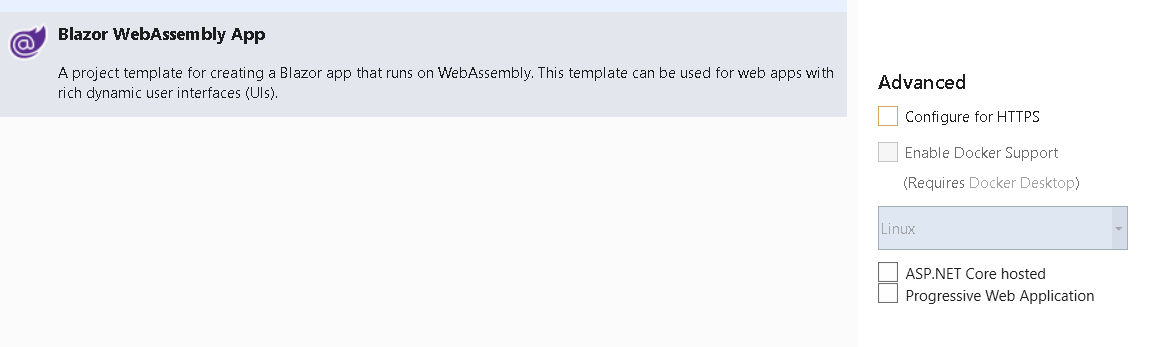
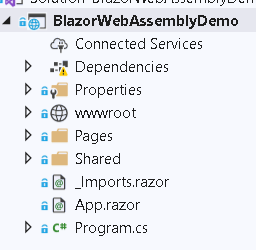
Demo : Create an WebAssembly Application

1. Open Visual studio and Create WebAssembly Project

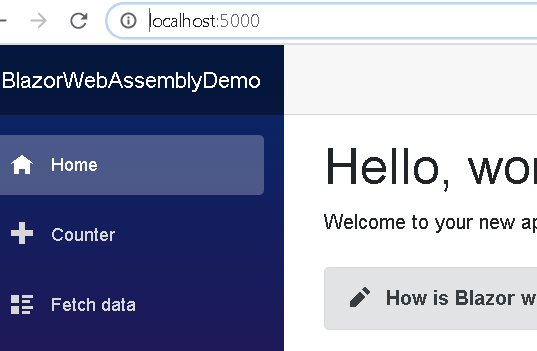




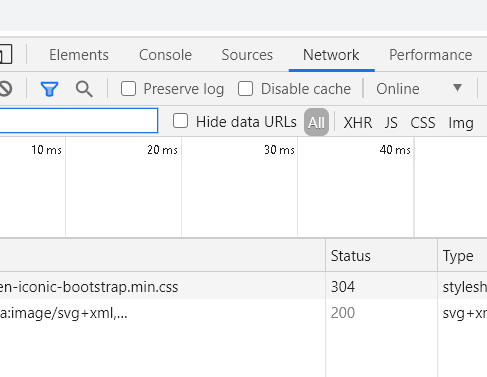
1. This will Create following project



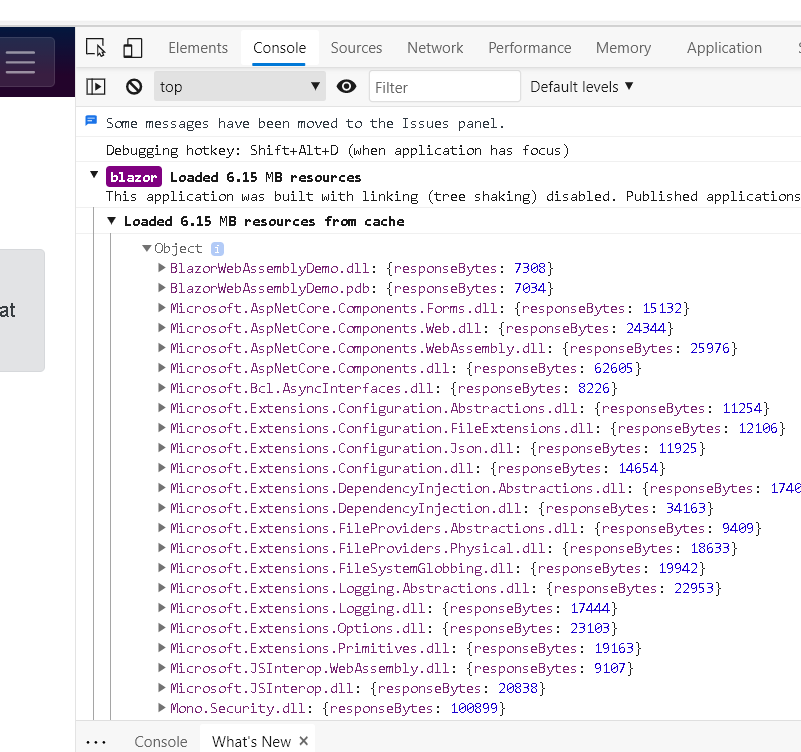
1. Run the application and see the network tab



1. Go to Network tab from Dev tools



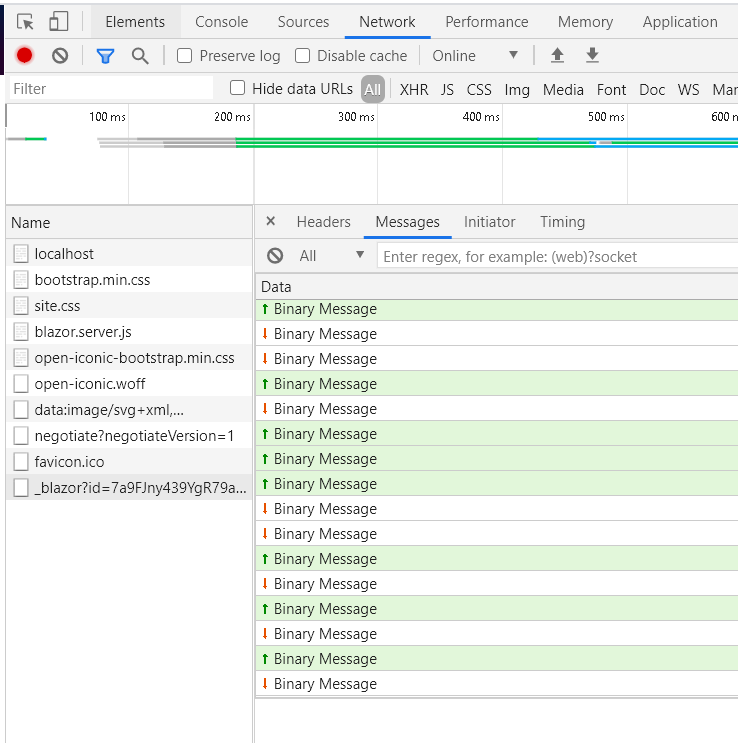
1. Go to Console section in MS Edge and Check the dll



* Blazor.webassembly. js : responsible for downloading other libs required
* Mono.wasm :

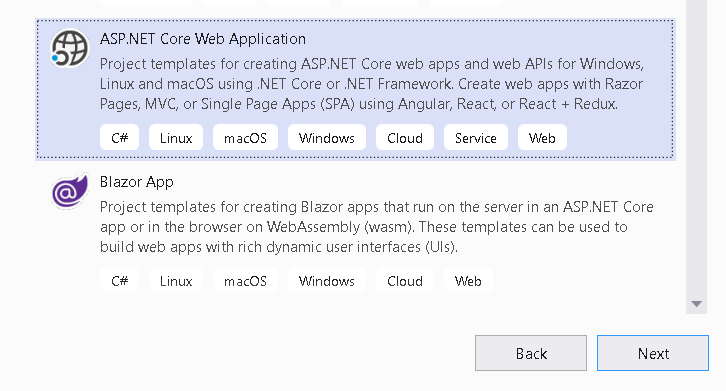
Demo : Blazor Server app

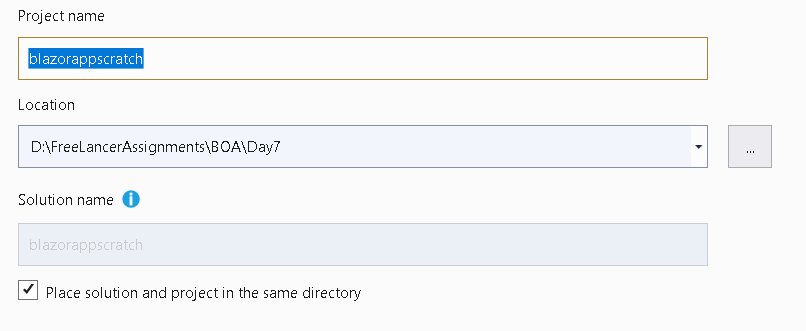
1. Create new Blazor Server app
2. Run the server app
3. Go to Dev Tools and check how its working



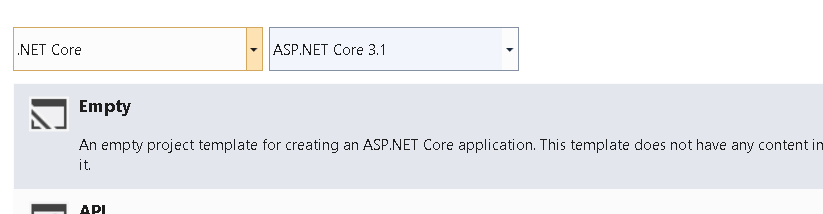
Blazor Create an application From Scratch

1. Add New ASP.Net Core Web Application

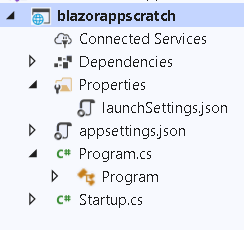




1. Create an Empty Project

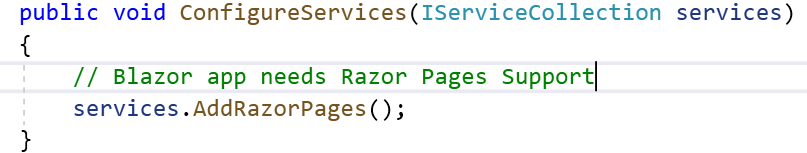


1. Project will have following structure

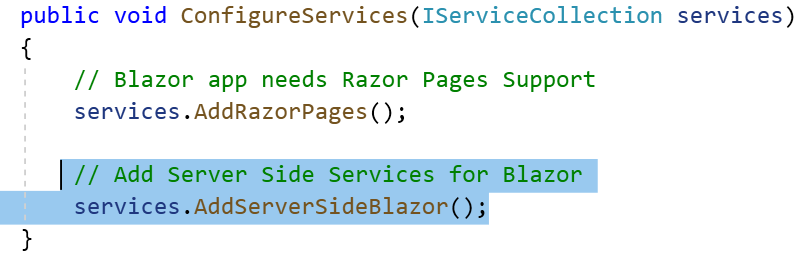


Configuring Application for Blazor Server

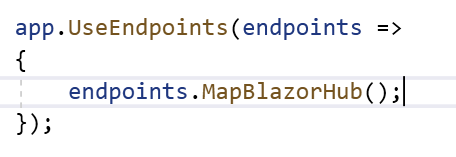
* Add Services and middleware to Startup class : AddRazorPages needs so that Razor Pages can work.



* Add Server Side Blazor Services to the Service collection.

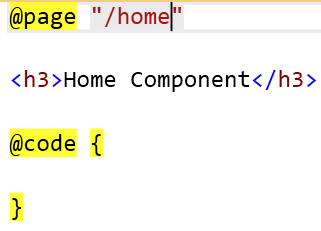


* Add endpoint.MapBlazorHub : Integrates Blazor with ASP.NET core endpoint routing. This will help SignalR to handle Persistent HTTP request.

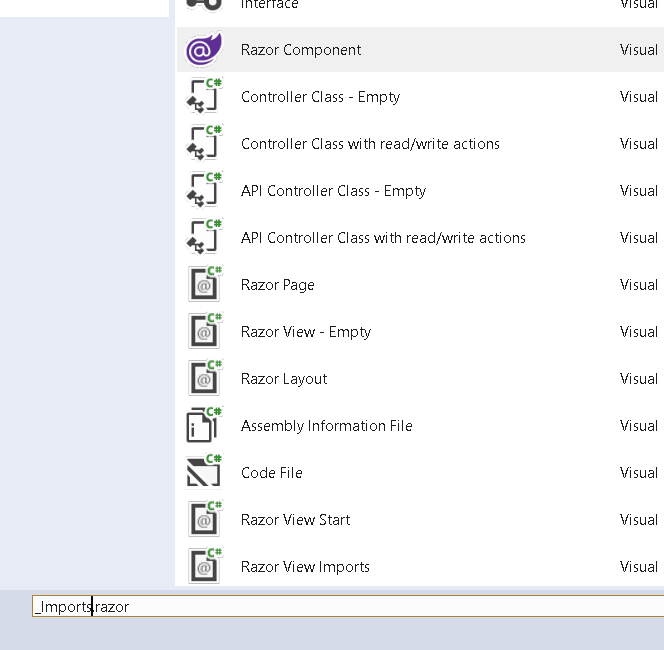


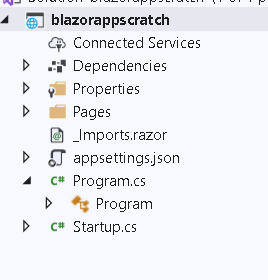
* Add Pages Folder and add Home Component



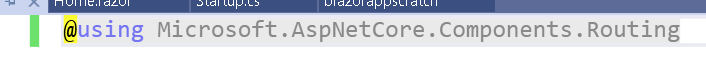


* \_Imports.razor : Use to place commonly used namespace in this file. So that you don’t have to import it separately.

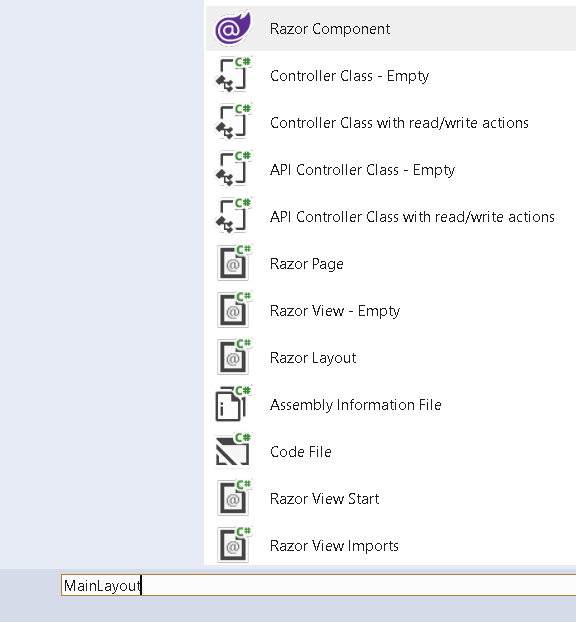




* Import Routing Feature : That is used for rendering and mapping routes to components . This is needed so that Urls are mapped to different razor components.



* Add New Folder : Shared and add MainLayout.razor .. this is our Main Component



Add Following code into that

@inherits LayoutComponentBase

<div class="sidebar">

@\*<NavMenu />\*@

</div>

<div class="main">

<div class="top-row px-4">

<a href="https://docs.microsoft.com/aspnet/" target="\_blank">About</a>

</div>

<div class="content px-4">

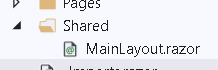
@Body

</div>

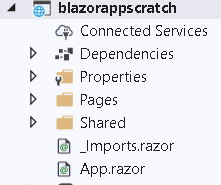
</div>

* Expose Shared folder





* Blazor uses Routing System for Selecting component based on URL . Add App.Razor to configure the same.



Add following code in App.Razor

<**Router** **AppAssembly**="@typeof(Program).Assembly">

<**Found** **Context**="routeData">

<**RouteView** **RouteData**="@routeData" **DefaultLayout**="@typeof(MainLayout)" />

</**Found**>

<**NotFound**>

<**LayoutView** **Layout**="@typeof(MainLayout)">

<p>Sorry, there's nothing at this address.</p>

</**LayoutView**>

</**NotFound**>

</**Router**>

Explaination

* specify the name of the .NET AppAssembly to use which in most cases is the current assembly:
* Context property is set to routeData which is received by RouteView Section which received Route Data and Default layout.
* RouteView Populates matching components contents inside the UI.
* Not Found Section just sets the same layout the sorry message.
* Fallback route – In case where there are only components and no views or Razor page then we have to use Fallback Route which will call a Razor page by name of \_Host.cshtml.

Fallback route has very low priority in routing matching so this route is initiated when other routes didn’t match.

@page "/"

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@namespace blazorappscratch.Pages

@{

Layout = null;

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>testblazorserver</title>

<base href="~/" />

<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />

<link href="css/site.css" rel="stylesheet" />

</head>

<body>

<app>

<component type="typeof(App)" render-mode="ServerPrerendered" />

</app>

<div id="blazor-error-ui">

<environment include="Staging,Production">

An error has occurred. This application may no longer respond until reloaded.

</environment>

<environment include="Development">

An unhandled exception has occurred. See browser dev tools for details.

</environment>

<a href="" class="reload">Reload</a>

<a class="dismiss">🗙</a>

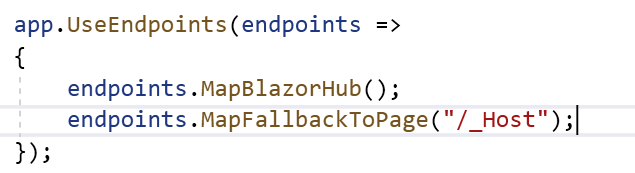
</div>

<script src="\_framework/blazor.server.js"></script>

</body>

</html>

* Add Fallback Path for the same.



* Explaination of Code

1. Component tag calls Router Component defined in App.Razor
2. Render model attribute sets the Rendermode to ServerPrerendered : it tells to render the component into static HTML and includes a marker for a Blazor Server side application. The HTML content of the component is sent again over the persistent HTTP connection.

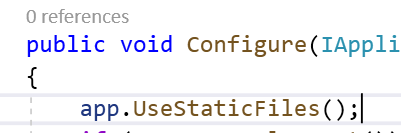
@page “/” : Specified URL of component

Script element specifies the name of JS file that establishes SignalR connection with the server. You don’t have to add this JS file.

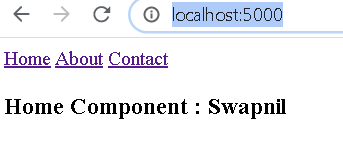
* How it will work or called

1. First the Fallback route which is \_Host is called. That forms top level html tags
2. Second Blazor router component which is App.razor called. This in turn will call MainLayout.razor. this Mainlayout will form Header, Footer
3. Inside it Component Code will show

* Add app.UseStaticFile in the Startup.cs



* Check the Output

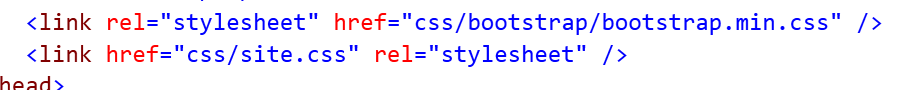


Enable Bootstrap

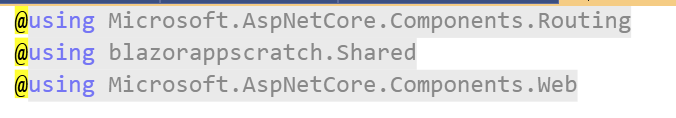
1. Include bootstrap files in the wwwroot folder



1. Apply it on \_Host.cshtml file



1. Include the Web Library



1. Check the Output

NavLink

* In-Built Component in Blazor and used for creating anchor tags for navigation
* Adv over normal anchor tag is it toggles an active CSS class with this you can assign active class .
* Also it renders anchor elements that are wired into the routing system.
* NavLinkMatch.All : Navlink is active when entire Current URL matches

1. Add NavLink

<div class="top-row pl-4 navbar navbar-dark">

<a class="navbar-brand" href="">testblazorserver</a>

<button class="navbar-toggler" **@onclick**="ToggleNavMenu">

<span class="navbar-toggler-icon"></span>

</button>

</div>

<div class="@NavMenuCssClass" **@onclick**="ToggleNavMenu">

<ul class="nav flex-column">

<li class="nav-item px-3">

<**NavLink** class="nav-link" href="" **Match**="NavLinkMatch.All">

<span class="oi oi-home" aria-hidden="true"></span> Home

</**NavLink**>

</li>

<li class="nav-item px-3">

<**NavLink** class="nav-link" href="counter">

<span class="oi oi-plus" aria-hidden="true"></span> Counter

</**NavLink**>

</li>

<li class="nav-item px-3">

<**NavLink** class="nav-link" href="fetchdata">

<span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data

</**NavLink**>

</li>

<li class="nav-item px-3">

<**NavLink** class="nav-link" href="products">

<span class="oi oi-list-rich" aria-hidden="true"></span> Manage Products

</**NavLink**>

</li>

@\*<li class="nav-item px-3">

<NavLink class="nav-link" href="products">

<span class="oi oi-list-rich" aria-hidden="true"></span> Parent

</NavLink>

</li>\*@

</ul>

</div>

@code {

private bool collapseNavMenu = true;

private string NavMenuCssClass => collapseNavMenu ? "collapse" : null;

private void ToggleNavMenu()

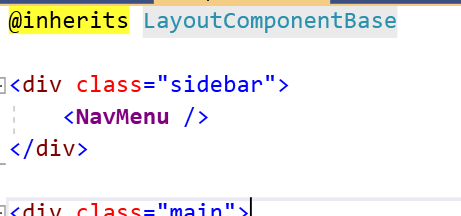
{

collapseNavMenu = !collapseNavMenu;

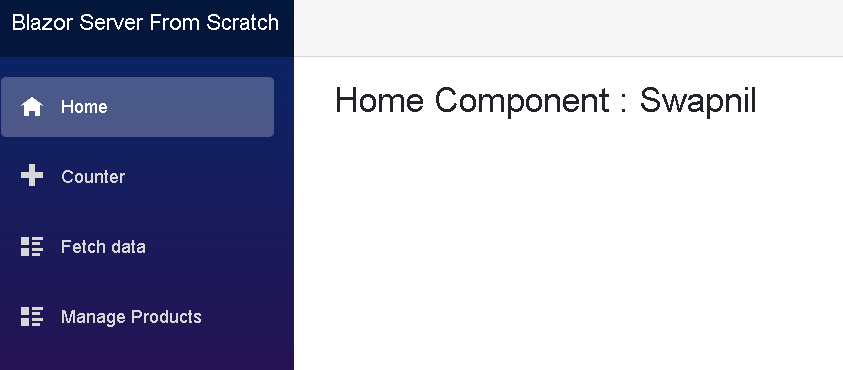
}

}

1. Add NavMenu in our main layout



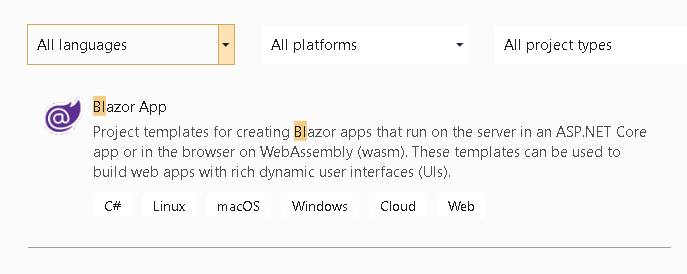
1. Check the Output

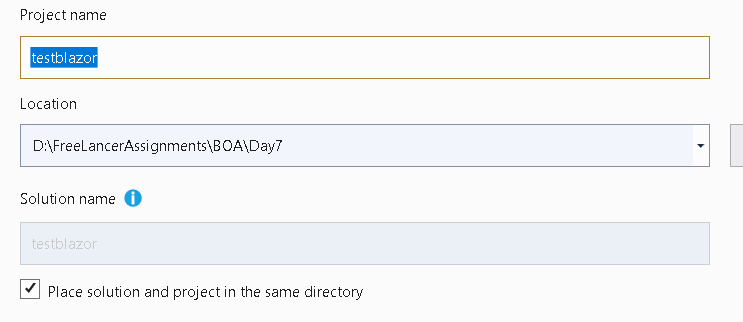


============================================================================

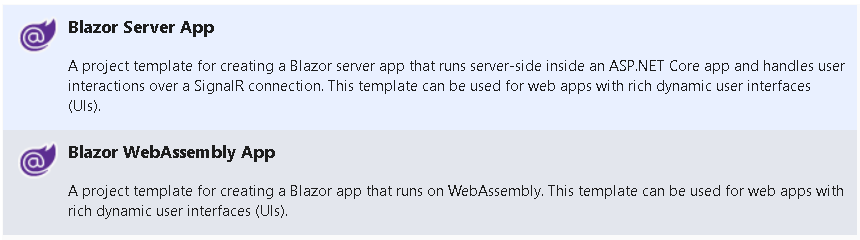
Demo

1. Add New Project





1. Add new Blazor Server App



What is Razor Component

* Building block of Razor application is Razor Component.
* Defined using .razor extension
* Usually grouped together to keep project organized.
* We don’t have @model class and model expression.
* Properties and methods that support components HTML are defined directly in an @code expression.
* Dependencies can be included using Inject
* Razor components are delivered to browser as part of Razor page or controller view.

Working with Data Bindings

* Each application receive input and display output.
* Data binding is the process where data is bound to HTML block with the logic @code blocks

Different Types of Binding

1. One Way Data Binding

* Class Variables and properties in the @code logic blocks are bound to HTML blocks for Display.
* An HTML block automatically displays any change in the value of a field .
* There is no input data from the HTML block bound to the @code logic

Real World Scenario

* We want to display number of words type , whenever user type in the textbox.

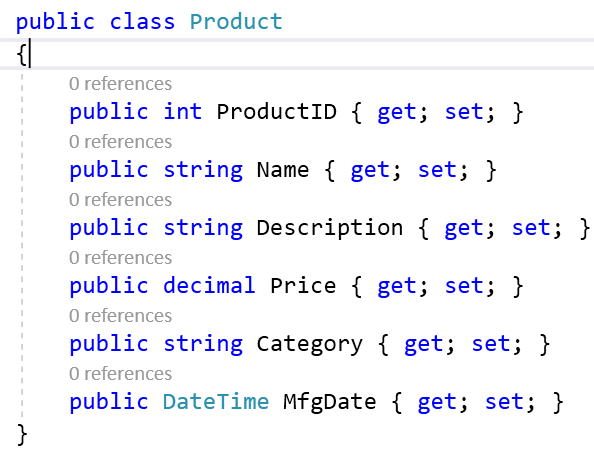
1. Two Way Data Binding

* Blazor supports 2 way data binding using @bind attribute.
* There is a bond between input data from HTML section and @code logic section. Change in input values of HTML section automatically changes the values in the @code logic section.

1. @bind=”@field”
2. @bind = “@property”
3. @bind=”eventname”
4. Attribute Binding

* Use to bind the value of an HTML attribute.

1. Add New Product class in Data folder



1. Add New Components Details.razor



1. Design the UI

@page "/details"

<h3>Details</h3>

<hr />

<div class="card bg-light mt-3" style="width:30rem ">

<div class="card-body">

<table class="table table-sm table-bordered table-striped">

<tr>

<td>Name</td>

<td><input **@bind**="@product.Name" **@bind:event**="oninput" /> </td>

</tr>

<tr>

<td>Price</td>

<td><input **@bind**="@product.Price" /></td>

</tr>

<tr>

<td>Category</td>

<td><input **@bind**="@product.Category" /></td>

</tr>

<tr>

<td>Description</td>

<td><input **@bind**="@product.Description" /></td>

</tr>

<tr>

<td> Mfg Date</td>

<td>

<input **@bind**="@product.MfgDate"

**@bind:format**="yyyy-MM-dd" />

</td>

</tr>

</table>

<ul>

<li>

@product.Name, @product.Price,@product.Category,@product.Description,

@product.MfgDate.ToString("yyyy-MM-dd"), @product.MfgDate.ToString("dddd")

</li>

</ul>

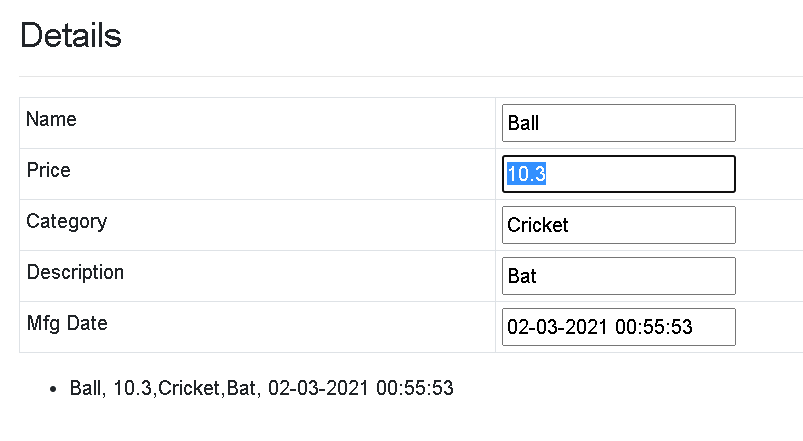
</div>

</div>

1. Check the Output : Data binding is happening when user press tab

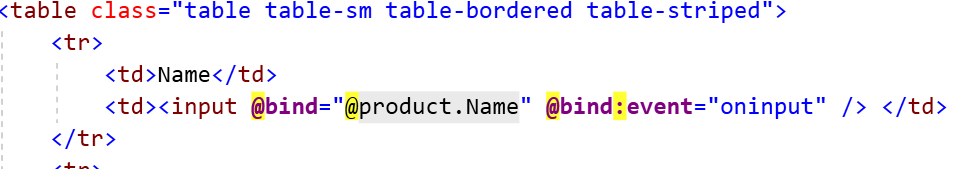
Textbox is updated in the UI only when the component is rendered not response to changing fields or property value. Since components render themselves after event handler code executes, field and property updates are usually reflected in the UI immediately after an event handler is triggered.

In this Case : When user types in the textbox and changes element focus , the onchange event is fired and currentvalue property set to the changed. @bind associates the current value of an expression and handles changes using registered handler.

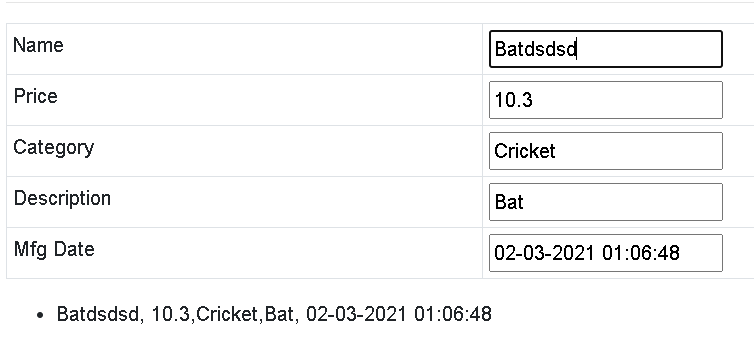


Bind a property or field on other events by also including an @bind:event attribute with an event parameter

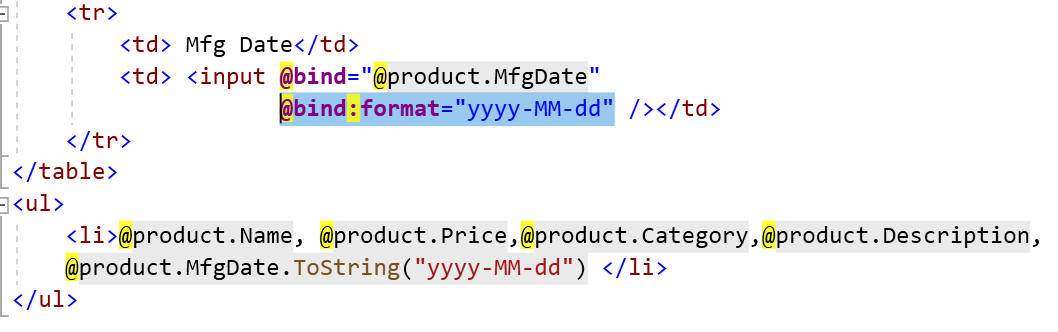
1. Add @bind:event on the input field



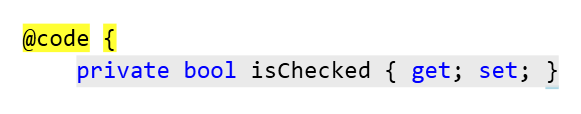
1. Check the Output



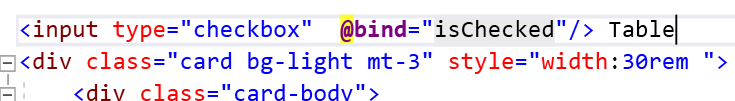
1. Add a bindformat on Date



1. Check the Output
2. Add variable isChecked in the code section



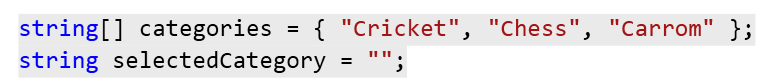
1. Attach it to Checkbox



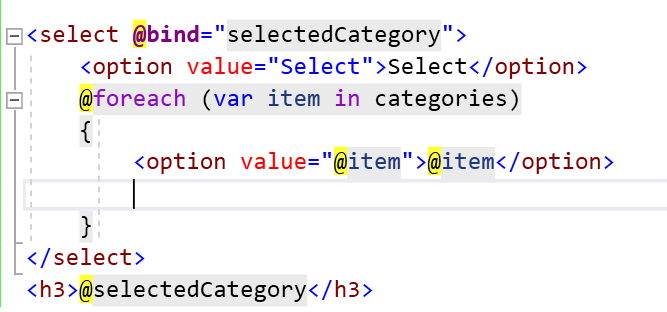
1. Check the Output

Demo : Bind To Select Control

1. Categories array in details.razor and Variable to hold selected Category



1. Add Select Control to display the data

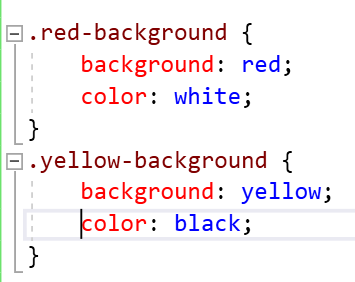


1. Check the Output

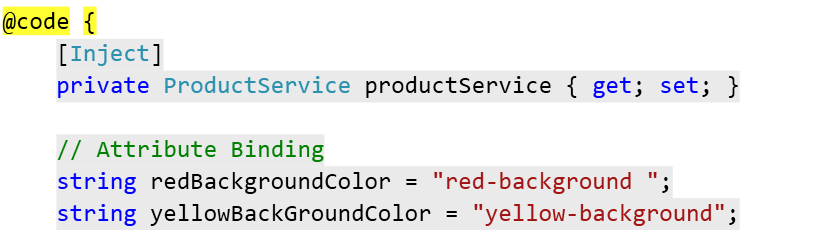
Demo : Attribute Binding

* Bind to an HTML attribute

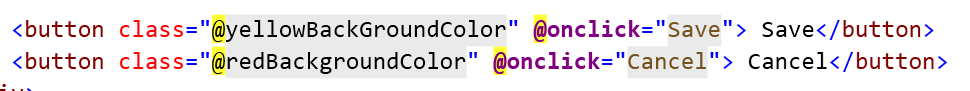
1. Add new classes in site.css file



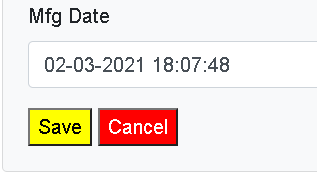
1. Add a variable to hold the classes.



1. Apply those variables to classes.



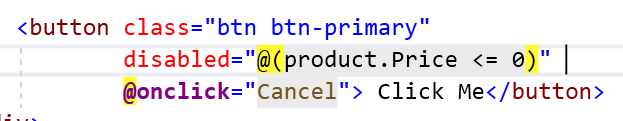
1. Check the Output



Conditional Attribute

* Use to control the browser by adding some attributes.

1. Add disable attribute on new button

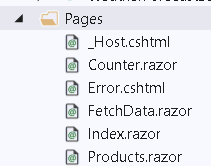


1. Check the Output

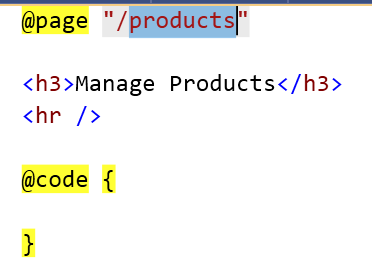
Event Handling in Blazor

* HTML element attribute named @on**{EVENT}** with a delegate typed.
* Event Handler Can be Asynchronous and return a Task.
* To Pass a parameter you can use Lambda Expression.

1. Add New Razor Component that will manage products.



1. Add URI for the same in the page



1. Change Model and make int nullable

public class Product

{

public int? ProductID { get; set; }

public string Name { get; set; }

public string Description { get; set; }

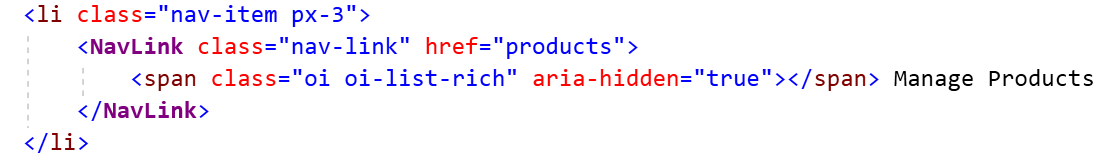
public decimal? Price { get; set; }

public string Category { get; set; }

public DateTime MfgDate { get; set; }

}

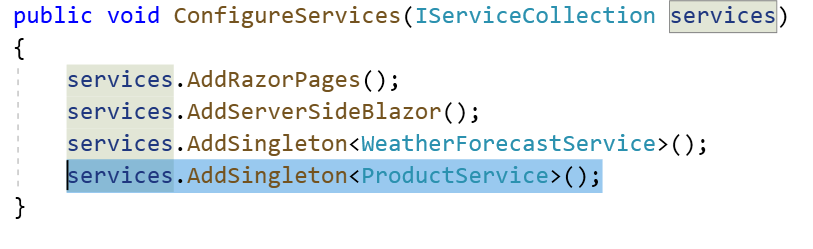
1. Add Link for the Product Page in NavBarMenu



1. Add ProductService and mock data



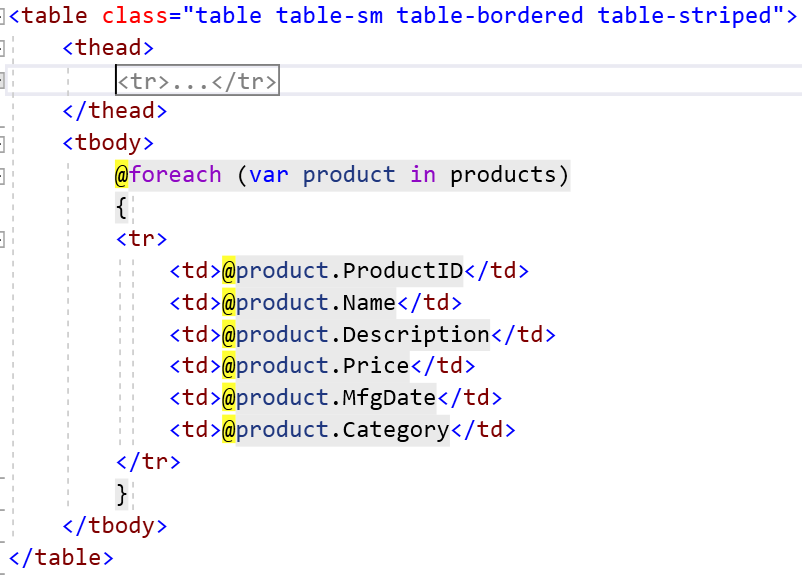
1. Activate ProductService object in startup.cs



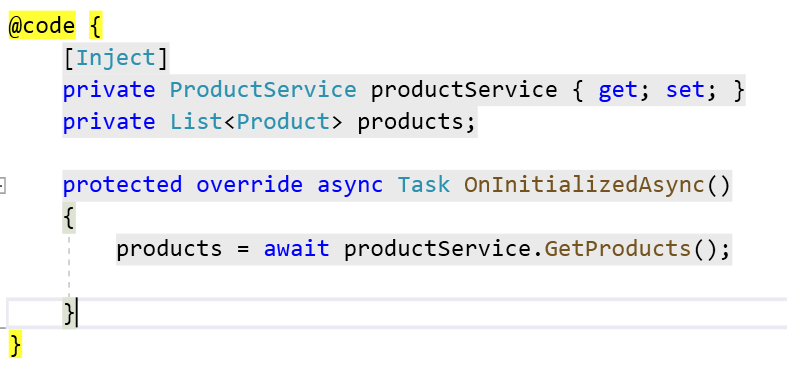
1. Add Data folder name in the import.razor



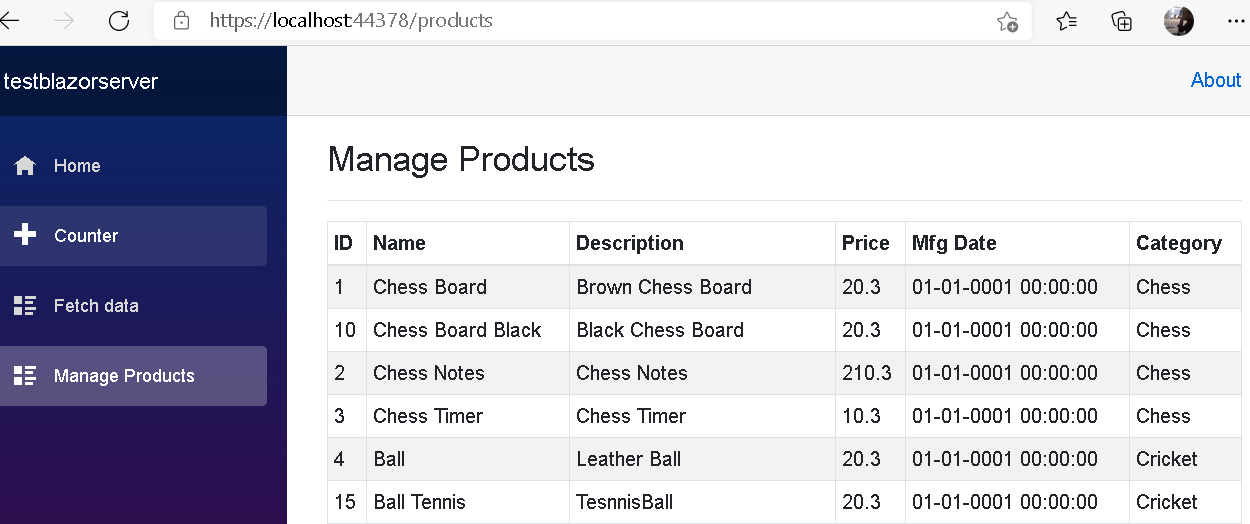
1. Inject Service in the component



1. Add a code



1. Check the Output

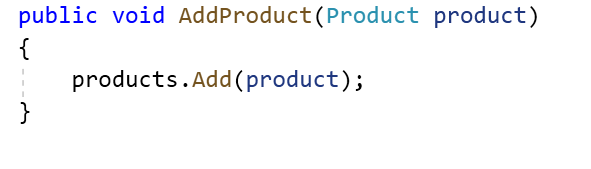


Add New Product

1. Add New Razor Component



1. Add new Method in the productservice



1. Add Code in Razor

@page "/add"

@inject NavigationManager NavigationManager

<h3>AddProduct : Event Handling</h3>

<div class="card bg-light mt-3" style="width:30rem ">

<div class="card-body">

<div class="form-group">

<label>ID</label>

<input type="text" class="form-control" **@bind**="product.ProductID" />

</div>

<div class="form-group">

<label>Name</label>

<input type="text" class="form-control" **@bind**="product.Name" />

</div>

<div class="form-group">

<label>Price</label>

<input type="text" class="form-control" **@bind**="product.Price" />

</div>

<div class="form-group">

<label>Description</label>

<input type="text" class="form-control" **@bind**="product.Description" />

</div>

<div class="form-group">

<label>Category</label>

<input type="text" class="form-control" **@bind**="product.Category" />

</div>

<div class="form-group">

<label>Mfg Date</label>

<input type="text" class="form-control" **@bind**="product.MfgDate" />

</div>

<button class="btn btn-primary" **@onclick**="Save"> Save</button>

<button class="btn btn-primary" **@onclick**="Cancel"> Cancel</button>

</div>

</div>

@code {

[Inject]

private ProductService productService { get; set; }

Product product = new Product

{

Name = "",

Description = "",

Category = "",

MfgDate = DateTime.Now

};

public void Save()

{

productService.AddProduct(product);

NavigationManager.NavigateTo("products");

}

public void Cancel()

{

NavigationManager.NavigateTo("products");

}

}

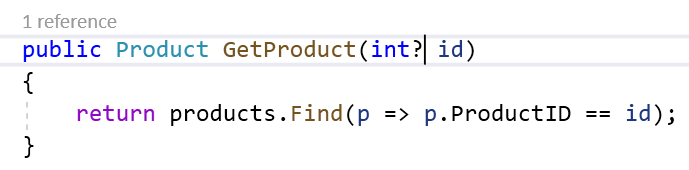
1. Check the Output : Now we will be able to add new product

How to Pass Parameter In the Events

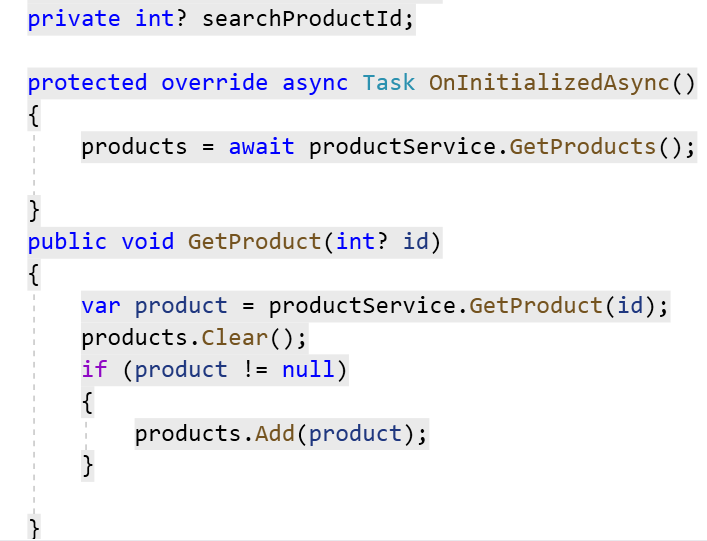
* You have to use Lambda Expression in order to pass data to Function

Demo : How to pass parameter

1. Add a method in ProductService



1. Create a method in the product.razor to call the GetProduct By ID



1. Change the UI Code

@page "/products"

<h3>Manage Products </h3>

<div class="card bg-light mt-3" style="width:50rem ">

<div class="card-body">

<**NavLink** class="btn btn-primary" href="add">

Add Product

</**NavLink**>

Search Product :

<input type="text" **@bind**="searchProductId" />

<button class="btn btn-primary"

**@onclick**="()=> GetProduct(searchProductId)">

Search

</button>

</div>

</div>

@if (products.Count == 0)

{

<div class="card bg-light mt-3" style="width:30rem ">

<div class="card-body">

<h4>No Records Found</h4>

</div>

</div>

}

else if (products.Count > 0)

{

<div class="card bg-light mt-3" style="width:50rem ">

<div class="card-body">

<table class="table table-sm table-bordered table-striped">

<thead>

<tr>

<th>Name</th>

<th>Description</th>

<th>Price</th>

<th>Mfg Date</th>

<th>Category</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var product in products)

{

<tr>

<td>@product.Name</td>

<td>@product.Description</td>

<td>@product.Price</td>

<td>@product.MfgDate</td>

<td>@product.Category</td>

<td>

@\*<button class="btn btn-info">Details</button>

<button class="btn btn-danger">Delete</button>

<button class="btn btn-warning">Edit</button>\*@

</td>

</tr>

}

</tbody>

</table>

</div>

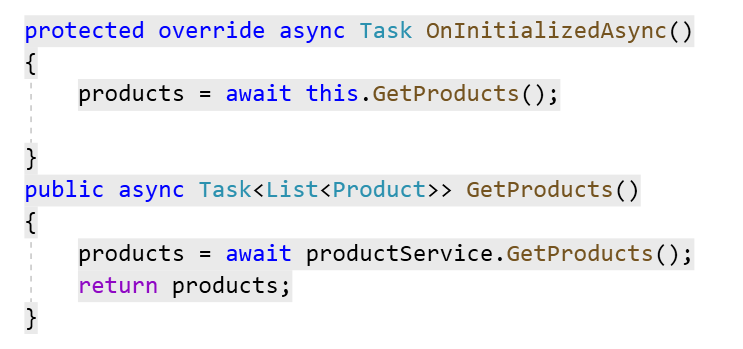
</div>

}

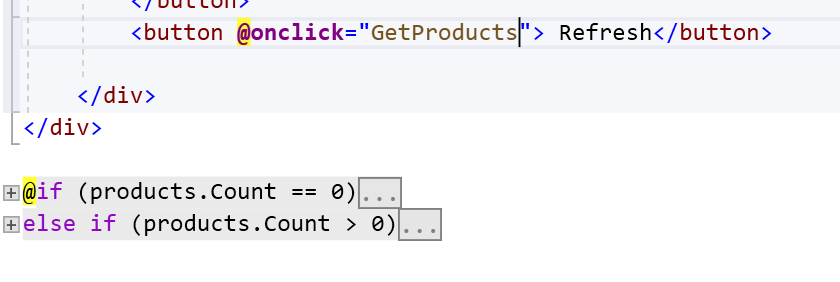
1. Check the Output

Add a Refresh button

1. Add GetProducts method and make the necessary changes



1. Call GetProducts method on Refresh button



1. Check the Output

EventCallback

* Its used to tell a parent , something happened in the Child.

Demo 1: Event Callback

1. Add Parent Component

@page "/counter"

<h3>ParentCounterComponent :@Count </h3>

<**Counter** **SetIncrementCount**="SetIncrementCount" ></**Counter**>

@code {

int Count = 0;

public void SetIncrementCount(int value)

{

Count = value;

}

}

1. Add Counter Component

<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class="btn btn-primary" **@onclick**="IncrementCount">Click me</button>

@code {

[Parameter]

public EventCallback<int> SetIncrementCount { get; set; }

private int currentCount = 0;

private void IncrementCount()

{

currentCount++;

SetIncrementCount.InvokeAsync(currentCount);

}

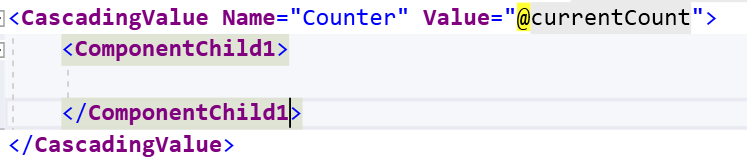
}

1. Check the Output

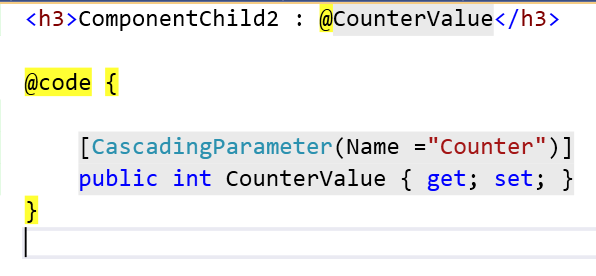
Cascading Values

* Its use to pass values from Parent to component to any leaf node

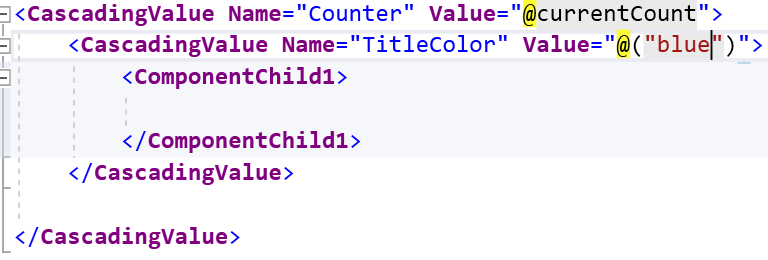
1. Add Component 1 and Component 2 and add the hierarchy for the same.



1. Go to ComponentChild 2 and Read the data



1. Check the Output
2. How to pass more parameters.



1. Use it in Component 2



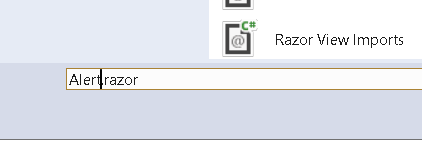
1. Check the Output

Component Parameter attribute

* This is required for components that want to expose their public properties to the parent component.
* Its use to pass data from Parent to child components.
* @ChildContent will hold content and needs to be type of RenderFragment because this is way Blazor engine passes it.
* For Example : Check SurveyComponent and Index.razor.

Demo

1. Add alert.razor component



1. Add Following code in Alert.razor

@if (Show)

{

<div class="alert alert-secondary alert-dismissable fade show mt-4"

role="alert">

@ChildContent

<button type="button" class="close" data-dismiss="alert"

aria-label="Close" **@onclick**="@Dismiss">

<span aria-hidden="true">&times;</span>

</button>

</div>

}

@code {

[Parameter]

public bool Show { get; set; }

[Parameter]

public RenderFragment ChildContent { get; set; }

public void Dismiss() => Show = false;

}

1. Add Alert Component in Product .razor

<div style="width:50rem">

<**Alert** **Show**="@ShowAlert">

<span class="oi oi-check mr-2" aria-hidden="true"></span>

<strong>Data is Deleted Successfully</strong>

</**Alert**>

</div>

1. Add a variable in the code section



Parent Child Communication

* Parent and child communicate through data binding.
* Alert Component communicates with the parent component through ShowAlert Property

Parent Component: product.razor

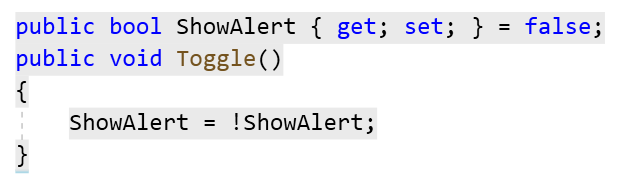
Show = ShowAlert

Parameter

Child Component: Alert.Razor

Show = Value

1. Add Toggle Method in products.razor



1. Check the Output

Timer Component

* Blazor Component is a class that inherits from ComponentBase class.
* This Component will call delegate Tick after certain number of seconds.
* Tick Parameter is a type of Action which is one of the built in delegate type of .NET
* Action<T > method returning void with one parameter of type T. This allows parent component to set the Action, So the child will execute an action.

Parent

Timer

5

TimeInSeconds

Toggle Function

Tick

1. Add Timer Component

public class Timer : ComponentBase

{

[Parameter]

public double TimeInSeconds { get; set; }

[Parameter]

public Action Tick { get; set; }

protected override void OnInitialized()

{

var timer = new System.Threading.Timer(

callback: (\_) => InvokeAsync(() => Tick?.Invoke()),

state: null, dueTime: TimeSpan.FromSeconds(TimeInSeconds),

period: Timeout.InfiniteTimeSpan

);

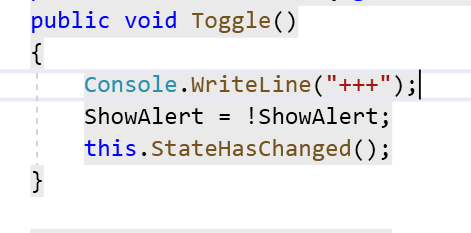
}

}

1. Call TimerComponent in products.razore



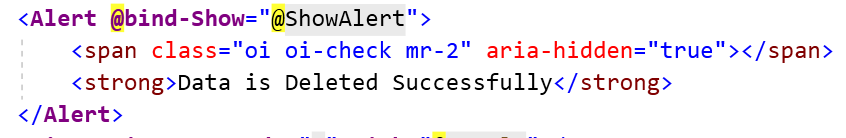
1. Add Toggle Method



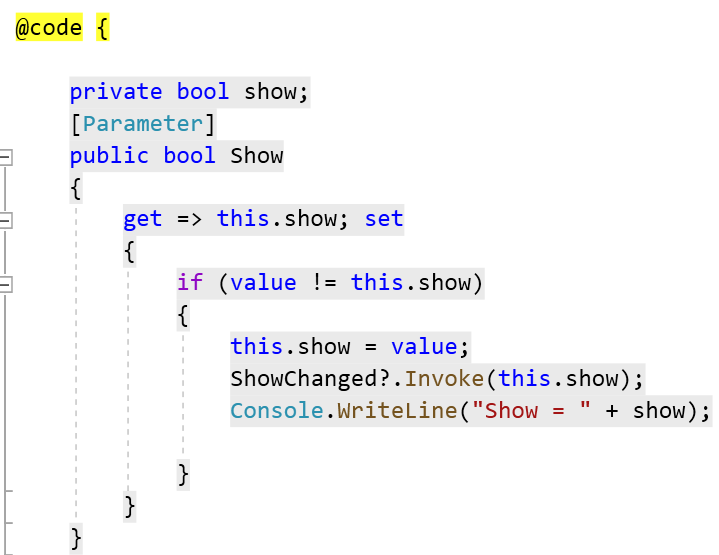
1. Check the Output : After 5 seconds , Alert box will disappear.

Use 2 Way Component to Component Data binding

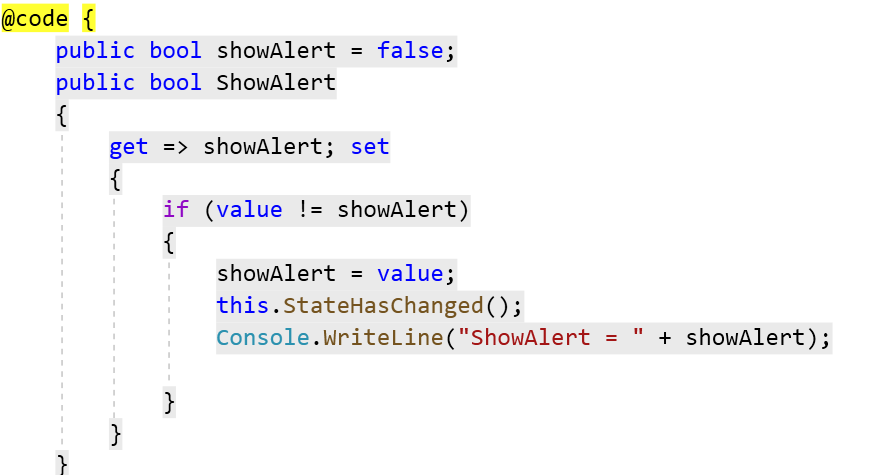
1. When user click the Alert box close button , it sets its own Show Property to false..
2. Problem is Parent Index Components ShowAlert stays true.
3. Changing the value of Show property wont change ShowAlert Property
4. With 2 way data binding changing value of Show Parameter will update value of showAlert property.
5. We have to use @bind—NameOfProperty syntax to data bind any property of a child component.
6. Add bind-Show Property on alert



1. Modify Property for the show in alert.razor



1. Modify Property for product.razor

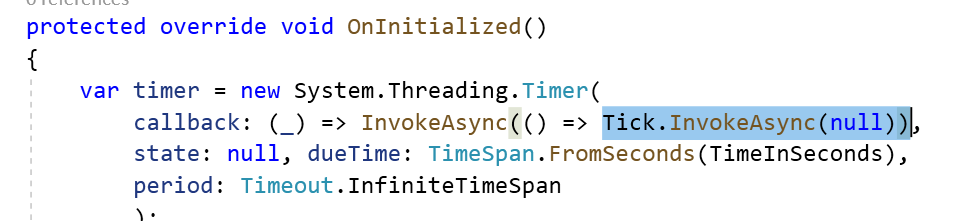


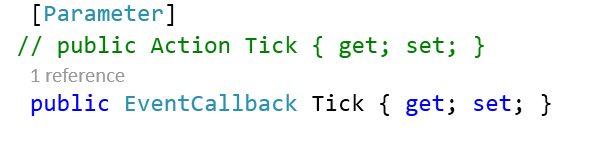
1. Check the Output

EventCallback

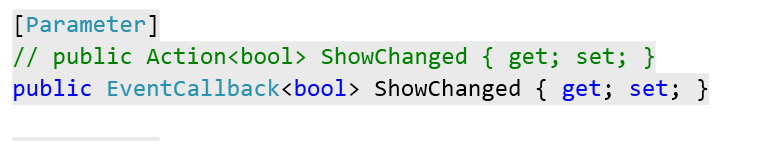
* We are using 2 way data binding between components with @bind-Show syntax and we used the ShowChanged callback to notify the parent component that Show property has changed.
* We also used StateHasChanged when the parents ShowAlert property get modified.
* Scenario where nested components wants to run a parent components method when a child component event occurs.
* Event argument types are permitted , specifying an event parameter in an event method definition is option and only necessary when event type is used in the method.
* To expose events across component use an EventCallback
* Parent component can assign a callback method to child components EventCallback.
* We should prefer EventCallback<T> over normal delegates for parent child relationship.

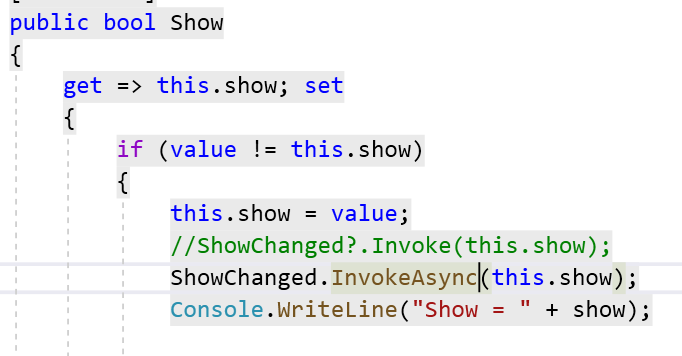
1. Add EventCallback in the Timer instead of Action





1. Go to Alert and change it





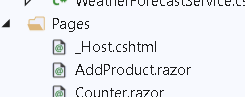
1. Remove the State Changed



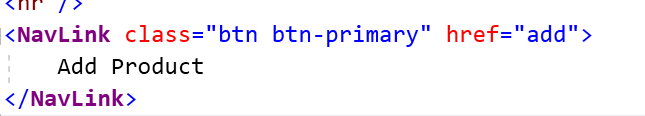
1. Check the Output

Demo

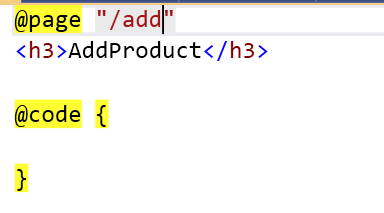
1. Add new Component AddProduct.razor



1. Add a Link in product.razor



1. Add a route for the same



Demo : Template Component

1. Add ListviewTemplate.razor

@\*Directive to specify type parameters\*@

@typeparam TItem

<ul>

@foreach (var item in Items)

{

@ItemTemplate(item)

}

</ul>

@code {

[Parameter]

public RenderFragment<TItem> ItemTemplate { get; set; }

[Parameter]

public IReadOnlyList<TItem> Items { get; set; }

}

1. Use this in Product.razor

<**testblazorserver.Templates.ListViewTemplate** **Items**="products">

<**ItemTemplate** **Context**="product">

<li>@product.Name</li>

</**ItemTemplate**>

</**testblazorserver.Templates.ListViewTemplate**>

1. Check the Output
2. Add TableViewTemplate.razor

@typeparam TItem

<table class="table">

<thead> <tr> @TableHeader</tr></thead>

<tbody>

@foreach (var item in Items)

{

<tr>@RowTemplate(item)</tr>

}

</tbody>

</table>

@code {

[Parameter]

public RenderFragment TableHeader { get; set; }

[Parameter]

public RenderFragment<TItem> RowTemplate { get; set; }

[Parameter]

public IReadOnlyList<TItem> Items { get; set; }

}

1. Add this in product.razor

<**testblazorserver.Templates.TableViewTemplate** **Items**="products">

<**TableHeader**>

<th>Name</th>

<th>Description</th>

<th>Price</th>

<th>Mfg Date</th>

<th>Category</th>

<th></th>

</**TableHeader**>

<**RowTemplate**>

<td>@context.Name</td>

<td>@context.Description</td>

<td>@context.Price</td>

<td>@context.MfgDate</td>

<td>@context.Category</td>

</**RowTemplate**>

</**testblazorserver.Templates.TableViewTemplate**>

1. Check the output