**Notes for Blazor (WebAssembly)**

1. Lifecycle of a Blazor Component (check **MoviesListGen.razor** component under BlazorMovies.Client\Shared)

* **OnInitialized** and **OnInitializedAsync** (to initialize the component (ex. to get data) after the HTTP request)
* **OnParametersSet** and **OnParametersSetAsync** (triggered any time a parameter is updated)
* **OnAfterRender** and **OnAfterRenderAsync** (triggered after component has been rendered)
* **ShouldRender** (define if a component has to be rendered again after rendering, ex. after the user performs some actions on the UI)
* **StateHasChanged()** to notify a change on the component

1. Dependency Injection refers to supply dependencies of a class from another class (services), and it can be performed using the **@inject** attribute.  
   Defailt services
   * HttpClient for HTTP requests to server
   * IJSRuntime to interact with JavaScript
   * NavigationManager to manage navigation through code

Lifecycle of a service

* Scoped, the service lives within a context (ex. during HTTP request). In the client side, it is like a singleton.
* Singleton
* Transient, different instances are created each time the service is requested

Let us create a new class for services in the client project called **Services.cs**

public class SigletonServices

{

. . . . . .

}

public class TransientServices

{

. . . . . . . . .

}

then we need to configure the DI in the **Program.cs** class in the following way



Then in the component we can inject the instance with

@inject SigletonService singleton  
@inject TransientService transient  
@inject IRepository repo

1. We can separate HTML code from c# code using partial classes, the partial class must have the same name as the component ex. **Counter.razor** and the class must be **Conter.razor.cs**, in this way we have a sort of code-behind, the class must be declared as partial, as following



And the component html will be



In a component it’s possible to invoke a method from JavaScript, ex.

[JSInvokable]

public static async Task<int> GetCurrentCount()

{

return await Task.FromResult(currentCountStatic);

}

And from JavaScript

function dotnetStaticInvokation() {

DotNet.invokeMethodAsync("BlazorMovies.Client", "GetCurrentCount")

.then(result => {

console.log("count from javascript", result);

});

To invoke an instance method the logic s the same it’s important to mark the c# method with the attribute **[JSInvokable]**.

1. **Using Identity Server**
   * DataContext class must inherit from IdentityDbContext   
     public class DataContext : IdentityDbContext //DbContext

Import the namespace from NuGet as

Microsoft.AspNetCore.Identity.EntityFrameworkCore

* + In DataContext make sure under the OnModelCreating to have the line  
    base.OnModelCreating(builder);  
    after the builder.Entity statements otherwise there will be an error during migration.
  + Under Startup.cs in ConfigureServices make sure to add the following lines  
     // Using Identity Server

services.AddIdentity<IdentityUser, IdentityRole>()

.AddEntityFrameworkStores<DataContext>()

.AddDefaultTokenProviders();

* + Finally run the command for a new migrations in order to create the table for IdentityServer. Open the terminal window, make sure to be in the Server folder then run  
    dotnet ef migrations add IdentityTables (or **Add-Migration IdentityTables**)  
    Then run  
    dotnet ef database update (or **Update-Database**)
  + To update db tools:  
    **dotnet tool install --global dotnet-ef --version 3.1.5**

Ref. <https://www.nuget.org/packages/dotnet-ef/>

The identity tables should be now created in our database.