ASP.NET Core Interview Questions and Answers

These interview questions are targeted for ASP.NET Core, ASP.NET Core MVC and Web API. You must know the answers of these ASP.NET Core interview questions to clear a .NET FullStack developer interview. C# Programming language is used to show examples.

1. Describe the ASP.NET Core.

ASP.NET Core is an open-source, cross-platform and high performance platform that allows you to build modern, Internet-connected and cloud enabled applications. With ASP.NET Core you can

- build web applications, IoT (Internet of things) apps, services and mobile Backends.
- run on .Net Core.
- You can do your development on Linux, Windows and MacOS.
- deploy your code to cloud or on-premises.

2. What are the benefits of using ASP.NET Core over ASP.NET?

ASP.NET Core comes with the following benefits over **ASP.NET**.

- Cross platform, provide ability to develop and run on Windows, Linux and MacOS.
- Open-source
- Unified Platform to develop Web UI and services.
- Built-in dependency injection.
- Ability to deploy on more than one server like IIS, Kestrel, Nginx, Docker, Apache etc
- cloud enabled framework, provide support for environment based configuration systems.
- Lightweight, High performance and modern HTTP request pipelines.
- well suited architecture for testability
- Integration of many client-side frameworks like Angular any version
- Blazor allow you to use C# code in browser with JavaScript code.

3. What is the role of Startup class?

Startup class is responsible for configuration related things as below.

- It configures the services which are required by the app.

- It defines the app's request handling pipeline as a series of middleware components.

```
// Startup class example
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

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

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        // other middleware components
    }
}
```

Startup class is specified inside the 'CreateHostBuilder' method when the host is created.
Multiple Startup classes can also be defined for different environments, At run time appropriate startup classes are used.

4. What is the role of ConfigureServices and Configure method?

ConfigureServices method is optional and defined inside startup class as mentioned in above code. It gets called by the host before the 'Configure' method to configure the app's services.
Configure method is used to add middleware components to the IApplicationBuilder instance that's available in Configure method. Configure method also specifies how the app responds to HTTP request and response. ApplicationBuilder instance's 'Use...' extension method is used to add one or more middleware components to request pipeline.
You can configure the services and middleware components without the Startup class and it's methods, by defining this configuration inside the Program class in **CreateHostBuilder** method.

5. Describe the Dependency Injection.

Dependency Injection is a Design Pattern that's used as a technique to achieve the Inversion of Control (IoC) between the classes and their dependencies. ASP.NET Core comes with a built-in Dependency Injection framework that makes configured services available throughout the application. You can configure the services inside the ConfigureServices method as below.

```
services.AddScoped();
```

A Service can be resolved using constructor injection and DI framework is responsible for the instance of this service at run time. For more visit ASP.NET Core Dependency Injection

5. Explain the request processing pipeline in ASP.NET Core.

For more about request processing pipeline for ASP.NET MVC visit Request Processing Pipeline.

6. What problems does Dependency Injection solve?

Let's understand Dependency Injection with this C# example. A class can use a direct dependency instance as below.

```
Public class A {
MyDependency dep = new MyDependency();

public void Test(){
dep.SomeMethod();
}
}
```

But these direct dependencies can be problematic for the following reasons.

- If you want to replace 'MyDependency' with a different implementation then the class must be modified.
- It's difficult to Unit Test.
- If MyDependency class has dependencies then it must be configured by class. If Multiple classes have dependency on 'MyDependency', the code becomes scattered.

DI framework solves these problems as below.

- Use Interfaces or base class to abstract the dependency implementation.
- Dependencies are registered in the Service Container provided by ASP.NET Core inside Startup class 'ConfigureServices' method.
- Dependencies are injected using constructor injection and the instance is created by DI and destroyed when no longer needed.

7. Describe the Service Lifetimes.

When Services are registered, there is a lifetime for every service. ASP.NET Core provides the following lifetimes.

- **Transient -** Services with transient lifetime are created each time they are requested from service container. So it's best suited for stateless, light weight services.
- **Scoped -** Services with scoped lifetime are created once per connection or client request. When using scoped service in middleware then inject the service via invoke or invokeAsync method. You should not inject the service via constructor injection as it treats the service behavior like Singleton.
- **Singleton** - Service with singleton lifetime is created once when first time the service is requested. For subsequent requests same instance is served by service container.

8. Explain the Middleware in ASP.NET Core.

The Request handling pipeline is a sequence of middleware components where each component performs the operation on request and either call the next middleware component or terminate the request. When a middleware component terminates the request, it's called **Terminal Middleware** as It prevents next middleware from processing the request. You can add a middleware component to the pipeline by calling **.Use...** extension method as below.

```
    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseRouting();
```

So **Middleware component** is program that's build into an app's pipeline to handle the request and response. Each middleware component can decide whether to pass the request to next component and to perform any operation before or after next component in pipeline.

9. What is Request delegate?

Request delegates handle each HTTP request and are used to build request pipeline. It can configured using Run, Map and Use extension methods. An request delegate can be a in-line as an anonymous method (called in-line middleware) or a reusable class. These classes or in-line methods are called middleware components.

10. What is Host in ASP.NET Core?

Host encapsulates all the resources for the app. On startup, ASP.NET Core application creates the host. The Resources which are encapsulated by the host include:

- HTTP Server implementation
- Dependency Injection
- Configuration
- Logging

- Middleware components

11. Describe the Generic Host and Web Host.

The host setup the server, request pipeline and responsible for app startup and lifetime management. There are two hosts:

- .NET Generic Host
- ASP.NET Core Web Host

.NET Generic Host is recommended and ASP.NET Core template builds a .NET Generic Host on app startup.
ASP.NET Core Web host is only used for backwards compatibility.

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

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

12. Describe the Servers in ASP.NET Core.

Server is required to run any application. **ASP.NET Core** provides an in-process HTTP server implementation to run the app. This server implementation listen for HTTP requests and surface them to the application as a set of request features composed into an HttpContext.
ASP.NET Core use the **Kestrel** web server by default. ASP.NET Core comes with:

- Default Kestrel web server that's cross platform HTTP server implementation.
- IIS HTTP Server that's in-process server for IIS.
- HTTP.sys server that's a Windows-only HTTP server and it's based on the HTTP.sys kernel driver and HTTP Server API.

13. How Configuration works in ASP.NET Core?

In ASP.NET Core, **Configuration** is implemented using various configuration providers. Configuration data is present in the form of key value pairs that can be read by configuration providers as key value from different configuration sources as below.

- appsettings.json - settings file

- Azure Key Vault
- Environment variables
- In-memory .Net objects
- Command Line Arguments
- Custom Providers

By default apps are configured to read the configuration data from appsettings.json, environment variables, command line arguments etc. While reading the data, values from environment variables override appsettings.json data values. 'CreateDefaultBuilder' method provide default configuration.

14. How to read values from Appsettings.json file?

You can read values from appsettings.json using below code.

```
class Test{
// requires using Microsoft.Extensions.Configuration;
 private readonly IConfiguration Configuration;
    public TestModel(IConfiguration configuration)
    {
        Configuration = configuration;
    }
// public void ReadValues(){
var val = Configuration["key"]; // reading direct key values
var name = Configuration["Employee:Name"]; // read complex values
}
}
```

Default configuration provider first load the values from appsettings.json and then from appsettings.Environment.json file.
Environment specific values override the values from appsettings.json file. In development environment appsettings.Development.json file values override the appsettings.json file values, same apply to production environment.
You can also read the appsettings.json values using options pattern described Read values from appsettings.json file.

15. What is the Options Pattern in ASP.NET Core?

Options Pattern allow you to access related configuration settings in Strongly typed way using some classes. When you are accessing the configuration settings with the isolated classes, The app should adhere these two principles.

- **Interface Segregation Principle (ISP) or Encapsulation:** The class the depend on the configurations, should depend only on the configuration settings that they use.
- **Separation of Concerns:** Settings for different classes should not be related or dependent on one another.

16. How to use multiple environments in ASP.NET Core?

ASP.NET Core use environment variables to configure application behavior based on runtime environment. launchSettings.json file sets `ASPNETCORE_ENVIRONMENT` to `Development` on local Machine. For more visit How to use multiple environments in ASP.NET Core

17. How Logging works in .NET Core and ASP.NET Core?

18. How Routing works in ASP.NET Core?

Routing is used to handle incoming HTTP requests for the app. Routing find matching executable endpoint for incoming requests. These endpoints are registered when app starts. Matching process use values from incoming request url to process the requests. You can configure the routing in middleware pipeline of configure method in startup class.

```
app.UseRouting(); // It adds route matching to middlware pipeline

// It adds endpoints execution to middleware pipeline
app.UseEndpoints(endpoints =>
{
endpoints.MapGet("/", async context =>
{
await context.Response.WriteAsync("Hello World!");
});
});
```

For more you can refer ASP.NET Core Routing

19. How to handle errors in ASP.NET Core?

ASP.NET Core provides a better way to handle the errors in Startup class as below.

```
if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}
else
{
app.UseExceptionHandler("/Error");
app.UseHsts();
}
```

For development environment, Developer exception page display detailed information about the exception. You should place this middleware before other middlewares for which you want to catch exceptions. For other environments `UseExceptionHandler` middleware loads the proper Error page.
You can configure error code specific pages in Startup class `Configure` method as below.

```
app.Use(async (context, next) =>
{
await next();
if (context.Response.StatusCode == 404)
{
    context.Request.Path = "/not-found";
    await next();
    }
    if (context.Response.StatusCode == 403 || context.Response.StatusCode == 503
|| context.Response.StatusCode == 500)
    {
    context.Request.Path = "/Home/Error";
    await next();
    }
    });
```

For more visit Error handling

20. How ASP.NET Core serve static files?

In ASP.NET Core, **Static files** such as CSS, images, JavaScript files, HTML are the served directly to the clients. ASP.NET Core template provides a root folder called `wwwroot` which contains all these static files. `UseStaticFiles()` method inside `Startup.Configure` enables the static files to be served to client.
You can serve files outside of this webroot folder by configuring Static File Middleware as following.

```
app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(env.ContentRootPath, "MyStaticFiles")), // MyStaticFiles
is new folder
        RequestPath = "/StaticFiles"  // this is requested path by client
    });
// now you can use your file as below
<img src="/StaticFiles/images/profile.jpg" class="img" alt="A red rose" />
 // profile.jpg is image inside MyStaticFiles/images folder
```

23. Explain Session and State management in ASP.NET Core

As we know HTTP is a stateless protocol. HTTP requests are independent and does not retain user values. There are different ways to maintain user state between multiple HTTP requests.

- **Cookies**
- **Session State**
- **TempData**
- **Query strings**
- **Hidden fields**
- **HttpContext.Items**
- **Cache**

24. Can ASP.NET Application be run in Docker containers?

Yes, you can run an ASP.NET application or .NET Core application in Docker containers.

- For Docker interview questions visit Docker Questions
- For more about .NET and Docker visit .NET and Docker and Docker images for ASP.NET Core

24. Explain Model Binding in ASP.NET Core.

25. Explain Custom Model Binding.

26. Describe Model Validation.

27. How to write custom ASP.NET Core middleware?

28. How to access HttpContext in ASP.NET Core?

29. Explain the Change Token.

30. How to used ASP.NET Core APIs in class library?

31. What is the Open Web Interface for .NET (OWIN)?

32. Describe the URL Rewriting Middleware in ASP.NET Core.

33. Describe the application model in ASP.NET Core.

34. Explain the Caching or Response caching in ASP.NET Core.

Caching significantly improves the performance of an application by reducing the number of calls to actual data source. It also improves the scalability. Response caching is best suited for data that changes infrequently. Caching makes the copy of data and store it instead of generating data from original source.
Response caching headers control the response caching. `ResponseCache` attribute sets these caching headers with additional properties. For more visit Caching in ASP.NET Core.

35. What is In-memory cache?

In-memory cache is the simplest way of caching by ASP.NET Core that stores the data in memory on web server.
Apps running on multiple server should ensure that sessions are sticky if they are using in-memory cache. Sticky Sessions responsible to redirect subsequent client requests to same server. In-memory cache can store any object but distributed cache only stores `byte[].`
`IMemoryCache` interface instance in the constructor enables the In-memory caching service via ASP.NET Core dependency Injection.

36. What is Distributed caching?

Applications running on multiple servers (Web Farm) should ensure that sessions are sticky. For Non-sticky sessions, cache consistency problems can occur. **Distributed caching** is implemented to avoid cache consistency issues. It offloads the memory to an external process. Distributed caching has certain advantages as below.

- Data is consistent across client requests to multiple server
- Data keeps alive during server restarts and deployments.
- Data does not use local memory

`IDistributedCache` interface instance from any constructor enable distributed caching service via Dependency Injection.

37. What is XSRF or CSRF? How to prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core?

Cross-Site Request Forgery (XSRF/CSRF) is an attack where attacker that acts as a trusted source send some data to a website and perform some action. An attacker is considered a trusted source because it uses the authenticated cookie information stored in browser.
For example a user visits some site 'www.abc.com' then browser performs authentication successfully and stores the user information in cookie and perform some actions, In between user visits some other malicious site 'www.bad-user.com' and this site contains some code to make a request to vulnerable site (www.abc.com). It's called cross site part of CSRF.
How to prevent CSRF?

- In ASP.NET Core 2.0 or later FormTaghelper automatically inject the antiforgery tokens into HTML form element.
- You can add manually antiforgery token in HTML forms by using `@Html.AntiForgeryToken()` and then you can validate it in controller by `ValidateAntiForgeryToken()` method.
- For more you can visit Prevent Cross-Site Request Forgery (XSRF/CSRF)

38. How to prevent Cross-Site Scripting (XSS) in ASP.NET Core?

39. How to enable Cross-Origin Requests (CORS) in ASP.NET Core?

40. What is the Area?

Area is used to divide large ASP.NET MVC application into multiple functional groups. In general, for a large application Models, Views and controllers are kept in separate folders to separate the functionality. But Area is a MVC structure that separate an application into multiple functional groupings. For example, for an e-commerce site Billing, Orders, search functionalities can be implemented using different areas.

41. Explain the Filters.

Filters provide the capability to run the code before or after the specific stage in request processing pipeline, it could be either MVC app or Web API service. Filters performs the tasks like Authorization, Caching implementation, Exception handling etc. ASP.NET Core also provide the option to create custom filters. There are 5 types of filters supported in ASP.NET Core Web apps or services.

- **Authorization filters** run before all or first and determine the user is authorized or not.
- **Resource filters** are executed after authorization. `OnResourceExecuting` filter runs the code before rest of filter pipeline and `OnResourceExecuted` runs the code after rest of filter pipeline.
- **Action filters** run the code immediately before and after the action method execution. Action filters can change the arguments passed to method and can change returned result.
- **Exception filters** used to handle the exceptions globally before wrting the response body
- **Result filters** allow to run the code just before or after successful execution of action results.

For more visit Filters in ASP.NET Core.

42. Describe the View components in ASP.NET Core.

43. How View compilation works in ASP.NET Core?

44. Explain Buffering and Streaming approaches to upload files in ASP.NET Core.

44. How does bundling and minification work in ASP.NET Core?

For more visit Bundling and Minification.

44. How will you improve performance of ASP.NET Core Application?

1. Describe the ASP.NET Core MVC.

ASP.NET Core MVC is a framework to build web applications and APIs. It's based on Model-View-Controller (MVC) Design Pattern. This design pattern separate an application into three main components known as Model, View and Controller. It also helps to achieve SoC (Separation of Concern) design principle.
ASP.NET Core MVC is light weight, open-source and testable framework to build web applications and services.

2. Explain the Model, View and Controller.

ASP.NET MVC has three main group of components Model, View and Controller, Each one has his own responsibilities as below.

- **Model -** It contains the business logic and represents the state of an application. It also performs the operation on the data and encapsulates the logic to persist an application state. **Strongly-typed Views** use **View-Model** pattern to display the data in the view.
- **View -** It's responsible to present the content via User interface. It does not contain much logic and use **Razor View Engine** to embed .NET code into view. If you need to perform much logic to display the data then prefer to use View Component, View Model or View Template for simplify view.
- **Controller -** It's responsible to handle user interactions, It works with model and select the view to display. Controller is the main entry point that decides the model with which it works and decide which view it needs to display. Hence it's name - Controller means controls user inputs and interactions.

3. Explain View-Model.

ViewModel is used to pass a complex data from controller to view. ViewModel data is prepared from different models and passed to view to display that data. For example, A Complex data model can be passed with the help of ViewModel.

```
Class Author{
public int Id {get;set;}
public Book Book {get;set;}
}
Class Book{
public string Name {get;set;}
public string PublisherName {get;set;}
}
```

This Author and Book data can be passed to view by creating Author ViewModel inside controller.

4. Explain strongly-typed views.

Strongly-typed views are tightly bound to a model. for example, In above question if you want to pass the author data to view then you need to write below code for type checking in your view. `@model Author`. Controller can pass strongly type model to view that enables type checking and intelliSense support in view.

5. Explain Partial Views.

6. How routing works in MVC application?

7. Describe Attribute based routing.

Attribute Routing gives you more control over the URIs in your web application. MVC 5 supports this attribute based routing where attributes are used to define the routes. You can manage resource hierarchies in better way using attribute based routing. Attribute based routing is used to create routes which are difficult to create using convention-based routing. For example below routes.

```
[Route("customers/{customerId}/orders")]
public IEnumerable GetOrdersByCustomer(int customerId) { ... }
        .
        .
        .
[Route("customers/{customerId}/orders/orderId")]
public IEnumerable GetOrdersByCustomer(int customerId, int orderId) { ... }
```

8. Explain dependency injection for controllers.

9. How ASP.NET Core supports dependency injection into views?

10. How will you unit test a controller?

11. What is Cache Tag Helper in ASP.NET Core MVC?

12. How validation works in MVC and how they follow DRY Pattern?

13. Describe the complete request processing pipeline for ASP.NET Core MVC.

Some General Interview Questions for ASP.NET Core

1. How much will you rate yourself in ASP.NET Core?

When you attend an interview, Interviewer may ask you to rate yourself in a specific Technology like ASP.NET Core, So It's depend on your knowledge and work experience in ASP.NET Core.

2. What challenges did you face while working on ASP.NET Core?

This question may be specific to your technology and completely depends on your past work experience. So you need to just explain the challenges you faced related to ASP.NET Core in your Project.

3. What was your role in the last Project related to ASP.NET Core?

It's based on your role and responsibilities assigned to you and what functionality you implemented using ASP.NET Core in your project. This question is generally asked in every interview.

4. How much experience do you have in ASP.NET Core?

Here you can tell about your overall work experience on ASP.NET Core.

5. Have you done any ASP.NET Core Certification or Training?

It depends on the candidate whether you have done any ASP.NET Core training or certification. Certifications or training are not essential but good to have.