ASP.NET Core Interview Questions & Answers

ASP.NET Core

By Jignesh Trivedi

An Author, Architect, and Microsoft MVP

DotNetTricks

ASP.NET Core Interview Questions & Answers

Release History

- Initial Release 1.0 - 30th Nov 2018
- Second Release 1.1 - 4th Jan 2019
- Third Release 2.0 - 20th Dec 2019
- Fourth Release 3.0 - 19th Mar 2021

About Dot Net Tricks

Dot Net Tricks is founded by Shailendra Chauhan (Microsoft MVP), in Jan 2010. Dot Net Tricks came into existence in the form of a blog post over various technologies including .NET, C#, SQL Server, ASP.NET, ASP.NET MVC, JavaScript, Angular, Node.js and Visual Studio, etc.

The company which is currently registered by a name of Dot Net Tricks Innovation Pvt. Ltd. came into shape in 2015. Dot Net Tricks website has an average footfall on the tune of 300k+ per month. The site has become a cornerstone when it comes to getting skilled-up on .NET technologies and we want to gain the same level of trust in other technologies. This is what we are striving for.

We have a very large number of trainees who have received training from our platforms and immediately got placement in some of the reputed firms testifying our claims of providing quality training. The website offers you a variety of free study material in the form of articles.

Unlimited Live Training Membership

Upgrade your skills set with hands-on real-time project-based training programs to build expertise on in-demand job skills and become industry competent. DotNetTricks Unlimited Live Training enables you to Become:

- **Full-stack JavaScript Developer** - JavaScript, Node.js, React, Angular
- **Full-stack .NET Developer** - .NET, MVC, ASP.NET Core, WebAPI
- **Cloud Engineer/Architect** - AWS, Microsoft Azure
- **Technical Architect** - Microservices, Design Patterns and Clean Architecture
- **DevOps Engineer** - DevOps, Docker and Kubernetes
- **Mobile Developer** - Xamarin, React Native

Learn more about Unlimited Live here: https://www.dotnettricks.com/membership

Self-Paced Training Membership

The most effective way to gain job-ready expertise for your career. Learn how to build highly scalable modern web applications & transform your coding skills. Learn to build projects and building expertise on .NET, JavaScript, Database, Cloud, DevOps, Docker, Front-end technologies and many more cutting edge technologies which are in industry demand today.

- 5,00+ hours of unlimited access to our premium content
- Real Hands-on Labs with integrated IDE
- Full access to training, study mode quizzes, and assignments
- Step-by-step learning with exclusive Learning Paths
- Become job-ready with interview prep sessions

Learn more about Self-paced training membership here: https://www.dotnettricks.com/plus-membership

Interview Q&A eBooks

Dot Net Tricks offer a wide range of eBooks on technical interviews Q&A. industry experts and coaches write all eBooks. These eBooks will help you to prepare yourself for your next job within a short time. We offer the eBooks in the following categories:

- .NET Development
- Front-end Development
- Cloud
- DevOps
- Programming Languages
- Database - SQL and NoSQL
- Mobile Development
- ML/AI and many more...

For other eBooks, do refer to https://www.dotnettricks.com/books

Corporate Training

Dot Net Tricks has a pool of mentors who help the corporation enhance their employment skills by changing the technology landscape. Dot Net Tricks offers customized training programs for new hires and experienced employees through online and classroom mode. As a trusted and resourceful training partner, Dot Net Tricks helps the corporation achieve success with its industry-leading instructional design and customer training initiatives.

Apart from these, we also provide on-demand boot camps and personalized project consultation.

For more details about Corporate Training, do refer to https://www.dotnettricks.com/corporate-training

Technical Recruiting

We provide a full technical staffing service, which suits our client's needs. Our specialized recruiters search worldwide to find highly skilled professionals that will fit our client's needs. If you are looking for a job change, do share your resume at hr@dotnettricks.com. Dot Net Tricks will help you to find your dream job in MNCs.

Join us today, learn to code, prepare yourself for interviews, and get hired!

Dedication

I would like to say many thanks to my mother Mrs Saryuben Trivedi and my wife Mrs Poorvi for their support. They deserve to have their name on the cover as much as I do for all their support made this possible. I would also like to say thanks to all my family members Girish Kumar Trivedi (father), Rakesh and Tejas (brother) for their continuous guidance and support to achieve my goals.

-Jignesh Trivedi

Introduction

What Where Author Qualification to Write This Book

Jignesh Trivedi is awarded as MVP by Microsoft for his exceptional contribution in Microsoft technologies under the category "Developer Technologies" for the year 2016, 2017, 2018 and 2019. He has more than 12 years of experience on Microsoft technologies such as ASP.NET, MVC, ASP.NET Core, TypeScript, Blazor, SQL server etc. and other technologies such as Angular and AngularJS, HTML, CSS, jQuery etc.

He is also a blogger and author of articles on various technologies. He is also a speaker and delivered talk on various technologies like ASP.NET Core, Angular, MVC etc. in public events.

What This Book Is

ASP.NET Core is an open-source, cross-platform framework for building web applications using C# and .NET. This book will teach you ASP.NET Core concepts from scratch to advance with the help of Interview Questions & Answers. Here, you will about the routing, tag helpers, middleware, session, dependency injection, authentication, authorization, unit testing and deployment. This book covers ASP.NET Core versions starting from 1.0 to 3.0.

What You'll Learn

This book is for .NET developers who are looking for a change or want to make a bright future in ASP.NET Core. This book covers the interview questions on the following topics:

- ASP.NET Core features details.
- How to configure middleware and routing.
- How to enable session and configure your application environment.
- A JS framework like a way to write server-side code to create and render the HTML using Tag Helper.
- How to reuse your code using Partial views and view component.
- Different ways to apply validations and binding data to server-side action methods.
- How to implement localization and globalization.
- Handle exceptions and logging errors.
- Dependency Injection implementation.
- Ways to secure your application and using filters.
- Using Unit testing frameworks and running tests.
- Various Deployment Options.

Our best wishes always with you for your learning and growth!

About the Author

Jignesh Trivedi - An Author, Architect, and Microsoft MVP

Jignesh Trivedi is working as a software developer with a leading organization and having more than 12 years of experience. He is very passionate about the Microsoft Technologies. He is author, speaker and MVP.

He has experience to develop enterprise application using Microsoft technologies such ASP.NET Core, C#, SQL server, etc. and other technologies such as Angular, node.js, etc.

He loves building great products and POC (proof of concepts) using the best available technologies. He loves to share his knowledge by contributing to Developer community.

He always tries to keep updated himself about new technologies and learning new skills and shared with other in simple manner. He hopes that this e-book helps you to crack interview of ASP.NET Core.

How to Contact Us

Although the author of this book has tried to make this book as accurate as it possible but if there is something that strikes you as odd, or you find an error in the book please drop a line via e-mail.

The e-mail addresses are listed as follows:

- mentor@dotnettricks.com
- info@dotnettricks.com

We are always happy to hear from our readers. Please provide your valuable feedback and comments!

You can follow us on YouTube, Facebook, Twitter, LinkedIn and Google Plus or subscribe to RSS feed.

Table of Contents

1

Introducing ASP.NET Core

Q1. What is the ASP.NET Core?

Ans. ASP.NET Core is not an upgraded version of ASP.NET. ASP.NET Core is completely rewriting that work with the .NET Core framework. It is much faster, configurable, modular, scalable, extensible and cross-platform support. It can work with both .NET Core and .net framework via .NET standard framework. It is best suitable for developing cloud-based such as web application, mobile application, IoT application.

Q2. What are the features provided by ASP.NET Core?

Ans. Following are the core features that are provided by the ASP.NET Core

- Built-in supports for Dependency Injection
- Built-in supports for the logging framework and it can be extensible
- Introduced new, fast and cross-platform web server - Kestrel. So, a web application can run without IIS, Apache and Nginx.
- Multiple hosting ways are supported
- It supports modularity, so the developer needs to include the module required by the application. However, .NET Core framework is also providing the meta package that includes the libraries
- Command-line supports to create, build and run the application
- There is no web.config file. We can store the custom configuration into an appsettings.json file
- There is no Global.asax file. We can now register and use the services in the startup class
- It has good support for asynchronous programming
- Support WebSocket and SignalR
- Protect against CSRF (Cross-Site Request Forgery)

Q3. What is Metapackage?

Ans. The framework .NET Core 2.0 introduced Metapackage that includes all the supported packages by ASP.NET code with their dependencies into one package. It helps us to do fast development as we don't require to include the individual ASP.NET Core packages. The assembly Microsoft.AspNetCore.All is a meta package provide by ASP.NET Core.

Q4. Can ASP.NET Core application work with full .NET 4.x Framework?

Ans. Yes. ASP.NET Core application works with full .NET framework via the .NET standard library.

Q5. Explain the advantages of ASP.NET Core over ASP.NET?

Ans. There are the following advantages of ASP.NET Core over ASP.NET:

- It is cross-platform, so it can be run on Windows, Linux and Mac.
- There is no dependency on framework installation because all the required dependencies are ship with our application
- ASP.NET Core can handle more request than the ASP.NET
- Multiple deployment options available withASP.NET Core

Q6. What is the use of the startup class in ASP.NET Core?

Ans. Startup class is an entry point of the ASP.NET Core application. Every .NET Core application must have this class. This class contains the application configuration related items. It is not necessary that the class's name must be "Startup", it can be anything. We can configure the startup class in the Program class.

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<TestClass>();
}
```

Q7. What is the use of ConfigureServices method of startup class?

Ans. ConfigureServices is an optional method of startup class. It can be used to configure the services that are used by the application. This method calls first when the application is requested for the first time. Using this method, we can add the services to the DI container, so services are available as a dependency in the controller constructor.

Q8. What is the use of the Configure method of startup class?

Ans. It defines how the application will respond to each HTTP request. We can configure the request pipeline by configuring the middleware. It accepts IApplicationBuilder as a parameter and also it has two optional parameters: IHostingEnvironment and ILoggerFactory. Using this method, we can configure built-in middleware such as routing, authentication, session, etc. as well as third-party middleware.

Q9. Can we configure the service and request pipeline without Startup class?

Ans. Yes, it can be achieved by defining ConfigureServices and Configure convenience methods on the host builder in the program class. If the multiple ConfigureServices method call exists, all methods are appended to one another. The last Configure method call is used when multiple Configure method calls exist.

```
public class Program
{
    public static void Main(string[] args)
    {
```

```
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                //webBuilder.UseStartup<Startup>();
                webBuilder
                .ConfigureServices(services =>
                {
                    //Configure Services
                })
                .Configure(app =>
                {
                    //configure the web application
                });
            });
}
```

Q10. What is new in ASP.NET Core 3.0 compared to Core 2.2?

Ans. Following features are introduced in ASP.NET Core 3.0

- Blazor: It is a new client framework for building interactive client-side web UI with .NET Core. You can create rich UI using C# code instead of JavaScript and provides almost all feature that SPA provides.
- New JSON serialization: The new JSON serialization is added under System.Text.Json namespace. It provides higher performance than Newtonsoft.Json.
- Two new Razor directives introduced: ASP.NET Core 3.0 introduced the following two new Razor directive.
    - @attribute: It applied the given attribute to the view or class generated the page
    - @implements: It is used to implements an interface for the generated class
- ASP.NET Core 3.0 IdentityServer4 supports authentication and authorization for SPA and API
- ASP.NET Core 3.0 template use .NET generic host (HostBuilder) instead of WebHostBuilder.
- .NET Core 3.0 introduces the new Worker Service app template that provides a starting point for writing long-running service
- It introduced a shared framework that automatically referenced when using the Microsoft.NET.Sdk.Web SDK in the project file
- Introduced pipes on HttpContext: It is possible to read the request body and write the response body using the System.IO.Pipelines API
- Health Checks use endpoint routing with the Generic Host

Following are the improvement in ASP.NET Core

- Template updated

    - Default web UI template update to removed cookie consent UI and remove references of CDN (instead of new template use local files)
    - Angular template update to support Angular 8
    - RCL (Razor class library) update to supports Razor component by default and new templates are added to supports pages and views
- Kestrel configuration has been updated for the migration to the Generic Host
- HTTP/2 protocol enable by default in Kestrel however for IIS and HTTP.sys, it is depending on the operating system
- Improved error reporting in IIS
- Improvements in performance
    - Reduced memory usage when using built-in DI container for scoped service
    - Reduced memory usage for WebSocket connection
    - Provides newly optimized and fully asynchronous JSON serializer
    - Reduced memory usage and throughput improvements in form parsing and for HTTPS connections
- Following assemblies removed from the ASP.NET Core shared framework
    - Newtonsoft.Json
    - Entity Framework Core

Q11. What is new in ASP.NET Core 5.0 compared to Core 3.1?

Ans. Following features are introduced in ASP.NET Core 5.0

- **ASP.NET Core MVC**
    - Model binding
        - Supports UTC time string to Datetime field
        - Supports recode type (introduced in C# 9.0)
    - Improvement in DynamicRouteValueTransformer(introduced in ASP.NET Core 3.1) - ASP.NET Core 5.0 can pass the state and filter the set of endpoints
    - The "Compare" attribute can be applied to properties on the Razor page model
    - The parameters and properties bound using "FromBody" attribute are "required" by default
- **Web API**
    - OpenAPI specification templates supported by default
    - Improvement in publishing flow with Azure API Management

- Web API template comes with pre-configured the Swagger UI page

- **Blazor**
  - Performance improvements in Blazor WebAssembly runtime
  - CSS isolation - Component specific CSS styles
  - Introduced new component: InputFile , InputRadio, InputRadioGroup
  - New Event supports: ontoggle, FocusAsync
  - Supports custom validation class attribute
  - The component supports IAsyncDisposable interface
  - DisplayName parameter support for InputDate, InputNumber and InputSelect component
  - Debugging improvements
  - Blazor WebAssembly prerendering
  - Lazy load assemblies
- **Authentication and authorization**
  - Microsoft.Identity.Web supports Azure Active Directory authentication
  - Allow anonymous access to an endpoint
  - Endpoint routing supports Authorization
  - Supports custom handling of authorization failures
- **Miscellaneous improvements**
  - Improvement in Startup class activation - pass additional parameters to Startup class which are initialized with host
  - Auto-refresh with dotnet watch
  - Improvement in Console logger formatter

Q12. Does ASP.NET Core support response compression?

Ans. Yes, ASP.NET Core supports response compression with Brotli compressed data format.

Q13. How can we do response compression in ASP.NET Core?

Ans. ASP.NET Core supports response compression with Brotli compressed data format. It provides the middleware for the same. To do response compression in ASP.NET Core, the middleware needs to inject in the response pipeline using AddResponseCompression method. We can also add one or more provides.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<BrotliCompressionProvider>();
        options.Providers.Add<GzipCompressionProvider>();
```

```
        options.MimeTypes =
            ResponseCompressionDefaults.MimeTypes.Concat(
                new[] { "image/svg+xml" });
    });
}
```

Q14. What is CoreCLR?

Ans. It is a .NET execution engine in .NET Core that performing various activity such as compilation to machine code, garbage collection, etc.

Q15. Where project static files are stored in ASP.NET Core?

Ans. All the static files located inside of wwwroot folder. It contains CSS, JS, images, fonts and other static content.

Q16. What is Swagger?

Ans. The Swagger provides the UI representation of the RESTful APIs without any implementation logic. It allows the users to understand the capabilities of a service without any code access. It is also known as OpenAPI. It helps us to test the Web API endpoint just like third party tool Postman and fiddler, however, it can be integrated with the codebase (can be added as middleware).

ASP.NET Core provides improved tooling and runtime experiences for creating swagger documents. Further information about swagger can be found here https://swagger.io/

2

Middleware and Routing

Q1. How to add middleware to the application request pipeline?

Ans. Using **IApplicationBuilder.Use, IApplicationBuilder.Map** and **IApplicationBuilder.Run** methods, we can add middleware delegate to the application request pipeline.

Q2. What is the difference between IApplicationBuilder.Use() and IApplicationBuilder.Run()?

Ans. We can use both the methods in Configure methods of startup class. Both are used to add middleware delegate to the application request pipeline. The middleware adds using IApplicationBuilder.Use may call the next middleware in the pipeline whereas the middleware adds using IApplicationBuilder. The run method never calls the subsequent ore next middleware. After IApplicationBuilder.Run method, system stop adding middleware in the request pipeline.

Q3. What is the use of "Map" extension while adding middleware to ASP.NET Core pipeline?

Ans. It is used for branching the pipeline. It branches the ASP.NET Core pipeline based on request path matching. If the request path starts with the given path, middleware on to that branch will execute.

```
public void Configure(IApplicationBuilder app)
{
    app.Map("/path1", Middleware1);
    app.Map("/path2", Middleware2);
}
```

Q4. What is the use of "MapWhen" extension while adding middleware to ASP.NET Core pipeline?

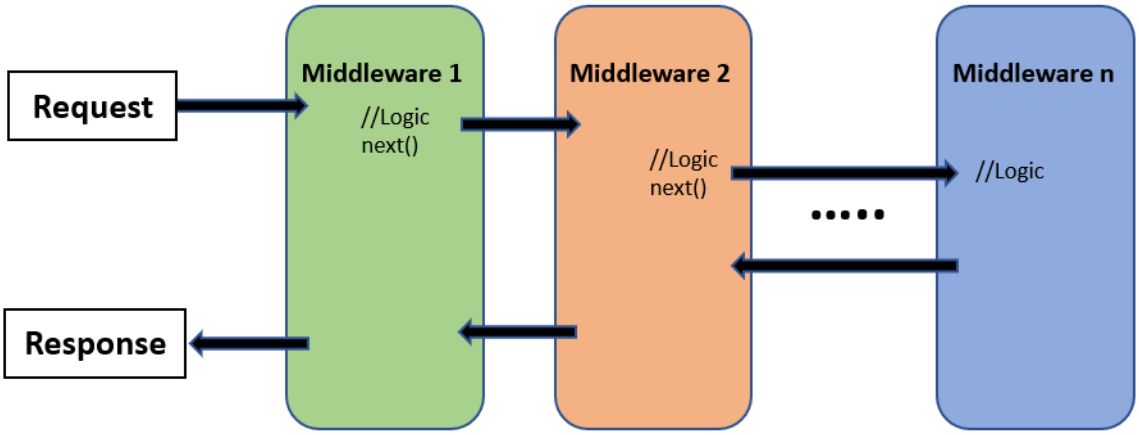
Ans. It is used for branching the pipeline. It branches the ASP.NET Core pipeline based on the result of the given predicate. we can use predicate of type Func<HttpContext, bool> to map request to the new branch. In the following example, middleware invoked if the request query string has the value "id".

```
public void Configure (IApplicationBuilder app)
{
```

```
    app.MapWhen(context => context.Request.Query.ContainsKey("id"),
                Middleware1);
}
```

Q5. What is middleware?

Ans. It is software which is injected into the application pipeline to handle request and responses. They are just chained to each other and form as a pipeline. The incoming requests are passes through this pipeline where all middleware is configured, and middleware can perform some action on the request before passes it to the next middleware. Same as for the responses, they are also passing through the middleware but in reverse order.



Q6. How can we configure the pipeline for middleware?

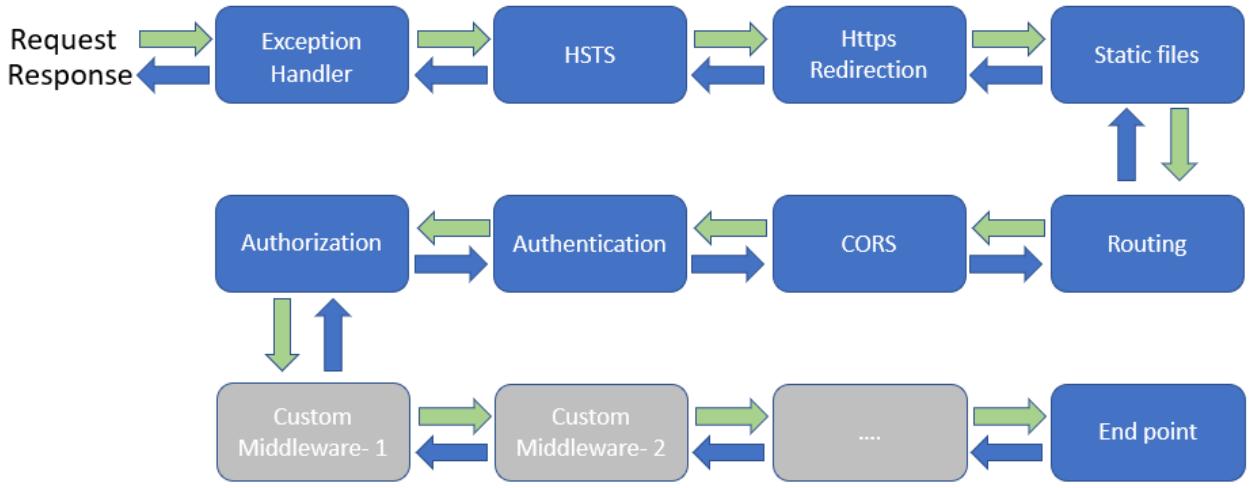
Ans. We can configure ASP.NET Core request/response pipeline using the Configure method of the Startup class by using the "Use*" methods. For example

```
public void Configure (IApplicationBuilder app)
{
   app.UseAuthentication();
   app.UseStaticFiles();
}
```

All the middleware(s) are added to the ASP.NET Core pipeline in the order they have been added.

Q7. Explain the order of request processing pipeline for ASP.NET Core MVC and Razor Pages apps

Ans. The following diagram explains the request/response processing pipeline for ASP.NET Core MVC and Razor Pages apps.



Q8. What is built-in middleware(s) available with ASP.NET Core?

Ans. Followings middleware ships with ASP.NET Core.

Middleware	Description	Method to invoke (IServiceCollection/IApplicationBuilder)
Authentication	Provide the authentication support	AddDefaultIdentity or AddIdentity/ UseAuthentication
Authorization	Provide the authorization support	AddAuthorization or AddAuthorizationCore or AddAuthorizationPolicyEvaluator / UseAuthorization
Cookie Policy	Used to configure the cookie policy	UseCookiePolicy
CORS	Configure Cross-origin resource sharing	AddCors /UseCors
Diagnostics	Configures diagnostics	UseDeveloperExceptionPage
Forwarded Headers	Forwards proxied headers onto the current request	UseForwaredHeaders
Health Check	Its check health for the ASP.NET Core app and its dependencies	AddHealthChecks / define endpoint for it using UseEndpoints
Header Propagation	Propagates HTTP headers from the incoming request to the outgoing HTTP Client requests.	
HTTP Method Override	Allows an incoming POST request to override the method	UseHttpMethodOverride
HTTPS Redirection	Redirect all HTTP request to HTTPS	UseHttpsRedirection
HTTP Strict Transport Security (HSTS)	It provides security enhancement (add special response header). It is available with only ASP.NET Core 2.1 or later.	UseHsts
MVC	Add MVC to request pipeline	AddMvc AddControllers AddRazorPages AddControllersWithViews / UseMvc

OWIN	Add Owin middleware for authentication	UseOwin
Response Caching	Provide support to cache the response	AddResponseCaching / UseResponseCaching
Response Compression	Enable compressing responses	AddResponseCompression /UseResponseCompression
Request Localization	Provide support for localization	AddLocalization/ UseRequestLocalization
Endpoint Routing	Define route and constrain	UseEndpoints
Session	Add session middleware to request pipeline	AddSession/UseSession
Static Files	Add support for accessing static files and directory browsing	UseStaticFiles
URL Rewriting	Provide URL rewriting and redirect the request	UseRewriter
Web Sockets	Enable the support for web socket	UseWebSockets

Note: you can learn more about the middleware describe above from the following link.
https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-5.0#built-in-middleware

Q9. How can you write custom middleware?

Ans. Yes, we can write our middleware. Generally, middleware is encapsulated in a class and extension method used to expose to the outer world. There are three easy steps to create middleware

Step 1: Creating a middleware class

```
public class MyMiddleware
{
    private readonly RequestDelegate _next;

    public MyMiddleware(RequestDelegate next)
    {
        _next = next;
    }
    public async Task InvokeAsync(HttpContext context)
    {
        //Write your code here

        // invoke the next middleware in the pipeline
        await _next(context);
    }
}
```

Step 2: Create an extension of IApplicationBuilder that expose our middleware

```
public static class MyMiddlewareExtensions
{
    public static IApplicationBuilder UseMyMiddleware(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<MyMiddleware>();
```

```
    }
}
```

Step 3: register middleware to an application using startup class Configure method

```
public class Startup
{
    public void Configure (IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseMyMiddleware();
    }
}
```

Q10. What is the default order of invoking middleware in the request pipeline?

Ans. The order that middleware components are added in the Startup.Configure method defines the order in which the middleware components are invoked in the request pipeline and reverse order for the response.

Q11. Does Static File Middleware compress the static files?

Ans. No.

Q12. What is the difference between Middleware and HttpModule

Ans. Following are the difference between Middleware and HttpModule

Middleware	HttpModule
It is independent of application events	It attaches to application events
Developers have full control over the middleware what executed and what order	The developer does not have any control on HttpModule order of execution
It configured via code in ConfigureServices method of the startup class	It configured using web.config or global.asax code
It hosts independently	It tied to System.Web

Q13. What is routing in the ASP.NET Core?

Ans. Routing is functionality that map incoming request to the route handler. The route can have values (extract them from URL) that used to process the request. Using the route, routing can find route handler based on URL. All the routes are registered when the application is started. There are two types of routing supported by ASP.NET Core

- The conventional routing
- Attribute routing

The Routing uses routes to map incoming request with the route handler and Generate URL that used in response. Mostly, the application having a single collection of routes and this collection are used for the process of the request. The RouteAsync method is used to map incoming request (that match the URL) with available in route collection.

Q14. How can we configure conventional routing?

Ans. The conventional routing can be configured by setting up routing middleware in the Configure method of the startup class.

```
app.UseMvc(routes =>
{
  routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

In the above example, the default route is added. It can match URL path such as "Employee/Details/10" and MVC will extract the route values: controller as an employee, action as Details and id as 10.
The ASP.NET Core MVC can also add the above conventional route using the convenience method UseMvcWithDefaultRoute.

```
app.UseMvcWithDefaultRoute();
```

We can add the route as many as we want.

Q15. What's happened MVC find two disambiguating actions through routing?

Ans. When MVC find two disambiguating actions, MVC chooses the best match or else throw an exception.

Q16. What is Attribute routing?

Ans. ASP.NET Core MVC provides a set of attributes to map actions directly with route templates. When we use attribute routing, we need not required any configuration i.e. we can call app.UseMvc() method in Configure method of startup class without pass route delegate.

Using Route attribute, we can achieve attribute routing.

```
[Route("")]
[Route("Home")]
[Route("Home/Index")]
public IActionResult Index()
{
   return View()
}
```

The above defined action method will be called for URL path "/","/Home", "/Home/Index".
When we define the route attribute on the controller, they are combined with the route defined on the action method.

Q17. Does ASP.NET Core support mix routing i.e. both conventional and attribute routing together?

Ans. Yes, it supports mixed routing, but the action defined with attribute routes cannot reach through conventional routes and vice-versa. If we mark the controller with attribute route, all actions under this controller are now attribute routed.

Q18. How can we define a route for areas?

Ans. The areas help us to organize related functionality into a physical group. For conventional route, MVC provides a method “MapAreaRoute" to define the route for areas.

```
app.UseMvc(routes =>
{
    routes.MapAreaRoute("default_route", "AreaName", "AreaName/{controller}/{action}/{id?}");
});
```

For attribute route, MVC provides an "Area" attribute to define an area for the route.
Example

```
[Area("User")]
public class HomeController : Controller
{
    ....
    ....
}
```

Q19. How attribute routing work with Http[Verb] attribute?

Ans. Attribute routing also uses Http[Verb] attributes to find out the correct route. If two or more action having the same route template defined than it uses the request HTTP verb to identify the correct route. For example, the following two action methods having the same route template but different HTTP verb (get and post) defined. If URL path like /employee and HTTP verb is “get” than ListEmployee method called and same as If URL path like /employee and HTTP verb is POST than CreateEmployee method called. The default HTTP verb for the method is GET.

```
[HttpGet("/employee")]
public IActionResult ListEmployee()
{
    // ...
}

[HttpPost("/employee")]
public IActionResult CreateEmployee(...)
{
    // ...
}
```

Q20. What are different HTTP verb templates supported by ASP.NET Core?

Ans. ASP.NET Core has the following HTTP verb templates:

- HttpGet

- HttpPost
- HttpPut
- HttpDelete
- HttpHead
- HttpPatch

Q21. What is route constraint and what is the use of it?

Ans. Route constraints match parameter type (associated in route template) with route value. If it does not match, the system will return with a 404 (not found) HTTP status code. There are many built-in route constraints provided by ASP.NET Core framework and you can also define custom route constraint. It helps you to reduce burden validating type in the action method.

Syntax:

```
[Route(URLPath/{parameterName: constrain (type)})]
```

Q22. Is it recommended to use route constraints for input validation?

Ans. It is not recommended to use route constraint for input validation. The system will return 404 (not found) HTTP status when the parameter value does not match with the route constraint. In the case of input validation, the system might accept 400 (bad request) HTTP status code.

Q23. What are built-in constraints supported by ASP.NET Core MVC?

Ans. Following are built-in route constraint supported by ASP.NET Core

Constraint	Description	Example
int	Allow only integer value	{id:int}
bool	Allow only boolean value (true and false)	{isActive:bool}
datetime	Allow only datetime value ((in the invariant culture)	{doj:datetime}
decimal	Allow only decimal value	{price:decimal}
double	Allow only double value	{heigh:double}
float	Allow only float value	{weight:float}
guid	Allow only guid value	{id:guid}
long	Allow only long value	{id:guid}
minlength(value)	String must contain at least x character	{password: minlength(8)}
maxlength(value)	String must not contain more than x characters	{password: maxlength(16)}
length(length)	String must contain exactly x characters	{empCode:length(8)}
length(min,max)	String must contain at least x character and not more than y character	{password:length(8,16)}
min(value)	Minimum int value must be x	{age:min(18)}
max(value)	Maximum int value must be not more than y	{age:max(60)}
range(min,max)	Int value must be in between x and y	{age:range(18,60)}

alpha	String must contain one or more alphabat character	{empCode:alpha}
regex(expression)	String must match to regular expression	{ empCode:^[a-zA-Z0-9_]$}
required	It enforces that parameter must have value	{name:required}

Q24. Can we define multiple route constraints for a route parameter?

Ans. Yes, multiple route constraint can be defined for route parameter using a colon (:) delimited. In the following example, the age parameter must be an integer value of 8 or greater.

```
[Route("test/{age:int:min(8)}")]
public void TestMethod(int age) { }
```

Q25. Can we create custom route constraints?

Ans. Yes, we can create custom route constraints by implementing the IRouteConstraint interface. This interface contains a single method, "Match" that returns true if the constraint is satisfied and otherwise false.

```
public class CustomRouteConstraint : IRouteConstraint
{
    public bool Match(HttpContext httpContext, IRouter route, string routeKey,
RouteValueDictionary values, RouteDirection routeDirection)
    {
        return true;
    }
}
```

After that custom route constraint registered with the app's ConstraintMap in the app's service container.

```
services.AddRouting(options =>
{
    options.ConstraintMap.Add("consname", typeof(CustomRouteConstraint));
});
```

After the registration, we can use it.

```
[HttpGet("{id:consname}")]
public IActionResult Index(int id)
{
}
```

Q26. Can we use ASP.NET Core reserved keywords as route names?

Ans. No, we cannot use reserved keywords as a route name or parameter name. These reserved keywords are action, area, controller, handler, and page.

Q27. What are the different places where we can define routing in ASP.NET Core?

Ans. Apps can configure routing using

- Controllers
- Razor pages

- SignalR
- gRPC services
- Endpoint enables middleware (example- Health checks)
- Lambdas and delegates registered with routing

Q28. What is endpoint routing?

Ans. The endpoint routing is a system that handles routing across different middleware such as MVC, Razor pages, gRPC, Blazor. Instead of the "AddMvc" middleware, you can use the "AddControllerWithView" middleware to get views and use the "AddController" middleware for adding API routing. It allows you to decouple the route matching logic from the MVC middleware by moving its own middleware. This endpoint middleware pass endpoint information to other middleware. For example, when endpoint information arrives at MVC middleware, controller and action determined using this information instead of URL.

Q29. How endpoint routing differs from earlier versions of routing?

Ans. There are the following differences from earlier versions.

- The endpoint routing does not use IRouter or Route based extensibility
- It does not support Microsoft.AspNetCore.Mvc.WebApiCompatShim that provides a compatibility shim to move ASP.NET 4.x Web API projects to ASP.NET Core
- It has different behaviour for the casing of generated URIs with conventional routes
  Consider following the default route

```
app.UseMvc(routes =>
{
      routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

  Following code is used to generate a link

```
var link = Url.Action("EmployeeList", "employee", new { id = 18 });
```

  When using IRouter-based routing, it generates URL: “/employee/EmployeeList/18”
  With Endpoint routing, it generates URL: “/Employee/EmployeeList/18” (Employee is capitalized)
- Endpoint routing generates an empty string when the controller and/or action method does not exist.

Q30. Can we define multiple routes for single action?

Ans. Yes, we can define multiple routes for a single action method by decorated with multiple route attribute. In the following example, two routes are defined for a single action.

```
[Route("test")]
[Route("test1")]
public IActionResult Index()
{
}
```

Q31. Can we have mixed routing (conventional and attribute routing) into one app?

Ans. Yes, ASP.NET Core app can mix the use of both conventional and attribute routing into one app. Typical use case of this to use the conventional route for the controller that serves HTML page and use attribute routing for serving REST APIs.

Q32. Does routing support token replacement in route templates

Ans. Yes, attribute and convention routes support token replacement by enclosing token with square brackets []. It supports "area", "controller" and "action" as a token value that replaced with actual area name, controller name or action name.

```
[Route("[controller]/[action]")]
public class HomeController : Controller
{
     public IActionResult Index()
     {

     }
}
```

Q33. What is the use of the NonAction attribute?

Ans. This attribute used to indicate, a controller's method is not an action method.

Q34. What is dynamic routing?

Ans. Dynamic routing is a new feature in Endpoint routing. It allows an application to decide the route at runtime. In other words, you can set endpoints (select controller or page) programmatically based on condition.

Q35. How can we achieve dynamic routing in ASP.NET Core MVC?

Ans. ASP.NET Core MVC provides DynamicRouteValueTransformer class that provides a way for dynamically manipulating route value to select controller action or page. It needs to use with MapDynamicControllerRoute<TTransformer> or MapDynamicPageRoute to implement custom logic.

Following are the steps to achieve dynamic routing based on the language selected

Step 1: Create a class and inherit it from DynamicRouteValueTransformer

```
public class TranslationTransformerExample : DynamicRouteValueTransformer
{
    private readonly TranslationUrl _translationUrl;

    public TranslationTransformerExample(TranslationUrl translationUrl)
    {
        _translationUrl = translationUrl;
    }

    public override async ValueTask<RouteValueDictionary> TransformAsync(HttpContext
httpContext, RouteValueDictionary values)
    {
```

```
        if (!values.ContainsKey("language") || !values.ContainsKey("controller") ||
!values.ContainsKey("action")) return values;

        var language = (string)values["language"];
        var controller = await _translationUrl.GetTranslationLanguageName(language,
(string)values["controller"]);
        if (controller == null)
            return values;
        values["controller"] = controller;

        var action = await _translationUrl.GetTranslationLanguageName(language,
(string)values["action"]);
        if (action == null)
            return values;
        values["action"] = action;

        return values;
    }
}

public class TranslationUrl
{
    private static Dictionary<string, Dictionary<string, string>> translationLanguageList =
new Dictionary<string, Dictionary<string, string>>
    {
        {
            "en", new Dictionary<string, string>
            {
                { "customer", "customer" },
                { "department", "department" },
                { "orders", "orders"}
            }
        },
        {
            "de", new Dictionary<string, string>
            {
                { "kundin", "customer" },
                { "abteilung", "department" },
                { "bestellungen", "orders"}
            }
        }
    };
    public async Task<string> GetTranslationLanguageName(string languageCode, string value)
    {
        var normalizedLang = languageCode.ToLowerInvariant();
        var normalizedValue = value.ToLowerInvariant();
        if (translationLanguageList.ContainsKey(normalizedLang) &&
translationLanguageList[normalizedLang].ContainsKey(normalizedValue))
        {
            return translationLanguageList[normalizedLang][normalizedValue];
        }

        return null;
    }
}
```

Step 3: Register endpoint with our custom DynamicRouteValueTransformer

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
          endpoints.MapDynamicControllerRoute<TranslationTransformerExample>
("{language}/{controller}/{action}");
     }
```

This code support following router definition (based on language)

- for English – /en/customer/orders
- for German – /de/kundin/bestellungen

Q36. What is the use of RequireHost or [Host] parameter?

Ans. The RequireHost or [Host] parameter applies a constraint to the route for a specific host. It can match the hostname, host with the wild card (subhost or subdomain) and host with a specific port. The RequireHost parameter can be defined with the endpoint route in the Configure method of Startup class. [Host] attribute can be applied on Controller or Controller's action method.

RequireHost Example

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseRouting();
    app.UseEndpoints(endpoints =>
    {
          endpoints.MapGet("/", context => context.Response.WriteAsync("Hi exmple!"))
            .RequireHost("example.com");
     }
}
```

[Host] Example

```
[Host("example.com")]
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return ControllerContext.MyDisplayRouteInfo();
    }

    [Host("example.com:8085")]
    public IActionResult AccessPrivately()
    {
        return ControllerContext.MyDisplayRouteInfo();
    }
}
```

3

Session and Environment

Q1. How can we enable Session in ASP.NET Core?

Ans. The middleware for the session is provided by the package Microsoft.AspNetCore.Session. To use the session in the ASP.NET Core application, we need to add this package to .csproj file and add the Session middleware to the ASP.NET Core request pipeline.

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSession();
        services.AddMvc();
    }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
       ….
        app.UseSession();
       ….
    }
}
```

Q2. How many types of session state is supported by ASP.NET Core?

Ans. There is two types of state available: InProc or in-memory and OutProc. For the OutProc session, there are many options available such as SQL Server, Redis server, etc.

Q3. How to access the session in ASP.NET Core application?

Ans. Using the HttpContext class, we can access the session in ASP.NET Core application. This class is defined under Microsoft.AspNetCore.Http namespace. There are three methods available to set the value to the session

- Set: It accepts byte array as argument and store byte array to a session
- SetInt32: It is an extension method that accepts the int. Internally, it converts an integer value to a byte array and call set method
- SetString: It is an extension method that accepts the string. Internally, it converts a string value to a byte array and call set method

Similarly, there are three methods to get the session value. All the method accepts the key name as a parameter.

- Get: It returns the byte array
- GetInt32: It returns the integer value
- GetString: It returns the string value

Example:

```
string name = "Jignesh Trivedi";
HttpContext.Session.Set("MyName" ,Encoding.Default.GetBytes(name));
string name1 = Encoding.Default.GetString(HttpContext.Session.Get("MyName"));
```

Q4. Why Session is stored in the form of a byte array in ASP.NET Core?

Ans. The main reason to stored bytes array in session to make sure that values are serializable and can be stored on the remote server.

Q5. Is there any sequence to call UseSession() method in the Configure method?

OR

Is session being enable if UseSession() method call last in Configure method of startup class?

Ans. There is no such sequence to call UseSession method in Configure method of startup class however this method must call before the UseMvc method.

Q6. Can we store the Complex data in the session?

Ans. Yes, we can store the complex data in the session. The idea behind storing complex data is to serialize the complex object and convert it into a string and store it in the session.

Example

```
public class User
{
    public string Name { get; set; }
    public int Id { get; set; }
}
public class HomeController : Controller
{
    public IActionResult TestMethod()
    {
        User u = new User() { Id = 1, Name = "Jignesh Trivedi" };
        HttpContext.Session.SetString("MyData", JsonConvert.SerializeObject(u));
        return View();
    }
}
```

Q7. How does ASP.NET Core track user session?

Ans. The session uses a cookie to identify and track request from a single browser. The default cookie name is ".AspNetCore.Session" and use a path of "/" as a domain is not specified with default cookies.

Q8. Can we change the name of the cookie used for the session?

Ans. Yes, we can change the name of the cookie used for session tracking by specifying SessionOptions in ConfigureServices method of the startup class. We can also change another default setting for cookie using SessionOptions.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSession(options =>
    {
        options.Cookie.Name = ".myAppName.Session";
        options.IdleTimeout = TimeSpan.FromSeconds(10);
        options.Cookie.IsEssential = true;
    });
}
```

Q9. How can we maintain the app state in ASP.NET Core?

Ans. Following are the way to maintain the app state in ASP.NET Core

- Session
- Cookies
- TempData
- Query String
- Hidden filed
- Cache
- By injecting singleton service

Q10. How to use TempData with ASP.NET Core?

Ans. Using the “TempData” attribute, we can use TempData with ASP.NET Core.

```
public class MyPageModel : PageModel
{
  [TempData]
  public string TestMessage { get; set; }
}
```

Q11. What is the lifetime for TempData?

Ans. At the end of the next request, TempData will be deleted.

Q12. Is there any way to keep or preserve TempData value to multiple requests?

Ans. Using the "Peek" and "Keep" methods, we can keep or preserve TempData value to the next request but to alive for multiple requests, we have to either peek or keep the method in every request.

Q13. How can we set an environment variable?

Ans. There are multiple ways to set the environment variable as given below:

- Using Operating System: Operating System is also providing a way to set up an environment variable. The environment variable is case-sensitive for macOS and Linux whereas it is case-insensitive for windows 10.
- Using command prompt or Windows PowerShell command: using Set and $env command, we can set environment variable
- Using Visual Studio: We can set environment variable using VS by specified in our project's debug profiles.
- The defined value in the launchSettings.json file

Q14. Do the keys for the environment variable are case-sensitive?

Ans. It is case-sensitive for macOS and Linux whereas it is case-insensitive for windows 10.

Q15. How determine the value of an environment variable programmatically?

Ans. Using Environment.GetEnvironmentVariable method, we can get the value of the environment variable programmatically.

```
string value = System.Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");
```

Q16. What are the various JSON files available in ASP.NET Core?

Ans. There are the following JSON files in ASP.NET Core:

- global.json
- launchsettings.json
- appsettings.json
- bundleconfig.json
- bower.json
- package.json

Q17. What is the use of “launchsettings.json” file?

Ans. This JSON file contains the project-specific settings with each Visual Studio profile configured to launch the application. In this file, we can also specify the version of the framework used by the application for compilation and debugging. It also contains information about environment variables that can be used.

Q18. What is the use of "appsettings.json" file?

Ans. This file is used to define application related settings such as connection string, any other custom settings. In ASP.NET, we define such settings into the web.config file.

Q19. What is the use of "bower.json" file?

Bower is the package manager for the web. So, this file holds the reference of the dependencies. The bower.json file holds the reference of dependencies that are provided by the Bower package manager.

4

Areas and Tag Helpers

Q1. What is Area in ASP.NET Core?

Ans. The area has allowed us to organize related functionality into a group. It provides the physical grouping and every group have their own set of Razor pages, model, view and controller. It allows us to define the same controller name in different areas.

Q2. Do nested areas are supported by ASP.NET Core?

Ans. No. Currently, nested areas are supported by the ASP.NET Core directly but it can be achieved by implement IViewLocationExpander. This interface helps us to modify view locations that search by the view engine.

Q3. How to generate links for controller action under the area?

Ans. To identify the controller under view, we need to pass the area name in the route argument.

Example:

```
@Html.ActionLink("Test Area", "Index", "Home", new { area = "Account" })
```

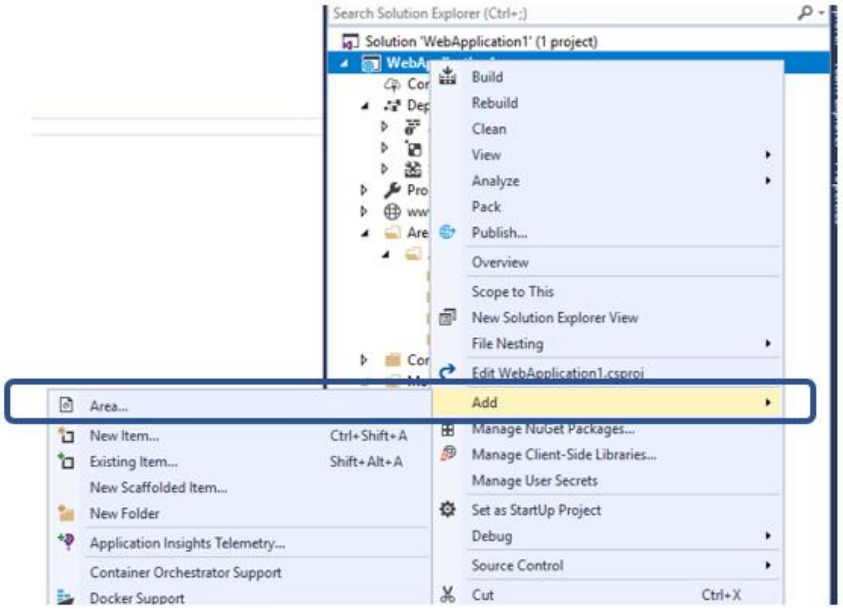
Q4. How to generate links for controller action under the area using tag helper?

Ans. The ASP.NET Core provides built-in tag helper to define action, controller and area. The "asp-controller" tag helper uses to define controller, "asp-action" tag helper uses to define action method and "asp-area" tag helper use to define area.

```
<a asp-area="Products" asp-controller="Item" asp-action="List">
    Product Item List
</a>
```

Q5. How can we add Area to ASP.NET Core application?

We can create Area by right click on the solution, select Add >> Area option. This creates a new area folder and under this folder Model, view, controller folder is created.



Q6. How to associate the controller with an Area?

Ans. Controller class decorated with the "Area" attribute to associate it with Area.

```
[Area("Test")]
public class TestController : Controller
{
...
...
}
```

Q7. Can we change the folder name "Areas" to any other name?

Ans. Yes, we can change the folder name "Areas" to any other name. Using the area name, ASP.NET Core set the default location where razor views are stored. If we change "Areas" to any other name, we must specify an area view location path. The following code changes the default area folder from "Areas" to "TestAreas".

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    ...
    services.Configure<RazorViewEngineOptions>(options =>
    {
        options.AreaViewLocationFormats.Clear();
        options.AreaViewLocationFormats.Add("/TestAreas/{2}/Views/{1}/{0}.cshtml");
        options.AreaViewLocationFormats.Add("/TestAreas/{2}/Views/Shared/{0}.cshtml");
        options.AreaViewLocationFormats.Add("/Views/Shared/{0}.cshtml");
    });
    ...
}
```

Q8. Explain about tag helper in ASP.NET Core?

Ans. It is a feature provided by the Razor view engine that enables us to write server-side code to create and render the HTML element in view (Razor). The tag-helper is C# classes that used to generate the view by adding the HTML element. The functionality of the tag helper is very similar to the HTML helper of ASP.NET MVC.

```
Example:
//HTML Helper
@Html.TextBoxFor(model => model.FirstName, new { @class = "form-control", placeholder = "Enter Your First Name" })

//content with tag helper
<input asp-for="FirstName" placeholder="Enter Your First Name" class="form-control" />

//Equivalent HTML
<input placeholder="Enter Your First Name" class="form-control" id="FirstName" name="FirstName" value="" type="text">
```

Q9. What are the advantages of tag helper?

Ans. There are the following advantages of tag helpers:

- The tag helper looks the same as the HTML attribute or directive, so any who working with UI can easily understand and edit razor view.
- They are provided More robust, reliable and maintainable code
- It executes the metadata such as DisplayName that using Data Annotation in the view model

Q10. What is the difference between HTML helper and Tag helper?

Ans. The functionality of the HTML helper and tag helper is nearly same but there are the following differences:

HTML Helper	Tag Helper
HTML helper starts with @ symbol that indication of staring of code	There is no such symbol or representation in tag helper
The Html helper used the anonymous object to represent the HTML attribute such as class, id, etc.	Tag helper is used as an HTML attribute, no such anonymous object is required
They are very difficult to understand and maintain	They are very much clear so easy to understand and maintain. The HTML code is more readable as compare to the HTML helper
The IntelliSense is very difficult with HTML helper	Visual Studio provides very rich IntelliSense for tag helper

Q11. How to add supports for Tag Helper to Razor view?

Ans. Using directive @addTagHelper define in _ViewImports.cshtml file, we can add the tag helper into the Razor. The _ViewImports.cshtml file used to provide the namespace that can be used in all the views.

Syntax:

```
@addTagHelper {name of the tag helper} {assembly name of the tag helper}
```

Example: _ViewImports.cshtml

```
@using TagHelper
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

In the above example, we have to use the wildcard character (*). It means that it adds the tag helpers define in "Microsoft.AspNetCore.Mvc.TagHelpers" assembly to the Razor view.

Q12. Can we disable Tag Helper at the element level?

Ans. We can disable Tag Helper at the element level using the opt-out character ("!"). This character must apply the opening and closing Html tags.

Example

```
<!span asp-validation-for="phone" class="divPhone"></!span>
```

Q13. Can we specify the prefix for tag helper?

Ans. Using @tagHelperPrefix, we can specify the tag prefix. This tag prefix also needs to specify into _ViewImports.cshtml file.

Example:

Views/_ViewImports.cshtml

```
@tagHelperPrefix th:
```

index.cshtml

```
<div class="form-group">
   <th:label asp-for="Firstname" class="col-md-6"></th:label>
</div>
```

Q14. Can we enable directory browsing through the code in ASP.NET Core?

Ans. Yes. We can enable directory browsing in ASP.NET Core using directory browsing middleware. We can invoke the middleware using UseDirectoryBrowser method in configure method of the startup class.

Example:

```
public class Startup
{
        public void ConfigureServices(IServiceCollection services)
        {
                ...
                ...
                services.AddDirectoryBrowser();
                ...
        }
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
                ...
                ...
```

```
                app.UseDirectoryBrowser(new DirectoryBrowserOptions
                {
                        FileProvider = new
PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
                        RequestPath = "/ProjectImages"
                });
                ...
                ...
        }
}
```

Q15. What is the use of "UseFileServer" in the Configure method of the startup class?

Ans. It is combination of UseDefaultFiles, UseStaticFiles and UseDirectoryBrowser. The "UseDefaultFiles" method enables the website to serve a default page without providing a qualified URI. The "UseStaticFiles" method enables the website to server static files stored in the project default directory. The default directory is "<content_root>/wwwroot", but we can change it via UseWebRoot method. The UseDefaultFiles method must be called before UseStaticFiles method.

Q16. What are built-in tag helpers provided with ASP.NET Core?

Ans. Following are Built-in ASP.NET Core Tag Helpers

Tag helper name	Description
Anchor Tag Helper	It generates standard HTML anchor (<a ... >) tag
Cache Tag Helper	It provides the ability to improve the performance of ASP.NET Core app by caching app content
Component Tag Helper	It is used to render a component from a page or view. (only available with Blazor)
Distributed Cache Tag Helper	It is very similar to Cache Tag Helper
Environment Tag Helper	It conditionally renders the content based on the current hosting environment
Form Tag Helper	It generates a form attribute value for the ASP.NET MVC Controller
FormAction Tag Helper	It generates a form action attribute and controls where a form submits its data
Form Tag Helper	It generates the HTML Form element with hidden Request Verification Token
Image Tag Helper	It generates <img> tag and provides cache-busting behaviour for static image files
Input Tag Helper	It generates HTML <input> element and binds model expression to input
Label Tag Helper	It generates an HTML label element
LinkTagHelper	It allows us to specify a CDN and fallback for the CSS file
ScriptTagHelper	It allows us to specify a CDN and fallback for the script file
Partial Tag Helper	It is used for rendering a partial view in Razor Pages
Select Tag Helper	It generates HTML <select> element and associate options element
Textarea Tag Helper	It generates HTML <textarea> element
Validation Message Tag Helper	It adds the HTML5 data-valmsg-for="property" attribute to the span element that attached with validation error messages on the input

Validation Summary Tag Helper	It generates element with asp-validation-summary attribute and list down all validation messages for the form

Q17. How can we add a custom tag helper in ASP.NET Core MVC?

Ans. Following are the step to create a custom tag helper in ASP.NET Core.

Step 1: Define Tag helper class. Using the HtmlTargetElement attribute, we can provide a tag helper target. It passes an attribute parameter that specifies the HTML element which contains an HTML attribute named.

```
[HtmlTargetElement("my-first-tag-helper")]
public class MyCustomTagHelper : TagHelper
{
   public string Name { get; set; }
   public override void Process(TagHelperContext context, TagHelperOutput output)
   {
      output.TagName = "CustumTagHelper";
      output.TagMode = TagMode.StartTagAndEndTag;

      var sb = new StringBuilder();
      sb.AppendFormat("<span>Hi! {0}</span>", this.Name);

      output.PreContent.SetHtmlContent(sb.ToString());
   }
}
```

Step 2: Add entry to _ViewImports.cshtml

```
@addTagHelper *, CustomTagHelper
```

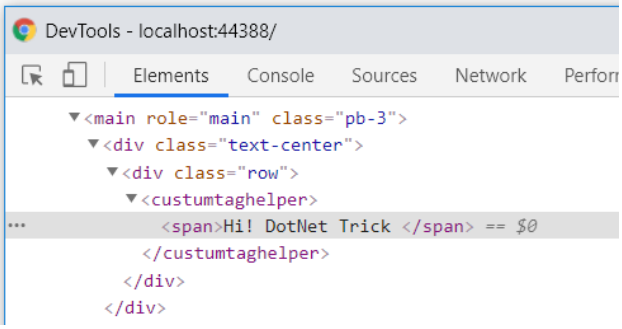
Use custom tag helper to Razor page

```
<div class="row">
   <my-first-tag-helper name="DotNet Trick ">

   </my-first-tag-helper>
</div>
```

Output:

Hi! DotNet Trick



5

Razor Pages and View Component

Q1. What are the Razor Pages in ASP.NET Core?

Ans. This is a new feature introduced in ASP.NET Core 2.0. It follows a page-centric development model just like ASP.NET web forms. It supports all the feature of ASP.NET Core.

Example:

```
@page
<h1> Hello, Book Reader!</h1>
<h2> This is Razor Pages </h2>
```

The Razor pages start with the @page directive. This directive handle request directly without passing through the controller. The Razor pages may have a code-behind file, but it is not really a code-behind file. It is a class inherited from PageModel class.

Q2. How can we prevent Razor pages from XSRF/CSRF attack?

Ans. We do not require to write any code to prevent Razor pages from XSRF/CSRF attack. The Antiforgery token generation and validation are done automatically in Razor Pages.

Q3. How can we enable Razor pages for ASP.NET Core app?

Ans. The Razor pages are enabled by adding middleware in ConfigureServices method of the Startup class.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    ...
    services.AddRazorPages();
    ...
}
```

Q4. What is the handler method in Razor pages?

Ans. It is the methods that automatically executed as a result of a request. The Razor pages support naming conventions to select the appropriate handler method to execute. It is On{Verb}[Async], where {Verb} is the HTTP method (Get and post), and [Async] is optional.

Q5. What are the handler methods available with Razor pages?

Ans. Following handler methods provided

- OnPost and OnPostAsync run in response to POST requests
- OnDelete and OnDeleteAsync run in response to DELETE requests
- OnGet and OnGetAsync run in response to GET requests
- OnHead run for HEAD requests when the handler is created

Q6. How can we do an automatic model binding in Razor pages?

Ans. The Razor pages provide the option to bind property automatically when posted the data using BindProperty attribute. By default, it only binds the properties only with non-GET verbs. we need to set SupportsGet property to true to bind a property on getting request.

Example:

```
public class Test1Model : PageModel
{
   [BindProperty]
   public string Name { get; set; }
}
```

Q7. How can we be binding the route data in Razor pages?

Ans. The route data used with Razor pages can be defined with the @Page directive. We can also define route parameter constrain such as the type of route data.

Example:

```
@page "{id}"
```

Example: Route with constrain (allow only int data for id)

```
@page "{id:int}"
```

Route parameter values store in RouteValueDictionary and we can access them using RouteData.Values property.

Example:

```
@RouteData.Values["id"]
```

Q8. What is the use of ViewData attribute in Razor pages?

Ans. This attribute is introduced in ASP.NET Core 2.1. Any Razor page model property decorated with ViewData attribute, their value is stored in ViewDataDictionary with property name as the key, so this property can be access in view via the Model property or the ViewData dictionary.

Example

```
[ViewData]
public string MyData { get; set; } = "This is Test";
```

In view

```
<h2>@Model.MyData</h2>
<h2>@ViewData["MyData"]</h2>
```

Q9. How can we apply filters in Razor pages?

Ans. There are two approaches:

1) Apply/register filter globally, so it applied to all the razor pages. By adding

```
public void ConfigureServices(IServiceCollection services)
{
        ...
   services.AddMvc(options =>
   {
      options.Filters.Add(new MyPageFilter());
   });
}
```

2) Apply Filter to individual page:

```
[MyPageFilter()]
public class ContactModel : PageModel
{
        ...
}
```

Q10. What is the RCL (Razor Class Library) project?

Ans. It is introduced with ASP.NET Core 2.1 and it allows us to reuse Razor pages/views (UI element) across multiple projects. It includes Razor pages, view components, data model, Razor views, and controllers. We can also create NuGet packages for this library.

Q11. What happened if view, partial view, or Razor Page found in both RCL and web application?

Ans. ASP.NET Core always gives the first precedence to local files (web application) if view, partial view, or Razor Page found in both RCL and web application.

Q12. What is the view component in ASP.NET Core?

Ans. It is a newly added feature and very similar to partial view. It is more powerful than the partial view. View component can work with both ASP.NET Core MVC and Razor pages. It is not used the data binding, but it renders based on the data provided. There is two part of view component: one is class and another return result (View in most cases).

Q13. What are the features provided by the ViewComponent?

Ans. The ViewComponent provides the following features:

- It follows SOC (separation-of-concerns) so provide maintainability and testability
- Also invoked from the Layout page
- It renders in chunk rather than a whole response
- May contains business logic and parameter
- It supports constructor dependency injection
- It does not take part in the controller life cycle

Q14. How can we create ViewComponent?

Ans. There is two part of view component: one is view component class and another return result (View in most cases). View component class is very similar to the controller class. It must be public, non-abstract and non-nested. The view component class must inherit from a ViewComponent base class or decorated with ViewComponent attribute.

```
public class CustomerList: ViewComponent
{
    public async Task<IViewComponentResult> InvokeAsync(int noOfEmployee)
    {
        ...
        ...
        return View();
    }
}
```

optionally, we can decorate the class with the ViewComponent attribute. Here, the class name doesn't need to be the same as the view component name. We can also set a view component by using the name property.

```
[ViewComponent(Name = "CustomerList1")]
public class CustomerList : ViewComponent
{
    public async Task<IViewComponentResult> InvokeAsync(int noOfEmployee)
    {
    ...
    ...
    return View();
    }
}
```

Using Component.InvokeAsync method, view component can be invoked from view. This method has two parameters: view component which needs to include and the parameter of the view component.

```
@await Component.InvokeAsync("CustomerList", new { noOfEmployee = 5 })
```

Q15. What is the default path of view for ViewComponent?

Ans. Same as the controller, views are searched in the following locations:

- View/{Controller Name}/Components/{View Component name}/{View Name}
- Views/Shared/Components/{View Component Name}/{View Name}

Q16. Can we have invoked view component from the controller?

Ans. Yes, we can invoke the view component from the controller.

Example:

```
public IActionResult IndexVC()
{
   return ViewComponent("CustomerList", new { noOfEmployee = 3 });
}
```

6

Model Binding and Validations

Q1. What is Model binding?

Ans. Model binding is a feature that maps the data from the HTTP request to the parameters of the controller actions method. The parameters can be a simple type like int, double, string or can be a complex type.

Q2. How does model binding work in ASP.NET Core application?

Ans. ASP.NET Core retrieves a route from the HTTP request and determines the action method be called. Then it binds the route data to the action method's parameters. It binds the request data to the parameter by name. If the parameter is a complex type, all the properties must have a public setter. Following are the data source used by a model binder to bind the value to the parameter.

- Form values - it is form data, that go with an HTTP request, using the POST method
- Route values - set of route values provided by MVC routing
- Query strings - data is part of the URL.

Example:
URL: http://abcd.com/my/edit/2

```
[Route("my/edit/{id?}")]
public IActionResult Edit(int? id)
{

}
```

In the above example, route data (i.e. “2”) is bind with an id parameter.
Note that all the data source describes above-stored data as name-value pairs.

The model binder used reflection to bind properties value and recursion to traverse the properties of complex types looks for a match. First, it will look for pattern parameter_name.property_name to bind values if it is not found then it uses property name to bind the value.

Q3. What are the characteristics of a complex type for binding the value?

Ans. For binding to happen

- The class must have public and default constructor (without parameter)
- Properties that need to bind must be writable and public
- Model binder uses the default constructor to initialize a complex type.

Q4. Can we control the behaviour of Model binding using attribute?

Ans. Yes, ASP.NET Core MVC provides some attribute that helps us to control the behaviour of Model binding. Following are the attributes.

- BindNever: It tells model binder to not bind value with the parameter
- BindRequired: This attribute mark parameter as required, and it adds error in the model state if binding not found
- FromHeader: It defines the binding source as a request header
- FromQuery: It defines the binding source as a query string
- FromRoute: It defines the binding source as route data
- FromForm: It defines the binding source as a form of data posted
- FromServices: It defines the binding source as services that injected as a dependency
- FromBody: Model binder find the parameter value for the request body
- ModelBinder: It tells the MVC to use a custom model binder to bind the parameter value

Q5. Can we create a custom model binder? If yes how?

Ans. Yes, we can create a custom model binder with ASP.NET Core MVC. Using the custom model binder, we can extend model binder functionality. It required when we need to transform the input before binding.

To create a custom model binder, the class must implement IModelBinder interface.
Example:

```
public class CustomModelBinder : IModelBinder
{
    public Task BindModelAsync(ModelBindingContext bindingContext)
    {
        throw new NotImplementedException();
    }
}
```

Q6. How can we use/register a custom model binder in ASP.NET Core?

Ans. There are two ways to register or use the custom model binder

- Using ModelBinder attribute
- By defining the binder provider and register it to ConfigureServices method of the startup class

Q7. How can apply a custom model binder using ModelBinder attribute?

Ans. There are two ways to apply a custom model binder using ModelBinder attribute.
1) Apply ModelBinder attribute to the model class
Example:

```
[ModelBinder(BinderType = typeof(CustomModelBinder))]
public class User
{
    //...
}
```

2) Apply ModelBinder attribute to individual action method

Example:

```
public IActionResult Contact([ModelBinder(BinderType = typeof(CustomModelBinder))] User u)
{
}
```

Q8. How can we register a custom model binder globally?

Ans. We can apply custom model binder globally by registering the binder to ConfigureServices method of startup class so model binder is available for all the action method.

We must create a binder provider (the class that implement IModelBinderProvider interface) to register a custom model binder globally.

Example:

```
public class CustomModelBinderProvider : IModelBinderProvider
{
    public IModelBinder GetBinder(ModelBinderProviderContext context)
    {
        if (context.Metadata.ModelType == typeof(User))
            return new CustomModelBinder();

        return null;
    }
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.ModelBinderProviders.Insert(0, new CustomModelBinderProvider());
    });
}
```

Q9. What is the use of BindProperty and BindProperties attribute?

Ans. BindProperty attribute is applied to the controller or page model class property to tell the model binder to bind the value of the matching request parameter with the property. To apply this attribute, a class property must define with public access modifier.

```
public class HomeController: Controller
{
    [BindProperty]
    public string Name { get; set; }
    ...
    ...
}
```

You can also bind complex property using BindProperty attribute.

```
[BindProperty]
public Employee EmployeeData { get; set; }
```

BindProperties attribute is introduced in ASP.NETCore 2.1 and it applied to the PageModel or controller class rather than applying to an individual property of a class. This attribute tells the model binder that all the public properties in the PageModel/controller taking part in model binding.

```
[BindProperties]
public class EmployeeModel : PageModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    ...
 }
```

Q10. What happened when "SupportsGet" property of BindProperty attribute is set to true?

Ans. When the SupportsGet property of BindProperty attribute is set to true, it allows only to bind a property to data from GET request.

Q11. How can we do model validation with ASP.NET Core?

Ans. ASP.NET Core also supports model validation using Data annotation. There are many built-in attributes are available such as Required, StringLength, Range, etc. and you can also create a custom validation attribute to validate our model.

Q12. How can we check our model is valid or not at the controller level?

Ans. The Model state represents validation errors on the submitted model. Model validation is done before to action method invoked, so by inspecting ModelState.IsValid, we can validate our submitted model. If this property set to false, your model is not valid.

Q13. What are the sources for model binding to get data from HTTP request?

Ans. Model binding gets the data from the following sources in HTTP request in the form of key-value pairs.

- Form (using FromFormattribute)
- Request body (using FromBody attribute)
- Route Data (using FromRoute attribute)
- Query string parameters (using FromQuery attribute)
- Request header (using FromHeader attribute)
- Uploaded files

Q14. Which part of the MVC framework responsible to set IsValid property of ModelState class?

Ans. ModelBinder

Q15. Can we validate the model manually in Controller class?

Ans. Base controller class ControllerBase has the method "TryValidateModel" that help us to validate the model manually in the Controller class. This method returns true if the model is valid.

Q16. How can we do Client-side validation?

Ans. The data annotation has built-in support for client-side validation. The client-side validation performs using jQuery Unobtrusive Validation script. The tag helper and Html helper have validation attributes that render HTML 5 data-attribute in the element that required for validation. Then jQuery Unobtrusive Validation parses data- attribute and perform the validation on the control when submitting the form.

Example:

```
<div class="col-md-10">
   <input asp-for="Name" class="form-control" />
   <span asp-validation-for="Name" class="text-danger"></span>
</div>
```

To perform the client-side validation, you must include jQuery and jQuery Unobtrusive Validation script file.

Q17. What are types that model binder can convert from source string?

Ans. The model binder can convert the following types from a source string

- Boolean
- Byte, SByte
- Char
- DateTime, DateTimeOffset, TimeSpan
- Enum
- Guid
- Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal
- Uri
- Version

Q18. What are built-in validation attributes provided with ASP.NET Core?

Ans. Following are Built-in ASP.NET Core validation attributes

Attribute name	Description
CreditCard	Validate value to credit card format
Compare	Compare two properties in model
EmailAddress	Validate value to email format
Phone	Validate value to phone number format
Range	Validate that value is in provided range or not
RegularExpression	Validate value with provided regular expression
Required	Validate value is not null

StringLength	Validate string length value does not exceed a specified length limit
Url	Validate value to Url format
Remote	Validate input on server-side by calling the action method

Q19. How can we create a custom validation attribute?

Ans. There are three steps to create a custom validation attribute

Step 1: Create a class and inherit it from ValidationAttribute

Step 2: Create a public constructor of the class if any data need outside of class

Step 3: override the IsValid method and write validation logic inside this method. Return ValidationResult.Success if validation was successful.

```
public class CustomValidationAttribute : ValidationAttribute
{
    private int _value;
    public CustomValidationAttribute(int value)
    {
        _value = value;
    }
    protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
    {
        return ValidationResult.Success;
    }
}
```

Q20. How can we disable client-side validation?

Ans. We can disable client-side validation by setting up ClientValidationEnabled property to false of AddViewOptions middleware.

```
services.AddMvc().AddViewOptions(options =>
{
    options.HtmlHelperOptions.ClientValidationEnabled = false;
});
```

7

Globalization and Localization

Q1. What is Internationalization in ASP.NET Core?

Ans. It involves Globalization and Localization. Globalization supports multiple cultures hence it used to create multilingual websites with ASP.NET Core. Localization is a process of translating user interface according to specific culture i.e., the application will display data based on the selected language.

Q2. What is a resource file naming convention?

Ans. Resource file name same as controller and view name with full type except for assembly name. For example, HomeController within demoApp.Controllers namespace, the naming convention for this file for English resource file is Controllers.HomeController.en-Us.resx.

Alternatively, we can put the resource file under a similar folder as the naming provided i.e. under folder Resources/Controllers/HomeController.en-US.resx

We can set the resource file path in the ConfigureService method.

Q3. How can we achieve localization in view?

Ans. The ASP.NET Core provides the IViewLocalizer service that is used to localize string into view. This interface finds the resource file under the view file path and the file name is the same as the view name.

Example:

```
@using Microsoft.AspNetCore.Mvc.Localization

@inject IViewLocalizer Localizer
<div>
    @Localizer[“TestAppText”]
</div>
```

Also, we need to register AddViewLocalization service in the ConfigureServices method of the startup class.

```
public void ConfigureServices(IServiceCollection services)
```

```
{
        services.AddMvc().AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix);
}
```

There is no option for a globally shared resource file, but it can be achieved by using IHtmlLocalizer service.

Q4. How can we access a localized resource string in Controller?

Ans. ASP.NET Core provides IStringLocalizer and IStringLocalizer<T> interface as dependency injection that use ResourceManager and ResourceReader to provide culture-specific resources at runtime. Using this interface, we can access localize string in the controller.

Example:

```
public class MyController: Controller
{
        private readonly IStringLocalizer<MyController> _localizer;

        public AboutController(IStringLocalizer<MyController> localizer)
        {
            _localizer = localizer;
        }

        [HttpGet]
        public string Get()
        {
            return _localizer["Title"];
        }
}
```

Q5. Can we localize error messages defined in Data Annotation?

Ans. Yes, it can localize using the service IStringLocalizer and register the service using AddDataAnnotationsLocalization method in ConfigureServices method of the Startup class.

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddMvc().AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix)
                . AddDataAnnotationsLocalization();
}
```

Example

```
public class ModelClass
{
        [Required(ErrorMessage = "RequiredName")]
        public string Name { get; set; }
}
```

Q6. How does the culture fallback mechanism work in ASP.NET Core?

Ans. The ASP.NET Core provides a strong culture fallback mechanism. It is starting from the requested culture, if not found, look for parent culture. If parent culture is not found, it retrieves the value from the default resource file. For example, if an application looking for fr-FR culture file. If it is not found, it looks for fr culture file and if the resource not found in the parent, it looks in the default culture file.

Q7. What happens when resource not found in a culture resource file?

Ans. If resource not found in the culture resource file, the resource manager will find resource value from the default resource file.

Q8. How can we return the resource key when the resource not found in the culture file?

Ans. If we want to return the resource key when a resource not found from a culture resource file, we must not have a default resource file.

Q9. Can we add Content-Language header With ASP.NET Core? If yes, how?

Ans. ASP.NET Core 3.0 allows you to add Content-Language header by setting the property ApplyCurrentCultureToResponseHeaders of RequestLocalizationOptions.

```
app.UseRequestLocalization(new RequestLocalizationOptions
{
   ApplyCurrentCultureToResponseHeaders = true
});
```

Q10. What is Neutral culture?

Ans. A culture that has a specified language, but not a region. For example, "en" and "es".

Q11. Is there any other way to set application culture other than Accept-Language header approach?

Ans. ASP.NET Core has QueryStringRequestCultureProvider that set application culture from query string but for same we have to pass query string parameters “culture” and “ui-culture”.

```
http://localhost:5000/?culture=fr-FR&ui-culture=fr-FR
```

8

Exception Handling and Logging

Q1. What is the "Developer Exception Page" in ASP.NET Core?

Ans. It is a feature that shows detailed information about the exception. This can be enabled using the Developer Exception Page middleware. This middleware defined in Microsoft.AspNetCore.Diagnostics assembly. To add this middleware to the request pipeline, we need to call IApplicationBuilder.UseDeveloperExceptionPage method in Configure method of startup class.

Example

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        ...
    }
    ...
    ...
}
```

Q2. How can we configure a custom exception handling page in ASP.NET Core?

Ans. Using the middleware "Exception Handler", we can add a custom exception handling page in ASP.NET Core. We can add this middleware by calling IApplicationBuilder.UseExceptionHandler method in Configure method of startup class.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
```

```
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }
    ...
    ...
}
```

Note that whatever route you mention here must available in the application.

Q3. Can we define custom exception without using defining error page?

Ans. Yes, we can define the custom exception handler page inline using lambda exception to UseExceptionHandler. It allows accessing the error before returning the response.

In the following example, the error status code and the custom error message has been written to the response object before returning the response.

```
app.UseExceptionHandler(errorApp =>
{
    errorApp.Run(async context =>
    {
        context.Response.StatusCode = 500;
        context.Response.ContentType = "text/html";

        await context.Response.WriteAsync("<html lang=\"en\"><body>\r\n");
        await context.Response.WriteAsync("ERROR!<br><br>\r\n");

        //Write your code for custom error here...

        await context.Response.WriteAsync("</body></html>\r\n");
        await context.Response.WriteAsync(new string(' ', 512)); // IE padding
    });
});
```

Q4. What is the use of UseStatusCodePages middleware?

Ans. ASP.NET Core does not provide status code page for HTTP error status code 400-599. Its returns the status code with an empty body. The UseStatusCodePages middleware is used to provide status code pages. This middleware must inject before request handling middleware.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
...
    app.UseStatusCodePages();
    app.UseHttpsRedirection();
...
}
```

Q5. Can we customize the response content type and text for UseStatusCodePages middleware?

Ans. Yes. The overload version of UseStatusCodePages method takes the content type and format string as a parameter.

```
app.UseStatusCodePages("text/plain", "Status code page, status code: {0}");
```

Q6. How to configure the logging framework in ASP.NET Core?

Ans. There are two easy steps to add logging support to ASP.NET Core application

1. We need to the dependency of "Microsoft.Extensions.Logging" to .csproj file, however, this dependency added by default for .NET framework 2.0 and above
2. The next step is to add a provider extension method to configure the logging method of WebHostBuilder instance under the main method of the program class

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureLogging((hostingContext, logging) =>
            {
                logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
                logging.AddConsole();
                logging.AddDebug();
            })
            .UseStartup<Startup>();
}
```

Q7. What are the possible values for Log level enum for the Logging framework?

Ans. Log level has the following possible values

- Trace (0)
- Debug (1)
- Information (2)
- Warning (3)
- Error (4)
- Critical (5)
- None (6)

Q8. How many built-in extension methods provided by the logging framework of ASP.NET Core?

Ans. There are many built-in extension methods provided by the logging framework of ASP.NET Core. Followings are the methods:

- LogCritical: Format and write a critical log message
- LogDebug: Format and write a debug log message
- LogError: Format and write an error log message
- LogInformation: Format and write an information log message
- LogTrace: Format and write a trace log message
- LogWarning: Format and write a warning log message
- Log: Format and write a log message specifying the log level. This is a base method and the methods describe above are called this method internally with different log level

Q9. What is the use of enum value LogLevel.None?

Ans. When the minimum log level is specified as LogLevel.None, the Logging framework suppress all the log.

Q10. How to set default minimum log level for Logging Framework?

Ans. Using LoggingBuilderExtensions.SetMinimumLevel method, we can set a minimum log level and is effective only if no rules are defined either by code or configuration. This method calls from the main method of program class.

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
WebHost.CreateDefaultBuilder(args)
    .ConfigureLogging((hostingContext, logging) =>
    {
        logging.SetMinimumLevel(LogLevel.Warning);
        logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
        logging.AddConsole();
        logging.AddDebug();
    })
    .UseStartup<Startup>();
```

In the above example, the minimum log level is set to warn so, the Logging framework ignores the log message that below mentioned warning level.

Q11. How to define filter rule for the Logging framework in ASP.NET Core?

Ans. We can specify the logging filter rules either by using a Configuration file or using C# code. We can specify the logging filter rules in any configuration file such as appsettings.json. The first step is to add a JSON configuration file and add a configuration section to the logging framework. In the following code example, I have followed the steps mention here.

Example:

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
WebHost.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((hostingContext, config) =>
```

```
    {
        var env = hostingContext.HostingEnvironment;
        config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json",
                    optional: true, reloadOnChange: true);
        config.AddEnvironmentVariables();
    })
    .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
        logging.AddConsole();
        logging.AddDebug();
    })
    .UseStartup<Startup>();
```

Appsettings.json

```
{
  "Logging": {
    "IncludeScopes": false,
    "Debug": {
      "LogLevel": {
        "Default": "Information"
      }
    },
    "Console": {
      "LogLevel": {
        "Microsoft.AspNetCore.Mvc.Razor": "Error",
        "Microsoft.AspNetCore.Mvc.Razor.Razor": "Error",
        "Default": "Information"
      }
    },
    "LogLevel": {
      "Default": "Debug"
    }
  }
}
```

The above example creates five filter rules

- One rule for Debug provider
- Three rules for Console Provider
- One rule for all provider

The same thing can be achieved using the C# code using the AddFilter method.

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
WebHost.CreateDefaultBuilder(args)
    .ConfigureLogging((hostingContext, logging) =>
    {
        logging.AddFilter("System", LogLevel.Debug);
        logging.AddFilter<Microsoft.Extensions.Logging.Debug.DebugLoggerProvider>("System",
LogLevel.Information);

logging.AddFilter<Microsoft.Extensions.Logging.Console.ConsoleLoggerProvider>("System",
LogLevel.Information);
```

```
logging.AddFilter<Microsoft.Extensions.Logging.Console.ConsoleLoggerProvider>("Microsoft.AspNetCore.Mvc.Razor", LogLevel.Error);

logging.AddFilter<Microsoft.Extensions.Logging.Console.ConsoleLoggerProvider>("Microsoft.AspNetCore.Mvc.Razor.Razor", LogLevel.Error);

logging.AddFilter<Microsoft.Extensions.Logging.Console.ConsoleLoggerProvider>("System", LogLevel.Information);
        ...
        ...
    })
    .UseStartup<Startup>();
```

Q12. What is log scope in the Logging framework?

Ans. It allows us to group a set of logical operations within the logging scope, so the logging framework attached the same data to each log of this set. To use log scope, first, we need to enable scopes for the provider.

```
.ConfigureLogging((hostingContext, logging) =>
{
       ...
       ...
       logging.AddConsole(options => options.IncludeScopes = true);
       ...
       ...
}
```

In controller Action

```
public IActionResult TestMethod()
{
    using (_logger.BeginScope("Log Scope Start"))
    {
        _logger.LogInformation("Example Scope");
        _logger.LogWarning("My Scope");
    }
    return View();
}
```

Q13. What are third party logging frameworks supported by ASP.NET Core?

Ans. Following are third party logging framework that supported by ASP.NET Core and all are available in GitHub repository.

- elmah.io
- Gelf
- JSNLog
- KissLog.net
- Loggr
- NLog
- Sentry
- Serilog
- Stackdriver

- Log4net

Q14. What are built-in logging providers supported by ASP.net Core?

Ans. ASP.NET Core provides the following built-in logging providers.

- Console (AddConsole)
- Debug (AddDebug)
- EventSource (AddEventSourceLogger)
- EventLog (AddEventLog)
- TraceSource (AddTraceSource)
- AzureAppServicesFile (AddAzureWebAppDiagnostics)
- AzureAppServicesBlob (AddAzureWebAppDiagnostics)
- ApplicationInsights

Q15. Can we use log service in Startup Class?

Ans. Yes. We can use log service in Startup class but writing the logs before completion of the DI container in ConfigureServices method of Startup class is not supported. Following places of startup class logger injection is not supported.

- Startup class's constructor
- Startup.ConfigureServices method

Q16. Can we change the log level in a running app?

Ans. Yes. Using reloading the configuration file, we can change the log level in the running app. The app can call "IConfigurationRoot.Reload" method to load the updated logging configuration.

9
Dependency Injection

Q1. How does dependency injection work in ASP.NET Core?

Ans. There is built-in support for dependency injection in ASP.NET Core. This support is not limited to middleware but also support Controllers, Views and Models. The services that are added to the service container are available as dependency service.

Q2. How many types of service containers available in ASP.NET Core?

Ans. There are two types of service container available in ASP.NET Core: Application Services and Framework Services. The framework services are service that is provided by the ASP.NET Core such as ILoggerFactory etc. The application services are the customs services created base on our requirement.

Q3. How can we inject the service dependency into the controller?

Ans. There are three easy steps to add custom service as a dependency on the controller.

Step 1: Create the service

```
public interface IHelloWorldService
{
    string SaysHello();
}

public class HelloWorldService: IHelloWorldService
{
    public string SaysHello()
    {
        return "Hello ";
    }
}
```

Step 2: Add this service to "Service container" (service can either added by singleton, transient or scoped)

```
public void ConfigureServices(IServiceCollection services)
{
    ….
    …
    services.AddTransient<IHelloWorldService, HelloWorldService>();
    …
    …
}
```

Step 3: Use this service as a dependency in the controller

```
public class HomeController: Controller
{
    IHelloWorldService _helloWorldService;
    public HomeController(IHelloWorldService helloWorldService)
    {
        _helloWorldService = helloWorldService;
    }
}
```

Q4. Can we inject the dependency to the individual action method of the controller?

Ans. Yes. Using the “FromServices" attribute, we can inject the dependency to the individual action method of the controller. This attribute tells the ASP.NET Core framework that parameter should be retrieved from the service container.

Example:

```
public IActionResult Index([FromServices] IHelloWorldService helloWorldService)
{
    ViewData["MyText"] = helloWorldService.SaysHello() + "Jignesh!";
    return View();
}
```

Q5. Can we get a service instance without dependency injection in the controller action method?

Ans. Yes. We can retrieve the service from the ASP.NET Core service container. Using the method "GetService" of "HttpContext.RequestServices" property, we can get dependent services configured with the Service container. This is also known as property injection.

Example:

```
public IActionResult Index1()
{
    var helloWorldService =
(IHelloWorldService)this.HttpContext.RequestServices.GetService(typeof(IHelloWorldService));
    ViewData["MyText"] = helloWorldService.SaysHello() + "Jignesh Trivedi!";
    return View("index");
}
```

Q6. How to specify a service life for register service that added as a dependency?

Ans. ASP.NET Core allows us to specify the lifetime for registered services. The service instance gets disposed of automatically based on a specified lifetime. So, we do not care about cleaning these dependencies, it will take care by the ASP.NET Core framework. There are three types of lifetimes.

Singleton

ASP.NET Core will create and share a single instance of the service through the application life. The service can be added as a singleton using the AddSingleton method of IServiceCollection. ASP.NET Core creates a service instance at the time of registration and subsequence request use this service instance. Here, we do not require to implement Singleton design pattern and single instance maintained by the ASP.NET Core itself.

Example:

```
services.AddSingleton<IHelloWorldService, HelloWorldService>();
```

Transient

ASP.NET Core will create and share an instance of the service every time to the application when we ask for it. The service can be added as Transient using AddTransient method of IServiceCollection. This lifetime can be used in stateless service. It is a way to add lightweight service.

Example:

```
services.AddTransient<IHelloWorldService, HelloWorldService>();
```

Scoped

ASP.NET Core will create and share an instance of the service per request to the application. It means that a single instance of service available per request. It will create a new instance in the new request. The service can be added as scoped using an AddScoped method of IServiceCollection. We need to take care of while, service registered via Scoped in middleware and inject the service in the Invoke or InvokeAsync methods. If we inject dependency via the constructor, it behaves like a singleton object.

```
services.AddScoped<IHelloWorldService, HelloWorldService>();
```

Q7. How can we inject the service dependency into the View?

Ans. There are three easy steps to add custom service as a dependency into the View.

Step 1: Create the service

```
public interface IHelloWorldService
{
    string SaysHello();
}

public class HelloWorldService: IHelloWorldService
{
    public string SaysHello()
    {
        return "Hello ";
    }
}
```

Step 2: Add this service to the Service container (service can either added by singleton, transient or scoped)

```
public void ConfigureServices(IServiceCollection services)
{
    ….
    …
    services.AddTransient<IHelloWorldService, HelloWorldService>();
    …
    …
}
```

Step 3: Inject the dependency to view using the @inject directive. Here we need to pass the service type and instance name that is used to access the service method.

```
@inject <type> <instance name>
```

Example:

```
@{
    ViewData["Title"] = "DIToView";
}
@inject DepedencyInjectionExample.Service.IHelloWorldService helloWorldService

<h4>DI To View</h4>

<h5>
    @helloWorldService.SaysHello() Reader!!!
</h5>
```

Q8. What point you will be taken care of while creating a service for DI?

Ans. Following are the best practices to create service for DI

- Do not create static methods
- Services able to obtain their own dependencies
- Avoid creating stateful service
- Avoid direct instantiation of dependent classes within services
- Make service classes small and well-factored, so can be easily tested

10
Security and Filters

Q1. What is the use of a filter in the ASP.NET Core application?

Ans. Filters allowed us to execute custom code after and before executing the action method. They are invoked at various stage of the application life-cycle. There are many built-in filters are available with ASP.NET Core application such as Authorize, HttpGet, HttpPost etc.

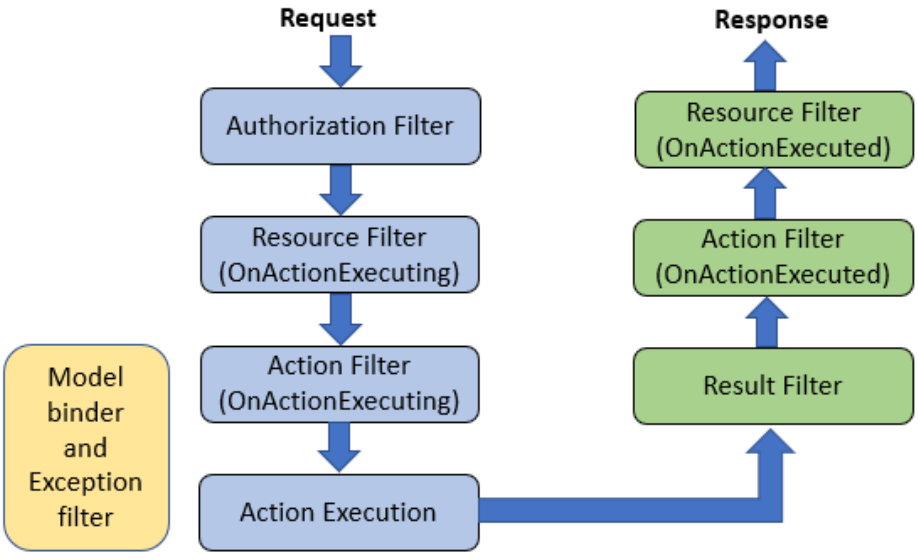
Q2. What are the different types of filters in ASP.NET Core?

Ans. Following filter types are provided by ASP.NET Core Application.

- **Authorization filters:** It helps us to find out whether the user is authorized for the current request. This executes first in the request pipeline.
- **Resource filters:** It executes immediately after the Authorization filters and before model binding happened.
- **Action filters:** It executes before and after the controller action method execute.
- **Exception filters:** It is used to handle the exception
- **Result filters:** It executes before and after the execution of controller action results.

Q3. In which sequence all the filters are invoked?

Ans. The following diagram shows the filter sequence, in which they are executed.



Q4. How can we create a custom filter in ASP.NET Core?

Ans. To implement a synchronous filter, the class must implement IActionFilter interface. This interface has two abstract methods: OnActionExecuting and OnActionExecuted. The method OnActionExecuting is called before the action executing and OnActionExecuted method is called after an action executed.

```
using Microsoft.AspNetCore.Mvc.Filters;
namespace CustomFilters
{
    public class MyActionFilter : IActionFilter
    {
        public void OnActionExecuted(ActionExecutedContext context)
        {
        }
        public void OnActionExecuting(ActionExecutingContext context)
        {
        }
    }
}
```

Q5. Does the filter support asynchronous implementation?

Ans. Yes, the filter supports asynchronous implementation. With asynchronous filters, there is only one method OnStageExecutionAsync to implement.

Q6. How to implement the asynchronous filter in ASP.NET Core?

Ans. To implement the asynchronous filter, the class must implement the IAsyncActionFilter interface. Following is the example.

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.Filters;
namespace CustomFilters
{
    public class AsyncActionFilter : IAsyncActionFilter
    {
        public async Task OnActionExecutionAsync(ActionExecutingContext context,
ActionExecutionDelegate next)
        {
            //To do : Code that execute before the action executes
            await next(); //Action Execution
            //To do : Code that execute after the action executes
        }
    }
}
```

Q7. Explain the Filter scope?

Ans. There are three different scopes for add filter to the ASP.NET request pipeline:

- To Action method: It is applied to action which decorated with the action filter
- To controller: It is applied to all action methods of the controller which decorated with the action filter
- Global filter: It is allied to all the controller's action methods

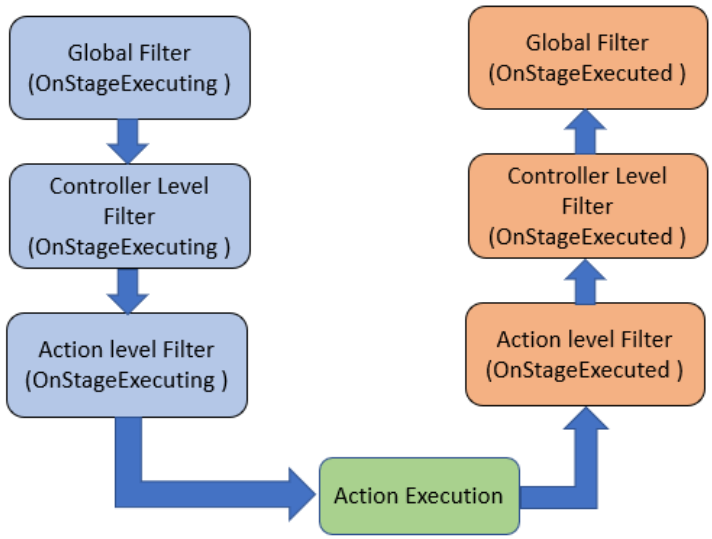
Q8. How to define filter at the Global level in ASP.NET Core?

Ans. To register the filter that applied globally, we need to add a filter to MvcOption.Filters collection in ConfigureServices method of startup class. We can add a filter by instance or by type. Following is a code example.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    ...
    services.AddMvc(options =>
    {
        //an instant
        options.Filters.Add(new MyActionFilter());
        //By the type
        options.Filters.Add(typeof(MyActionFilter));
    });
}
```

Q9. What is the Default order of filter execution?

Ans. The global filter first applied to the request and then the controller level filter applied and finally, the action method level filter applied. The following diagram shows the default order of execution of filters in a different scope.



Q10. Can we override the default order of execution of filter in ASP.NET Core?

Ans. Yes, we can override the default order of execution of filter by using implement IOrderedFilter interface. This interface has a property called "Order" that is used to define the order. The filter with a lower Order property value will execute first. ASP.NET Core application sorted the filter by order first and then the scope of the filter.

Example:

```
using Microsoft.AspNetCore.Mvc.Filters;
namespace WebApplication1
{
    public class MyActionFilter : IActionFilter, IOrderedFilter
```

```
    {
        public int Order { get; set; }
        public void OnActionExecuted(ActionExecutedContext context)
        {
        }
        public void OnActionExecuting(ActionExecutingContext context)
        {
        }
    }
}
```

Q11. Does the built-in filter also implement the interface IOrderFilter?

Ans. Yes, the built-in filter implements the IOrderFilter interface and the default value for the order property is set to 0 (zero).

Q12. Can we cancel the execution of filter or short-circuiting filters?

Ans. Yes, by setting up the Result property of the context parameter (ActionExecutingContext or ActionExecutedContext), we can short-circuit the filters.

Example:

```
public void OnActionExecuted(ActionExecutedContext context)
{
    context.Result = new ContentResult()
    {
        Content = "Short circuit filter"
    };
}
```

Q13. Can we inject the dependency into the filter attribute?

Ans. The filter that applied directly to the controller or its action cannot have constructor dependency. This is a limitation of the filter attribute. But there is a way to overcome this limitation.

Q14. Can you apply a filter attribute having constructor dependency to the controller or action?

Ans. The filter that applied directly to the controller or its action cannot have constructor dependency. This is a limitation of the filter attribute. But there is a way to overcome this limitation. There is a built-in attribute available, with the help of them we can apply such filter to controller or action method. Following are the built-in attribute.

- ServiceFilterAttribute
- TypeFilterAttribute

Q15. What is characteristic of the Authorization filter?

Following are the characteristic of Authorization Filter

- The first filter that run-in pipeline
- Control access to an action method
- Has a before call method but not after call

Q16. Are the action filters applied to Razor pages?

No, They are not applied to Razor pages. Razor pages supports IAsyncPageFilter or IPageFilter.

Q17. What is an exception filter in ASP.NET Core?

Ans. It is used to handle the exception that occurred during the execution code and it provides a place on that we can do something before anything written to the response body. It helps us to apply the error handling policy for the application. It implements either IAsyncExceptionFilter or IExceptionFilter interface. Both the interface has a single method that needs to override.

Example:

```
public class CustomExceptionFilterAttribute : Attribute, IExceptionFilter
{
    public void OnException(ExceptionContext context)
    {
        var exception = context.Exception;
        //Do something with exception

        // log exception here....
        var result = new ViewResult { ViewName = "Error" };
        context.Result = result;
        context.ExceptionHandled = true;
    }
}
```

Q18. Can Exception Filter have before call method?

Ans. No. It does not have a before call method.

Q19. What is a limitation of the Exception filter?

Ans. Exception filters are not able to handle exceptions when they occurred in following are

- Controller Constructor
- Another action filter such as Authorization filter, Resource filters, Result filters, etc.
- Result execution
- Model binding

Q20. How the exception filter differs from an action filter?

Ans. It is a special type of action filter that helps us to catch an unhandled exception. It does not have before and after events. It only has OnException or OnExceptionAsync method that triggered when an unhandled exception occurred.

Q21. What kind of exception can be caught using an exception filter?

Ans. It can catch unhandled exceptions that occur in Razor Page or controller creation, model binding, action filters, or action methods.

Q22. What is the use of Result Filter in ASP.NET Core?

Ans. It is used to run the custom code before and after the execution of the action result. It is executed if the action method has been successfully executed. If we want to add the logic that executed before and after a view result is executed, a result filter can be used. For example, modify the view result before render the view.

Q23. What are Authentication and Authorization?

Ans. Authentication is a process of checking user identity and user identity can be checked against any trusted source such database or other 3rd party services such as Facebook, LinkedIn, etc.

Authorization is a process of checking user rights i.e., determines whether the user can do this. ASP.NET Core has built-in support of identity providers however it can also support 3rd party identity service such as Twitter, Facebook, Microsoft account etc.

Q24. How can we configure a built-in identity service for ASP.NET Core application?

Ans. By calling AddDefaultIdentity or AddIdentity in ConfigureService method of the startup class, we can add a built-in identity service to the request pipeline. This service can be enabled by calling UseAuthentication in Configure method of the startup class.

Using the following code, we can Configure identity middleware.

```
public void ConfigureServices(IServiceCollection services)
{
        ...
        services.AddDefaultIdentity<IdentityUser>()
             .AddEntityFrameworkStores<ApplicationDbContext>();
        ...
}
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
        ...
        app.UseAuthentication();
        ...
}
```

By decorating a controller or action method with Authorize attribute that you apply authorization.

Q25. What is the difference between AddIdentity and AddDefaultIdentity?

Ans. Both are added default identity system configuration for user and role, however, the AddDefaultIdentity method adds more default configuration to the system such as default login page, registration page, etc. We can configure a default configuration with the AddIdentity method.

Q26. How can we override the default configuration for Identity?

Ans. ASP.NET Core allows us to override the default configuration by changing the value of IdentityOption in ConfigureServices method of the startup class. If we are not adding configuration, the ASP.NET Core application uses the default configuration.

Example:

```
public void ConfigureServices(IServiceCollection services)
{
    // Overide the configuration
    services.Configure<IdentityOptions>(options =>
    {
        // Password settings
        options.Password.RequireDigit = true;
        options.Password.RequiredLength = 8;
        options.Password.RequireNonAlphanumeric = true;
    });
}
```

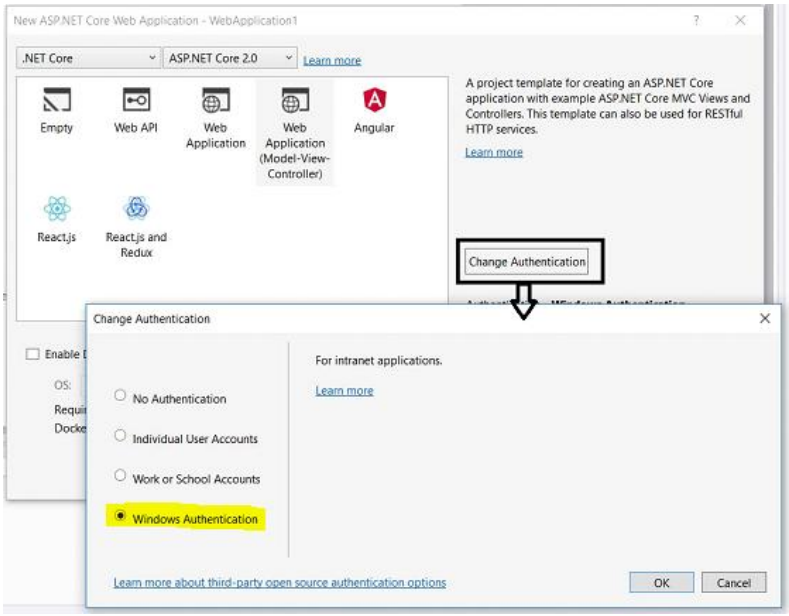
Using this identity option, we can override the following default configuration

- PasswordOptions (Password Policy)
- LockoutOptions(User's lockout)
- UserOptions (User validation settings)
- SignInOptions(Sign in settings)
- ConfigureApplicationCookie(Cookie settings for Application)

Q27. How can we configure windows authentication in ASP.NET Core?

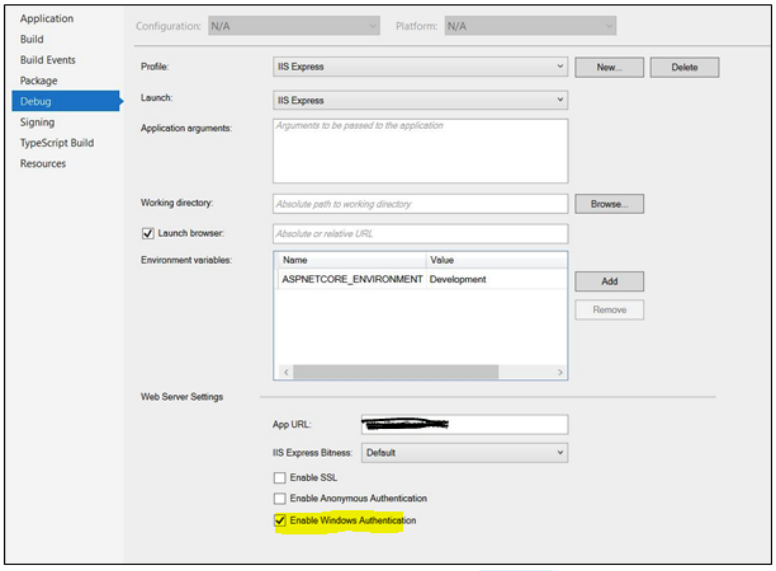
Ans. We can configure Windows authentication in ASP.NET Core application in one of the following ways

1) Configure windows authentication when they created. We can configure Windows authentication for the ASP.NET Core application when it created by either Visual Studio or using CLI.



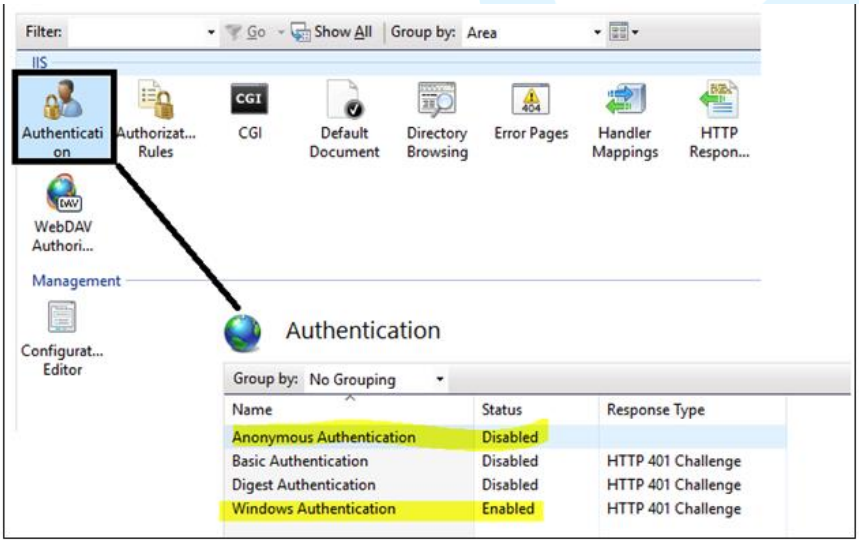
```
>dotnet new MVC --auth windows
```

2) It can be configured manually using Visual Studio project properties. Open Visual Studio project properties >> go to Debug tab and checked the "Enable Windows Authentication" checkbox.



3) We can also configure it through IIS

Select the application and Go to Feature view >> select the "Authentication" module and enable Windows Authentication.



Q28. How can we enforce HTTPS for all requests in ASP.NET Core?

Ans. ASP.NET Core provides middleware that redirects HTTP requests to HTTPS. We can inject the middleware into the request pipeline using the UseHttpsRedirection method of the IApplicationBuilder object.

Q29. Is there any alternative approach for enforcing Https other than injecting middleware?

Ans. Yes, HTTPS Redirection middleware is to use URL rewriting middleware (AddRedirectToHttps), we can also use URL rewriting middleware and set additional rule for redirection.

Q30. How can we enable CORS in ASP.NET Core app?

Ans. There are multiple ways to enable CORS in the ASP.NET app.

- Enable CORS with attributes

We can use the EnableCors attribute to enable CORS. It can be applied to Razor Page, Controller and action method.

- Enable Cors with endpoint routing

using the RequireCors extension method, we can enable the CORS for a defined route with endpoint routing

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/test", async context => context.Response.WriteAsync("test"))
        .RequireCors("policy name");
});
```

- Enable CORS using middleware

ASP.NET Core provides CORS middleware. If we use CORS middleware, CORS policies to all the apps endpoints.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddCors(options =>
    {
        options.AddPolicy("MyAllowSpecificOrigins",
        builder =>
        {
            builder.WithOrigins("http://abc.com",
                                "http://www.xyz.com")
                                .AllowAnyHeader()
                                .AllowAnyMethod();
        });
    });
    ...
}
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseCors();
    ...
}
```

Q31. How to prevent ASP.NET Core app from Cross-Site Request Forgery (CSRF)?

Ans. It also is known as Session riding and pronounced as XSRF. In this method of attack, the attacker copies forge as a trusted source and send the data to the site. The site processes this information believe that this is coming from a trusted source.

ASP.NET Core provides tag helper "asp-anti forgery" that enable /disable anti-forgery for the page. If this tag helper value is set to "true", it means that it enables the anti-forgery for this page.

```
<form asp-controller="Account" asp-action="Login" asp-antiforgery="true">
...
...
</form>
```

using the ValidateAntiForgeryToken attribute, we can check whether the token is valid or not at the server side (controller/action method).

```
[ValidateAntiForgeryToken]
public class AccountController : Controller
{
    //...
    //...
}
```

Q32. How can we achieve Role-based authentication in ASP.NET Core?

Ans. We can define role based authentication with Authorize attribute.

```
[Authorize(Roles = "Admin")]
public class HomeController : Controller
{

}
```

Q33. How can we achieve Policy-based authentication in ASP.NET Core?

Ans. We can define policy-based authentication with Authorize attribute and policies can be defined when registered as part of authorize service configuration (ConfigureServices method).

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddAuthorization(options =>
  {
    options.AddPolicy("UserPolicy", policy =>
      policy.UserRequireCustomClaim(ClaimTypes.Email);
```

```
        policy.UserRequireCustomClaim(ClaimTypes.DateOfBirth);
    });
}
```

```
[Authorize(Policy = "UserPolicy")]
public class HomeController : Controller
{

}
```

Q34. How can we achieve Claim based authentication in ASP.NET Core?

Ans. Claim based authentication can be achieved using policy-based authentication. You can define multiple claims when registered as part of authorize service configuration (ConfigureServices method).

Q35. How can we forcefully redirect to HTTPS for all requests?

Ans. Using built-in middleware "UseHttpsRedirection", we can forcefully redirect to HTTPS for all requests.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
...
    app.UseHttpsRedirection();
...
}
```

How can we configure/change the default port for HTTPS redirection?
The AddHttpsRedirection method provides the way to configure the default port for HTTPS redirection.

```
public void ConfigureServices(IServiceCollection services)
{
...
    services.AddHttpsRedirection(options =>
    {
        options.RedirectStatusCode = StatusCodes.Status307TemporaryRedirect;
        options.HttpsPort = 5001;
    });
...
}
```

11

Unit Testing

Q1. What Unit Testing frameworks can be used with ASP.NET Core?

Ans. Mainly there are three unit testing frameworks people use with ASP.NET Core.

- MSTest
- xUnit
- NUnit

Q2. How can we write a unit test with the MSTest framework?

Ans. To write a unit test using the MSTest framework, create a class and this denotes with TestClass attribute. Create test method under this class as many as you want but denote the methods with the TestMethod attribute.

Example

```
[TestClass]
public class UnitTest1
{
    [TestMethod]
    public void TestMethod1()
    {

    }
}
```

Q3. Can we create an MSTest project using command line interface (CLI)?

Ans. Yes, using the following command we can create a MSTest project with CLI.

```
dotnet new MSTest
```

Q4. How can we verify the test in MSTest?

Ans. We can verify the test using the Assert class. This class has a couple of methods that help us to verify the actual result with the expected result. For example, the AreEqual method used to verify specified values must equal otherwise it throws an exception.

Example:

```
[TestMethod]
public void TestMethod1()
{
    int expected = 1;
    int actual = 1;
    Assert.AreEqual(expected, actual);
```

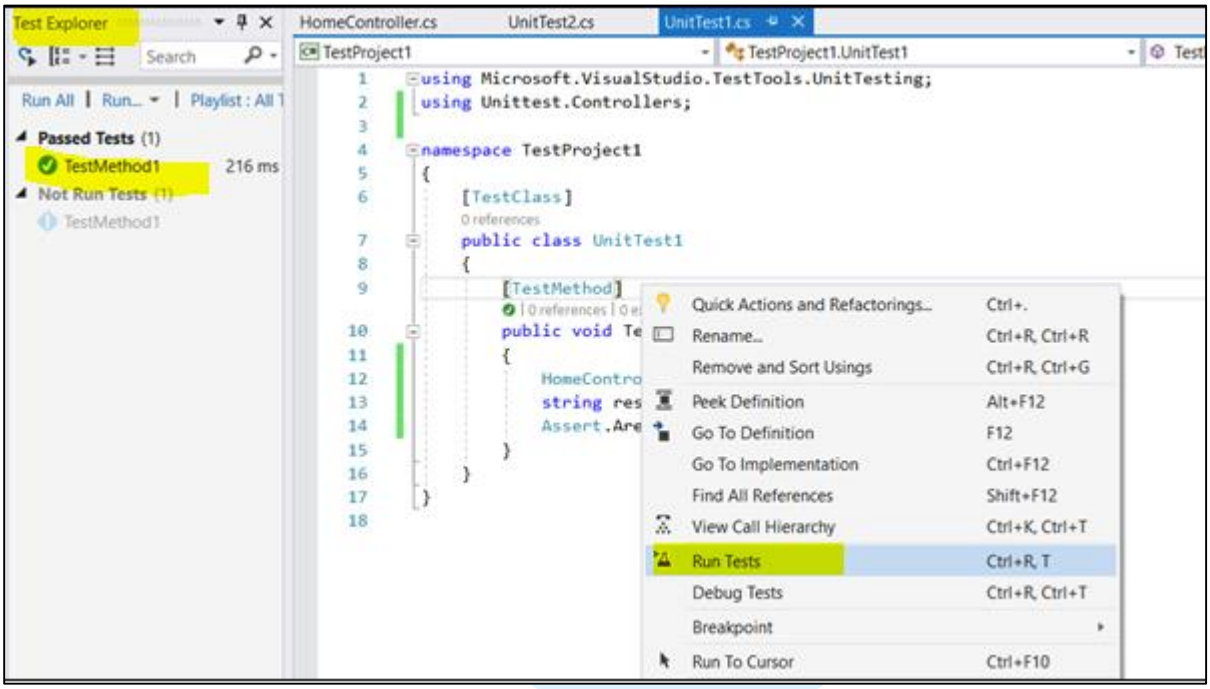
```
}
```

Q5. How to run the unit test?

Ans. We can run test either from CLI or Test Explore. Using the following command, we can run all the test within the project.

```
dotnet test
```

We can also run the test from test explore or from individual test method from the code reference.



Q6. Can we write a data-driven test using MSTest?

Ans. Yes, we can write a data-driven test in MSTest by using DataTestMethod and DataRow attribute. The DataTestMethod attribute tells the test framework that this is test method and the DataRow attribute specify the input value to run the unit test.

```
[DataTestMethod]
[DataRow(1,"Test")]
[DataRow(2, "Test 1")]
public void TestMethod2(int Id, string name)
{
    //do something
}
```

Q7. How can we write a unit test with xUnit framework?

Ans. The xUnit is an open-source and community-focused unit testing framework. To write a unit test using the xUnit framework, create a class and create a test method under this class as many as you want but denote the methods with the Fact attribute.

```
[Fact]
public void Test1()
{
    //do something.
}
```

Q8. How can we verify the test in xUnit?

Ans. We can verify the test using the Assert class. This class has a couple of methods that help us to verify the actual result with the expected result. For example, the "Equal" method is used to verify that specified values must equal otherwise it throws an exception.

Example:

```
[Fact]
public void Test1()
{
    string test1 = "Test1";
    string test2 = "Test2";
    Assert.Equal(test1, test2);
}
```

Q9. Can we write a data-driven test using xUnit?

Ans. Yes, we can write a data-driven test in xUnit by using Theory and InlineData attribute. The Theory attribute tells the test framework that this is a test method and the InlineData attribute specify the input value to run the unit test.

Example:

```
[Theory]
[InlineData(1, "Test Data 1")]
[InlineData(2, "Test Data 2")]
public void Test2(int id, string name)
{
    //Do something
}
```

Q10. How can we write a unit test with NUnit framework?

Ans. NUnit is an open-source unit test framework that supports all the .NET languages. It is very similar to MSTest. The template for NUnit is not by default available, so we need to install the template using CLI or using NuGet package manager.

```
//Using CLI
>dotnet new - I NUnit3.DotNetNew.Template

//Using NuGet package manager
PM> Install-Package NUnit -Version 3.9.0
```

To write a unit test using NUnit framework, create a class that denotes with TestFixture attribute and create a test method under this class as many as you want but denote the methods with "Test" attribute.

Example:

```
[TestFixture]
public class TestClass
{
     [Test]
     public void Test1()
     {
       //Do something
     }
}
```

Q11. How can we verify the test in NUnit?

Ans. We can verify the test using the Assert class. This class has a couple of methods that help us to verify the actual result with the expected result. For example, "AreEqual" method is used to verify that specified values must equal otherwise it throws an exception.

Example:

```
[Test]
public void Test1()
{
    string test1 = "Test1";
    string test2 = "Test2";
    Assert.AreEqual(test1, test2);
}
```

Q12. Can we write a data-driven test using NUnit?

Ans. Yes, we can write data-driven test in NUnit by using TestCase attribute. The TestCase attribute specifies the input value to run the unit test.

```
[TestCase(1, "TestData 1")]
[TestCase(2, "TestData 2")]
public void TestMethod(int Id, string name)
{
   //do something
}
```

Q13. What is the use of the setup attribute in NUnit framework?

Ans. This attribute is used to identify the method that called before executing each test. In other words, it helps us to write the test initialization that required before the test executes.

Q14. How can we create a test for the controller that has service dependency?

Ans. We can get the service instance object using the GetService method of the service provider class. To test the controller having such a dependency, we can create an instance of a service provider class and get the required service instance.

In the following example, Controller having a dependency on ILogger service. So, I have created an object of the service provider and using the service provider class object of such services can be created.

```
public void TestMethod1()
{
    var serviceProvider = new ServiceCollection()
        .AddLogging()
        .BuildServiceProvider();

    var factory = serviceProvider.GetService<ILoggerFactory>();

    var logger = factory.CreateLogger<HomeController>();
    HomeController home = new HomeController(logger);
    ...
    ...
}
```

Q15. What is a Mock (or moq) object in a unit test?

Ans. A mock object is a dummy object that acts as a real object and this object can be controlled in test code. The moq (https://github.com/moq/moq4) is a library that allows creating mock objects in code. This library is also available on NuGet (https://www.nuget.org/packages/Moq/). This mock object supports all three test frameworks.

Example:

```
public void Test1()
{
     var mock = new Mock<IDataRepository>();
     mock.Setup(p => p.GetNameUsingId(100)).Returns("This is test");
     HomeController home = new HomeController(mock.Object);
     string result = home.GetNameUsingId(100);
     //Verify the result
}
```

Q16. What are the Integration tests?

Ans. It ensures that app components are correctly functional at the level that includes the supporting infrastructure like database and file system. ASP.NET Core supports integration tests using a unit test framework.

Q17. How Integration tests different from a unit test?

Ans. Following are the difference between Integration tests and unit test

- Unit tests use to test isolated app component such as individual class methods. Integration tests use to test two or more app components work together to produce an expected result
- Unit tests use fabricated components (Mock object) whereas Integration tests use infrastructure components such as actual database and file system
- Integration tests may take a longer time to execute as they interact with actual infrastructure components
- It is recommended to use a unit test instead of integration tests whereas possible and it is not recommended to use integration tests to check every possible permutation of data

Q18. What is the load test and stress test?

Ans. Load testing and stress testing are used to test web app performance and scalability. Load testing test web app can handle a specified number of user load for a certain scenario. The stress testing test web app scalability and stability when the app running under extreme conditions for a long period.

Q19. What are the tools used for web performance testing?

Ans. Following are the tools that can be used for web performance testing

- Visual Studio (load test project)
- Azure DevOps Test Plans service
- Apache JMeter
- ApacheBench (ab)
- Gatling
- Locust
- West Wind WebSurge
- Netling
- Vegeta

12
Deployment

Q1. What are the different techniques for hosting an ASP.NET Core application?

Ans. ASP.NET Core can be hosted in multiple ways such as

- Hosted with IIS/Azure App service
- Hosted with Nginx (Linux)
- Hosted with Apache (Linux)
- Hosted as a Windows service
- Hosted using HTTP.sys / Kestrel server

Q2. What is Kestrel?

Ans. The Kestrel is a web server that supports cross-platform. It included in the default in ASP.NET Core project templates. It supports the following features:

- HTTPS
- Opaque upgrade used to enable WebSockets
- Unix sockets for high performance behind Nginx

Q3. How to add Kestrel server in ASP.NET Core application?

Ans. The definition of Kestrel web server is under Microsoft.AspNetCore.Server.Kestrel package, however, it is included in the Microsoft.AspNetCore.App meta package.

In Program class (entry point of the project), the template code calls WebHost.CreateDefaultBuilder method that calls UseKestrel method to use Kestrel server.

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseKestrel(options =>
```

```
        {
            //set properties of Kestrel server
        });
}
```

Q4. Can we bind the TCP socket with Kestrel server?

Ans. Yes. Using Listen method, we can bind to a TCP socket and also, we can configure the SSL certificate. Same as it has ListenUnixSocket method to bind the Unix socket.

Q5. What is HTTP.sys server?

Ans. It is a Windows-based web server for ASP.NET Core. It is an alternative to Kestrel Server and it has some features that are not supported by Kestrel. It does not work with IIS due to it is incompatibility with the ASP.NET Core modules.

Q6. What are the features are supported by HTTP.sys server?

Ans. The following features are supported by the HTTP.sys server.

- Windows Authentication
- Response caching
- WebSockets
- Direct file transmission
- Port sharing
- HTTPS with SNI (Server Name Indication)
- HTTP/2 over TLS

It supports Windows 7 and Windows Server 2008 R2 and later

Q7. How to host an application using HTTP.sys server?

Ans. Following are the step to configure HTTP.sys server for ASP.NET Core application

- Add a reference to Microsoft.AspNetCore.Server.HttpSys assembly that contains the definition of HTTP.sys server, however, this library included in meta package.
- Configure HTTP.sys server using UseHttpSys method of WebHostBuilder in the main method of Program class. We can also specify the options for HTTP.sys server.

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
    .UseHttpSys(options =>
    {
        //specify the option for HTTP.sys server
    });
```

Q8. How to host ASP.NET Core application as a Windows service?

Ans. Followings are the step to host ASP.NET Core application as a Windows service

- Install Microsoft.AspNetCore.Hosting.WindowsServices package from NuGet

- This package contains RunAsService extension method that specified web application hosted as windows service

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args)
            .Build()
            .RunAsService();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

We can also handle service start / stop event for windows service.

Q9. What is ASP.NET Core Module?

Ans. It allows the application to run behind IIS in a reverse proxy configuration, however, IIS provides advanced web app security and manageability features. It can be work only with windows version Windows 7 or later and Windows Server 2008 R2 or later. This module is incompatible with HTTP.sys.

Q10. What is in-process and out-of-process hosting model in ASP.NET Core?

Ans. In-Process It is the default hosting model in ASP.NET Core. Here, IISHttpServer used instead of the Kestrel server. It hosts ASP.NET Core application inside the IIS worker process (w3wp.exe).
If your app is running out-side the ASP.NET Core worker process (using Kestrel server), it is referring as Out-of-process hosting.

References

This book has been written by referring to the following sites:

1. https://docs.microsoft.com/en-us/aspnet/core - Microsoft Docs - ASP.NET Core
2. https://stackoverflow.com/questions/tagged/asp.net-core - Stack Overflow - ASP.NET Core
3. https://www.dotnettricks.com/learn/aspnetcore - Dot Net Tricks - ASP.NET Core