آموزش منطق view در فریمورک Blazor

به طور پیش فرض، قالب های blazor framework، با استفاده از @functions ،کد منطق منظر را درون قالب razor تولید می کنند.

- در پشت صحنه، فریمورک blazor یک کلاس شامل کد C# برای تولید درخت اشیای منظر و یک کد C# برای نمایش منطق منظر ، تولید می کند.
- بیشتر برنامه نویسان تمایل ندارند منظر و منطق را در یک فایل ترکیب کنند.

```
@page "/"

<!-- View (HTML) -->

@functions {
    // Logic (C#)
}
```

روش های بسیاری برای تفکیک منظر (view) و منطق (logic) وجود دارد که در این مقاله مورد بحث قرار خواهند گرفت.

کلاس پایه در فریم ورک Blazor

یک روش برای تفکیک منظر و منطق، ساخت کلاس پایه (Base) است.

- کلاس پایه شامل تمامی منطق منظر (view logic) می باشد (C#).
- مولفه بلیزر از این کلاس مشتق می شود و منظر را اضافه می کند.

نگاهی به این مثال بیاندازیم که منظر و منطق منظر را در یک فایل قرار داده است.

```
@page "/jsinterop"
@using BlazorApplication.Pages
@using Microsoft.JSInterop

<h1>JavaScript Interop Demo</h1>

<hr />

<button class="btn btn-primary" onclick="@CallJSMethod">Call JS Method</button>
<button class="btn btn-primary" onclick="@CallCSMethod">Call C# Method</button>
<br />
<br />
<p id="demop">JavaScript function called from C#</p>
```

```
<br />
<p>@message</p>

@functions {
   protected static string message { get; set; }

   protected void CallCSMethod()
   {
      JSRuntime.Current.InvokeAsync<bool>("CSMethod");
   }

   protected void CallJSMethod()
   {
      JSRuntime.Current.InvokeAsync<bool>("JSMethod");
   }

   [JSInvokable]
   public static void CSCallBackMethod()
   {
      message = "C# function called from  JavaScript ";
   }
}
```

حال، برای تفکیک منظر از منطق منظر، با ایجاد یک کلاس مولفه جدید در فریمورک بلیزر، کد C# را به درون کلاس پایه انتقال دهید، و تمامی کد های C# تعریف شده در @functions را انتقال دهید.

```
using Microsoft.JSInterop;
using Microsoft.AspNetCore.Blazor.Components;

namespace BlazorApplication.Pages
{
   public class JSDemoModel : BlazorComponent
   {
      protected static string message { get; set; }

      protected void CallCSMethod()
      {
         JSRuntime.Current.InvokeAsync<bool>("CSMethod");
      }

      protected void CallJSMethod()
      {
         JSRuntime.Current.InvokeAsync<bool>("JSMethod");
      }

      [JSInvokable]
      public static void CSCallBackMethod()
      {
         message = "C# function called from JavaScript ";
      }
   }
}
```

می توان از بخش @inherits برای ارائه فریم ورک بلیزر به همراه یک تجربه code-behind که نشانگر منظر(view markup) را از کد منطق منظر(view logic code) تفکیک می کند، استفاده کرد.

```
@page "/jsinterop"
@using BlazorApplication.Pages
@using Microsoft.JSInterop
@inherits JSDemoModel

<h1>JavaScript Interop Demo</h1>

<hr />

<button class="btn btn-primary" onclick="@CallJSMethod">Call JS Method</button>
<button class="btn btn-primary" onclick="@CallCSMethod">Call C# Method</button>
<br />
<br />
<p id="demop">JavaScript function called from C#</p>
<br />
<p>@message</p>
```

تزریق وابستگی در فریمورک Blazor

فریمورک بلیزر از تزریق وابستگی (Dependency Injection) پشتیبانی می کند، و DI یکی از روش های تفکیک منطق است که مستقل از منظر می باشد و از جمله آنها می توان به core business logic و data access logic اشاره کرد.

کلاس های جزئی در فریم ورک Blazor

در حال حاظر، فریم ورک بلیزر از کلاس های جزئی (Partial Classes) برای مولفه های خود پشتیبانی نمی کند. بنابراین، نمی توانید یک فایل مجزا بسازید و همان نام کلاس را به عنوان مولفه استفاده کنید.