28/4/25

1- Templated Components

2- Component

3- Parameters

↳ simple

↳ Route

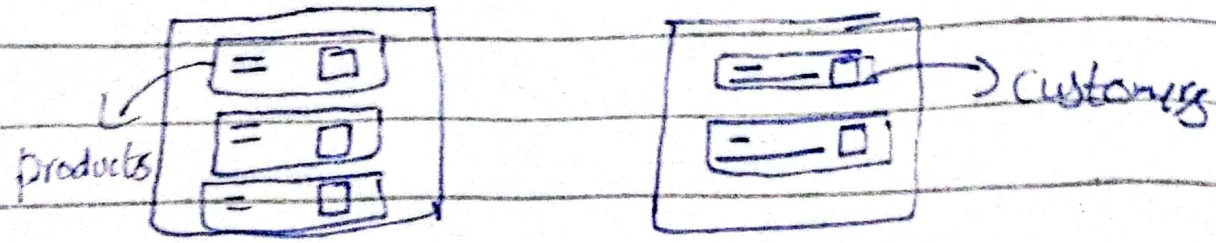
↳ Cascading.

4- Event Call Back

5- Generic Component

6- Generic Templated Component

⊙ Generic Components



[Parameter]
List<Products>

[Parameter]
List<Customers>

(Multiple Components Problem)

⇒ Solution is Generic Components.

: List<TItem> .....

⊙ Implementation:

: @type Param TItem (Generic List. Razor)
(In child component)
(Call it in parent component)

<GenericList TItem = "String" / int > data = Somedata />

List<String> Somedata = { " " };

⊙ Make a Generic Data to represent Customer and Product Data.

⇒ In Parent:

: List <Product>
↳ Name
↳ Price
↳ Color

, List <Customer>
↳ Name
↳ Address

: <GenericTable TItem="Product"
data = Products>
<GenericTable TItem="Customer"
data=Customers>

⊙ In child:

@typeParam TItem

<table>
@foreach(var x in data)
{ <tr>
@foreach (var p in typeof(x).GetProperties)
<td>
p.GetValue(x)

@code { </table></tr></td>
[parameter]
List<TItem> data = new List<TItem>;

}

=> We also wanted to make different view for different components so we use Generic Templated Components.

=> Child

⊙ @type param TItem

```
<div>
@foreach(TItem p in data)
{
    @MySection(p);
}
</div>
```

<tables
foreach(var
I in
data)
@DataTemplate
</tables

```
@code
{
    [Parameter]
    RenderFragment<TItem>
```

:@code
[Parameter]
List<TItem> data =
new...
[Parameter]
RenderFragment
DataTemplate
<TItem>
(get;set;

⊙ Parent:

```
<GenericTable TItem="Product"
    data=Products>
    :DataTemplate
    <MySection data=customers>
        <div>
        <table>
            <tr>
                <td>
                    data.name
                </td>
                <td> data.Price
```

only this changes

5/5/25

(*) Next Monday (50%. Backend of Project)

Data Binding :

→ If Data source changes, then textview changes it is called one way data binding

→ If we change textView, data source changes, then it is called two-way data binding.

: string some data = "_____"

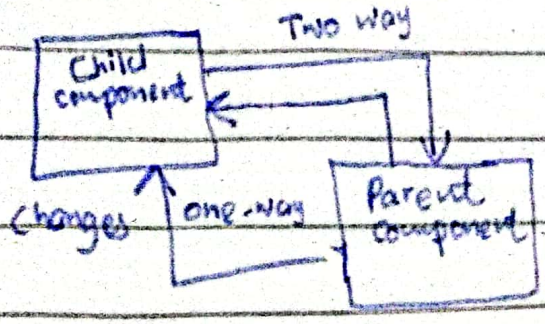
changes ↓ one-way ↓↑ Two-way

TextView TextView

Type :

→ **Simple binding** (one data changes other)

→ **Component data binding** (one compon Chang other)



* @page "/childcomponent"

<button @onclick="ChangeText">Update</button>

@code
{
    string idata = "some data";
    public void ChangeText()
    {
        Console.WriteLine("Change Data");
    }
}

* For One-way:

→ Make component BindingExample

@page "/"

<h3>Binding Example</h3>

<h1>@myData</h1>

<button onclick="changeValue">Change the value</button>

<input type="text" value="@myData" />

@code {
    string myData = "some data";
    void changeValue()
    {
        myData = "This is text changed";
    }
}

(*) For Two-way:

Just change

<input type="text" @bind-value="@myData" />

: Press "Tab" (Focus change event)

⇒ For our own event, we use:

<input type="text" @bind-value "@myData"
@bind-value:event ="oninput"/>

⊙ Short syntax:

@bind ="@myData" @bind:event
="oninput"/>

⊙ Check for CheckBox now:

<h2>@isChecked</h2>

<input type="checkbox" @bind="@isChecked"
@bind:event = "oninput" />

@code {

// Source
string myData = "Some Data"
bool isChecked = true;

:

void ChangeValue()
{ myData = "_____";
isChecked = !isChecked;
}

⊙ For Component Nov.

⊙ For one-way component:

→ Make Parent and child component:

→ Parent component

<h1> @Parent Property </h1>
<ChildComponent /> </ChildComponent>

@code {

public string ParentProperty
{get; set;} =
"Default child"

}

→ Child Component

<h1> @ child Property </h1>

@code

{ public string Child.Property
{get; set;} = "---";

}

→ PC :

<button onClick="ChangeValue"

⊙ CC :

<button @onclick=childvaluechange> ...

</button>

@code

{

Void childvaluechange()

{ childProperty = "change child Text"

}

}

⊙ Perform If we click on child, it changes in parent component.

⊙ PC:

@code

{ public ChildProperty {get; set};

X Not correct

Void onclickCC()

{

childProperty = "Child Data changed"

}

For one way:

⊙ CC:

```
@code
{
    [Parameter]
    public string ChildProperty { get; set; }
}
```

⊙ PC:

```
<ChildComponent ChildProperty="@ParentProperty" />
</ChildComponent>
```

⇒ ⊙ Now For Two way:

⊙ CC:

```
@code
{
    [Parameter]
    public EventCallBack<string> ChildPropertyChanged { get; set; }
    void ChildValueChange()
    {
        ChildProperty = "...";
        ChildPropertyChanged.InvokeAsync(ChildProperty);
    }
}
```

⊙ PC:

```
<ChildComponent @bind-ChildProperty="@ParentProperty" />
</ChildComponent>
```

⊙ ChildProperty named as "myec"

PC:

< ChildComponent @bind-ChildProperty
= "@ParentProperty"
@bind-ChildProperty:event="myec" >
</ChildComponent>

⊙ Cascading Parameter:

CSS (Internal, Inline, external)

→ Effect at multiple places

⊙ PC:

<h3> Parent Component </h3>

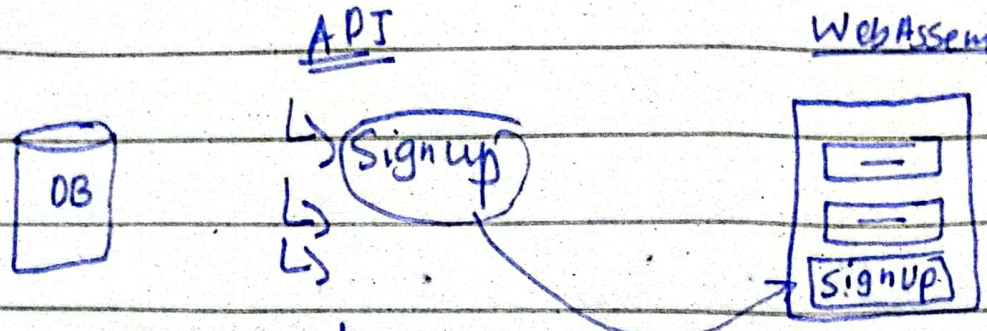
<CascadingValue Name="m"
Value="@message"

⊙ CC:

[CascadingParameter...

12/5/25 **Front End & Back End connection**

* In WebAssembly, our work is on client side.
* In project, we are using two projects API and WebAssembly



* Frontend is connected with API's with backend.

API:

-> Product Model (Name, Price, Id)

Calling Services { -> Product Controller (CRUD)

-> ProductService Class (CRUD) (Repository Pattern) (In Service Folder)
(GetAll, GetById, Add, Update .....)

-> In Program.cs:

builder.Services.AddSingleton<ProductServices>();

(It is separately working)

WebAssembly:

-> Same Product Class (Models Folder)

: It converts data as JSON

→ ProductService class (service class)
→ Using HttpClient in Product services.

: await _http.GetFromJsonAsync<List<Product>>("http...//Link")
(Get, Post, Put, Delete)

⊙ Component Life Cycle Method (override)
: OnInitializedAsync()
(In ProductList Component)

: For Add Product what is used?

⊛ public async Task CreateProduct(Product p)
{
await _http.PostAsJsonAsync
("https:.....", product);
}

:correct:
→ Make Form for html
Use Blazor Form Component (EditForm)

<EditForm Model="currentproduct"
<InputText @bind-value = ...
<InputNumber @ :---

(R) www.mysite.com

security purpose ↓ Allow to access

www.mybackendapi.com

(we need to define that from mysite only access is possible)

→ builder.Services.AddCors(options =>

options.AddPolicy("AllowBlazor client")

:

Use

: app.Cors("AllowBlazor client")

14/5/25

⊛ JS Interoperability :

→ Some library is based on Javascript and if we wanted to call it in Blazor project, then we use Javascript interoperability.

→ we want to call Javascript function in .NET and .NET function into Javascript.

⊙ Today, we wanted to call Javascript function into .NET.

```
function showAlert( )
{
    alert("....")
}
```

⊙ **Implementation :**

→ WebAssembly (BlazorFW)

→ .js folder → Javascript File.

- function showMessage()
  {
  alert("this is a message");
  }

(Now we wanted to call this Js function into counter.)

- counter:

@inject IJSRuntime js

:

<button @onclick="CallJS">Call JS function</...>

@code
{ : async Task
void CallJS()
{
await js.InvokeVoidAsync("showMessage")
}
}

- index.html:

<script src="js/site.js"></script>

: (Now alert window will be shown).

⇒ Now function has some data:

```
function showMessage(message)
{

}
```

⊙ counter:

```
await js.invokeVoidAsync("...", "...",
```

⇒ JS ⇒ (send) .NET

```
function showMessage(message)
{
    conso.Le.writeLine(message);
}
```

```
counter:
@code int data {get; set}
    async Task CallJS()
    {
        int x = await js.InvokeAsync
                    <int>
                    ("ShowMessage");
    }
```

19/5/25

⇒ Book chapter (Javascript Interoperability).

(*) @inject ..... (component)

(*) Two methods:

(i) Void Async (Not return anything)

(ii) Async (Return some parameter)

(@) Javascript function is there which get Window size (height, width) and send it to .NET and show its components.

: [JSInvokable]

: .NET:

```
<h3> Width: @w </h3>
<h3> Height: @h </h3>
```

: Create window size class and receive object there by Invoke async

(+) call in Javascript method:

(i) Static Method

(ii) Instance Method

(#) Restful API's (Representational State Transfer)

(Issue init, for different Categories, there is different Attributes)

⇒ For this issue, we use queries from client side and fetch required data from server. So multiple endpoints issue resolved.

⇒ GraphQL is also used for this technique to overcome multiple endpoints issue.

⊙ Implementation:

⇒ For static Method:

```
[JSInvokable]
public static string GetMessageFromDotNet()
{
    return "Message static ....";
}
```

⊙ In js:

```
function callDotNetStaticMethod()
    var message =
DotNet.invokeMethod("BWA Javascript Example",
                    "GetMessageFromDotNet")
    alert(message);
```

⊙ In mc (main component):

```
<button onclick="callDotNetStaticMethod" />...
```

* [JSInvokable]
public static int ReturnNumberFromDotNet(1)
{
return i=I;
}

→ In js:

function callDotNetStaticNumberMethod(i)
{
var message = DotNet.InvokeMethod("ProjectName", "FunctionName", i)
}

* Delay can also be used for async static Method

→ In C#:

[JSInvokable]
public static async Task<string> GetAsyncMessageFromDotNet()
{
await Task.Delay(1900);
return "Async message from .NET";
}

① OData:

Steps. ① ⇒ Category, Product Class with properties in web api Project.

② : Microsoft.AspNetCore.OData

③ → Product Controller:

```
class ProductsController:
        ODataController
{ private static randomly
        (function) ....
```

④

```
[EnableQuery]
public IActionResult
        Get()
{ return OK(Products);
}
```

⑤ We run this and write filter in URL:

: local .... / products ? filter = category

eg = "Electronics"

: The above query will be sent to api method which will filter data according to query.

(*) Now we wanted to select specific data: (select = Name, Price)

: products? filter = Instock && price > 100
eq = "True"

: products? filter = Instock eq true and price gt 100
& select = Name, category Price

→ OData make our work easier.

SCD

21/5/25

⊙ Instance Method:

→ Make Project

→ Add component myComponent

: mc

<h3> MyComponent </h3>

: static function called without class

@ code
[JSInvokable]
public void showComponentMessage()

: This is instance method

{
Console.WriteLine("Instance method called from JS")
}

Steps

①: Now call JSInvokable

②: We make class object instance and then call it.

@ code

{
private DotNetObjectReference<MyComponent> objectReference;
protected override void OnInitialized()
{
objectReference = DotNetObjectReference.Create(this);

: Add runtime inject.

@page "/mc"
inject IJSRuntime js
<h3> My Component</h3>

<button @onclick="callJSMethod">
Call JS method</button>

@code{
private DotNetobj....
:
:

public void callJSMethod()
{ js.InvokeVoidAsync
("recive Component
Instance",
objectReference);
}

: Folder → js → New item
site.js ← Javascript ←

→ site.js:
function recive ComponentInstance
(objref)

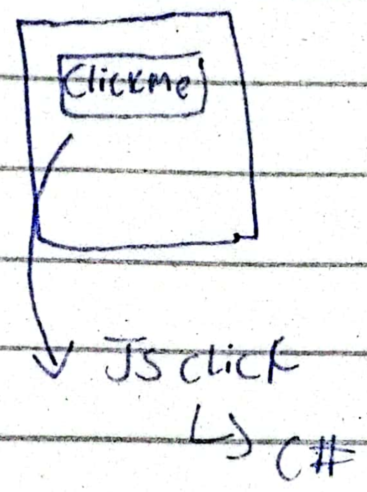
; No Assembly Name required NOW

{ objref. invokeMethodAsync
            ("Show Component Message");
}

: Add JS file in index :

<script src=js/site.js </script>

(*)



Main
⇒ Component :

<h3>Main Component </h3>

<button onclick = " Click JS Method" >
            ClickMe </button>

@ code

{ private DotNetObjectReference <MainComponent>

protected override OnInitialized {
    :
    }

```
public void ClickJsMethod()
{
    js.invokevoidasync
    .
    .
    .
}
```

⇒ correct.

→ @page "/clicker"

<h3> Clicker </h3>

<button> Click Me </button>

: Add class ClickHandler.

```
public class ClickHandler
{
    public int count {get; set;}
    [JSInvokable]
    public void TrackClick()
    {
        Console.WriteLine($" Button
            is clicked
            {count}
            times");
    }
}
```

→ clicker:

```
<h3> Clicker </h3>
<button id="trackButton"> Click Me
</button>
```

```
@inject IJSRuntime js;
@code
{
    //Step 2
    private DotNetObjectReference<ClickHandler>
        objectReference;

    protected override void
        OnInitialized()
    {
        object... = ...Create(new
            ClickHandler());
    }

    protected override Task OnAfter
        RenderAsync
        (bool firstRender)
    {
        if(firstRender)
        {
            await js.InvokeVoidAsync
                ("f1", objectReference);
        }
    }

    : Resources Deallocater:
    public void Dispose()
    {
        objectReference.Dispose();
    }
```

Good Practice to Deallocate resources.

⇒ site.js:

: We receive in F1 function:

```
function F1(o){
    var button = document.getElementById
                 ("trackButton");
    button.addEventListener
                 ("click", function(){
        o.InvokeMethodAsync
                 ("TrackClick");
    });
```

: check in inspect Console.

: Another Function to x and y

```
public void TrackClickWithXandY
                 (int x, int y)
{   count++;
    Console.WriteLine($"button is
                 clicked {count} times.
                 (x) and (y)
}
```

(*) Clicker:

<button id="TrackXandY" > Click Me to Get X and Y </button>

@code
{
// Call DotNetReference / ClickHandler
// Then initialize

protected override
:

Site.js:

function f2(){
var button = document.getElementById("TrackXandY")

button.addEventListener("click", function(event)
{
0.InvokeMethodAsync("TrackXandY", event.clientX, event.clientY)
});}

⊙ clicker

<button id="trackButton2"> Click me 4 button
Track x and y

@code{

'Same code as used before'

}

⊙ Local Storage:

⇒ Wanted to save temporary thing on browser, we can store in key, value pair.

: From Book (Chapter #4)

⇒ Functions to add, update and clear data.

⇒ js function localstorage set, get function

⊙ Progressive Web App (PWA):

⇒ It can be installed and run on desktop without internet and offline can be used.

⊕ Singleton state can be maintained by Dependency Injection.