**ASP.NET Core - Intermediate**

# 1 | ASP.NET Core Internals

## a | ASP.NET Core Internals

1. Create New Project MyMVA Web Application.
2. Set authentication to Individual user accounts (gives ability for users to login to the application. We can manage which logged in user can access which part of the application)
3. Till 20:00 is Basics of Startup class.
4. In Program.cs
   1. CreateDefault builder will load files like appsettings, loats its valie in configuration.
   2. It also loads environment variables.
5. In Home Page, index file.
   1. At top add @inject Microsoft.Extensions.Configuration.IConfiguration Configuration
      1. This makes available configuration to this page.
   2. Clear all divs.
   3. Add header tag with <h1>This is my Machine Name : @Configuration["computername"]</h1>
      1. This will find computername in configuration.
6. In appsettings.json
   1. Add Appsetting. "MyTwitterKey" : "TwitterSecretKey",
7. In Home Page, index file.
   1. Add header tag with <h1>This is my Machine Name : @Configuration["MyTwitterKey "]</h1>
8. Run and Check
9. This is in Development Env, but in Production Env we don’t want the same value.
10. We will override it with Env Variable.
11. Right click Project and select Properties, go to Debug Tab.
12. In Environment variables.
    1. The key ENVIRONMENT is set to development, if this is not set the by default it is production Env.
    2. Add New Key “MyTwitterKey” to value “TwitterSecretKeyFromEnvVar”
13. Run and check.
    1. Now we will get “MyTwitterKey” from Env Variable.
14. Appsettings.json is not the best secure way to store variables. By mistake it may be commited by developers. And it becomes public.
15. To avoid this we can use user Secrets
16. Right click Project select manage User Secret.
    1. We get a file secrets.json, this is not a encrypted file, it is not in source control.
    2. If a key exist in secret.json and in Env variable., then Env Variable will be selected.
    3. If it only exist in Scerets.json it will be picked.
17. Run and Check.
18. Now we will change log level in Env Variable.
    1. Add Value “LoggingLogLevelDefault” and key “Trace”
19. Run and check, now we will get trace level debugging. (just like in Application Insights)

# 1 | Tag Helpers

## a | Tag Helpers

1. Create New Project WebApplication24 Web Application., Set authentication to Individual user accounts.
2. In Register.cshtml
   1. The form uses tag helper, we see tag helper for div, label, input, span,etc.
3. In \_Layout.cshtml.
   1. We see tags environment, these are not HTML5 tags, these are higher level tag helpers.
      1. These generates a normal html, depending on development Env.
4. These Tag Helpers become active in \_ViewImports.
   1. @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers
   2. \*, this is the assembly, which is pulled.
   3. Else we define Namespace.( without \*,)
5. We will Learn how to make custom Tag Helper.
6. Add New Class RepeatetagHelper , inherit it from TagHelper
7. Overide ProcessAsync Method in Parent class.
   1. Add Forloop, for Count.
   2. Add output.Content.AppendHtml(await output.GetChildContentAsync(useCachedResult: false));
8. Go to \_ViewImports and add RepeatetagHelper.
9. Add Repeat Tag to Index Page, with count 6, add p tag with content
10. Run and Check.
    1. The p tag will be repeated Count Times.
11. We will check tag helpers created by other developers, and we can use it.
    1. DamianEdwards.
    2. DavidPaquette (dpaquette)
    3. Check their Github, Examples and Source.

# 1 | Entity Framwork Core

## a | Entity Framwork Core

1. POCO : Plain Old CLR Object
2. Create new console app data-seeding , copy code from ref proj. (Ist Proj Example)
3. In Proj add Nuget Packages : (Note : For Data Seeding to work install version 2.1)
   1. Microsoft.EntityFrameworkCore
   2. Microsoft.EntityFrameworkCore.SqlServer
   3. Microsoft.EntityFrameworkCore.Tools
4. Open data-seeding Proj, Program class.
   1. Here we have simple POCO Class, which act as models.
   2. BloggingContext is the entry point of data access for the application, which is derived from DbContext.
   3. It have DbSet Properties, which makes tables.
   4. We Overide OnConfiguring to specify server details.
   5. We Overide OnModelCreating
5. Now Db is Not Created.
6. Open package manager console.
   1. Type Add-Migration Initial and Enter.
   2. We will see that Migrations folder is created
7. In Migrations Folder, Open first initial,cs file.
   1. It Has Up and Down Method
   2. Up Method Has FLUENT Api calls to Add the changes.(Creation Od Db, Tables.)
   3. Down Method Has FLUENT Api calls to Remove the changes.
8. Open package manager console.
   1. Type Update-Database and Enter.
   2. This will generate or update the DB.
9. In Startup.cs, in OnModelCreating
   1. Change first Theme color to AliceBlue in HasData
10. Open package manager console.
    1. Type Add-Migration Alice and Enter.
    2. We will see that in Migrations folder, Migration File \*\_Alice.cs has Changes to change Red To Alice Blue.
    3. Type Update-Database and Enter.
    4. Changes will be reflected in DB
    5. Type Script-Migration, this will generate a .sql file, which we can share.
11. Run and Check.
12. Create new console app flexible-mapping , copy code from ref proj. (2nd Proj Example)
13. In Proj add Nuget Packages : (Note : For Data Seeding to work install version 2.1)
    1. Microsoft.EntityFrameworkCore
    2. Microsoft.EntityFrameworkCore.SqlServer
    3. Microsoft.EntityFrameworkCore.Tools
    4. Microsoft.Extentions.Logging
    5. Microsoft.Extentions.Logging.Console
14. Open flexible-mapping Proj, Program class.
15. What is Flexible Mapping.
    1. <https://gavilan.blog/2018/04/21/flexible-mapping-example-entity-framework-core-2-0/>
    2. Summary : We can use Flexible Mapping to associate columns of our SQL Server table with fields of our model. The advantage of this is that it allows us to apply transformations to our values prior to Entity Framework inserting them into the database.
16. Here we see one Entity Type, ( Model) Blog.
    1. Here for this model we have used field mapping for Url Property.
    2. Add \_url field.
    3. Make Url Property Readonly, remove Set;, and add method SetUrl.
17. We need to configure this \_url field.In On ModelCreatin Method.
    1. Add modelBuilder.Entity<Blog>().Property<string>("Url").HasField("\_url");
    2. This says property “Url” is backed by filed “\_url”
18. What is Shadow Properties : <https://docs.microsoft.com/en-us/ef/core/modeling/shadow-properties>
19. Check SetupDatabase Method.
    1. For testing we have apis to delete database and create them.
       1. db.Database.EnsureDeleted(); db.Database.EnsureCreated();
20. Run and Check.
21. Add New Entity Type(Model) Post
    1. Add Properties Id, Title, Body.
    2. In Blog Model Add Collection of posts.
22. Run and Check.
    1. Post Table will be created. Has blogId Column
23. Add Prop BlogFK to Post. We want this to be the Foreign key
24. In OnModelCreating Method
    1. Add modelBuilder.Entity<Blog>().HasMany(b => b.Posts).WithOne().HasForeignKey(p => p.BlogFK);
    2. This will make sure that BlogFK Is the foireign Key.
25. Run and Check. BlogFK Is the foireign Key.
26. Create new console app query-filters , copy code from ref proj. (3rd Proj Example)
27. In Proj add Nuget Packages : (Note : For Data Seeding to work install version 2.1)
    1. Microsoft.EntityFrameworkCore
    2. Microsoft.EntityFrameworkCore.SqlServer
    3. Microsoft.Extentions.Logging
    4. Microsoft.Extentions.Logging.Console
28. Open query-filters Proj, Program class.
29. New Feature in EF Core 2.0
    1. This is to improve a feature , called Eager Loading
    2. What is Eager Loading :
       1. <https://docs.microsoft.com/en-us/ef/core/querying/related-data>
       2. You can use the Include method to specify related data to be included in query results. In the following example, the blogs that are returned in the results will have their Posts property populated with the related posts. (eg .Include(blog => blog.Posts))
    3. In Main Method, check line 18.
       1. It says when you query for Blog, bring Instances of Posts as parts of results.
    4. Here when we ask to include Posts, we want to apply a filter. (e.g. We want Posts from Last 1 Month, etc.), here we use query filter
30. In OnModelCreating method
    1. We write our first filter. modelBuilder.Entity<Post>().HasQueryFilter(p => !p.IsDeleted);
    2. Comment it for now.
31. Run and Check, we will see that no filter is applied.
32. Then Uncomment the Is deleted filter.
33. Run and Check, we will see that filter is applied.
34. If we want to Ignore the QueryFilters. Add .IgnoreQueryFilters() to the queries.
35. Run and Check, we will see queryFilter is not applied.
36. Now we add Query Filter For Multi-Tenancy
    1. modelBuilder.Entity<Blog>().HasQueryFilter(b => EF.Property<string>(b, "TenantId") == \_tenantId); where \_tenantId is a readonly field of BloggingContext.
    2. Here we access a Ststic API of EF, EF.Property, here we check tenantId of Blog should be equql to \_tenantId in DbContect.
37. Uncomment IgnoreQueryFilters() , and run.
38. Run and Check.