**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)

# 2 | 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.

# 3 | 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.

# 4 | Authentication and Authorization

## a | Authentication and Authorization

1. Create a .Net Core Application Authentication\_Authorization,
2. In Change Authentication use Individual User Accounts.
3. In Startup we will see Authentication is setup in middleware.
   1. In ConfigureServices Method services.AddIdentity is done
   2. In Configure method, app.UseAuthentication(); is used
      1. It is after UseStaticFiles, since we donot want to check if user is logged in while serving static files.
      2. It is before UseMvc, since we want to check if user is logged in while serving a Page.
4. Run And Check
5. VS will ask to create SSL Certificate, and in installed locally on machine, say yes, again say yes.
6. Check in address bar, we will see https://
7. GO to Register page, Enter email id : krunalrele@gmail.com, password : Gctlab@123 and click Register.
8. We will get Error, we need to set connection string. And press Apply Migration button, once done, Refresh the page.
9. Now go to your account.
   1. We can edit info,
   2. change password
   3. do Two factor Authentication
      1. but doing 2FA using long code is tedious. So we will do it using QR Code.
      2. Goto the documentation link provided in the page.
10. Adding QR code patch
    1. We will do it using javascript libraries.
    2. Goto <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity-enable-qrcodes?view=aspnetcore-2.0> and follow instructions.
    3. Download qrcode.js library and pu it in lib folder in wwwroot
    4. Make changes in Pages/Account/Manage/EnableAuthenticator.cshtml and cshtml.cs
11. Run and Check, we will see Qr code is there.
12. Goto <https://gauth.apps.gbraad.nl> and here you check two factor authentication in Dev Mode.
    1. Add Account name and Secret Key.
    2. Then you will get code.
    3. Put that code in your website, and click ok, we have our 2FA setup.
    4. It gives us some recovery code. In case we lost phone with authenticator app.
    5. 8d0d2446 c9f8e530 these are one time use password. Reset Recovery codes gives new 10 Recovery password.
13. Now Logout and login it will ask you for code.
14. When you you select Remember this computer, it will again not ask for the code.
15. Till Now AspNetUsers and AspNetUserTokens are filled.

## b | Enabling Authentication using External Providers

1. Can we login to website using Microsoft, Google, Facebook, Twitter account, YES
2. Go in Login Page, We see on Right Side , we see documentation link for Using other Service to login.
   1. <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/index?view=aspnetcore-2.0>
3. We will do with Microsoft account. Click on Microsoft link and follow instructions.
   1. Go to <https://apps.dev.microsoft.com/> and add an app with app name. this will provision a new application for me.
   2. In Application Registration Page,
      1. Click Generate New Password. Copy the password”jxAB4?+\_qlvdqZRYGZ8482)” we will need to put this in our application.
      2. Copy Application Id “a9e4c03c-6c9e-4cc9-9ad9-be192c8f0c0a”
      3. This Id and password is used to contact with Microsoft, that I an this registered application, check these passwords for me.
      4. Click on Add Platforms, select web.
      5. In Redirect Url add localhost url with port, “https://localhost:44322/” + “signin-microsoft”
         1. This is the callback Url, once Microsoft is done authenticating, acll abck to this url.
         2. Click save.
4. We donot want to put the app Id and password, in appsetting.json, we put it in secrets.json (Rightclick Project select Manage User Secrets)
   1. Add "Authentication:Microsoft:ApplicationId": "a9e4c03c-6c9e-4cc9-9ad9-be192c8f0c0a", "Authentication:Microsoft:Password" : "jxAB4?+\_qlvdqZRYGZ8482)"
   2. This will not get checked in source control.
   3. For setting it on another server , set the environment variable.
5. Go in startup class in Configure Services method, under addIdentity Servce call,
   1. Add Microsoft Auth Service and give it AppId and Password.
6. Run and Check.
   1. Goto Login Page, We will see Microsoft Button in Right side, Click it.
   2. Login through Microsoft Account,
   3. Then click register giving email id,
7. We Are done signing in using Externalauthentication.

## c | Introduction to Authorization

1. Create a .Net Core Application IdentityAuthorizationSample,
2. In Change Authentication use Individual User Accounts.
3. In Startup class in ConfigureServices Method.
   1. Set Dbcontext service to use in Memory Database AuthSample
4. Now to seed data in configure services.
   1. Add SampleData,cs in Data Folder
      1. Add static Method InitializeData to with parameters IServiceProviders and ILogger factory
      2. First we define a service scope for Dependency injection provider using IServiceScopeFactory.
      3. We are going to be working with services, userManger , logging etc, this is a workaround for constructor injection (Like we normally do), because we are interacting with that provider directly., and we need to declare our scope.
      4. Check id Dev Env if so donot do the initialization
      5. Then we get RoleManager service, add two roles.
      6. Then we Get UserManager service, add a user.
   2. In Startup class in Configure Method, after app.UseAuthentication(), call
      1. SampleData.InitializeData(app.ApplicationServices, loggerFactory);
5. Rightclick Project and select Edit .csproj
   1. Add <DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools" Version="2.0.2" />
   2. When we go to cmd. And type dotnet watch run this will run the program, and if changes made to cs files , it will recompile and run automatically.
6. Run and Check, we will be able to login Using the user name we setup in sample data.
7. Now to check data in DB, User Local server and set connection string.
8. Then in Configure Method before Initializing data, Ensure.DB.Deleted, Ensure.DB.Created
9. Run And Check.
10. In Users Table, user will be added, Roles table two roles will be added.
11. Now We will Check Authorization.
12. We want to allow a user to about page only when logged in
    1. Go to About Page Model.and decorate AboutModel class with [Authorize] attribute
    2. Run and Check, without logging in when we click on Aboutus page, it redirects us to Login Page, here in Url we see the return url is about page. And when we login it redirects us to the about page.
13. Now we want only Admin Role to access the About Page.
    1. Go to About Page Model.and decorate AboutModel class with [Authorize(Roles = “Admin”)] attribute
    2. Now Run and Check, even if you are logged in , you will see access denied page, when you click on About Page.
14. Now we give test user Admin Role.
    1. Go to SampleData, add await userManager.AddToRoleAsync(user, "Admin");,this will add user role as admin.
    2. Run and Check, Logout and login, goto about page, you will be able to access the page. Since user is given admin role.
    3. We will see that along with User , roles Table, UserRoles Table aslo has an entry. Stating users role.
15. Now we want About page should be accessed be people of specific country, for this we enforce a Policy.
    1. In Startup class in ConfigureServices Method, after services.addIdentity,
       1. Add services.AddAuthorization
       2. And configure it with configure.AddPolicy("CanadiansOnly", policy => policy.RequireClaim(ClaimTypes.Country, "Canada"));
          1. This will ensure that user is from Canada, to have access to About page.
       3. And configure it with configure.AddPolicy("CanadiansOnly", policy => policy.RequireClaim(ClaimTypes.Country
          1. This will ensure that user has a country associated, to have access to About page.
       4. We will use the one with specific country Canada.
    2. Go to About Page Model.and decorate AboutModel class with [Authorize(Policy = “CanadiansOnly”)] attribute
    3. Run and Check, Logout and login, try to access About page, it will say Access denied, since even if user is admin, he is not Canadian.
16. Now we will grant a claim to our Test user, that he is from Canada.
    1. In Sampledata add await userManager.AddClaimAsync(user, new Claim(ClaimTypes.Country, "Canada"));
    2. Run and Check, Logout and login, try to access About page, user can access it, since he is Canadian.
    3. We will see that along with User , roles , UserRoles, Tables UserClaims Table aslo has an entry. Stating users Claims.
17. Now we want About page should be accessed be people of specific country(Canada) or Should be admin (Canadian OR Admin)
    1. Add Requirements Folder to project. Add Class CanadianReqirement to it
       1. Derive it from AuthorizationHandler<CanadianRequirement>, IAuthorizationRequirement, and implement IAuthorizationRequirement
       2. Define Contructor.
       3. In method HandleRequirementAsync which came from interface implementation.
          1. Check admin role and Claim of country, if yes then succeed.
       4. In Startup class add the new policy
          1. configure.AddPolicy("CanadianOrAdmin", policy => policy.AddRequirements(new CanadianRequirement()));
       5. Go to About Page Model.and decorate AboutModel class with [Authorize(Policy = “CanadianOrAdmin”)] attribute, this enforce the policy.
    2. Run and Check, Logout and login, try to access About page, user can access it, since he is Canadian and Admin.
    3. In SampleData Comment country claim.
    4. Run and Check, Logout and login, try to access About page, user can access it, since he is Admin.
    5. In SampleData Comment Admin Assignment.uncomment Canada claim.
    6. Run and Check, it will not work,
       1. In CanadianRequirement class where we chaeck Country isa Canada,in if statement where we check claimType, it should be claim.Type, and not claim.ValueType.
    7. Run and Check, Logout and login, try to access About page, user can access it, since he is Canadian.
18. Now we want to hide About link, if we donot have authorization.
    1. Go to \_Layout page,
       1. Add at top @using Microsoft.AspNetCore.Authorization,
       2. inject @inject IAuthorizationService AuthorizationService
       3. whre we show about p[age link
          1. add if((await AuthorizationService.AuthorizeAsync(User, "CanadianOrAdmin")).Succeeded), if succeed then show about link.
    2. Run and Check, Logout , you will not see the About Link and then login, now about link will appear

# 5 | Web API and Swagger

## a | Getting Started with Web API

1. Create a .Net Core Web API Application WebAPISample,
2. Run and Check in PostMan
3. Add Model class TicketItem in Proj
   1. Add Id, Concert, artist, Available Properties in this class
4. Below model class add Ticketcontext class derived from DbContext, write ctor,
   1. Add DbSet of TicketItem, TicketItems
5. In Startup class, in COnfigureServices
   1. Add dbcontext to use in Memory Db
6. Add Empty Web api Controller TickerCOntroller
   1. Write ctor pass ticket context and give it to local field.
   2. In ctor if ticket item count is Zero add a ticket.
7. Add Get all Method return list if tickets
   1. Add return \_context.TicketItems.AsNoTracking().ToList();
      1. AsNoTracking disables tracking, as we just want list of tickets, we donot intend to change them, makes more efficient.
   2. Add [HttpGet] attribute above method, this is explicitly telling api handliers inside MVC that we are handling HTTPGet verb with this method.
8. Run and check api/ticket in Postman
9. In Ticketcontroller Add Method Get By Id.
   1. Get ticket by id, if not found return NotFound() (404)
   2. If found return objectResult of ticket (200)
   3. Add Attribute [HttpGet("{Id}", Name ="GetTicket")]
      1. We set that we will have Id in Url, and Name is used in point thirteen.
10. Run and Check in Postman. /Ticket/1 (200) , /Ticket/2 (404)
11. In Ticketcontroller add Create Method.
    1. It takes Ticket item as input
    2. If null return BadRequest() (400)
    3. AddTicket, save changes.
    4. Add return CreatedAtRoute("GetTicket", new { id = ticket.Id }, ticket);
       1. First argument is Name of Route
       2. Second argument is anonomous object giving id
       3. Third Argument is actual object we are returning.
    5. Where does the ticket object come from…?
       1. It comes from body.
       2. Add
       3. [FromBody] Attribute to the parameter.
    6. Add Attribute [HttpPost] on method.
12. Run and Check. In postman.
    1. In body add new ticket in json, select post, and run. /api/Ticket
    2. Now ticket will be added.
    3. Now check GET. You will see it added.
13. Now Add Update Method
    1. It takes Id, Ticket ad Parameters
    2. Add [HttpPut("{id}")] attribute over the method.
    3. Do update and return new NoContentResult(); (204). This says, we are done. With no content to return.
14. Run and Check. In postman.
15. Now Add Delete Method
    1. It takes Id Parameters
    2. Add [HttpDelete("{id}")] attribute over the method.
    3. Do Delete and return new NoContentResult(); (204). This says, we are done. With no content to return.
16. Run and Check. In postman.

## b | Web APIs and Swagger

1. We donot have any documentation of our api, for others to use.
2. We can add Swagger capabilities.
   1. Swagger is a standard that we use to define our APIs, so that these RESTFUL endpoints can be found, discovered and executed easily with little documentation, and a helpful user interface to interact with it.
3. A Library called Swashbuckle, allows us to add Swagger, Swagger is not a Microsoft Library, it is a opensource found on GitHub.
4. Go to Nuget Package manager of our Prev Proj WebAPISample, Add Swashbuckle.AspNetCore to our Proj.
5. Swashbuckle is a service we will add. In Startup Classs.
   1. Add Service.
6. In Configure Method. We put it n Http Pipeline
   1. Add app.UseSwagger();
   2. Add app.UseSwaggerUI configure it with s.SwaggerEndpoint("v1/swagger.json", "Ticket API V1");, above is the path where swagger will be there.
7. Run and Check.
   1. Goto /swagger/v1/swagger.json, we will find json, shows all our apis.
   2. Goto /Swagger we have a UI, we can play with it, add ticket,
8. We have Docs on Mocrosoft docs.

# 5 | Publishing & Deployment – Azure & Docker

## a | Publishing & Deployment – Azure & Docker