

**WebAssembly**

WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

A Route is a URL pattern and Routing is a pattern matching process that monitors the requests and determines what to do with each request.

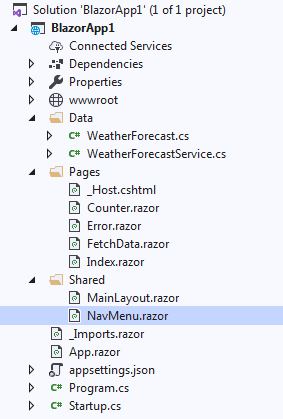
**@page directive**

Using @page directive, you can define the routing in Blazor component. The @page directives are internally converted into RouteAttribute when template is compiled.

In Blazor, Component can have multiple routes. If we require that component can render from multiple route values, we need to define all routes with multiple @page directives.

1. @page "/multiple"
2. @page "/multiple1"

|  |
| --- |
| 1. \_Host.cshtml (Master page of blazor app)  2. App.razor  3. MainLayout.razor  4. NavMenu.razor |



Flow of Application :-

1. Startup. cs

app.UseRouting();

|  |
| --- |
| app.UseEndpoints(endpoints =>  {  endpoints.MapBlazorHub();  endpoints.MapFallbackToPage("/\_Host");  }); |

* By convention, the *host* page is usually named *\_Host.cshtml*.
* The route specified in the host file is called a *fallback route* because it operates with a low priority in route matching.
* The fallback route is considered when other routes don't match

**Route templates**

* The Router component enables routing to each component with a specified route.
* The Router component appears in the *App.razor* file:

|  |
| --- |
| <**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**> |

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

There are two NavLinkMatch options that you can assign to the Match attribute of the <NavLink> element:

* NavLinkMatch.All – The NavLink is active when it matches the entire current URL.
* NavLinkMatch.Prefix (default) – The NavLink is active when it matches any prefix of the current URL.

# Core Blazor layouts

Technically, a layout is just another component. A layout is defined in a Razor template or in C# code

|  |
| --- |
| @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> |

To turn a *component* into a *layout*, the component:

* Inherits from LayoutComponentBase, which defines a Body property for the rendered content inside the layout.
* Uses the Razor syntax @Body to specify the location in the layout markup where the content is rendered.

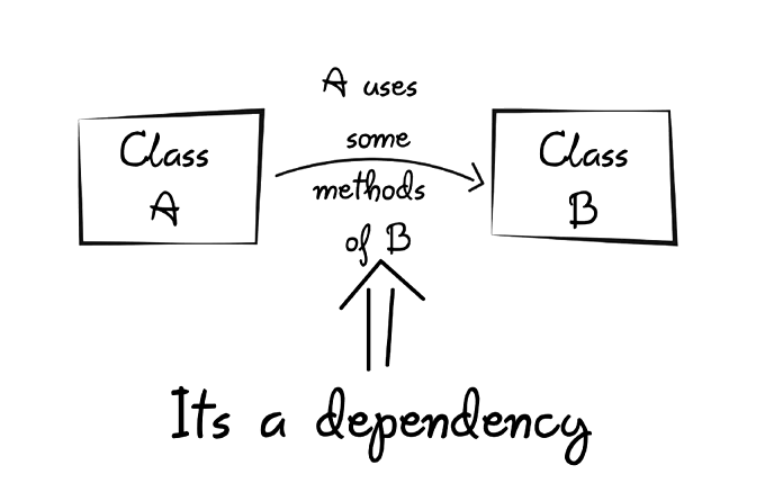
# Passing parameters in route

|  |  |
| --- | --- |
| |  | | --- | | * You can pass param using {} curly braces * Give multiple routing to prevent mandatory parameter passing * [Parameter] attribute is mandatory if using property as parameter in routing |   @page "/authors/authorDetails/{authorid}"  @page "/authors/authorDetails"  @page "/author"  <h3>AuthorDetails</h3>  <p>@AuthorID/ J.K rowling / harry potter</p>  @code {  [Parameter]  public string AuthorID { get; set; }  } |

**Passing data type parameters**

|  |
| --- |
| @page "/authors/authorDetails/{authorid}"  @page "/authors/authorDetails/{AuthorIDInt:int}"  @page "/authors/authorDetails"  @page "/author"  <h3>AuthorDetails</h3>  <p>@AuthorID/ J.K rowling string / harry potter</p>  <p>@AuthorIDInt/ book2 int/ book topic</p>  @code {  [Parameter]  public string AuthorID { get; set; }  [Parameter]  public int AuthorIDInt { get; set; }  } |

**Dependency Injection**

****

**Steps :**

1. Add services.AddSingleton<Interface name, class implementing this interface>(); [startup.cs]
2. @inject IAuthorService IauthorService; in razor component

|  |  |
| --- | --- |
| @inject IAuthorService IauthorService;   |  | | --- | | Works like **class** - declare props,methods in it  Works like **constructor**:- starting point |   @code {  [Parameter]  public int AuthorId { get; set; }  public Author AuthorDetails { get; set; }    protected override async Task OnInitializedAsync()  {  AuthorDetails = await IauthorService.GetAuthorById(AuthorId);  }  } |

**FORMS AND VALIDATIONS**

<**EditForm** **Model**="@author" **OnValidSubmit**="@SaveAuthor">

<**InputText** class="form-control col-3" **@bind-Value**="author.FirstName" placeholder="FirstName" />

USE <**DataAnnotationsValidator** /> to apply validations

|  |
| --- |
| <**EditForm** **Model**="@author" **OnValidSubmit**="@SaveAuthor">    <**DataAnnotationsValidator** />  <**ValidationSummary** />  <div class="col-12 row">  <label class="col-2 font-weight-bold">First name</label>  <**InputText** class="form-control col-3" **@bind-Value**="author.FirstName" placeholder="FirstName" />  <**ValidationMessage** **For**="@(() => author.FirstName)" />  <br />  <div class="col-12 row">  <span class="col-2"></span>  <input type="submit" class="form-control col-1 btn btn-primary" value="Save" />  <span>&nbsp;</span>  <input type="submit" class="form-control col-1 btn btn-primary" value="Clear" />  </div>  </**EditForm**> |

**To show validation error messages: -** <**ValidationSummary** />

validation against particular field only :-

<**ValidationMessage** **For**="@(() => author.FirstName)" />

**Add JavaScript in Razor component**

@inject IJSRuntime JSRuntime;

//To call js function from js file reference to which u can include in \_Host.cshtml

await JSRuntime.InvokeVoidAsync("saveMessage", firstname, lastname);

//best method to assign default values

protected override async Task OnAfterRenderAsync(bool firstRender)

{

if (firstRender && Cities == null)

{

Cities = await JSRuntime.InvokeAsync<string[]>("getCities");

}

StateHasChanged();

}

**Components**

Two types of components:-

* 1. Pages component : - Having Routing at top with @Directive
  2. Shareable component :- No directives at top , so these can be nested with others

Example of child reusable shareable component

Component name: -**Servervalidations.razor**

|  |
| --- |
| @if (IsVisible)  {    <div id="divValidationMessage" class="col-10 alert alert-danger row">  <div class="col-11" style="text-align:left">  <span>@ChildContent has been saved successfully</span>  </div>  </div>  }  @code {  [Parameter]  public bool IsVisible { get; set; }  [Parameter]  public RenderFragment ChildContent { get; set; }  } |

Parent component :- **Test.razor**

|  |
| --- |
| <**ServerValidations** **IsVisible**="IsVisible">  <span>Authors : </span>@RecordName  </**ServerValidations**>  @code {  public bool IsVisible { get; set; }  public string RecordName { get; set; }    private async Task SaveAuthor()  {  IauthorService.SaveAuthor(author);  IsVisible = true;  var firstname = author.FirstName;  var lastname = author.LastName;  RecordName = firstname + " " + lastname;  author = new Author();    }  } |