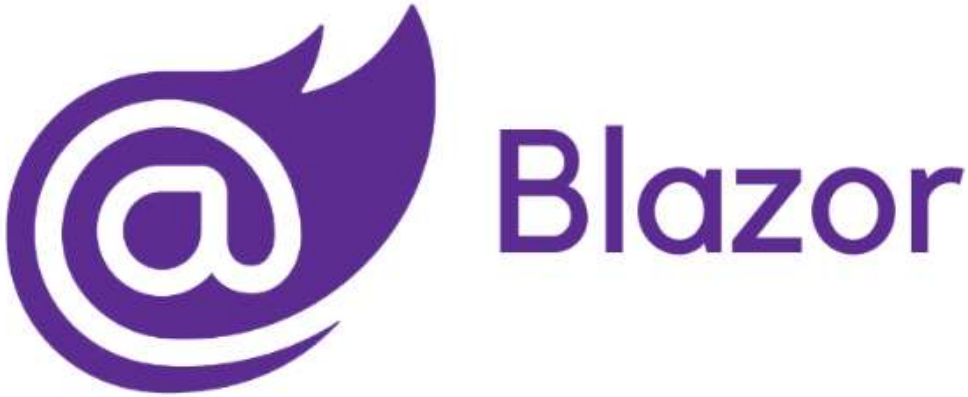
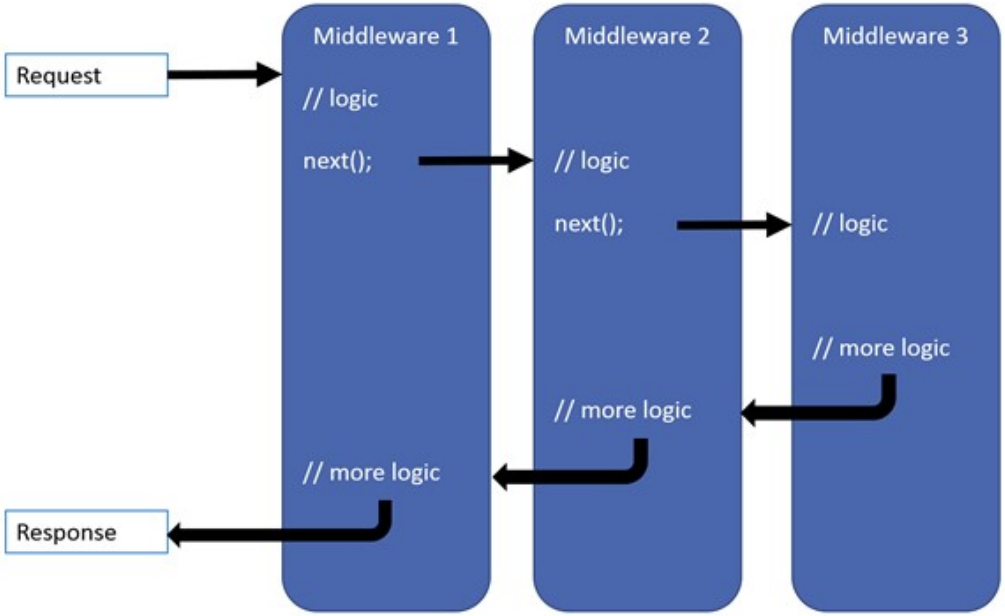


Identity (Autenticación y Autorización)

Program.cs (middelware)

¿Qué es un Middleware?

Un **middleware** es un componente de software que se ubica en la canalización de solicitudes HTTP. Cada solicitud que llega al servidor pasa a través de una serie de middlewares, los cuales pueden realizar operaciones antes y/o después de que se procesen las solicitudes y respuestas. El middleware puede ser encargado de realizar funciones como autenticación, manejo de errores, registro de logs, compresión de respuesta, entre otros.



https://learn.microsoft.com/es-es/aspnet/core/fundamentals/middleware/?view=aspnetcore-8.0

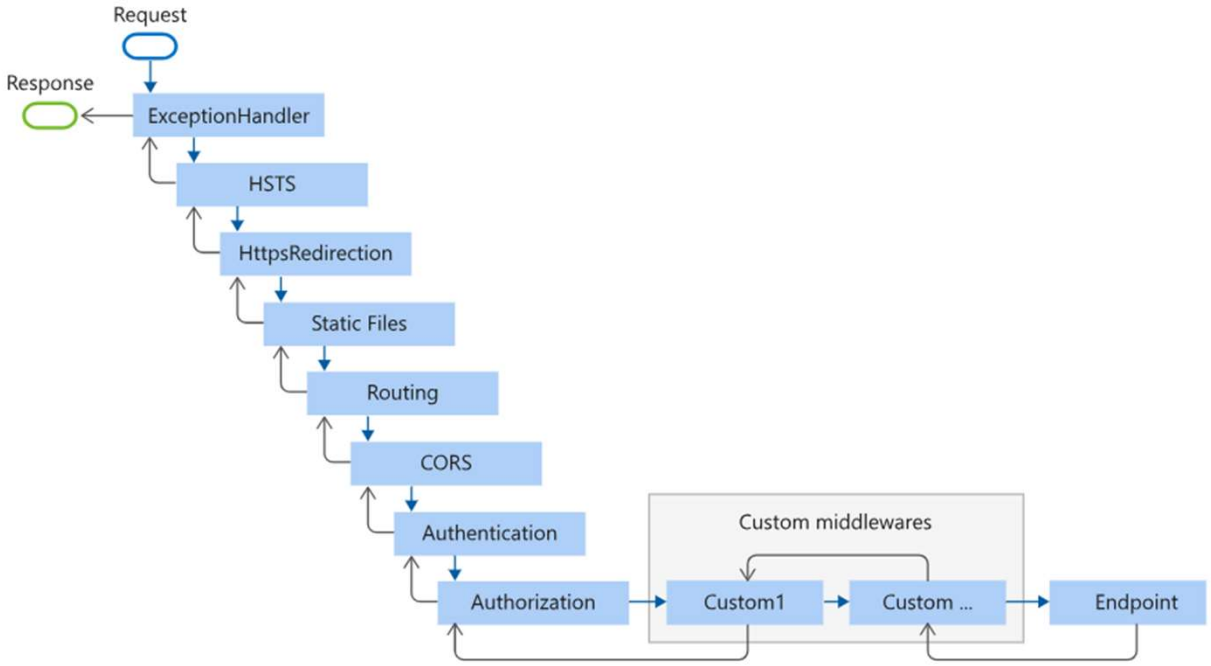
Program.cs (middelware)

¿Cómo funciona en una aplicación Blazor Server?

- En una aplicación **Blazor Server**, el middleware es responsable de manejar las solicitudes HTTP desde los clientes.

- El ciclo de vida de la solicitud se gestiona a través de una pila de middleware en el servidor.

- Cuando un cliente realiza una solicitud, esta pasa a través de los diferentes middlewares que se han registrado en la aplicación antes de que se llegue a la lógica principal, y la respuesta sigue el mismo camino de vuelta.



https://learn.microsoft.com/es-es/aspnet/core/fundamentals/middleware/?view=aspnetcore-8.0

Program.cs (middelware): añadir los servicios a la aplicacíon

```
using LeccionAuthentication.Components;
using LeccionAuthentication.Components.Account;
using LeccionAuthentication.Data;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
```

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

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

builder.Services.AddCascadingAuthenticationState();
builder.Services.AddScoped<IdentityUserAccessor>();
builder.Services.AddScoped<IdentityRedirectManager>();
builder.Services.AddScoped<AuthenticationStateProvider, IdentityRevalidatingAuthenticationStateProvider>();

builder.Services.AddAuthentication(options =>
    {
        options.DefaultScheme = IdentityConstants.ApplicationScheme;
        options.DefaultSignInScheme = IdentityConstants.ExternalScheme;
    })
    .AddIdentityCookies();

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddIdentityCore<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddSignInManager()
    .AddDefaultTokenProviders();

builder.Services.AddSingleton<IEmailSender<ApplicationUser>, IdentityNoOpEmailSender>();
```

Explicación del Program.cs (middelware)

1. Inicialización de la aplicación

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

Esta línea inicializa un **WebApplicationBuilder**, que se encarga de configurar los servicios, el middleware y construir la instancia final de la aplicación

2. Registro de Servicios.

2.1. Componentes Razor e Interacción del servidor

codebuilder.Services.**AddRazorComponents**()
.**AddInteractiveServerComponents**();

AddRazorComponents: Registra el soporte para componentes Razor (componentes Blazor que generan la interfaz de usuario)

.**AddInteractiveServerComponents**: Habilita el modo de renderizado interactivo de Blazor Server, permitiendo que los componentes interactúen con el servidor

2.2. Estado de autenticación en cascada

codebuilder.Services.**AddCascadingAuthenticationState**();

Este servicio se encarga de hacer que el estado de autenticación esté disponible como un parámetro en cascada para los componentes Blazor que lo necesiten

Explicación del Program.cs (middelware)

2.3. Servicios personalizados

```
builder.Services.AddScoped<IdentityUserAccessor>();
builder.Services.AddScoped<IdentityRedirectManager>();
builder.Services.AddScoped<AuthenticationStateProvider, IdentityRevalidatingAuthenticationStateProvider>();
```

IdentityUserAccessor y **IdentityRedirectManager** son servicios personalizados (definidos en la aplicación) que gestionan tareas relacionadas con el acceso a la identidad del usuario y la redirección dentro de la aplicación.

AuthenticationStateProvider: IdentityRevalidatingAuthenticationStateProvider es un proveedor personalizado que actualiza periódicamente el estado de autenticación del usuario.

2.4. Configuración de autenticación e identidad

```
builder.Services.AddAuthentication(options =>   {
      options.DefaultScheme = IdentityConstants.ApplicationScheme;
      options.DefaultSignInScheme = IdentityConstants.ExternalScheme;
})
.AddIdentityCookies();
```

Configura los esquemas de autenticación para la aplicación.

IdentityConstants.ApplicationScheme se usa para el inicio de sesión de la aplicación, mientras que **IdentityConstants.ExternalScheme** es para inicios de sesión externos (como Google o Facebook)
AddIdentityCookies: Añade autenticación basada en cookies para Identity.

Explicación del Program.cs (middelware)

2.5. Configuración de la base de datos

var connectionString = builder.Configuration.**GetConnectionString**("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<**ApplicationDbContext**>(options => options.UseSqlServer(connectionString));

builder.Services.**AddDatabaseDeveloperPageExceptionFilter**();

La cadena de conexión se obtiene de la configuración (generalmente del archivo **appsettings.json**), y Entity Framework se configura para usar **SQL Server** como proveedor de base de datos.

ApplicationDbContext: Es una clase personalizada de DbContext que representa la base de datos.

AddDatabaseDeveloperPageExceptionFilter: Muestra páginas de excepción detalladas para errores relacionados con la base de datos durante el desarrollo.

2.6. Configuración de ASP.NET Identity

builder.Services.**AddIdentityCore**<ApplicationUser>(options => options.SignIn.**RequireConfirmedAccount** = true)
.**AddEntityFrameworkStores**<ApplicationDbContext>()
.**AddSignInManager**()
.**AddDefaultTokenProviders**();

AddIdentityCore<ApplicationUser>: Registra ASP.NET Identity con una clase personalizada de usuario ApplicationUser.
RequireConfirmedAccount: Obliga a que la cuenta esté confirmada por correo electrónico para permitir el inicio de sesión.
AddEntityFrameworkStores: Conecta Identity con Entity Framework, utilizando ApplicationDbContext para persistir los datos de los usuarios.
AddDefaultTokenProviders: Proporciona soporte de tokens para restablecer contraseñas, confirmación de correos electrónicos, etc.

Explicación del Program.cs (middelware)

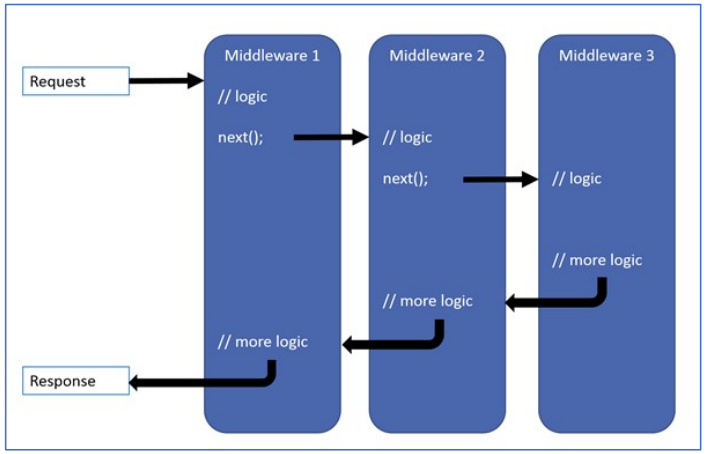
2.6. Servicio de envío de correos electrónicos

builder.Services.AddSingleton<**IEmailSender**<ApplicationUser>, **IdentityNoOpEmailSender**>();

Registra un servicio de envío de correos electrónicos de tipo "No-Op" (No operación), que actúa como un marcador de posición para una funcionalidad de envío de correos que puede implementarse más adelante.

Program.cs (middelware): configurar la canalización (pipeline) de la solicitud HTTP

```
//Esta línea finaliza la configuración y construye la instancia de la aplicación web
var app = builder.Build();

// Si el entorno es el de dessarrollo
if (app.Environment.IsDevelopment())
{
   //Para ejecutar migraciones de base de datos directamente desde el entorno de desarrollo
   app.UseMigrationsEndPoint();
}
else // Si el entorno es producción u otro diferente a desarrollo
{
    app.UseExceptionHandler("/Error", createScopeForErrors: true);

// The default HSTS (HTTP Strict Transport Security Protocol) value is 30 days.
//You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection(); //Redirección a HTTPS
app.UseStaticFiles(); //Entrega de archivos estáticos
app.UseAntiforgery(); //
app.MapRazorComponents<App>() //Configuración de componentes Razor interactivos
    .AddInteractiveServerRenderMode();

// Habilitación de endpoints para el sistema de autenticación (Identity)
app.MapAdditionalIdentityEndpoints();

app.Run();
```



Las configuraciones incluyen la **gestión de errores**, la redirección a **HTTPS**, la entrega de **archivos estáticos**, la protección contra **CSRF**, la configuración de **componentes Razor** interactivos y la habilitación de **endpoints** para el sistema de **autenticación** (Identity).