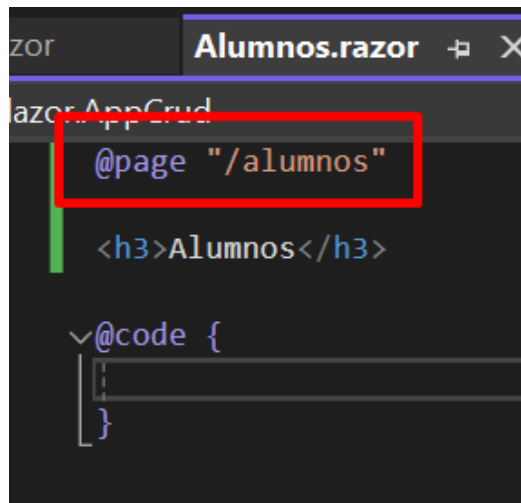


3) Crear contenido dinámico en una aplicación Blazor

- 1) En base a la página: Weather.razor vamos a crear una página que liste a los alumnos.
- 2) Sobre la carpeta pages hacemos botón derecho y creamos un componente Razor con nombre Alumnos.razor



- 3) El primer paso es identificar a qué url va a reaccionar nuestra página para lo cual definimos lo siguiente:

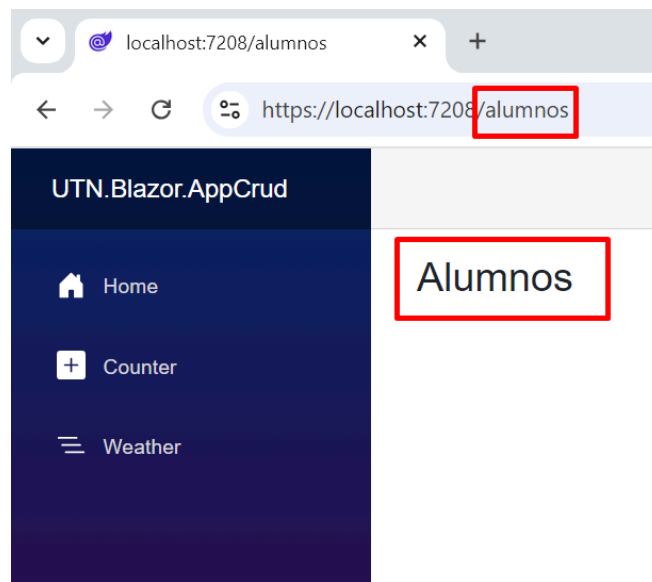


```
@page "/alumnos"

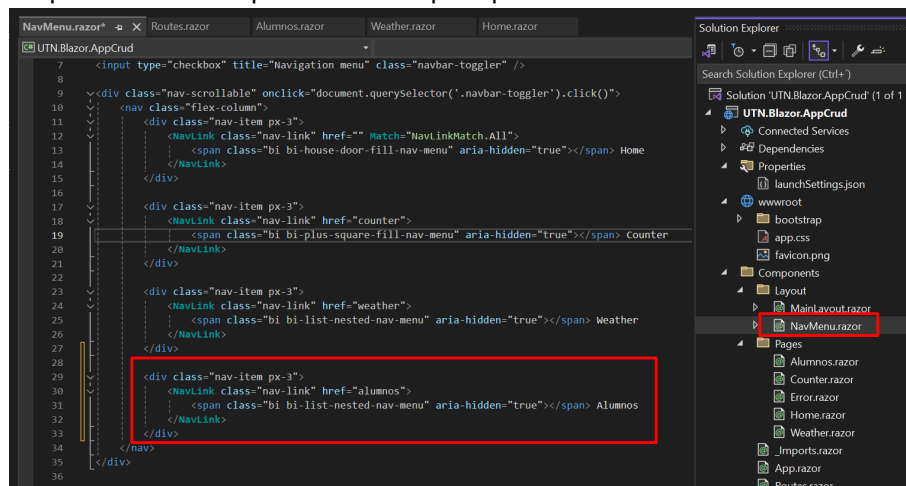
<h3>Alumnos</h3>

@code {
}
```

Notar que más allá de que es una buena práctica que exista una correspondencia entre el nombre del componente razor y la url, no es un requisito para el funcionamiento.



- 4) A continuación vamos a agregar nuestra nueva página a las opciones del menú lateral (left side menu) para que los usuarios que accedan sepan que existe:



Notar que esta parte de la funcionalidad de nuestro sitio está dentro de la carpeta Layout, esto es debido a que es compartida por todas las páginas. Adicionalmente podemos ver que el archivo principal es MainLayout y este mediante un identificador invoca al componente que renderiza ("dibuja") el menú:

```
@inherits LayoutComponentBase

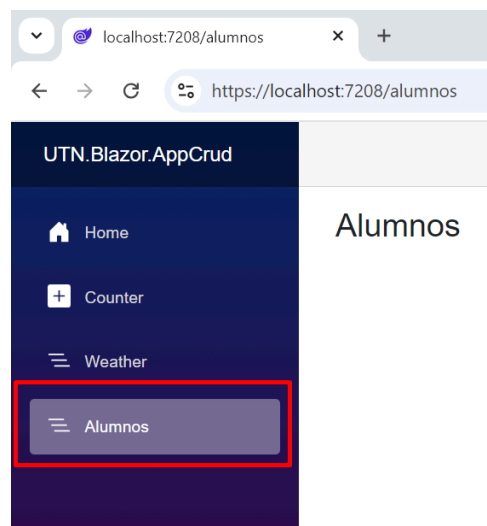
<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <a href="https://learn.microsoft.com/aspnet/core/" target="_blank">About</a>
        </div>

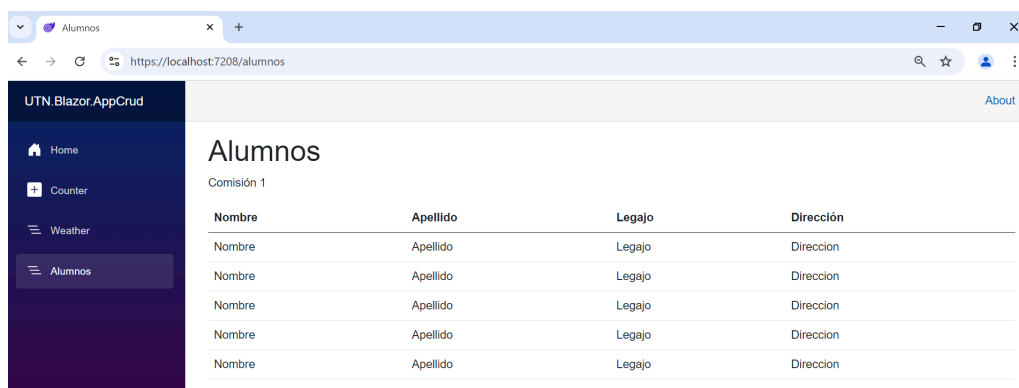
        <article class="content px-4">
            @Body
        </article>
    </main>
</div>

<div id="blazor-error-ui">
    An unhandled error has occurred.
    <a href="" class="reload">Reload</a>
    <a class="dismiss">✕</a>
</div>
```

Si todo quedo correctamente deberíamos ver lo siguiente:



5) Analice detenidamente el componente weather.razor para lograr un comportamiento similar pero listado alumnos como se muestra a continuación:



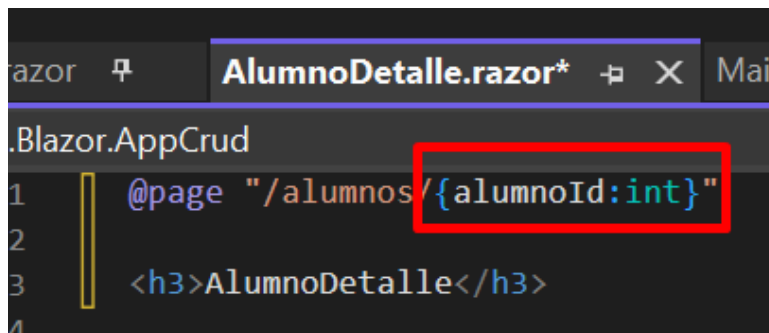
Nombre	Apellido	Legajo	Dirección
Nombre	Apellido	Legajo	Dirección
Nombre	Apellido	Legajo	Dirección
Nombre	Apellido	Legajo	Dirección
Nombre	Apellido	Legajo	Dirección
Nombre	Apellido	Legajo	Dirección

Para el ejemplo planteamos la siguiente entidad:

```
private class Alumno
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public string Legajo { get; set; }
    public string Direccion { get; set; }
}
```

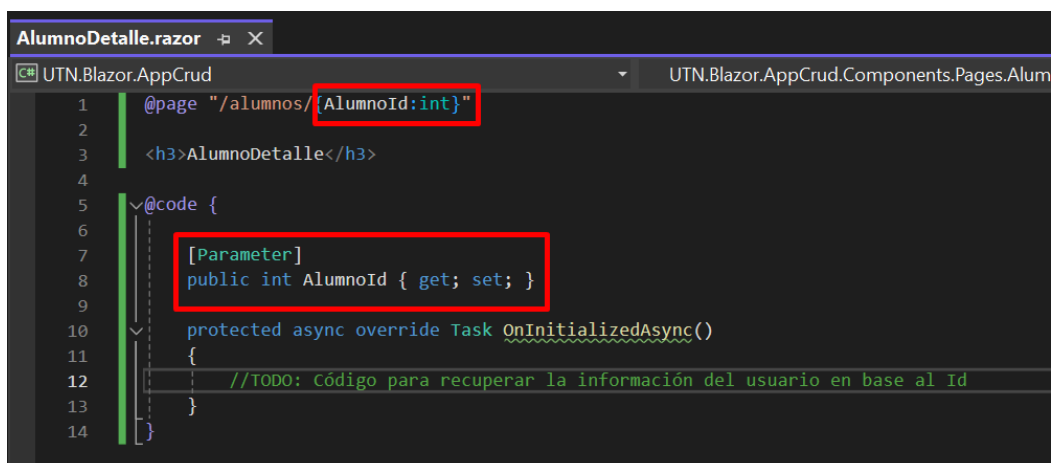
4) Crear la página de detalle

- 1) De la misma forma que hicimos con el componente de razón de Alumnos, vamos a generar uno que sea AlumnoDetalle.razor
- 2) Notar que ahora vamos a tener una página de detalle por cada alumno existente, por lo que para lograr un comportamiento que sea dinámico en la URL tenemos que poder de alguna forma identificar qué alumno en si queremos cargar. Esto lo logramos de la siguiente forma:



```
AlumnoDetalle.razor*
UTN.Blazor.AppCrud
1 @page "/alumnos/{alumnoId:int}"
2
3 <h3>AlumnoDetalle</h3>
4
```

y lo podemos acceder de la siguiente manera:



```
AlumnoDetalle.razor
UTN.Blazor.AppCrud
1 @page "/alumnos/{AlumnoId:int}"
2
3 <h3>AlumnoDetalle</h3>
4
5 @code {
6
7     [Parameter]
8     public int AlumnoId { get; set; }
9
10    protected async override Task OnInitializedAsync()
11    {
12        //TODO: Código para recuperar la información del usuario en base al Id
13    }
14 }
```

5) Armar un formulario de edición

- 1) Ayudándonos en los componentes de Blazor: EditForm y InputText armar un formulario HTML que utilice la capa de datos realizada en la práctica anterior de EF Core para persistir los datos del alumno.
- 2) Utilizando DataAnnotationsValidator / ValidationMessage validar que todos los campos sean de ingreso obligatorio.

ANEXO

Diferencias entre Templates

<https://stackoverflow.com/questions/77690847/what-is-the-difference-between-blazor-web-appnew-in-net-8-and-blazor-server-a>



When you continue with the Server App you will see it can only be created with dotnet 7 and below, the Web App only with dotnet 8.

13



They are templates from 2 different generations. The new Web App can deliver everything the older ServerApp does and it adds a few more options.



See [New Blazor Web App template](#)



Share Improve this answer Follow

edited Dec 20, 2023 at 23:37

answered Dec 20, 2023 at 12:33



Henk Holterman

271k ● 31 ● 347 ● 530

Name	Description	Render location	Interactive
Static Server	Static server-side rendering (static SSR)	Server	✗
Interactive Server	Interactive server-side rendering (interactive SSR) using Blazor Server	Server	✓
Interactive WebAssembly	Client-side rendering (CSR) using Blazor WebAssembly+	Client	✓
Interactive Auto	Interactive SSR using Blazor Server initially and then CSR on subsequent visits after the Blazor bundle is downloaded	Server, then client	✓