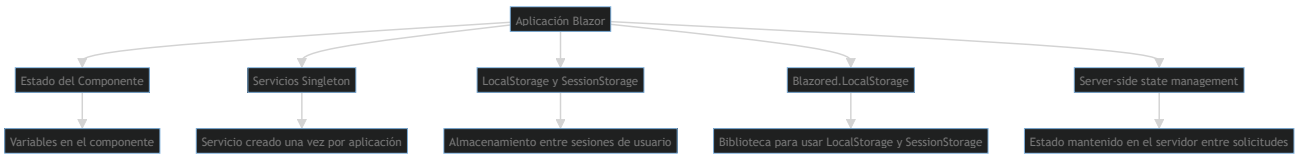
Login y Manejo de estado en Blazor

Manejo de estado en Blazor

El manejo de estado en Blazor se refiere a la forma en que tu aplicación maneja y mantiene la información del usuario y de la aplicación entre las solicitudes del usuario. Esto es especialmente importante en las aplicaciones de una sola página (SPA), como las aplicaciones Blazor, donde la aplicación se ejecuta en el navegador del cliente.

En Blazor, hay varias formas de manejar el estado:

1. **Estado del componente**: El estado del componente se mantiene dentro del componente y se pierde cuando el componente se destruye. Este es el tipo de estado más simple y se puede manejar simplemente con variables en el componente.
2. **Servicios Singleton**: Los servicios Singleton son servicios que se crean una vez por aplicación y se pueden usar para mantener el estado que se comparte entre varios componentes.
3. **LocalStorage y SessionStorage**: Estos son mecanismos proporcionados por el navegador que permiten almacenar el estado entre sesiones de usuario. LocalStorage mantiene la información incluso después de cerrar y abrir el navegador, mientras que SessionStorage mantiene la información solo mientras el navegador está abierto.
4. **Blazored.LocalStorage**: Es una biblioteca que proporciona una forma fácil de usar LocalStorage y SessionStorage en Blazor.
5. **Server-side state management**: En las aplicaciones Blazor Server, el estado se puede mantener en el servidor entre las solicitudes del usuario. Esto se puede hacer usando SignalR para mantener una conexión en tiempo real entre el cliente y el servidor.



Local Storage, Browser Storage y Memory Storage in Blazor

Claro, puedo proporcionar más detalles sobre estos tres tipos de almacenamiento en Blazor.

1. **Local Storage**: Local Storage es una característica del navegador web que permite a las aplicaciones web almacenar datos en el lado del cliente. Los datos almacenados en Local Storage son persistentes, lo que significa que los datos se mantienen incluso después de cerrar y abrir el navegador. En Blazor, puedes usar Local Storage para almacenar datos que necesitas mantener entre las sesiones del usuario, como la información de autenticación del usuario. Para usar Local Storage en Blazor, puedes usar la API de JavaScript Interop para interactuar con Local Storage directamente, o puedes usar una biblioteca como Blazored.LocalStorage que proporciona una interfaz .NET para Local Storage.
2. **Browser Storage (Session Storage)**: Session Storage es similar a Local Storage en que permite a las aplicaciones web almacenar datos en el lado del cliente. Sin embargo, a diferencia de Local Storage, los datos almacenados en Session Storage se pierden cuando se cierra la sesión del navegador. Esto lo hace útil para almacenar datos que solo necesitas mantener mientras el usuario está navegando en tu aplicación, como el estado de la interfaz de usuario. Al igual que con Local Storage, puedes usar la API de JavaScript Interop para interactuar con Session Storage directamente, o puedes usar una biblioteca como Blazored.SessionStorage.
3. **Memory Storage**: En Blazor, el almacenamiento en memoria se refiere a los datos que se almacenan en la memoria del servidor o del cliente mientras la aplicación está en ejecución. Los datos almacenados en la memoria se pierden cuando se cierra la aplicación o cuando se recicla el proceso del servidor. En Blazor Server, puedes usar el almacenamiento en memoria para mantener los datos entre las solicitudes del usuario mientras mantienes una conexión en tiempo real entre el cliente y el servidor con SignalR. En Blazor WebAssembly, el almacenamiento en memoria se limita a la vida útil de la aplicación en el navegador del cliente.

Es importante tener en cuenta que cada uno de estos métodos de almacenamiento tiene sus propias limitaciones y consideraciones de seguridad. Por ejemplo, tanto Local Storage como Session Storage están limitados en tamaño (generalmente a unos pocos MB), y los datos almacenados en ellos son accesibles para cualquier código JavaScript que se ejecute en la misma página, lo que puede ser un problema de seguridad si estás almacenando datos sensibles. Por otro lado, el almacenamiento en memoria puede ser más seguro ya que los datos se mantienen en el servidor, pero también puede ser más costoso en términos de uso de la memoria.

localstorage para guardar el token de acceso

LocalStorage es conveniente en este caso de uso por varias razones:

1. **Persistencia**: LocalStorage permite almacenar datos de manera persistente en el navegador del cliente. Esto significa que los datos, como el token de acceso y la información del usuario, se mantendrán incluso después de cerrar y abrir el navegador. Esto es útil para mantener al usuario autenticado entre sesiones.
2. **Facilidad de uso**: Con la biblioteca Blazored.LocalStorage, es fácil interactuar con LocalStorage desde Blazor. Puedes almacenar y recuperar datos con simples llamadas a métodos.

3. **Almacenamiento en el lado del cliente**: Al almacenar los datos en el navegador del cliente, reduces la carga en tu servidor y evitas tener que transmitir estos datos entre el cliente y el servidor.
4. **Compatibilidad**: LocalStorage es compatible con todos los navegadores modernos, lo que significa que tu aplicación funcionará para la mayoría de los usuarios.

Sin embargo, es importante tener en cuenta que LocalStorage tiene sus limitaciones y consideraciones de seguridad. Los datos almacenados en LocalStorage son accesibles para cualquier código JavaScript que se ejecute en la misma página, lo que puede ser un problema de seguridad si estás almacenando datos sensibles. Además, LocalStorage está limitado en tamaño (generalmente a unos pocos MB), por lo que no es adecuado para almacenar grandes cantidades de datos.

Uso de Local Storage, Session Storage y Memory Storage en Blazor

1. **Local Storage**: Para usar Local Storage en Blazor, puedes usar la biblioteca Blazored.LocalStorage. Primero, necesitas instalarla en tu proyecto con el siguiente comando:

```
dotnet add package Blazored.LocalStorage
```

Luego, puedes usarla en tu componente de la siguiente manera:

```
@page "/localstorage"
@inject Blazored.LocalStorage.ILocalStorageService localStorage

<button @onclick="SaveData">Save Data</button>

@code {
    private async Task SaveData()
    {
        await localStorage.SetItemAsync("key", "value");
    }
}
```

2. **Session Storage**: Para usar Session Storage en Blazor, puedes usar la biblioteca Blazored.SessionStorage. Primero, necesitas instalarla en tu proyecto con el siguiente comando:

```
dotnet add package Blazored.SessionStorage
```

Luego, puedes usarla en tu componente de la siguiente manera:

```
@page "/sessionstorage"
@inject Blazored.SessionStorage.ISessionStorageService sessionStorage
```

```
<button @onclick="SaveData">Save Data</button>

@code {
    private async Task SaveData()
    {
        await sessionStorage.SetItemAsync("key", "value");
    }
}
```

3. **Memory Storage**: Para el almacenamiento en memoria, puedes usar una variable en tu componente que mantendrá los datos mientras el componente esté en ejecución. Aquí hay un ejemplo:

```
@page "/memorystorage"

<button @onclick="SaveData">Save Data</button>

@code {
    private string data;

    private void SaveData()
    {
        data = "value";
    }
}
```

Para agregar estos componentes a tu aplicación Blazor, simplemente necesitas crear los archivos `.razor` correspondientes en la carpeta `Pages` de tu proyecto y pegar el código correspondiente. Recuerda que también necesitas registrar los servicios de Blazored.LocalStorage y Blazored.SessionStorage en tu método `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddBlazoredLocalStorage();
    services.AddBlazoredSessionStorage();
}
```

Por último, asegúrate de tener las siguientes directivas `@using` en tu archivo `_Imports.razor`:

```
@using Blazored.LocalStorage
@using Blazored.SessionStorage
```

Estos ejemplos son muy básicos y solo almacenan una cadena estática. En una aplicación real, probablemente querrías almacenar datos más complejos y manejar errores y casos límite.

Ejemplo de uso de Local Storage en una aplicación Blazor para el login

Para loguearte y almacenar el token en LocalStorage, necesitarás utilizar la biblioteca Blazored.LocalStorage. Asegúrate de haberla instalado en tu proyecto con el comando `dotnet add package Blazored.LocalStorage`.

Primero, inyecta el servicio `ILocalStorageService` en tu componente `Login.razor`:

```
@inject Blazored.LocalStorage.ILocalStorageService localStorage
```

Luego, después de obtener el token de acceso en tu método `HandleValidSubmit`, puedes almacenarlo en LocalStorage de la siguiente manera:

```
await localStorage.SetItemAsync("accessToken", token);
```

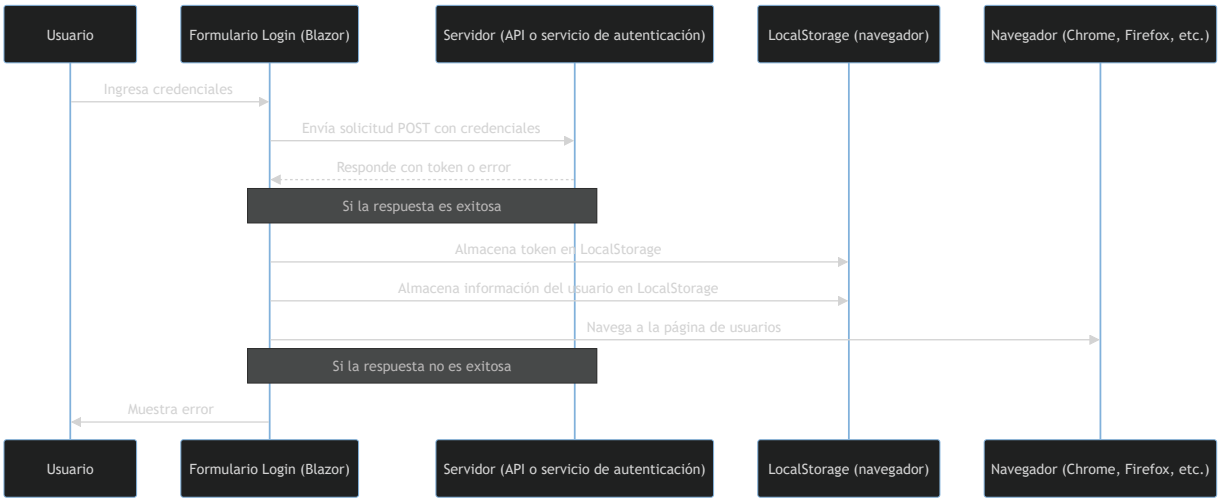
Para almacenar la información del usuario logeado, puedes crear un objeto con la información que deseas almacenar y luego convertirlo a JSON para almacenarlo en LocalStorage. Por ejemplo, si tienes un objeto `User` con las propiedades `Email` y `Name`, puedes hacer lo siguiente:

```
var user = new { Email = loginForm.Email, Name = "User Name" };
var userJson = JsonSerializer.Serialize(user);
await localStorage.SetItemAsync("user", userJson);
```

Aquí está el código completo de cómo podrías hacerlo en tu método `HandleValidSubmit`:

```
private async Task HandleValidSubmit()
{
    var loginRequest = new { Email = loginForm.Email, Password = loginForm.Password };
    var data = new StringContent(JsonSerializer.Serialize(loginRequest), Encoding.UTF8, "applicat

    var response = await Http.PostAsync("/auth/login", data);

    if (response.IsSuccessStatusCode)
    {
        var responseContent = await response.Content.ReadAsStringAsync();
        var responseBody = JsonSerializer.Deserialize<Dictionary<string, string>>(responseContent

        if (responseBody.TryGetValue("AccessToken", out string token))
        {
            jwtToken = new JwtSecurityToken(token);
```

```
            // Almacenar el token en LocalStorage
            await localStorage.SetItemAsync("accessToken", token);

            // Almacenar la información del usuario en LocalStorage
            var user = new { Email = loginForm.Email, Name = "User Name" };
            var userJson = JsonSerializer.Serialize(user);
            await localStorage.SetItemAsync("user", userJson);

            // Navegar a la página de usuarios
            NavigationManager.NavigateTo("/users");
        }
        else
        {
            // Mostrar algún error en caso de no poder obtener el token
        }
    }
    else
    {
        // Mostrar un error al usuario en caso de que las credenciales sean incorrectas, etc.
    }
}
```



Recuerda que este es un ejemplo básico y que deberías manejar los errores y los casos límite adecuadamente en tu aplicación. Además, asegúrate de que la información del usuario que estás almacenando en LocalStorage no es sensible, ya que LocalStorage es accesible para cualquier código JavaScript que se ejecute en la misma página.