Blazor WebAssembly

SPA

Single Page Application

Se trata de una aplicación de una sola página, al igual que React o Angular.

WebAssembly es como un hermano de JavaScript. Son las únicas 2 alternativas que se ejecutan del lado del cliente.

Es el ensamblador para la web.

C# genera código y compila para WebAssembly.

También con Blazor se puede ejecutar desde el lado del servidor.

No se pueden realizar páginas web progresivas (eso solo con javascript), nativas, que se descargan y ejecutan enteramente en la máquina.

La desventaja es que debe descargar al navegador, y al inicio tarda mas que una app server.

Es de hace 2 años. Usa .Net 5

Autenticación

***OAuth 2.0*** y ***OpenID Connect***

OpenIdConnect como protocolo de autenticación

OOut como protocolo de autorización.

Dice que ambos son la estrategia de autenticación recomendada en aplicaciones SPA. Podemos ya olvidarnos de las cookies.

Http se considera insegura. Hoy día es todo Https

Elegí para esto la autenticación de Microsoft.

Lo único que ve el usuario es la página index.html

Si tenés un index.razor se le puede meter code behind, creando una clase index.razor.cs. Al igual que hice ya con los css.

Los componentes se pueden usar como páginas o como componentes de otras páginas.

@page "/"

<**PageTitle**>Index</**PageTitle**>

<h1>Hello, world!</h1>

<**Counter**></**Counter**>

Welcome to your new app.

<**SurveyPrompt** **Title**="How is Blazor working for you?" />

Por ejemplo, ahí agregué el componente counter, que era una página, desde la página principal y se dispone en pantalla.

Clase 2

Ese tiempo extra que lleva cargar una pagina webAssembly aconseja camuflarlo con alguna pantalla de bienvenida como para que el usuario sienta que entró, cuando la página en realidad se sigue cargando.

Desde WebAssembly (blazor, react, o angular) no se puede hacer una conexión directa a BBDD. Hay que utilizar una API.

Comienza con el proyecto de la pizzería.

Poné que le hospede la webAssembly en una web Api. Crea 2 proyectos. Ahí pondrá la conexión a BBDD. Será PWA pero no tilda esa opción para hacerlo manualmente.

En Shared están las entidades.

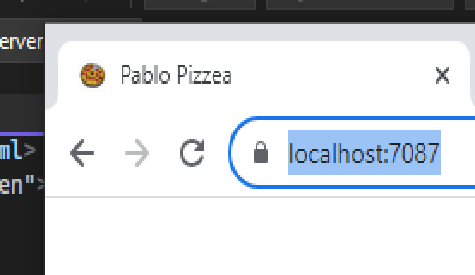
MainLayout es el equivalente del masterPage de webForms. Es lo común que se verá en todas las páginas.

Copia una carpeta Font, para importar estilos y lo pone en root/css

38:52 / 2:48:26

Pone eso en el index.html. Es el enlace a la imagen que será el ícono de la app.

<link rel="icon" href="Images/pizza-de-pepperoni-comida-pan-y-pasta-12622863.jpg" />



Recomienda borrar esa línea. Para elminar “estilos aislados”.

<link href="PabloPizzea.Client.styles.css" rel="stylesheet" />

Eso de ahí es lo que se muestra al cargarse la página, podría modificarse.

<div id="app">

<svg class="loading-progress">

<circle r="40%" cx="50%" cy="50%" />

<circle r="40%" cx="50%" cy="50%" />

</svg>

<div class="loading-progress-text"></div>

</div>

Contexto. Es la representación de la base de datos.

Hace code-first.

public class PizzaStoreContext:DbContext

{

public DbSet<PizzaSpecial> Specials { get; set; }

public PizzaStoreContext(DbContextOptions options):base(options) {}

}

Con eso hace referencia a que creará una tabla llamada Specials

En Models, simula el ingreso de pizzas al sistema a través de un array, eso lo hace en la clase SeedData. Luego, lo agrega al contexto (o sea, a la BBDD):

var Specials = new PizzaSpecial[]

{

new PizzaSpecial()

{

Name="Pizza Napolitana",

Description="Tomate, aho y albahaca",

BasePrice=1850,

ImageUrl="/images/Pizzas/napolitana.png"

},

ETC, ETC, ETC

};

context.Specials.AddRange(Specials);

context.SaveChanges();